EXTERNAL LABELING AS A FRAMEWORK FOR ACCESS CONTROL

by

Thomas H. Rozenbroek
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Information Technology

Committee:

[illegible]	Dr. Edgar Sibley, Dissertation Director
Alexander Levis	Dr. Alexander Levis, Committee Member
[illegible]	Dr. Duminda Wijesekera, Committee Member
Sanjeev Setia	Dr. Sanjeev Setia, Committee Member
[illegible]	Dr. Daniel Menascé, Senior Associate Dean
[illegible]	Dr. Lloyd J. Griffiths, Dean, Volgenau School of Engineering

Date: 04/13/2012

Spring Semester 2012
George Mason University
Fairfax, VA

External Labeling as a Framework for Access Control

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Thomas H. Rozenbroek
Master of Science
Naval Postgraduate School, 2012
Master of Science
Strayer University, 1996
Bachelor of Engineering
Stevens Institute of Technology, 1983

Director: Edgar Sibley, University Professor
Department of Information and Software Engineering

Spring Semester 2012
George Mason University
Fairfax, VA

DEDICATION

This dissertation is dedicated to my loving wife, Elizabeth. Her understanding, patience, and unwavering support allowed me to complete these studies. With my love always.

ACKNOWLEDGEMENTS

I wish to acknowledge and thank the many people, who were critical to completing my dissertation.

First, I would like to thank my dissertation director, Dr. Edgar Sibley, and committee members, Dr. Alexander Levis, Dr. Duminda Wijesekera, and Dr. Sanjeev Setia. Gentlemen, thank you for your direction, support, suggestions and guidance. Without your involvement, I would have been lost.

Second, I would like to thank the facility and staff at George Mason University. You added greatly to my time at GMU.

Finally, I would like to thank my family. Your support, tolerance, and patience during my studies were critical to my completing these studies.

Thank you one and all.

TABLE OF CONTENTS

LIST OF TABLES

<u>**LIST OF FIGURES**</u>

LIST OF ABBREVIATIONS AND SYMBOLS

Abbreviation / Symbol	*Definition*
$\wedge\,(A, B, \ldots)$	Logically ‘AND’ all elements
$\vee\,(A, B, \ldots)$	Logically ‘OR’ all elements
$\cap\,(A, B\ \ldots)$	The Intersection of any element that is common to every set
$\cup\,(A, B, \ldots)$	The Union of all elements contained in every set
¬(A)	Negates A
ABAC	Attribute Based Access Control
ACL	Access Control List
CND	Computer Network Defense
DDN	Defense Data Network
DoD	Department of Defense
DoDD	Department of Defense Directive
DoDI	Department of Defense Instruction
DoE	Department of Energy
FIPS	Federal Information Processing Standard
FTP	File Transfer Protocol
GOSAC-N	Government Open Source Access Control - Navy
HIPAA	Health Insurance Portability and Accountability Act
I&A	Identification and Authentication
IAC	Information Assurance Control
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISOC	Internet Society
IT	Information Technology
MAC	Mandatory Access Control
NIST	National Institute of Standards and Technology
NNPI	Naval Nuclear Propulsion Information
NO_FORN	No Foreign Access
NSA	National Security Agency
PBAC	Policy Based Access Control
PII	Personnel Identifying Information
RBAC	Role Based Access Control
RFC	Request For Comment

RI	Reference Implementation
SFF	Secure File Format
SP 800	Special Publications 800 (series)
TCP	Transmission Control Protocol
UML	Unified Modeling Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language

ABSTRACT

EXTERNAL LABELING AS A FRAMEWORK FOR MANAGING OBJECTS

Thomas H. Rozenbroek, Ph.D.

George Mason University, 2012

Dissertation Director: Dr. Edgar H. Sibley

With the ever increasing volume of data existing on and passing through on-line resources together with a growing number of legitimate users of that information and potential adversaries, the need for better security and safeguards is immediate and critical. Currently, most of the security and safeguards afforded on-line information are provided externally by the infrastructure and are based on security information that is also maintained by that infrastructure. As the infrastructure increases in size and interconnection, the more insecure the movement of information throughout the infrastructure becomes. The interconnection of different infrastructures means that there is a need for greater need for coordination between the infrastructures. Unfortunately, this is not always possible.

An alternative to strict reliance on the infrastructure is to include security attributes along with the objects that need to be secured. It is possible to improve the security of this information by attaching the external security labels to these objects. These external

labels, which contain the required security information, are transferred as an integral part of the object's migration throughout the infrastructure. This dissertation presents a framework for using external labels that will provide better safeguards for securing information. This framework is object based and as such is applicable to anything, virtual or real-world, that can be represented or treated as 'an object'. It discusses how each entity within the infrastructure must be labeled to support the increase in security as well as provide the framework for assessing the user and system labels against those of the information objects.

This dissertation presents and details the key features of the labeling solutions and explains the reasons why each of the features is necessary for the labeling framework to secure objects. The framework is based on securely attaching labels to the objects, while still allowing for the separation of the labels from the object. This separation must take place without the lessening the security afforded the objects. The second feature of the framework is the treatment of the object labels, themselves. The framework applies labels to the objects being protected, the users requesting access to the objects, and the end user and intermediate systems handling the objects. This provides for better management of the environment and therefore greater security for the objects. The final key feature of the framework is abstract nature of the objects and their labels. This framework places no limitation on either the objects being secured or the content of the labels. Any information that can be treated as an object can be handled by this framework. Also, any rules that can be modeled can be supported by the framework. This framework as proposed by this dissertation includes several types of labels that can be used to secure objects. This types of labels presented can be easily extended to meet the unique needs of the infrastructure without lessening the framework, itself.

Additionally, this dissertation extends the use of labels to address security problems beyond simple access control. It demonstrates how object labeling can be used to secure multiple objects in a confederated manner, rather than as individual objects. Information is no longer being processed in small collections, but rather as large collections of information gathered from numerous sources. This framework is able to be managed these large collections in an effective manner. Further extensions include using labels to handle data aggregation and the avoidance of sensitivity escalation. Having access to larger collections increases the risk that too much information can be collocated or accessed at the same time. This dissertation presents tools and techniques for using the framework to minimize and control how information is aggregated in order to reduce these risks. Also, the framework can be used to insure that information aggregates don’t result in the creation of information set which are “more” sensitive than the original information.

1. INTRODUCTION

The Study of Security as it relates to information assurance and system/network security is a large, complex, and ever changing field. While the security field is changing, there are several constants that form the foundation for all work in this field. Harris details the "Security Triad" [1]; in it, each leg of this triad represents a focus area for security; Confidentiality, Integrity, and Availability. The U. S. Department of Defense adds two additional legs, authentication and non-repudiation to these three. DoD Directive 3600.1 [2] defines "Information Assurance" as:

> *Measures that protect and defend information and information systems by ensuring their availability, integrity, authentication, confidentiality, and non-repudiation. This includes providing for restoration of information systems by incorporating protection, detection, and reaction capabilities. Note: CND provides operational direction and guidance through global network operations and defense for employment of IA in response to a CND alert or specific threat.*[1]

In one way or another, every security solution is implemented to provide one or more of these areas of concern.

Unfortunately, traditional use of external metadata as object tags or labels does not lend itself to autonomous security. The reasons for this are numerous, but are most often

[1] [2] Section E2.1.14 pg 13

related to the weak linkage between the object labels and the objects to which they are "attached”. This linkage is considered “weak” because it can be easily broken. Once the links are broken, the integrity of the link is destroyed. With the integrity of the link destroyed, the external metadata is of no use for security operations. For example, transporting an object between two hosting environments can cause the link between object and its label to be broken simply because one or more elements of the metadata have different ontology in the two hosting environments or they may not be captured at all.

In a homogenous processing environment[2], the linkage between an object and its external metadata can be more easily maintained, but only as a deliberate action on the part of the transport mechanism and only if the metadata values that are passed as part of the transport mechanism have the same meaning in both processing environments. For example, in the Linux/Unix operating systems, users are tracking using a User ID or UID, which is typically denoted by an unsigned integer. Each unique user on the system is assigned a different UID value.[3] If the user has different UIDs on the different systems or if the same UID on different systems is assigned to different users, then the transport mechanism must account for this and change the UID as part of the transport process. Some transport mechanisms, such as the File Transfer Protocol or FTP[4], perform these actions; other transport mechanisms do not.

[2] A local area network with host systems all using the same version of the same operating systems and using the same version of the operating systems based transport mechanism is an example of a homogenous processing environment.

In the case of a heterogeneous processing environment[3], the integrity and security of these linkages is strained even further. This is due to difference in how the different processing environments handle these external attributes. In an extremely diverse (and therefore heterogeneous processing) hosting environment, such as the Internet or a cloud computing facility, the integrity of the links is considerably harder to maintained.

My framework avoids many of the security problems that heterogeneous environments create by abstracting the objects and object labels. This eliminates the reliance on the underlying infrastructure for many attributes and features. I have investigated object labels, their association to the objects to which they are attached, and the security requirements for using object labels to provide security. I present a solution for binding security labels to the objects. One of the goals of my research is to present a working framework that can be used to implement object security based exclusively on the object labels. This removes the dependence on operating system level file attributes.

After a labeling framework has been presented, I extended to use of object labels to allow then to address the securing of multiple labeled objects. Securing of multiple labeled objects introduces additional challenges and issues that need to be addressed. Examples of these challenges include the aggregation of information, and escalation of sensitivity and escalation avoidance.

[3] A local area network with host systems running different operating systems is an example of a heterogeneous processing environment.

As part of my dissertation, I have presented a framework for implementing label based security as part of the existing infrastructure. My framework differs from many secure labeling solutions, like Trusted Solaris or SELinux, in that it can be hosted as part of existing operating environments. My framework is tolerant of the infrastructure not being completely “label aware” or secure. It further supports infrastructures with different levels of trust.

In cyberspace, the use of metadata tags is ubiquitous. File system attributes, such as creation date, owner, group, and permissions, are external metadata that are used to describe characteristics of or rules for gaining access to the file with which they are associated. These attributes form the foundation by which operating systems control access to the system objects. A key limitation of this approach is the inflexibility, which is imposed by using these operating system level controls. The approach that my framework takes is to implement labeling of information objects, implement labeling and user and system objects and establish the rules by which users are granted access to the objects. By defining the rules for access in this manner, the framework can be tailored to meet the needs of the infrastructure. A second key feature of my approach is my framework is not constrained by the type or format of information being protected or by a pre-established set of label attributes. My framework will protect any type of information and any number of information types concurrently. Additionally, the data that is carried by the object labels is not pre-determined, rather it is established when the framework is implemented. The framework abstracts both objects and object labels to permit greater flexibility.

With a security framework developed that is based exclusively on object labels, I studied two extensions. The first of these extensions focuses on the security of objects when taken as a collective. I present how object labels can be used to control and protect aggregation of data objects. The second extension includes time and history in managing the security of objects.

Data aggregation is the basis for creating information. The value of facts and single pieces of data is greatly increased if they can be collected and considered as a whole. This will lead to situations where the amount of collected data needs to be considered in totality. I have included a section on information aggregation and how labels should be used to insure that the aggregated information is properly secured. For example, if information with different level of classification is being aggregated, the resulting object should be labeled so as to account for the classification of each object in the aggregate.

In the second extension, I have considered how history and time should be considered when securing objects. I have shown that controlling when information should be released can account for the time since last object was accessed. Also, I have shown how labels can be used to make access decisions that take into account the information that the user already has. This permits user knowledge and "Chinese wall security problems" to be addressed is a secured manner.

It is also important to outline what is not covered in my dissertation. It does not attempting to offer a universal or "One Size Fits All" solution for object security. The range of problems and different requirements for security make that impractical.

Additionally, my dissertation includes sample sets of labels and modeling to demonstrate and clarify the work being presented. Neither the models nor the pseudo-code should be considered to be production ready. Rather that they should serve as aids in visualizing and verifying the models presented as well as the guides for developing production solutions to specific problems

2. <u>BACKGROUND/HISTORY</u>

Over the last 40 years, there has been much discussion about how security labels could be used to protect information. Published works by Denning [5], Bell and La Padula [6], Biba [7], and Brewer and Nash [8] as well as publications by the U. S. Department of Defense [9] [10] and the Internet Engineering Task Force [11] [12] have all discussed labeling and proposed abstract and theoretical applications for labels. However, there are very few practical implementations that make use of labels as a security mechanism. In fact, a Defence R&D Canada – Ottawa report [13], published in 2005, states:

> *"Although some research and development has been conducted into security labeling over the past thirty years, much of it as part of MultiLevel Security (MLS) initiatives, there is currently little commercial support for security labels and trusted binding mechanism."*[4]

This report goes on to state that:

> *"Furthermore, no security labeling standard or trusted binding mechanism has emerged as a de-facto standard suitable for a variety of object classes."*[5]

Much of the reason for the lack of standardization in this area could be due to the lack of need for any. When labeling was being discussed by Denning, Biba, and others in the

[4] [13] Pg iii

[5] [13] Pg iii

1970s, there was no global computer network. Computing was mainframe based and the requirement to share information between two or more computers was almost non-existent.

It was not until the early 1980s, the first computer networks began to appear. However, their ubiquity and openness was far from what it is today. RFC 1296 [14] states that in August of 1981, there were 213 network connected hosts for the entire Internet.[6] The DDN Directory [15], published in 1984, included a network diagram that only requires the back of the front cover to clearly depicts the all of the military, commercial, and academic sites that composed the entire ARPAnet/MILnet. A section of this directory enumerates all 371 registered hosts. By Jan 1992, that number had risen to only 727,000 [14]. Of these, 243,020 addresses were assigned to educational institution. The focus at this time was not on security, but rather on the exchange of information. In 1988, Comer wrote:

> *"To appreciate internet technology, think of how it affects research. Imagine for a minute the effects of interconnecting all the computers used by scientists. Any scientist would be able to exchange data resulting from an experiment with any other scientist. It would be possible to establish national data centers to collect data from natural phenomena and make the data available to all scientists. Computer services and programs available at one location could be used by scientists at other locations. As a result, the speed with which scientific investigations proceed would increase. In short, the changes would be dramatic."*[7]

---

[6] At this time, this "global" network was referred to as the "Defense Data Network" (DDN). Over the years, it has gone through several name changes. It has also been known as the "APRAnet" and "MILnet."

[7] [16] pg 1-2

Throughout the rest of his book, *“Internetworking with TCP/IP”* [16], Comer discusses protocols for exchanging information as well as other communications concerns, but does not discuss security. In fact, the word, ‘security’, does not appear in the book’s index. The focus was exclusively on exchanging information.

During this same time period, several network specifications [12] [17] were published that did include provision for including security labels or tags as part of the network traffic. However, the standards for network traffic didn’t include provisions for securing the traffic. This means that all of network traffic was transmitted using plaintext without the use of encryption. The lack of protections, such as encryption, meant that any safeguards that these labels and tags would have provided could be easily circumvented. The use of ‘Network Sniffers’[8] can provide a complete transcript of any communication traffic being exchanged. It would not be until safeguards, like encryption, were incorporated into network communications that using security tags and labels would have had any value as a security mechanism.

A second reason for the lack of available products and support is the diversity of information types that would need to be supported by a single solution. And while there are some commercial and open-source products that utilize labeling, the value provided

[8] Network Sniffers are dedicated hardware devices or programs loaded on computers that are designed to capture, decompose, analyze, and display network traffic. Numerous commercial and open source offerings are available. Additionally, many operating systems provide the functionality to capture, store and display network traffic without additional software.

by the use of security labels can only be realized by the use of those products, exclusively.

In the physical world, the need to protect information from unauthorized disclosure is older than this country. Evidence of the use of various techniques to protect information can be found throughout antiquity. Coded message, discovered in the tombs of Egyptian nobility and ciphers used by Roman armies under Caesar are just two examples of how information has been protected throughout history.[18] From its formation, this country has had the requirement to protect information from unintended parties. This country's founders clearly recognized the need to protect information both on the battlefield and in the conduct of government. George Washington was known to "label" communications as a means of protecting it from disclosure. [19] Additionally, the Constitution of the United States includes provisions to permit the withholding records and correspondence from the public disclosure.[9]

While the need for secrecy has been recognized for all of this country's history, it would not be until the First World War that the first formal system of classifying information was put in place. [20] Under this system, there were 3 levels of classification. Since this time, this country's classification systems have under gone numerous changes. In fact, there has not always been one single set of rules for classify information or even a single classification hierarchy. Within the U. S. federal government, different departments in

[9] U. S. Constitution Article I Section 5

the executive branch have had different classification systems. Today, the U. S. government has a single set of classification levels and rules for determining what should be given a specific level. Executive Order 12958 [21] was originally signed by President Clinton in 1995, and was amended by President Bush with the signing of Executive Orders 13292 [22] in 2003. There are 3 levels of classification: TOP SECRET, SECRET, and CONFIDENTIAL. Any information not deemed critical enough to warrant a CONFIDENTIAL or higher classification is considered UNCLASSIFIED. This is not to say that the unauthorized release of unclassified information does not have a negative impact on the nation's security, it does.

In addition to the hierarchical classification structure detailed by Executive Order 12958 et al, there are additional sets of labels generally referred to as "Categories" that can be applied to objects. While Categories are not hierarchical in nature, they are used to limit access to information. Examples of category labels include, but are not limited to: PII[10], HIPAA[11], NNPI[12], NO_FORN[13], etc. Conditions detailed by both the Classification Levels (or simply Levels) and Category labels must be satisfied in order that access to the information to be granted.

---

[10] Personally Identifiable Information (PII)

11 Used to identify information protected from disclosure by the Health Insurance Portability and Accountability Act (HIPAA)

12 Used to identify Navy Nuclear Propulsion Information (NNPI)

13 Used to identify information that needs to be protected from disclosure to non-United States citizens (NO_FORN)

3. SECURITY CONTROLS THAT EXTERNAL LABELS CAN PROVIDE

3.1. Introduction

My dissertation considers security problems that can be addressed by using external security labels. These problems can be assigned to one of three groups; *Simple Access Control*, *Information Aggregation*, and *Aggregation Avoidance*. Each group, while making use of object labeling to provide security to the object, does so in a different manner. Also, it is worth noting that simply applying labels to objects does not increase their security. The application of security labels to object must be done in conjunction with rules for interpreting those labels, as well as processing elements within the infrastructure.

At this point, it is helpful to have a good understanding of the different types of objects and labels that are presented. *Appendix A - Labelling Framework* is a primer on how objects and object labels are related and represented in my dissertation.

For each of the three groups, *Appendix E - Example Problems that can be solved with Labeled Objects* discusses example problems that can be addressed by using object labels and the supporting framework. Additionally, *Appendix F - Label Based Access Control Demonstration System* details a demonstration system that implements several of the concepts that are presented in this dissertation.

3.2. Simple Access Control

Confidentiality is one of the three key attributes for information security solutions. It focuses on insuring that access to information is granted only to those individuals and processes that are authorized to have access and denying access to those that do not.

Simple Access Control, uses object labels to provide “simple” access control for the object. This type of usage is in line with the DoD safeguards for protecting classified documents [23] [24], or the National Security Agency’s (NSA) “Rainbow Series” publications.[14] In general, simple access control focuses on each object, individually. There is no consideration given to any other objects in the system. Traditional access control mechanisms provided by a computing environment also follow this model. (Access to a file or process is based solely on the attributes of the each file or process and the permissions of the user making the request.)

3.2.1. Host Based Solutions

In an object label centric security environment, security information about the object is carried by the labels connected to the object. Security information about the user and the

[14] The Rainbow Series discusses more than just system and information integrity and confidentiality. Topics, like covert channels, are also discussed. However, they are outside of the scope of my research.

user’s system are carried by a second set of labels that are associated with that user and system and connected to any request for access that the user makes or is made on the user’s behalf. The User/System labels are compared against the labels that are attached to the object and based on a set of access rules, the user is granted access to the object, if the rules are satisfied. If the access rules are not satisfied, then access to the object is denied.

3.2.2.Cross Domain Solutions

Cross Domain Solutions represent a network level implementation of simple access control, rather than a host or system based version of simple access control. They aren’t co-located on the system with the information management system, but are placed at strategic locations throughout the infrastructure. Usually, they act as gateway or firewalls between two networks with vastly different security postures and act as ‘checkpoints’ through which the object must pass.

3.3.Information Aggregation

3.3.1.Introduction

One of the key benefits of the Internet and computers in general is their inherent ability to collect, store, process, and present a large volume of information from multiple sources. By having a large volume of information available to a computer user, the aggregation of information is inevitable. Object labeling and the use of an associated framework can provide greater control over the aggregation of information.

Dictionary.com defines an aggregation as:

"a group or mass of distinct or varied things, persons, etc" [25]

When applied to information, it can be viewed as the collecting of information from one or more data sources with the expectation that the aggregated information has a value equal to or greater than the information, if considered as individual entities.

Information Aggregation, uses object labels and external processing to create new labels that reflect the security characteristics of the aggregation of the original objects. The security attributes of this new aggregated object are carried by additional object labels that are attached to the aggregated object. The actual values of the new security attributes are calculated by an external process, and are based on the attribute values of the original object labels. Rather than determining whether access to the object should be granted, the primary focus of this group is on re-labeling the collections of objects as a whole.

Once a new aggregated object has been created, the new object label is used to determine if access should be granted. Because the creation of an aggregated object and determining if access to the aggregated object should be granted are separate activities, I address the problem of accessing an aggregated object as two problems. The first is the creation of the aggregated object with new aggregation labels and the second is an access control based on the aggregation labels. The first problem is discussed here. The second problem becomes a case of applying simple access control using the aggregated labels and that has been previously discussed.

An additional advantage of keeping the creation of aggregated labels separate from the access control is these two operations can be conducted at different times or by different systems.

When considering information aggregation, there are two different types of aggregation that can take place. These involve either

Concatenated Aggregation; or
Cumulative Aggregation.

3.3.2.Concatenated Aggregation

Concatenated Aggregation is the simpler form of aggregation. For this type of aggregation, there is no increase in security level or lowering of the integrity level as a result of aggregating multiple data objects. For Concatenated Aggregation, the labels that are created are simply a concatenation or superset of all of the labels of the individual data objects. Once concatenated, set theory is applied to produce an aggregated label that reflects the concatenated values. A well documented example of Concatenated Aggregation can be seen in how DoD 5200.1-PH DoD Guide to Marking Classified Documents [23] handled document labeling.

While DoD 5200.1-PH only discusses classified documents, the guidance that it provides can be equally applied to other forms of Concatenated Aggregation. When the classification level of the individual objects is aggregated, then the resulting label represents the highest classification of any object in the collection.[15] If the measures of integrity of the information in an object is carried by an object label are being concatenated, then the aggregated label for integrity would carried the lowest integrity for any information in the aggregated object.[16]

3.3.3.Cumulative Aggregation

Cumulative Aggregation refers to the aggregation of information that produces a result that may carry a higher level of classification. Quist, who refers to aggregation as "compilation", states:

> *A compilation [aggregation] of many different items of information classified at one level (e.g., Confidential) should be classified at a higher level (e.g., Secret) if the total damage caused by the unauthorized release of all of the items of information would equal or exceed the damage caused by the release of one item of information classified at that higher level.*[17]

This type of aggregation has been associated with "Mosaic Theory" and "Compilation Theory" Whichever theory is applied, the ramification of this type of aggregation are far reaching. From the requirement to protect the information differently to the rules for public release, this type of aggregation represents a challenge to the information

[15] This is in keeping with the labeling work done by Bell and La Padula

[16] This is in keeping with the labeling work done by Biba

[17] [26]Pg 15

assurance community. In many cases, a collection of objects can't simply be treated as individuals, but rather must be viewed as a whole.

As a practical matter, the rules for determining are what point does the classification level change is very poorly defined. In fact, a May 2005 Department of Energy (DoE) communiqué [27] goes so far as to state that:

> *"Compilations do not usually have a basis in guidance and are, therefore, more difficult to determine."* [18]

and goes on to state:

> *"Compilation, on the other hand, consists of unclassified facts that by selection, arrangement, or completeness of the information add sufficient value to merit classification. Compilations may involve information over a period of time and, hence, require greater effort to identify the classified information"*[19]

This assessment reverses an assessment made by Quist[26]. Quist presented the argument that aggregation (compilation) of unclassified information would have no impact in its classification level. His paper offered this discussion:

> *"Let us assume that the damage caused by the release of an item of Confidential information would a "1" on a arbitrary scale of damage. (For Secret and Top Secret information, the damage value would be greater.) The release of an unclassified item of information would cause no (zero) damage to our national security (by the definition of what constitutes classified information). Therefore, no matter how many items of unclassified information are compiled (added together), the sum of the*

[18] [27] Pg 1

[19] [27] Pg 4

> *damages caused by their release would still be zero and the compilation should not be classified."*[20]

Perhaps the biggest change to have occurred between the publication of Quest's work and the DoE communiqué is an acknowledgement that the inappropriate release of unclassified information can still be damaging to national security. The introduction and use of categories, such as "For Official Use Only", "Restricted Data", and "Controlled Unclassified Information" reflect the importance of some types of unclassified information and the need to insure its protection.

However, the labeling of unclassified information does not address the problem of information aggregation. Beyond the labeling of the information, rules for determining how much unclassified information must aggregated before a classified object is created need to be established. Finally, there needs to be a framework within which the rules can be applied to the information in the labels. The formulation of the rules for assessing the security classifications is not presented as part of this dissertation, but the framework for processing these rules is.

[20] [26] Pg 5

3.4. Aggregation Avoidance

3.4.1. Introduction

For simple access control, the execution of the rules for granting access to an object will always return the same result. For many problems, this is sufficient. However, as the volume of information that is available on-line increases, the need to insure that information consumers do not amass too much information too quickly also increases.

Aggregation Avoidance, uses object labels and external processing to actively regulate or limit access to collections of labeled objects. While the *Simple Access Control* group's use of object labels also focuses on regulating or limiting access, Aggregation Avoidance adds a "history" element to the processing. It is intended to make access decisions based not just on the object's labels, but takes into account the prior object access operations that have been performed. This history, which may extend over a wide range of times based on the security requirements of the protected objects, is intended to regulate the release of information. Aggregation Avoidance builds on the work done for Information Aggregation.

Two problems in aggregation avoidance are presented as part of my dissertation.

3.4.2.Informational “Critical Mass” Avoidance

As introduced as part of the Cumulative Aggregation, it is possible for the label values of an aggregated object to be different than that of any of the individual objects from which the aggregate is composed. While it is desirable to be able to label the aggregated object with a new and accurate security label, there is also a need to prevent the creation of aggregated objects with higher label values.[21] The need for this can be found in Executive Orders, and DoD polices/directives. All of which mandate that classified/sensitive information be subject to more stringent protections to insure that unauthorized release of this information does not occur. For DoD IT systems, the current guidance from DoDI 8500.2[28] prescribes a different set of information assurance controls (IACs) for classified and unclassified systems. Additionally, as part of the certification and accreditation process, each system’s accreditation includes a statement concerning the level and type of information that the system can process. Processing information with a security level higher than that for which the system is accredited is considered a security breach or “spillage”. A spillage is said to have occurred if even a single classified document or object has been transferred inappropriately.

A second type of security breach results from too much information being hosted on a single system. In this case, none of the individual information objects are classified high enough to violates the system’s accreditation statement. However, when considered in

[21] In the case of document classification, this would mean a higher classification level.

the aggregate, their aggregated security level exceeds that for which the system is accredited and a spillage has occurred. Quist [29] presents a case where the operational readiness of an individual military unit is unclassified, however when collected and aggregated together, the result (a single statement of operational readiness for the entire U. S. military) is a TOP SECRET document. If the system hosting these readiness reports was not cleared for TOP SECRET, then a security breach has occurred. Clearly, there is a requirement to insure that too much information (informational "critical mass") is not collected in a single location.

The U.S. Nuclear Regulatory Commission (NRC) defines "Critical Mass" as:

> *"The smallest mass of fissionable material that will support a self-sustaining chain reaction." [30]*

Applying the same principle in terms of Information Assurance, one can define "Informational Critical Mass" as the minimum amount of aggregated information required to change the level of security classification or sensitivity of the entire set of information. For example, the current U. S. classification system, there would be 3 Informational Critical Mass Values. There is one when aggregated UNCLASSIFIED information becomes CONFIDENTIAL, one when aggregated CONFIDENTIAL information becomes SECRET, and finally one when aggregated SECRET information becomes TOP SECRET. Beyond the U. S. Classification system, informational critical mass can occur in any information system that stores information.

Being able to execute the rules for information aggregation before the information objects are co-located can prevent informational critical mass from being reached. The preemptive prevention of too much information being co-located prevents security breaches and spillages from occurring.

3.4.3.Chinese Wall Security Problem

A special case of informational critical mass avoidance is the "Chinese Wall Security Problem." Brewer and Nash [8] define the Chinese Wall Security Problem in the context of a consulting firm that is supporting two competitors. When an employee of the consulting firm is working with the information about one of the competitors, the employee's access to information about and from all of the firm's competitors must be prevented. If the employee is no longer working with the firm's information, then any of the information for its competitors is now available to that employee. Once the employee begins working on a new company's information, a new set of limitation on the information available to that analyst are now in effect. Chinese Wall Security problems become increasingly complex when suppliers and supporting organization become part of the analysis.[22] Using object labels as a security mechanism within the framework

[22] For example in the automotive industry, a parts supplier may provide "spark plugs" to two or more automobile manufacturer. Access to information about the spark plug provider must be included in for all automobile manufacturers that use that spark plug. However, the information about spark plugs must be limited to that automobile manufacturer being analyzed. In the case, when the spark plug supplier is the subject of the analysis, information about all of the spark plugs using by all of the automobile manufacturers can be made available to the analysis.

developed for my dissertation can address the security needs created by Chinese wall security problems.

A further element of this problem, employee memory, can be addressed. When original discussed, having information for multiple competitors simultaneously was a violation of the policy. However, an employee could return the first company's information and immediately have access to the second company's information. This violated the "spirit" of separating competitor's information. This framework can enforce a minimum time between accessing competitor's information, thus giving the employee time to forget what they already know.

4. REQUIREMENTS FOR A LABEL BASED SECURITY

4.1. Introduction

The focus of my dissertation is on securing information by using object labels. In order for object labels to be used for the securing of objects, several key requirements must be satisfied. These requirements are:

Object Integrity;
Strong Linkage;
Separation of the Label from the Object;
Object Security Labels;
User and System Security Labels;
Labeling Algebra;
Reference Monitors;
Network Framework; and
Interoperable;
and Cost Effective.

4.2. Object Integrity

The first condition for which any security solution must provide is the integrity of the object being protected. The U. S. Department of Defense (DoD) and the National Institute of Standards and Technology (NIST) identify the integrity of information as being a key component of security. DoDI 8500.2 [28] and NIST Special Publication 800-53 [31], both contains security controls that focus on insuring integrity. Other guidance and directives including DoDD 8500.01E [32] and DoDD 8570 [33] mandate compliance with those controls. If the security solution does not preserve the integrity of the object, then the solution has automatically failed.

This does not mean that as part of the process of securing an object or information that the object or information can't be altered or changed. Many solutions used to secure information do just that. But rather that for any operation performed on an object, there must a reciprocal operation, which returns the object to its original state. For a security solution, this means that all performed operations must be mathematically valid and reversible. An example of this is encryption. Information is routinely encrypted to protect it from unauthorized exposure. The process involves taking the original message, called "plaintext", and encrypting to produce "ciphertext". The encryption process makes use of a mathematically validated process and one or more encryption keys. When access to the original plaintext is required, the ciphertext is decrypted to reproduce the original plaintext. If this decryption process were not possible then encryption would be an unacceptable solution for protecting information.

For my dissertation, this requirement is directed more towards preventing alteration of information from occurring. *Appendix B - Secure File Format* discusses the Secure File Format (SFF) and how it can be used to address many of the requirements for this thesis. With regards to Object Integrity, Secure File Format defines a technique for "wrappering" the object to be protected by to creating a "Type 1" SSF object. From an integrity perspective, wrappering the object in this manner, removes any issues related to the original format of the object allows any solution that makes use of the Secure File Format to treat all objects identically. Figure 1 depicts how a Type 1 wrappered object is represented

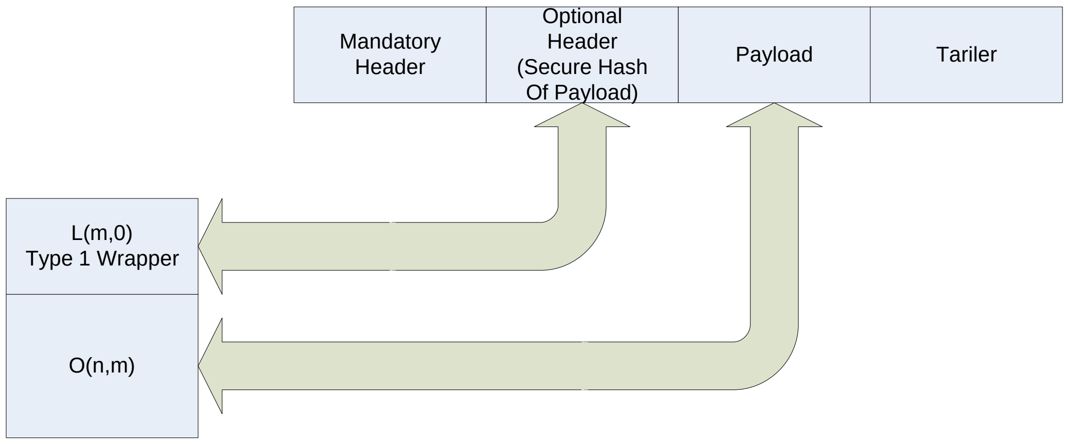


Figure 1- A Type 1 Wrappered Object

4.3. Strong Linkage

The second condition that must be satisfied for object labeling to be useful as a security mechanism is being able to 'strongly link' a label containing security information to an object. It is this attachment that forms the basis of trust that the label is applicable to that object, and that no alterations to either the object or its label have occurred. Mager [13] clarifies the role and value of binding by expressing that:

> *"A security label can be deemed trusted if it is bound or linked to the object, such that this binding can later be validated by a third party. This binding is defined as a trusted process of inseparably associating one or*

more data items that can be validated by another party. The trusted process s typically accomplished using cryptographic techniques."[23]

For my dissertation, 'strong linkage' is intended to mean that the label and its object are bound in such a way that any change to either the object or its label negates the validity of the attachment between the two. The rationale behind this requirement is based on preventing a label that is less restrictive from being attached to an object needing greater security. For example, if a label that asserts that the attached object is CONFIDENTIAL could be attached to a SECRET object without being detected, then any request from a user with a CONFIDENTIAL clearance would be honored and a SECRET object would be released inappropriately.

In order to address this requirement, features of the Secure File Format work done by Rozenbroek [34] will be employed. One of the key features of the Secure File Format is the ability to strongly associate one or more objects with a security label. The objects are stored in the payload field and are cryptographically hashed. The generated hash value or message digest is then included in the security label. The security label, which includes the security attributes and the message digest for the payload are also cryptographically hashed. This forms a second message digest, which is insures the integrity (strong linkage) between the security information and the object(s) in the payload. Secure File Format discusses how Secure File Format uses cryptographic hashing to provide the strong linkage.

[23] [13] pg9, Section 2.6.3

4.4. Separation of Label and Object

The third property that needs to be addressed is the ability to separate the security label from its object without loss of the validity of their bond. Being able to separate the label from its object provides my framework with several features that greatly improve security as well as reduce the impact that labeling has on the infrastructure.

First, by working with just the labels and not the entire objects (labels and labeled object), only the label need be exposed while the access control decision and/or aggregation are being processed. This greatly improves the security of the object and its information because it is only after the determination of whether to grant access has been made that the object is accessed. The ability to make the access determinations before starting to work with the object has different effects based on how the labels are being processed.

In the case where the simple access control is being implemented, the labeled object is only exposed after the access control decision has been reached and access to the object has been granted. In the cases where the aggregation would produce an unacceptable object, the labeled objects would never be accessed and the aggregation would not be created. For information distributed across the infrastructure, this would mean that none of the information would be transferred to a common location. In the case where the requested operation would cause a critical mass threshold to be exceeded, the operation would be disallowed. It is only when accessing the object or operation is deemed appropriate or necessary that the labeled object is engaged or the operation is allowed.

Second, by working with just the labels, the amount of data that needs to be processed is greatly reduced. This will lessen the impact that the framework would have on the infrastructure.

An additional benefit to being able to engaging the security labels independently of the object is being able to process the entire communication path before exposing the object. In larger or distributed infrastructures, this means allowing the labels to be processed by every network security checkpoint in the infrastructure before releasing information.

4.5. Object Security Labels

4.5.1. Introduction

So far, security labels have been discussed in a very abstract manner; there has been no discussion on the format of a label or what information that can be conveyed via security labels.

The first requirement for defining the format and designing the content of the security label is to decide on a lexicon for the labels. As part of the specification for a Secure File Format object, Rozenbroek defined several types of wrappers, which include an XML object that is to be used for either a Security Header or other management information or instructions. Since the Secure File Format approach for binding management and security labels to objects is being used as part of my dissertation, the use of XML is appropriate.

XML, or the Extensible Markup Language [35], is a standard lexicon for defining formats for the storage and exchange of information. It is widely accepted and there are numerous applications that make use of XML for both exchanging information and the management of configuration and system data. In addition to using XML for object and user/system security labels, XML is used for defining the rules for granting access as well as building rules for complex labels and aggregates.

4.5.2.XML Labels for Security and Management

The specification for the Extensible Markup Language is controlled by the World Wide Web[24] (W3C) and defines a rich and powerful environment for storing and exchanging information. One key feature that distinguishes XML from other markup languages like HTML [36] is the separation of data from presentation. In HTML, the defined tags are used primarily for formatting the presentation of the information. These tags are embedded in the document. In the case of XML, there is a very small set of pre-defined tags. The tag set for any given XML document is defined for that document and are tailored to meet the needs. My dissertation defines several different XML formats that will be used to provide object security.

[24] http://www.w3c.org

Figure 2 shows the basic XML representation of an object label.

```
<Object_Label>
    <Object_ID>{Object_ID}</Object_ID>
    <Label>
        <Name>{Name}</Name>
        <Type>{Type}</Type>
        <Value>{Value}</Value>
    </Label>
    …
    <Label>
        <Name>{Name}</Name>
        <Type>{Type}</Type>
        <Value>{Value}</Value>
    </Label>
</Object_Label>
```

Figure 2 - XML Representation of an Object Label

The <Object_ID>{*Object_ID*}</Object_ID> tag is used to convey an object identification value. This field is not used for determining if access to the object should be provided, but rather for identifying the object to the processing system. For detailed information about the object, an informational label should be used.

In addition to the elements defined here, well-formatted XML comments (<!-- {*Comments*} -->) can be included at any point in the label that is allowed by the XML specification. When the label is being processed, the embedded comments are ignored by a process making use of the labels.

The <Label>{*Label Information*}</Label> tag is used to denote those elements that form each individual label. Depending on the security requirements, there will be one or more <Label> tags for each <Object_Label>.

The <Name>{*Name*}</Name> tag specifies a unique name for the label being defined. With the exception of informational labels, for each named label specified by an access rule, there must be a corresponding label defined for each user and for each system. If there either the user or system label lacks a corresponding named label then any comparison operation fails and access to the object is denied.

The <Type>{*Type*}</Type> tag defines what type of label is being represented. Types of Labels Supported describes in detail the format and purpose of each of the different types of tag. Table 1 details the four values that are defined for {*Type*}.

Table 1 - Possible Values for {Type}

{Type} Value	***Type of Label***
HIER	Defines a Hierarchal Label
CATE	Defines a Category Label
COND	Defines an Conditional Label
INFO	Defines a Informational Label

The <Value>{Value}</Value> tag defines the value for the label. The nature of this field is determined by the {Type} field value and is discussed in more detailed in Types of Labels Supported.

4.5.3.Multi-Valued Labels

While being able to attach a single label to an object increases the security, there are numerous cases where being able to attach a combination of label types is valuable. Figure 3 indicates how labels with multiple named values are structured. Within any given label, there are no restrictions on the number and type of labels that can be stored. For example, a label attached to an information object may include a classification label and an integrity label. This is the case, where two hierarchical labels would be used to address a "Bell-LaPadula" type security requirement and a "Biba" type integrity requirement simultaneously. All of the conditions defined by the labels are available for determining access to the object.

For this example, a typical label would resemble:

```
<Object_Label>
    <Object_ID>Protected_Object</Object_ID>
    <Label>
        <Name>Classification</Name>
        <Type>HIER</Type>
        <Value>SECRET</Value>
    </Label>
    <Label>
        <Name>Integrity</Name>
        <Type>HIER</Type>
        <Value>HIGH</Value>
    </Label>
</Object_Label>
```

Figure 3 - Sample XML Object Label

In this case, the object, *Protected_Object* carries a label, *Classification*, which is set to *SECRET* and a second label, *Integrity*, which is set to *HIGH*.

4.6.<u>User and System Labels</u>

4.6.1.Introduction

The second type of labels that required as those that will be associated with the user and the user's system. These labels convey security information about the user, their system, and any intermediate systems invoked with labeled communications. It is important to note that a user can either be an end-user (human being) or a computer process. From a label security perspective, there is no difference between the two. As with object labels, the Extensible Markup Language is used to capture and transport both user and system labels.

The separation of user and system labels is a key feature of this approach. It allows for simplification of implementation by permitting the qualities of the user to be considered independently of the computer systems being employed. For example, a user may have a SECRET clearance and therefore access to information labeled as "SECRET" or lower. However, access to this information must be granted only when the request is made from a system that is also approved for SECRET or higher information. If the request was made from a system not approved for SECRET information, then the request must be denied, despite the fact that the request was made by a user with a SECRET (or higher) clearance. Without this separation of users and systems, a unique set of labels would need to be created and maintained for each user/system combination.

A further reason for using system labels in addition to user labels is that their use allows network topology to be taken into account. If only a single set of user/system labels were being consideration, it would preclude the inclusion of the qualities of the transport path in making access decisions. Because the user and each system are considered separately, it is possible to account for the network path that the object traverses. For example, an end-user has a laptop computer that is allowed to store PII[25] information. If the user wants to send an email containing Privacy Act information to her supervisor (also using a host cleared for Privacy Act information), then she will need to use an email server that is approved (e.g. labeled) to handle Privacy Act information as well. Even though both the employee and her supervisor are using approved host machines, the use of a properly labeled intermediate host (the email server) is required to properly safeguard the transmitted information. Figure 4 shows the topology of this example.

[25] PII – Personnel Identifying Information. The Privacy Act of 1974 [37] as amended is the U. S. federal status that establishes the criteria for what is considered PII, what safeguards must be in place for is protection, and the penalties for not doing so.

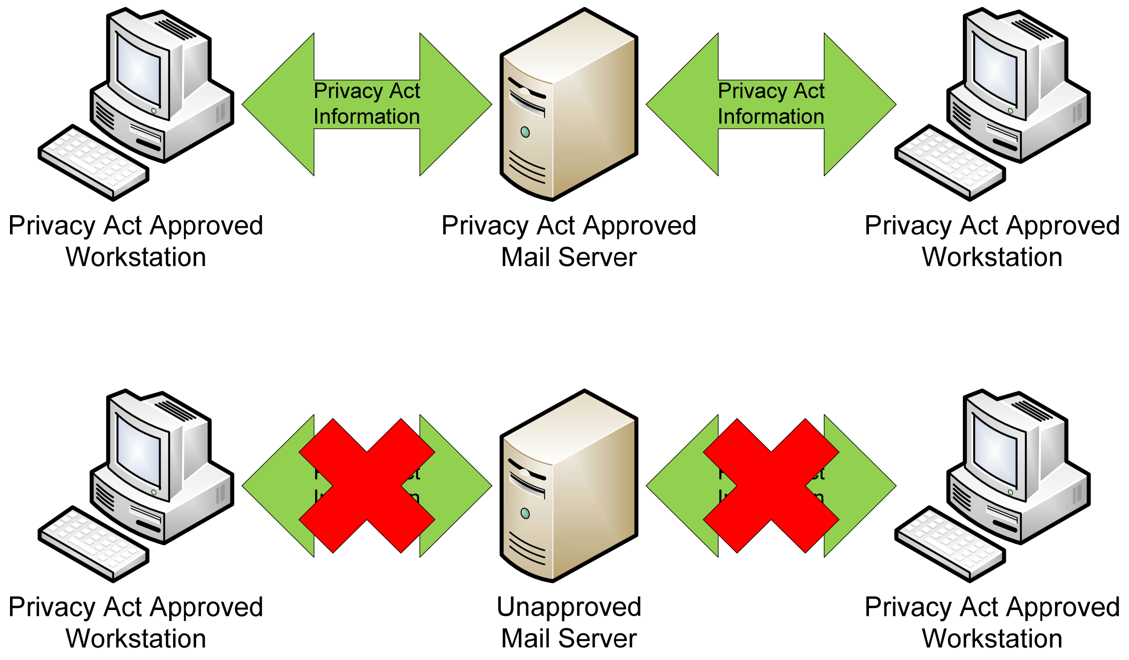

Privacy Act Information
Privacy Act Information
Privacy Act Approved Workstation
Privacy Act Approved Mail Server
Privacy Act Approved Workstation
Privacy Act Approved Workstation
Unapproved Mail Server
Privacy Act Approved Workstation

Figure 4 - Intermediate System Labeling

Because every system involved with the information exchange was labeled, their abilities to properly handle the information were taken into consideration. If the user and system were labeled with a single label, inclusion of the email server would have been more difficult.

4.6.2.XML Labels for Security and Management

The specification for the Extensible Markup Language (XML) [35] is controlled by the World Wide Web Consortium (W3C)[26] and defines a rich and powerful environment for

26 http://www.w3c.org

exchanging information. One key feature that distinguishes XML from other markup languages like HTML [36] is the separation of data from presentation. In HTML, the defined tags are used primarily for formatting the presentation of the information. These tags are embedded in the content of the document. HTML tags are usually used as a means to control presentation more than organizing information. By contrast, XML does not consider the format of presentation, but used XML tags to provide semantics to the information in the document. How a document is displayed, if it is displayed at all, is controlled by an external mechanism. With XML, there is a very small set of pre-defined tags. The tag set for any given XML document is defined for that document and are tailored to meet the document's needs.

Because user and system labels will be compared against object labels, their syntax must track very closely. Figure 5 shows the XML representation for both the user and system label.

```
<User_Label|System_Label>
    <User_ID>{User_ID}</User_ID>
    <Label>
        <Name>{Name}</Name>
        <Type>{Type}</Type>
        <Value>{Value}</Value>
    </Label>
    …
    <Label>
        <Name>{Name}</Name>
        <Type>{Type}</Type>
        <Value>{Value}</Value>
    </Label>
</User_Label|System_Label>
```

Figure 5 - XML Representation of a User Label

The <User_ID>{*User_ID*}</User_ID> tag is used to convey a user identification value. This field is not used for determining if the access to the requested object should be granted. Instead, it conveys information that is intended to make the infrastructure easier to manage. The information in this tag should be used for activities such as auditing. For more detailed information about the user, an informational type label should be used.

The <System_ID>{*System_ID*}</System_ID> tag is used to convey a system identification value. This field is not used for determining if access to the object should be granted. Like the User_ID tag, it should be used for activities, such as auditing. For detailed information about the system, an informational type label should be used.

In addition to the elements defined here, well-formatted XML comments (<!-- {*Comments*} --> can be included at any point in the label that is allowed by the XML specification. When the label is being processed, the embedded comments are ignored by a process making use of the labels.

The <Name>{*Name*}</Name> tag defines a unique name for the label being defined. The {Name} value defined is the label is used as part of the comparison with the object label to determine if access should be granted. The user and system label names used are required to track with the object label names. One note, it is permissible for a user and/or system label set to have additional named labels. This will allow for a more complete description of the user and their systems. In the case, where there are additional user and system labels, only those labels identified by the access rules are used. The additional user and system labels are ignored.

The <Type>{*Type*}</Type> tag defines what type of label is being represented. *Appendix D - Types of Labels Supported* describes in detail the format and purpose of each of the different types of tag. Table 2 details the four values that are defined for {*Type*}.

Table 2 - Possible Values for {Type}

{Type} Value	***Type of Label***
HIER	Defines a Hierarchal Label
CATE	Defines a Category Label
COND	Defines an Conditional Label
INFO	Defines a Informational Label

The <Value>{*Value*}</Value> tag defines the value for the label. The nature of this field is determined by the {*Type*} field value and will be discussed in more detailed in *Appendix D - Types of Labels Supported*.

In general, User and System labels are combined to form a single User/System_Label for processing purposes. When this is done, a new "User/System_Label" is created by collecting all elements that are common to both the User and System labels. The new label represents the intersection of values for each named label. For example, if a User has a SECRET clearance and the System has a CONFIDENTIAL clearance, then User/System_Label is assigned a CONFIDENTIAL clearance. (The workstation is lacking a SECRET clearance.) If a User is a member of the groups $\{A\}$, $\{C\}$, $\{D\}$, and $\{E\}$ and the System is a member of groups $\{A\}$, $\{B\}$, and $\{D\}$, then User/System is a member of groups $\{A\}$, and $\{D\}$. (The user is not part of groups $\{B\}$. The system is not part of groups $\{C\}$ and $\{E\}$.) In general, the User/System_Label is not created or stored outside the application involved with the access determination process. Formally, this can be expressed as:

$$User/System\_Label_i \equiv \cap \left(User\_Label, System\_Label_j\right)_i$$

Where i – is the index of the label being generated
j- is the index of the system being considered

In a large or geographical dispersed infrastructure, the task of maintaining user and system labels may be excessive and expensive. To account for this, user and system labels can be provided by the user/system locally or by a trusted third party infrastructure, such as PKI [38] or LDAP [39] [40] [41]. In either case, user and system labels must be supplied in the prescribed formats. The use of third party infrastructures does allow for the user and system labels to be realm specific.

4.6.3.Multi-Valued Labels

As with object labels, a single user/system label may contain one or more named labels. The syntax for User/System_Label is:

```
<User/System_Label>
    <Object_ID>Combined_User_System_ID</Object_ID>
    <Label>
        <Name>Classification</Name>
        <Type>HIER</Type>
        <Value>Confidential</Value>
    </Label>
    …
    <Label>
        <Name>Groups</Name>
        <Type>CATE</Type>
        <Value>A</Value>
        <Value>D</Value>
    </Label>
</User/System_Label>
```

Figure 6 - XML Representation of a User/System Label

4.7. Labeling Algebra

4.7.1. Introduction

Thus far, the XML schemas for expressing conditions encapsulated by labels that are bound to objects have been presented. In this section, the second component, the Labeling Algebra for processing the labels is presented. Labeling Algebra is the logic that is used to determine if access to the labeled object should be granted based on the object's own labels and those of the user and systems.

Access to the secured objects is granted if conditions detailed by the attached object labels have been met by the user and systems labels. In general, this can be represented by this statement;

> *When compared against the object's labels, if the user and systems labels satisfy the conditions defined by the access rules for the object then access to the labeled object is granted, otherwise access is denied.*

The process for determination of whether the rules have been satisfied is discussed in the following sections.

4.7.2. Access to Labeled Objects

In order to be granted access to an object, a set of access rules must be successfully satisfied. This can modeled formally as:

$$(Access\_Rules) \Leftrightarrow \; GRANT\_ACCESS$$

The rules for gaining access to an object are captures as one or more tests. If any one of these tests is TRUE, then access is granted. The access rules can be formally modeled as:

$$\vee \, (Test_i)_{i=1,m} \Leftrightarrow Access\_Rules$$

Where m is the number of test that is part of the Access_Rules

For each test, there are one or more rules that are evaluated. All of the rules must be satisfied for the Test return a “TRUE”. Tests are formally modeled as:

$$\wedge \left(Rule_j\right)_{j=1,n(i)} \Leftrightarrow \; Test_i$$

Where n(i) is the number of rules that are part of the ith Test

For each rule, a comparison between an object label and a user/system label is performed. If the comparison operator yields a positive result then the Rule returns a “TRUE”. Rules are formally modeled as:

$$Operator_j\left(Object\_Label_j, User/System\_Label_j\right) \Leftrightarrow \; Rule_j$$

Where $Operator_j$ is the operator used to evaluate the j^{th} Rule

At this point, access to a labeled object has been decomposed formally into a collection of simple operations based on individual labels. Assessment of these operations yields TRUE/FALSE values for the each label’s rule.

The rules are collected and assessed to produce the results for a test. If all of the rules return TRUE values, then the result of the test is TRUE.

The rules are collected and assessed to produce the results for the access rules. If any of the rules returns a TRUE value then the results for access rules is TRUE.

If the access rules are TRUE, then Access is Granted.

The logic used to capture this hierarchy is XML-based and is designed to align with the information carried by the object, user and system labels.

4.7.3.Simple Access Rules for Object Labels

Introduction

The simple access rules for granting access to labeled objects are those operations that are executed against a single type of object label. Because each type of label conveys the object security metadata differently, the algebra for label evaluation must be tailored to the type of label being evaluated. However, the processing of the different labels of labels will ultimately yield results that are common. For access to labeled objects, the two possible results of a label evaluation are shown in Table 3.

Table 3 - Possible Results from Labeling Algebra Operation

Result Value	***Result Condition***
GRANT	Permission to perform the requested operation on the labeled object is granted
DENY	Permission to perform the requested operation on the labeled object is denied

Because each test produces a single Boolean result, controlling access to objects is a matter of defining tests based on need and then processing the results with Boolean algebra. Figure 7 shows the structure of an Access Rule file.

```
<Access_Rules>
    <Test>
        <Testname>{Testname}</Testname>
        <Rule>
            <Name>{Name}</Name>
            <Type>{Type}</Type>
            <Operator>{Operator}</Operator>
        </Rule>
        …
        <Rule>
            <Name>{Name}</Name>
            <Type>{Type}</Type>
            <Operator>{Operator}</Operator>
        </Rule>
    </Test>
    …
    <Test>
        <Testname>{Testname}</Testname>
        <Rule>
            <Name>{Name}</Name>
            <Type>{Type}</Type>
            <Operator>{Operator}</Operator>
        </Rule>
        …
        <Rule>
            <Name>{Name}</Name>
            <Type>{Type}</Type>
            <Operator>{Operator}</Operator>
        </Rule>
    </Test>
</Access_Rules>
```

Figure 7 - XML Representation of an Access Rule

Figure 7 shows that <Access_Rules> are composed of one or more test, each of which contains one or more rules.

The <Test>{Test}</Test> tags are used to delineate each of the tests.

The <Rule>{*Rule*}</Rule> tag is used to delineate each of the rules that must be satisfied for the test return a TRUE result.

If all of the rules in a test are TRUE, then the test is assessed as TRUE. If any of the Rules fails to produce a TRUE result, then the test is assessed as FALSE. Each rule is used to assess a different tag, but every rule returns the same two results; TRUE or FALSE. This allows for tests to be tailored to the type of label being evaluated.

The test's results are then collected and assessed. If one or more of the tests has been assessed as TRUE, then the Access_Rules are also TRUE and access to the object is granted. If none of the tests is assessed as being TRUE, then the Access_Rules is assessed as FALSE and access to the object is denied.

The <Name>{*Name*}</Name> tag is used to designate the name of the rule. The <Name> tag is also used to associate the rule with which labels in the object and user/system labels will be used for the rule assessment.

The <Type>{*Type*}</Type> tag is used to designate which set of operators will be invoked when processing the rule. There are 2 values for {*Type*} that are defined. Table 4 defines these values and how they should be processed.

Table 4 - Possible Values for the {Type} tag in a Rule

{Type} Value	***Processing Rules***
HIER	Process the Rule using the hierarchical rules
CATE	Process the Rule using the categorical rules

The <Operator>{*Operator*}</Operator> tag is used to identify which operator is used along with the object and user/system labels in assessing the rule. The possible {*Operator*} values are determined by the {*Type*} tag.

Hierarchical Rules

For assessment of hierarchical labels, the each rule identifies the name of the label values under consideration, the type of labels being considered, and the operator used. Figure 8 shows the format of a Rule used to evaluate a hierarchical label.

```
<Rule>
    <Name>{Name}</Name>
    <Type>HIER</Type>
    <Operator>{Operator}</Operator>
</Rule>
```

Figure 8 - XML Representation of a Hierarchical Access Rule

The <Name>{*Name*}</Name> tag identifies the object and user/system labels that are to be assessed using the <Operator> tag.

The <Type>{*Type*}</Type> is always set to “HEIR”. This denotes the type of labels being evaluation and the potential operator that can be used.

The <Operator>{*Operator*}</Operator> tag is used to identify the operation that will be used to assess the user/system label against the object label. If the Operator is satisfied then that Rule returns a “TRUE”. Otherwise, a “FALSE” is returned. There are six operations that have been identified for use with hierarchical labels. Table 5 shows these operations.

Table 5 - Operations for Hierarchal Labels

Operation	***Language***	***Syntax***
(EQ)	Equals	$(User/Sytem\_Label = Object\_Label) \leftrightarrow \text{TRUE}$
(GT)	Greater Than	$(User/Sytem\_Label > Object\_Label) \leftrightarrow TRUE$
(GE)	Greater Than or Equal	$(User/Sytem\_Label \geq Object\_Label) \leftrightarrow TRUE$
(LT)	Less Than	$(User/Sytem\_Label < Object\_Label) \leftrightarrow TRUE$
(LE)	Less Than or Equal	$(User/Sytem\_Label \leq Object\_Label) \leftrightarrow TRUE$
(NE)	Not Equal	$(User/Sytem\_Label \neq Object\_Label) \leftrightarrow TRUE$

Categorical Rules

For assessing categorical labels, there is no concept of hierarchy. Instead, categorical labels embrace the concept of inclusive in a group. Figure 9 shows the format of a Rule used to evaluate a categorical label.

```
<Rule>
    <Name>{Name}</Name>
    <Type>CATE</Type>
    <Operator>{Operator}</Operator>
</Rule>
```

Figure 9 - XML Representation of a Categorical Label

The <Name>{*Name*}</Name> tag identifies the object and user/system tags that are to be assessed using the <Operator> tag.

The <Type>CATE</Type> denotes that this rule is applicable to a categorical label and that both the object and user/system labels must be defined as categorical labels. If either the object or user/system label is not categorical, then the Rule is set to FALSE.

The <Operator>{*Operator*}</Operator> tag is used to identify the operation that will be used to assess the user/system label against the object label. If the Operator is satisfied then that Rule returns a “TRUE”. Otherwise, a “FALSE” is returned. For simple categorical labels, there are two cases to consider. In the first case, the user and system labels contain at least one of the values associated with the object. In the second case, the user and system must contain all of the label values associated with the object.

Table 6 - Category Label Operators

Operator	***Description***
ANY	Each set of User and Systems labels must include at least one of the categories identified by the Object labels
ALL	Each set of User and Systems labels must include all of the categories identified by the Object labels

4.8.Reference Monitors

4.8.1.Introduction

Beyond simply labeling objects and having an algebra for processing the object labels, there needs to be applications for managing the security of the objects. These applications will make use of the security labels and labeling algebra to decide if the labeled object should be released. As the security labels are external to the labeled object and can be removed without any loss of integrity or security, they can be processed in advance of the labeled object being exposed.

RFC 1457 [42] states:

> *"In general, security labeling by itself does not provide sufficient data security; it must be complemented by other security mechanisms."*[27]

27[42] pg 1

While this statement is almost self-evident, it establishes the need for external infrastructure mechanism that work with an object's security labels, the user and system labels, and the rules for access, in order to provide the complete security mechanism. The DoD 5200.28 [9], sometimes referred to as the "Orange Book", mandates the use of a "Reference Monitor" as a means of controlling access to system objects at the "B3" level. The Reference Monitor concept was originally proposed by J. Anderson in an Air Force Study[43]. As documented by the Orange Book, the Reference Monitor "enforces the authorized access relationships between subjects and objects of a system"[28]. Reference Models or "Security Kernels" as they are often called, are used to implement access policies with a system. In most cases, Reference Monitors are used to enforce Mandatory Access Controls (MACs). Mandatory Access Control is defined as:

> *A means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e. clearance) of the subjects to access information of such sensitivity.[45]*

What is important to take from this definition is that information about the objects and subjects[29] is represented as labels and that it is only the information in the labels is used to determine access. Many of the models used in the development of Reference Monitors are based on the work done by Bell and La Padula.[46] Figure 10 is a reprint of Figure 1

28 [44] Section 6.1

[29] For my dissertation, subjects refer to users and systems.

from Ames, Gasser, and Schell's paper on Security Kernels. It shows the role of the Reference Monitor in controlling a subject's access to objects.

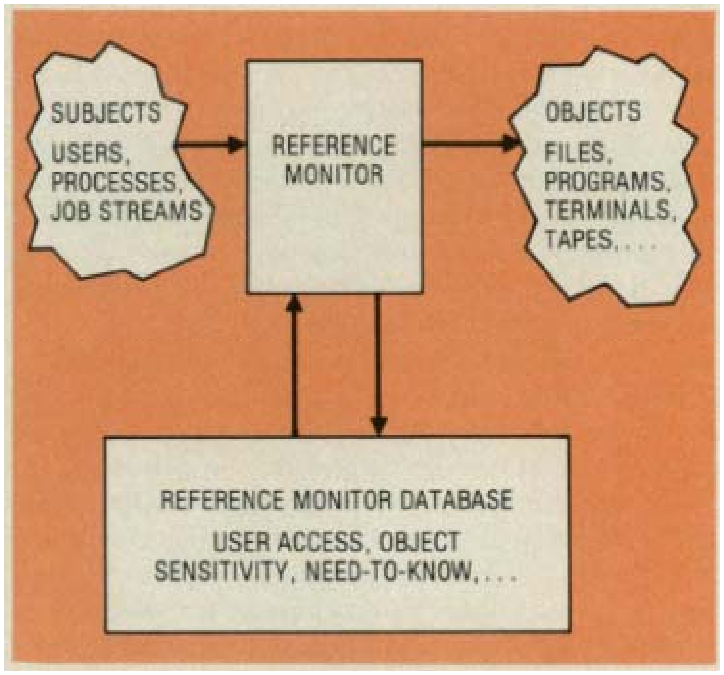


Figure 10 - (Figure 1. Reference Monitor)[32]

Figure 11 shows the how labels and rules are used in my dissertation's interpretation of the Reference Monitor. Subjects are replaced by users and systems with their security information carried by the user/system labels. Objects, the information being protected, are still present and their information is carried by the object labels.

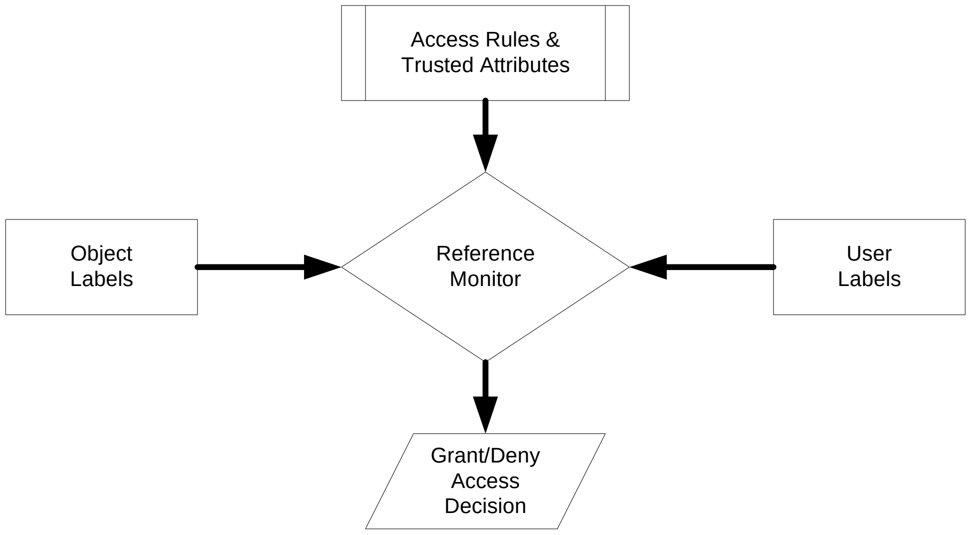


Figure 11 - The Reference Monitor

In order for a Reference Monitor to be effective, its placement is critical. Ames, Gasser, and Schell state “that all access to information must be mediated by the kernel [Reference Monitor]”[46]. If users are able to circumvent the Reference Monitor, then it is ineffective as a security mechanism.

4.8.2.Detailed Discussion

For a Reference Monitor to be of value for controlling access to object, it must interact with other components in the infrastructure. Figure 12 shows the interconnection between it and other elements of the infrastructure as well as the internal structure of the Reference Monitor. Beyond the elements shown, the Reference Monitor may interact with some form of auditing system. For the purposes of clarity, these are not included because they are not integral to the Reference Monitor's operation. For my dissertation, the Reference Monitor is being divided into three parts.

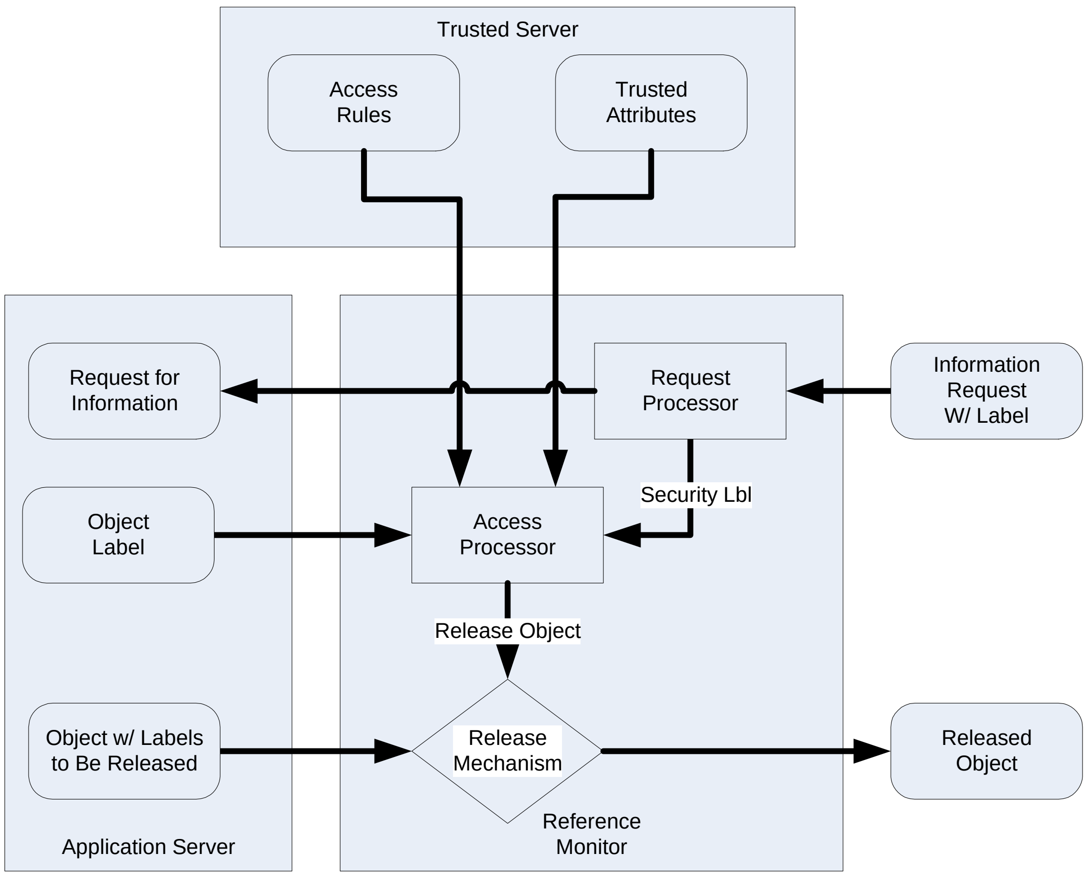


Figure 12 - Decomposed Reference Monitor

There are two external components with which the Reference Monitor will interact as part of the label based access control process. The two external components are the "Trusted Server" and the "Application Server." The three internal components of the Reference Monitor are the "Request Processor", the "Access Processor", and the

“Release Mechanism.” Figure 13 shows the sequence of activities associated with the Reference Monitor’s internal components and the external components with which it interacts.

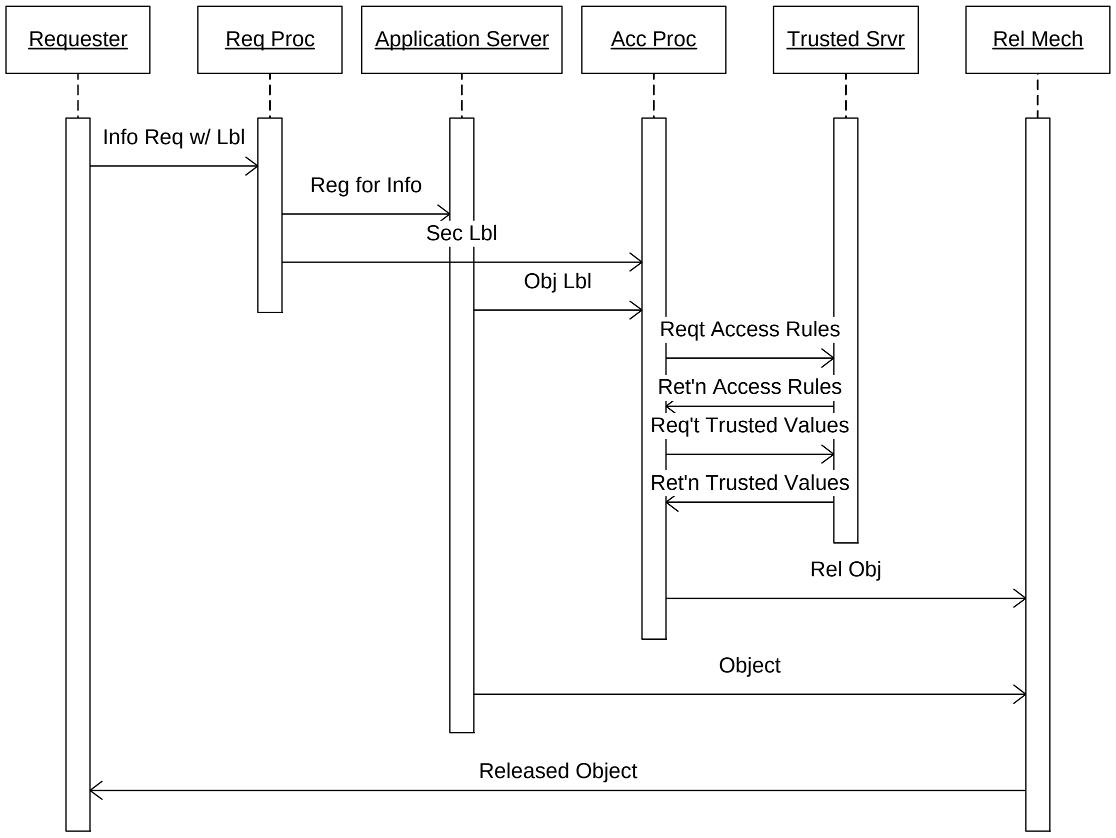


Figure 13 - Sequence Diagram for Decomposed Reference Monitor

The “Trusted Server”, which was originally called the “Reference Monitor Database”, is an external mechanism that will provide the Reference Monitor with rules that will be used. These rules are critical to the operations of the Reference Monitor and safeguards will need to be taken to insure that they are always available and have not been modified without proper authorization. The second function of the Trusted Server is to provide "Trusted Attributes" securely. Trusted Attributes is that information that is required to process conditional labels. Because Trusted Attributes are used as part of the access process, the integrity is as important as the access rules. For example, if an object's label provides for automatic declassification after a given date, then providing the Reference Monitor with an inaccurate date may allow a classified document to be released prematurely as unclassified. This represents a data spillage.

The "Application Server" is the second external mechanism that interacts with the Reference Monitor. My dissertation does not discuss the Application Server with much detail. The Reference Monitor expects to have limited interaction with Application Server, so the internal workings of this server are not greatly affected by the Reference Monitor. However, the Application Server must be isolated from any interaction except that which has been vetted by the Reference Monitor. The interaction between the Reference Monitor and Application Server will be discussed as part of the Reference Monitor discussion.

The first part of the Reference Monitor to be discussed is the Request Processor. The Request Processor acts as the front-end to the Reference Monitor. It interacts with the user requesting an object by receiving the labeled request object. The request processor is responsible for removing the external object labels and generating the request object that will be processed by the application server. The type of service being requested will determine the type of server that will be used. As the use of object labels should not alter the content of the original object, conventional service providers, such as web servers, file servers and database servers, can be used. The request processor's second responsibility is to separate the security labels from the request object and provide them to the Access Controller. It makes a copy of the user/system label. It passes the original labeled object onto the Application Server and provides the user/system label to Access Processor. For every request for information that the Application Server receives, it return both a labeled object and a copy of the object's label. The object's label is provided to the Access Controller as part of the process for making access control decision. The labeled object is provided to the Release Mechanism.

The second part of the Reference Monitor is the "Access Controller". The Access Processor is the central security component in the Reference Monitor. It is responsible for assembling all of the information required to make access decisions. If any conditional operations are required, the Access Controller resolved the conditional operations to create the new object and user/system labels that will be used in the determination. Once the required information has been collected and/or pre-processed, the Access Processor processed the information in order to determine if access to the object should be granted or denied. Beyond making this critical decision, the access controller is not involved with any of the processing needed to fulfill the request. The access decision is then communicated with the "Release Mechanism."

The final part of the Reference Monitor is the "Release Mechanism". The Release Mechanism is responsible for executing the Access Controller's decision. If the "Grant

Access" result is provided, then the Release Mechanism releases the Application Server's results back to the requestor.[30] If the "Deny Access" result is provided, the Release Mechanism results in an access denied message and terminates the session.

4.9. Network Framework

4.9.1. Introduction

It is worth restating that at the time that security models, such as Bell-LaPadula and Biba were developed, the major computing resources were mainframe computers with very little connectivity. Therefore, there was little need to consider the infrastructure upon which the information was hosted. By virtue of the information having been loaded, the hardware is authorized to have access to that information. Since then, the typical computing environment has changed dramatically. Today the majority of computing resources are not centrally located and operating in a standalone environment; they are interconnected via local area networks to form a network of connected computing resources, ranging from personnel computers, smartphones, and netbooks on the lower end to large mainframes, and supercomputers on the upper end. These local area networks are connected to larger networks. All of which are connected to create a

30 As with the Access Controller, the Release Mechanism may be required to perform ancillary operations, such as adding additional wrappers containing audit trail and other security information. This is not part of this discussion and does not affect the work presented.

network of networks, collectively called “the Internet”. Currently estimates place the total number of Internet users at just over 2 Billion[31] with just over 818 Million Internet connected devices[32]. Because of this shift in computing paradigm, security labels must be applied, not just to users, but to any infrastructure components over which the information object may travel. One key feature that the Internet does provide is consistency at the network protocol level. Unlike computing environments which are highly heterogeneous, the protocols used to exchange information over the Internet are more homogeneous across the network. Much of the credit for this consistency can be attributed to the Internet Engineering Task Force (IETF)[33] and the publication of “Requests for Comments” or RFC. RFCs are used to specify the protocols that are used for information exchange in an Internet environment. They are implementation independent and as such remove much of the complexity. Each RFC is written to address a single topic. By keeping this “as small as possible” approach to defining standards, the Internet has been able to adapt to changes and mature to meet the changing needs of its world-wide community. However, while the underlying protocols and standards are common throughout the Internet, the level of trust, access and security associated with different parts of the Internet varies greatly.

[31] Internet World Stats (http://www.internetworldstats.com) states the total number of Internet users at 2,095,006,005 as of March 31,2011[47]

[32] The Internet Systems Consortium (http://www.isc.org) states the total number of hosts advertised in the DNS at 818,374,269 as of Jan 2011[48]. This number does not account for the any hosts that don’t have a DNS registration, such are private network protected by NATing firewalls, which would only present a small number of registered addresses.

[33] http://www.ietf.org

While the use of Reference Monitors, which front-end application servers, may be sufficient for an intranet environment, especially those were all of the systems are considered to be equally secure and able to handle any information that is present on the intranet. It does not scale to an extranet or Internet environment. For the larger and more heterogeneous network environments, a tiered system of Reference Monitors is required. Using Reference Monitors in this manner offers greater flexibility and allows more complicated networking environments to be realized. A key requirement for the Reference Monitors within a given infrastructure is the need to work together. The National Computer Security Center's Trusted Network Interpretation [10], sometimes known as the "Red Book", describes the all of the Reference Monitors in a network as a single abstract concept as the "Network Reference Monitor". The Network Reference Monitor controls which users are granted access to which objects. As with a single Reference Monitor, all requests for labeled information must be routed through one or more of the Reference Monitors in the network, in order for the Network Reference Monitor concept to work properly.

An additional consideration for securing objects in a network environment is the systems that compose the network. The need to control access to information is based as much on the system as the user of the system. Figure 14 shows a Reference Monitor, updated to reflect the need to include the hardware systems. An example of why the need to consider the hardware as well as the user is the user with a TOP SECRET clearance attempting to read a SECRET document. If the user is working at a workstation that has been approved to handle SECRET (or TOP SECRET) information, then that user will be granted access to the information. If the same user with the same clearance were to attempt to read the same SECRET document from their home computer, which has not been cleared to read SECRET document, that same request would be denied.

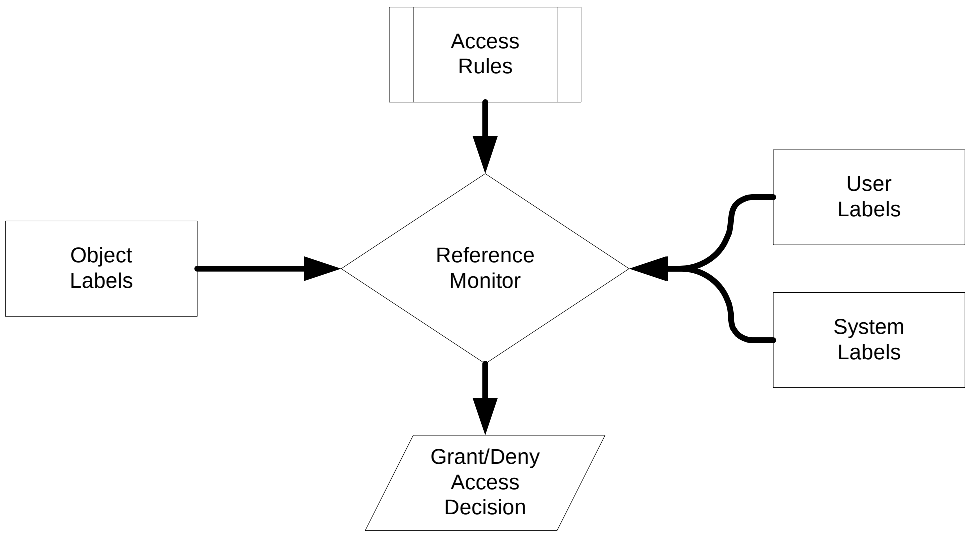


Figure 14 - Updated Reference Monitor

4.9.2.Network Topologies

As networks increase in size, they become increasingly more segregated. Most organizations will subdivide their networks to match the business functions supported by the hosts on that segment of the network as well as the need to afford some portions of the network greater security. A typical corporate network may include subnetworks for engineering, finance, human resources, and sales. Additionally, the organization may host its "public presence" on an external network segment called a "DMZ". The DMZ will host web servers and other resources that the organization wants to make available to customers and other parties outside of the company, while providing additional security to the internal corporate network. Figure 15 shows a notional corporate network.

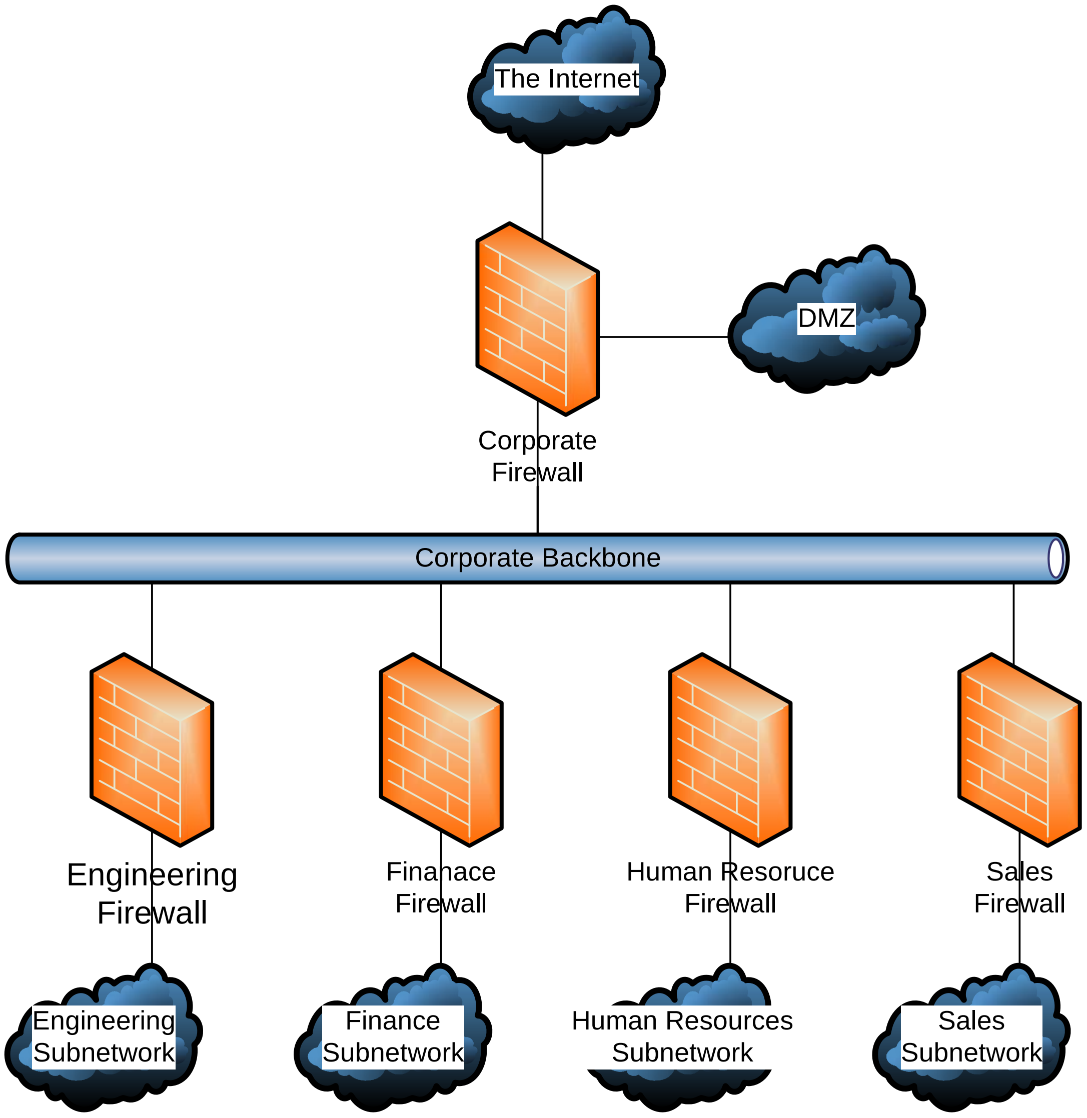


Figure 15 - Notional Corporate Network.

Between each of the subnetworks and the corporate backbone is a router or firewall as well as other network security appliances.[34] The role of the firewalls and routers is to control access to the subnetwork and thus the information on the subnetwork, while still providing the needed conductivity. For example, the engineers on the engineering subnetwork need access to the Internet in order to download software updates, check on-line engineering journals, and conduct research, but the organization needs to prevent the corporate engineering information from being accessed from outside the engineering department. One of the major shortcomings of this approach is that it requires the firewalls to be aware of when it is acceptable to release information and when it is not. Firewalls can be configured to control the type of network traffic that can traverse the firewall. They can control to which hosts that traffic can be sent or received. However, while a firewall can determine if a given type of traffic is considered acceptable, they are very limited in determining if acceptable traffic types[35] are transporting information that should not be released. Some of the new firewalls are able to perform "deep packet inspection." However, this inspection is resource intensive and can't understand the meaning of what is being exchanged. Object labels can provide the necessary "understanding" about the information being passed without the need to analyze every

[34] These appliances may include Intrusion Detection Systems (IDS), network traffic monitors, virus scanners, and other network level hardware and software components designed to improve the security of the network.

[35] Acceptable traffic types may include web traffic, email, streaming media, database queries, etc. Each of these acceptable traffic types is controlled by enabling or disabling pre-assigned TCP and UDP ports at the firewall. Controlling which hosts can send and receive what type of traffic is manage by determining which network addresses can accept what type of traffic.

byte of information in the packet. By upgrading the firewalls and router so that they are label aware controlling access to information objects can be realized at the object level.

4.9.3.Network Proxies

When a Reference Monitor is able to include object labels in its decision making process, it is called a “Network Proxy” and operates in one of two configurations; Transfer Nodes and Request Proxies. The concept of Network Proxies is discussed in RFC 1457[42]. In this RFC, Network Proxies are presented as either end-user systems or intermediate systems. The key idea being that security based on labeling must account for the network topology, not just the user. Unfortunately, it does not detail the use of labels as a means of doing much more than network traffic control.

While Reference Monitors are used to protect information and are installed between the information and the user, Network Proxies are installed on the system hosting the information and are installed throughout the network to control the transport of labeled objects between different parts of the network or infrastructure. For the notional corporate network, shown in Figure 15, each of the subnetworks would have a Network Proxy installed on each of the Firewalls as well as any network connected information source that is using labels to secure information. Because of this, a request for information will be handled by more than one Network Proxy.

A key feature of Network Proxies is that they establish associations between themselves, other Network Proxies, Information Providers[36], and the end user hosts. These associations are the basis for determining how information will be passed between the different parts of the communication path. There are two types of associations: weak or strong. If the association allows the information to be transferred between nodes in the communication path before the end-to-end connectivity has been established, then the association is weak. If the end-to-end connectivity is confirmed before any information is transferred, then the association is strong.

In a weak association, it is not necessary for all of the associations in the network path to be verified before a labeled object is transferred. Before each transfer, the object label of the response object is compared against the user/system labels to insure that the object is not inappropriately released. At each transfer in the process, the connection of the next transfer is tested. If the next association is acceptable, then the labeled object is forwarded to the next Network Proxy. Figure 16 shows the sequence of exchanges that transpire when weak association is used.

[36] Information Providers are Reference Monitors that are front-ending the system that is providing the information.

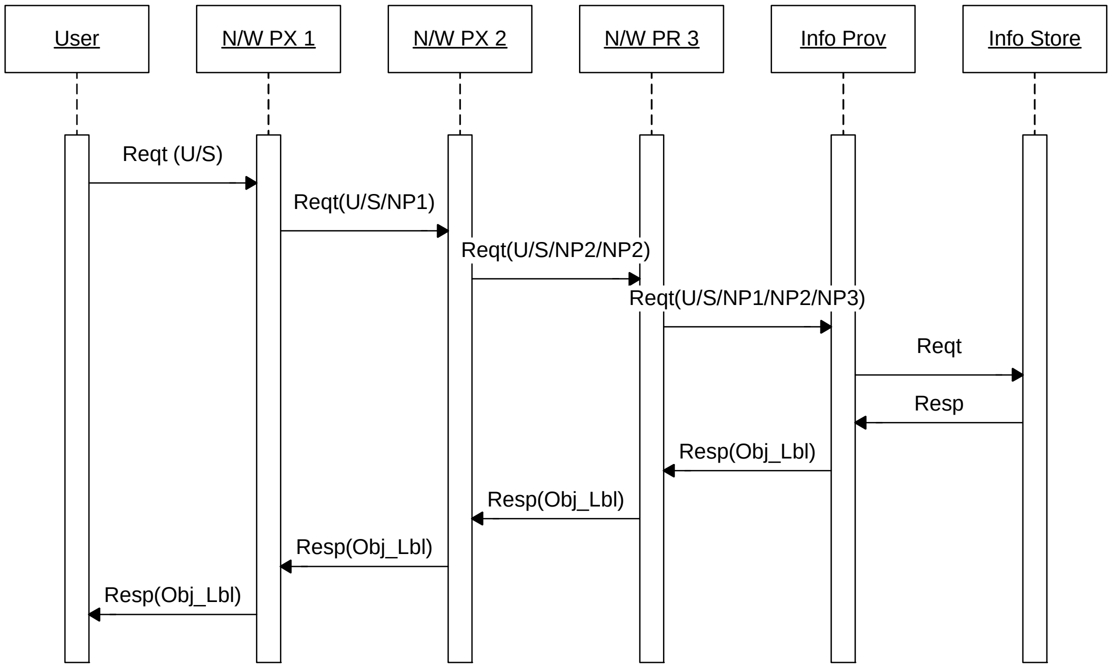


Figure 16 - Sequence Diagram for Weak Association

In a weak association, the user submits a request, along with the request, their user label and system label are sent. The first Network Proxy forwards the request adding their system label to the request. This is repeated for all Network Proxies, until the request reaches the information provider. At the information provider, the request is processed and the attached user/system label is evaluated. If the user/system label is acceptable, then the labeled object is generated and returned to the last Network Proxy. The last Network Proxy then forwards the labeled object back to the next Network Proxy in the chain. This is repeated until the labeled object is returned to the user. The advantage of weak association is that it is better able to support low-bandwidth or intermittent network connections. The disadvantage is that it can leave a labeled object stranded on an intermediate Network Proxy.

In a strong association, the entire network path is validated. For this to happen, each connection in the communications path is assessed and all must be considered acceptable before the labeled object is released by the Information Provider. Only after the complete network path has been validated is the transfer of the labeled object initiated. Figure 17 shows the sequence of exchanges that transpire when strong association is used.

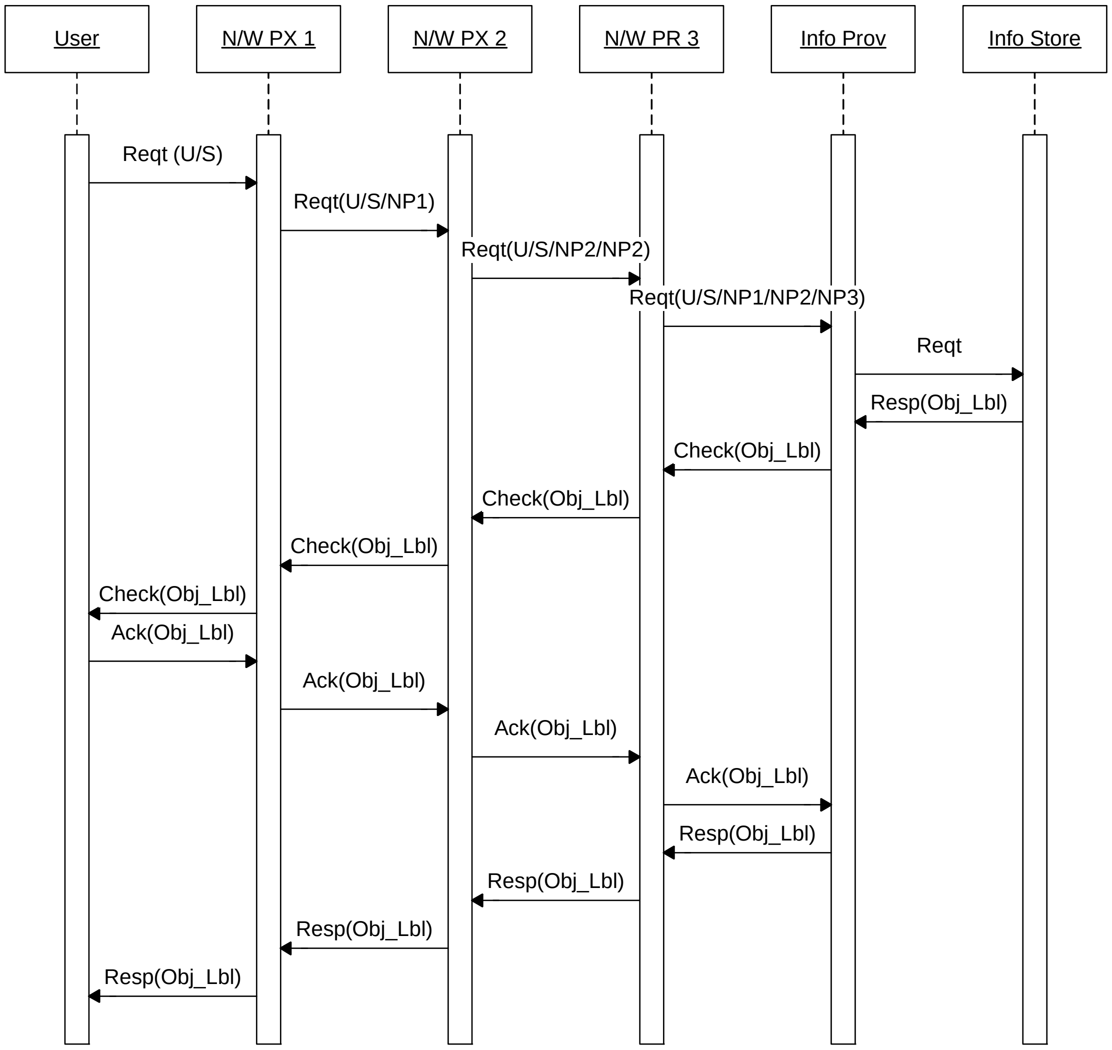


Figure 17 - Sequence Diagram for Strong Association

In a strong association, the user submits a request, along with the request, their user label and system label are sent. The first Network Proxy forwards the request adding their system label to the request. This is repeated for all Network Proxies until the request reaches the Information Provider. At the Information Provider, the request is processed and the attached user/system label is evaluated. If the user/system label is acceptable, then the labeled object is generated. The object's label is then attached to a "Check Path" object and the labeled Check Path object is then returned to the last Network Proxy. The last Network Proxy evaluates the response object's label against the user/system label. If the user/system label is acceptable, then the Network Proxy forwards the Check Path object back to the next Network Proxy in the chain. This is repeated until the Check Path object is returned to the user's system. At the Network Proxy on the user's system, the Check Path object is acknowledged and sent back to the Information Provider through the chain of Network Proxies. Once the acknowledgement is received by the Information Provider, the original labeled object is submitted into the chain. The advantage of using strong association is that the communication channel is confirmed before the labeled object is placed on the network. The disadvantage is that the response time is greater and the quality of the network is a factor.

The choice of association to be used will depend on the network topology, the robustness of the network connections as well as the criticality and sensitivity of the information being requested. Larger networks with less robust connections will favor weak associations. Stronger associations are preferred when the information is more critical and/or highly sensitive.

One network security feature that Network Proxies facilitate is security within the network, versus security between endpoints. With Network Proxies, a network environment can be partitioned based on the level and control and security that each partition network affords. Figure 18 shows how Network Proxies can be deployed to form an infrastructure that allows for control of information throughout the entire network, not just at the Information Provider and End User system.

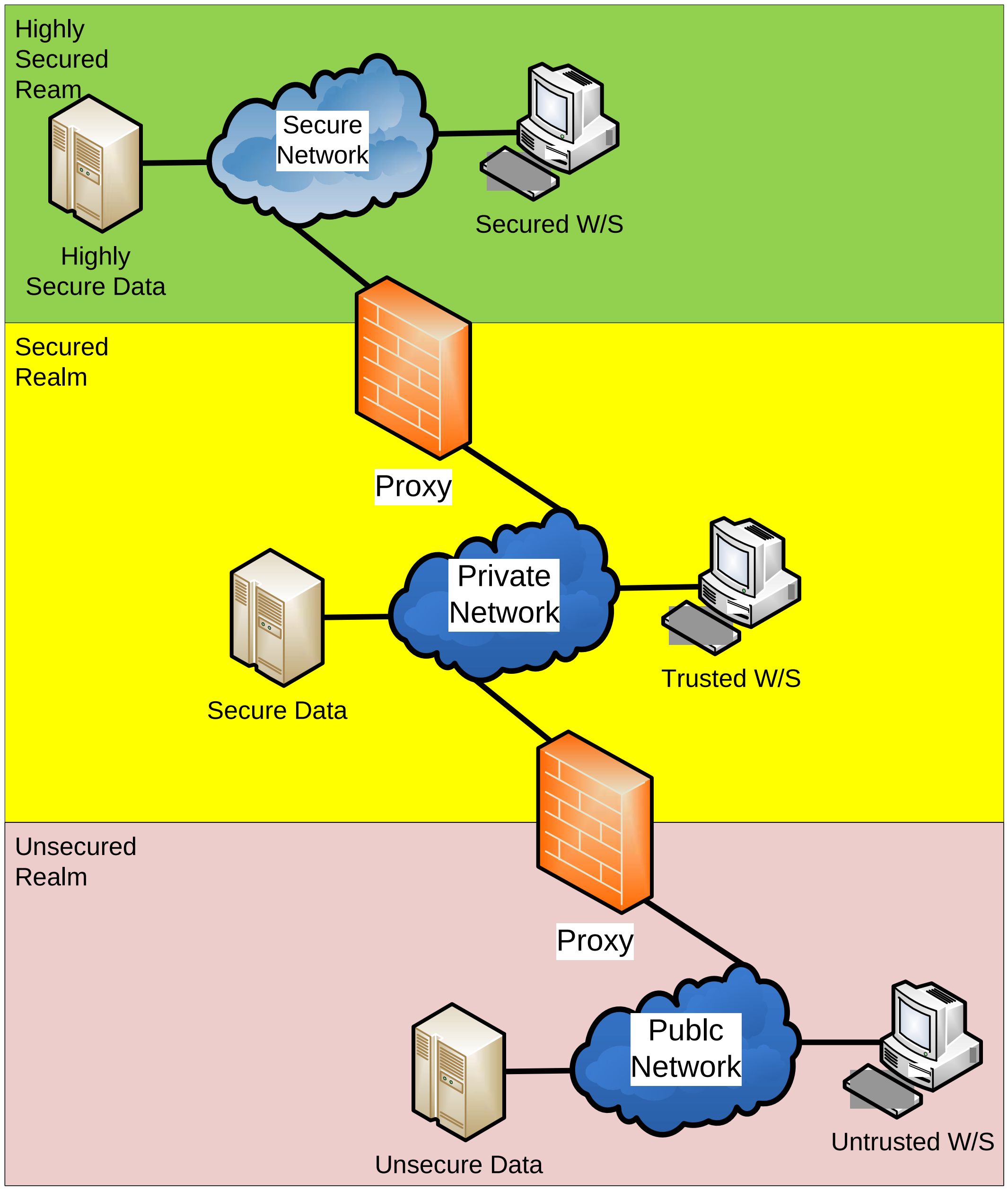


Figure 18 - Notional Placement of Network Proxies to form a Controlled Network

A key component of creating an infrastructure using Network Proxies is insuring that Network Proxies can identify themselves and validate that other Network Proxies and Information Providers are legitimate. For smaller collections of Network Proxies, static lists maintained locally on each proxy are a viable administration tool. As the collections grow larger and more geographically dispersed, alternative mechanisms must be incorporated. There are numerous mechanisms that are already available. These include centralized server solutions such as Kerberos [49], centralized infrastructure solutions such as Public Key Identification (PKI)[38] [50], and decentralized solutions such as openPGP [51].

There are two varieties of Network Proxy; Transfer Nodes and Request Proxies

A Transfer Node is used to transfer the labeled object between two other nodes in the network. As it is a store and forward location or "queue" in the network, it serves as a point at which decisions, about how the object being transferred, are made. Transfer Nodes are used should be used when the security posture of the infrastructure changes greatly, but there is still a common understanding of the meanings of the object labels.

Request Proxies differ from Transfer Nodes in that a Transfer Node simply forwards the labeled object onto the next node; a Request Proxy is used as an entry point into the labeling infrastructure for request from users or systems that are not label aware or there is a major change in the use of labels and/or rules. They work by accepting a request from either a known or unknown user. They then create an appropriate set of user labels for that user. In the case of a known user, a third party infrastructure tools, such as an

LDAP server, can be used to provide the user label. For an unknown user, a simple default label is used. For large networks that are broken up into regions with vastly different security characteristics, request proxies can be used as gatekeepers or Cross Domain Solutions at the boundaries. It is worth noting that if within a region there are label aware resources, then a Transfer Node should be used at the interconnection. Otherwise, every host in the network region will be treated identically.

4.9.4.Transfer Nodes

Transfer Nodes are used to transfer labeled objects between the end user and the ultimate destination, the information request server or Network Proxy. They allow a large network environment, such as the Internet, to be broken into smaller regions or segments. By compartmentizing a network with transfer nodes, the migration of information objects can be controlled based exclusively on the external labels. Figure 19 shows a sample network.

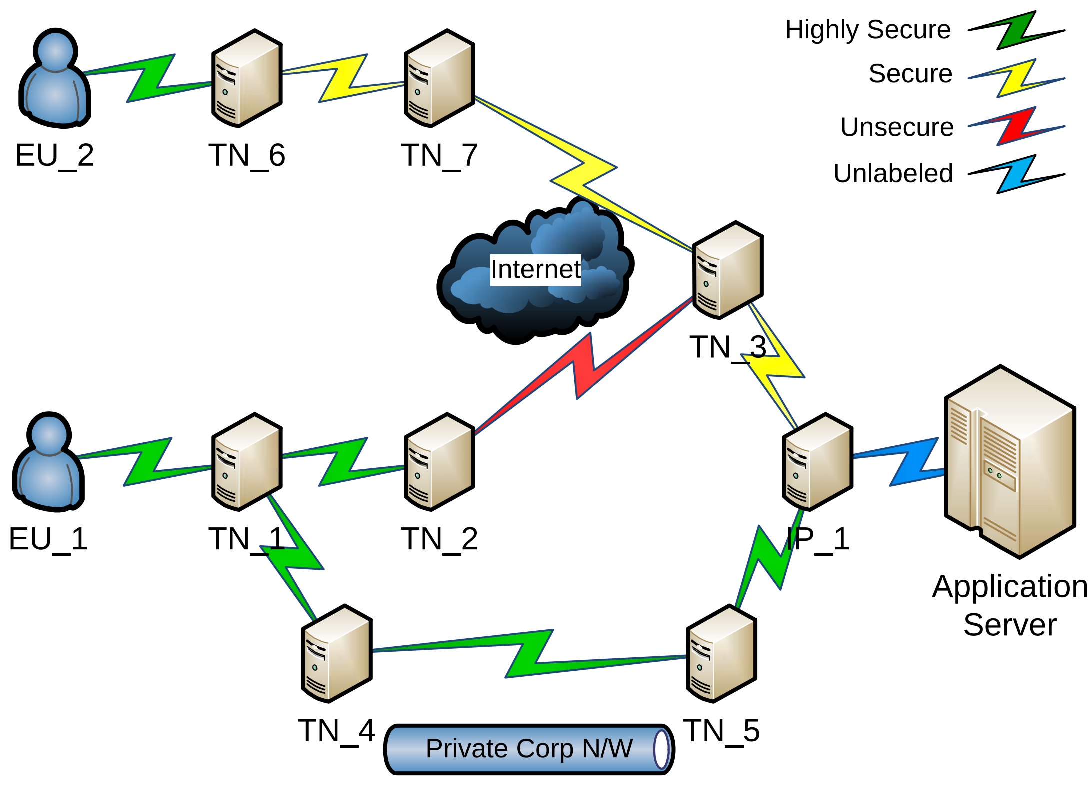


Figure 19 - Sample Network with Transfer Nodes

In Figure 19, there are 7 transfer nodes, identified as TN_1 through TN_7. Additionally, there are 2 End User (EU_1 and EU_2), an Information Provider (IP_1), and the Application Server. Finally, there are 8 associations that are depicted. The association between RM_1 and the Application Server, shown in blue, is an unlabeled object transfer; it is included for completeness, but does not require an association to be established for information exchange.

In the first example, the end user has requested an information transfer that requires a highly secured connection for the information transfer. The green associations indicate connections that are highly secure and can support this request. They show that by transferring the request and information objects via the private corporation network, it is possible to create associations end-to-end that are secure enough to support the information transfer. Because a highly secure connection can’t be established between TN_2 and TN_3, or TN_3 and IP_1, the Internet based connection is not a viable path for this transfer.

Because only an unsecure association can be between TN_2 and TN_3, any connection that makes use of this network path can’t be used. In the real world, this might represent the transfer of classified information via the Internet. In this example, the yellow associations indicate that some types of “restricted” information can be sent via the Internet, but the end user request exceeds that level of security.

In a second example, end user, EU_2, is requesting information that requires only a secure connection. Because either highly secure connections (EU_2 to TN_6) or secure connections (TN_6 to TN_7 and TN_7 to TN_3) can be established this network path can support the requested information transfer and the transfer is allowed. In this example, the second secure connection (TN_7 and TN_3) may have been enabled by the use of encryption on the Internet portion of the network path.

In either of these examples, the association could have been either weak or strong. For a strong association, four separate connections, EU_1 to TN_1, TN_1 to TN_4, TN_4 to _TN_5 and TN_5 to RM_1 or EU_2 to TN_6, TN_6 to TN_7, TN_7 to TN_3 and TN_3 to RM_1) would need to be established before the information was transferred.

If a weak association was being established, then the information transfer would have begun once the first association was established. In the case of the first example, this would mean that the information would have been sent from EU_1 to TN_1 and then from TN_1 to TN_2. It is only when the association between TN_2 and TN_3 could not be established that the connection was aborted. Weak associations carry the added responsibility for purging the information request that was stranded at node TN_2.

4.9.5.Request Proxies

The second type of Network Proxy is the Request Proxy. The Request Proxy is an "entry point" into the label based framework; it permits users and systems that are not label aware to make use of information that is secured by the framework. Figure 15 and Figure 18 shows notional networks. The first figure shows a corporate network, the second shows a larger network. In both cases, portions of the network can be treated as a single user or system. For example, any user of a resource in the corporate DMZ should not have the same level of access to resources behind the corporate firewall. Because of this, extending the labeling framework out into the DMZ provides no greater security. A request proxy can be installed as part of the corporate firewall. This extends the label based framework without incurring the cost of "labeling" every user and every system that the DMZ supports.

However, users of resources in the corporate DMZ may or may not be a known user. An unknown user would be someone, who does not have a predetermined relationship with the corporation. This might be someone using the corporate web server to comparison shop. A known user has a predetermined relationship with the corporate. An online shopper, who has bought from the company in the past, is an example of a known user. Figure 20 shows the sequence diagram for a known user session.

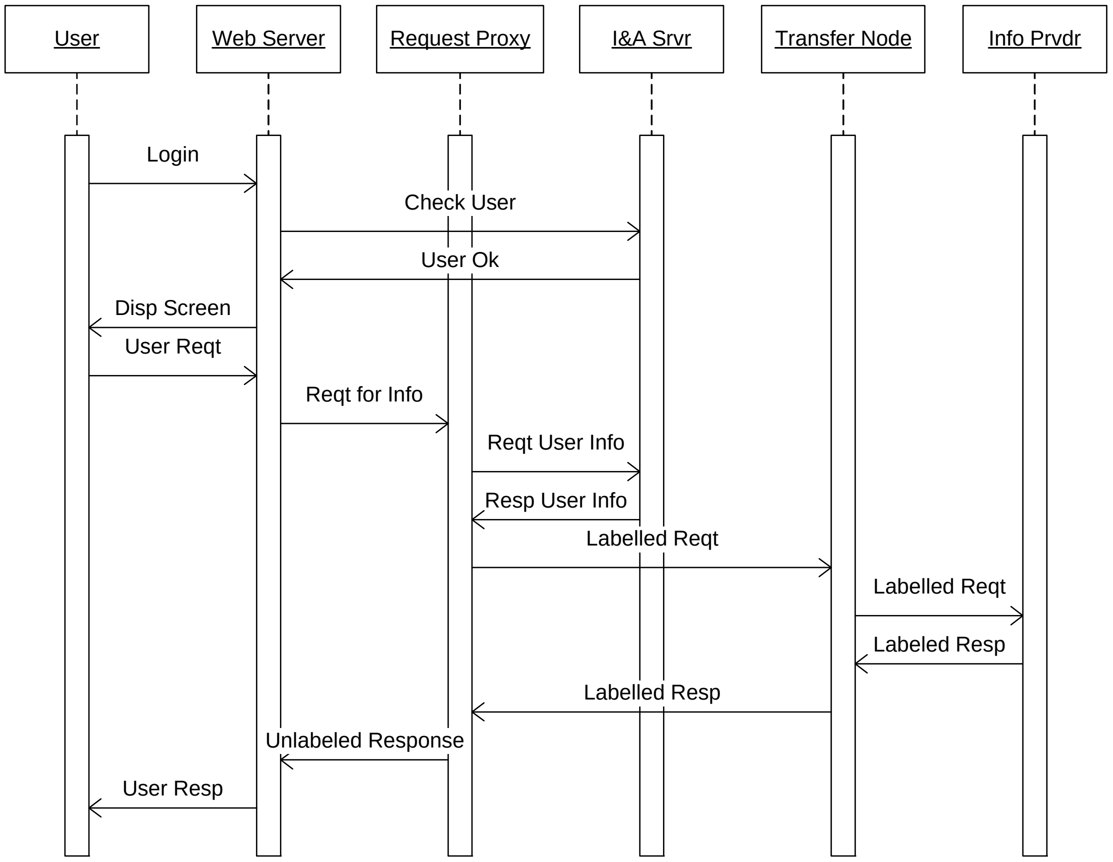


Figure 20 - DMZ Transaction for a Known User

The known user would log onto the corporate web server. Once their identity is confirmed, the known user could request their account information (or any other transaction requiring access to the sales database). The web server would forward the request to the Request Proxy, which is part of the corporate firewall. The request proxy would create a user label based on the user's login and a generic system label. The generic system label would reflect the fact that the web server in the DMZ was forwarding the request. At this point, regular labeled transaction can be executed. The labeled request would be forwarded to the Transfer Node in the Sales Firewall. The request would be evaluated and if approved then forwarded to the Information Provider in front of the sales database.

For an unknown user, Figure 21 shows the transactions. For this case, there is no initial interaction with the Identification and Authentication (I&A) server to validate the user. Instead, the web server begins by presenting those resources that are available to an unknown user. When a request for additional information is made that requires access to labeled information, the web server submits a 'Reqt for Info'. This request includes an indication that the user is unknown. (This is either clearly indicated or inferred by the absence of a valid User ID.) For an unknown user, both a generic system label and a generic user label are returned by the I&A server. After that, the forward and request process would be identical. The key feature of the Request Proxy is that extends the use of labels to users and systems that are not label aware. By doing so, my label based framework is able to treat this larger community without the need for special security practices.

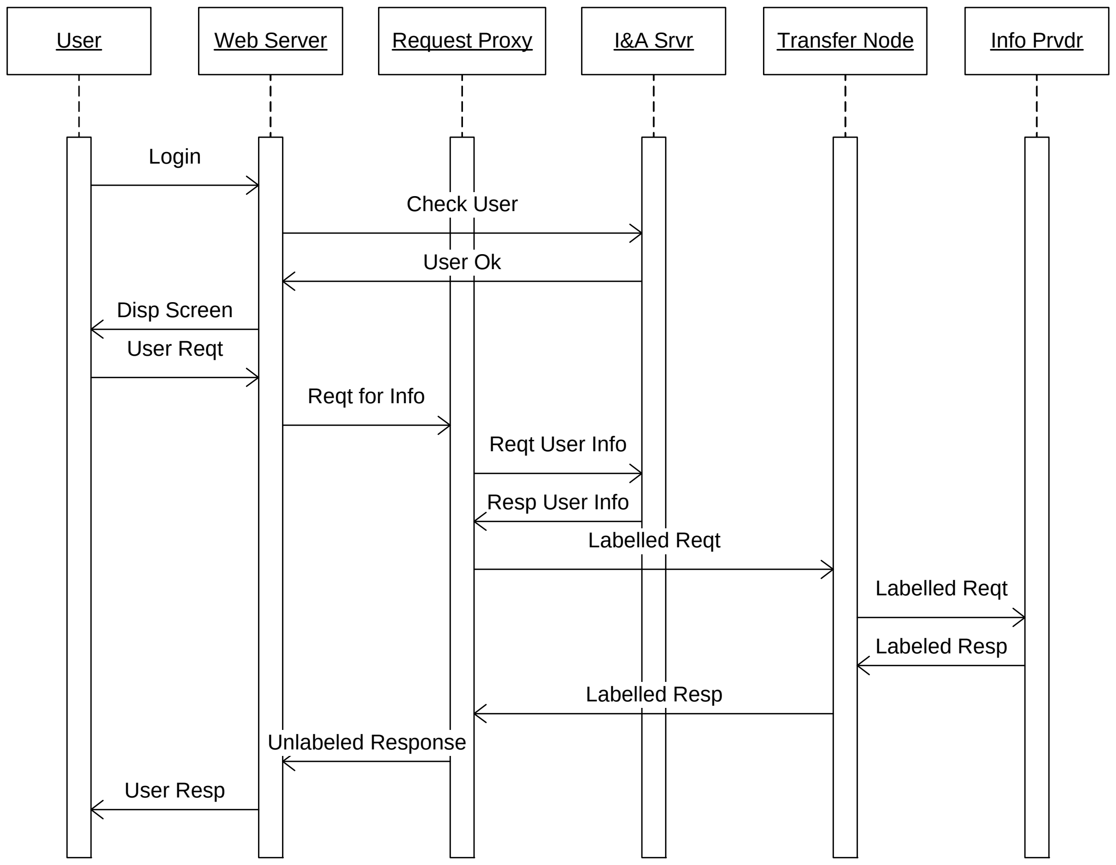


Figure 21- DMZ Transaction for a Unknown User

4.10. <u>Interoperability</u>

4.10.1.Introduction

This section presents a discussion on how this object labeling framework can be integrated with external access control solutions to form a more complete access control solution. For this discussion, the Government Open Source Access Control – Navy (GOSAC-N) system was selected as a representative external solution. GOSAC-N is an open source access control system that is maintained by Technica Corporation[37] and posted to the U. S. Government's Forge.mil[38] website. The GOSAC-N system is available as a reference implementation (RI) that demonstrates a "navy" website support three communities of interest (COIs) and how GOSAC-N is able to limit the content returned to the user based on user and object attributes. In the case of GOSAC-N, neither the user attributes nor the object attributes are conveyed with the request or object. Instead, they are maintained as part of the system infrastructure. The key focus of GOSAC-N is the policy management for access control determination.

4.10.2.GOSAC-N System Architecture

The GOSAC-N system is designed to enforce access control by making access control decision based on the user identification and a pre-established set of rules for the object being protected. Unlike traditional access control systems that rely on statics technology, such as Role Based Access Control (RBAC) or Access Control Lists (ACLs), GOSAC-N relies on newer technologies for implementing access control policies. These new technologies, which include Attribute Based Access Control (ABAC) and Policy Based Access Control (PBAC), are intended to permit a system make access control determination more dynamically. Figure 22 shows the GOSAC-N system architecture.

[37] http://www.technicacorp.com

[38] http://www.forge.mil

The GOSAC-N architecture is defined in detail in the GOSAC-N: Technica PBAC Reference Implementation [52]. For this discussion, there are two components of interest. The first is the Policy Enforcement Point (PEP) and the second is the Policy Decision Service (PDS) perform the equivalent functions to that of the Reference Monitors in the Object Labeling Framework.

The Policy Enforcement Point has the responsibility for enforcing the decisions that are made by the Policy Decision Service. It serves both as the point of presence for the user making requests of the system and also as the source for the system's response. When compared against the Object Labeling Framework, its functions are performed by the Request Processor and the Release Mechanism. Unlike the Request Processor and the Release Mechanism, the PEP is highly tailored to support the type of information being processed.

The Policy Decision Service has the responsibility for making the access control determinations. In this manner, it functions much like the Access Processor in the Object Label Framework. Unlike the Access Control Processor, none of the information that is used in making the determination is provided by either the labels associated with the object being requested or the initial query from the user. While user attributes are used in making access determination, they are based on information that is not provided as part of the query, but rather by the infrastructure.

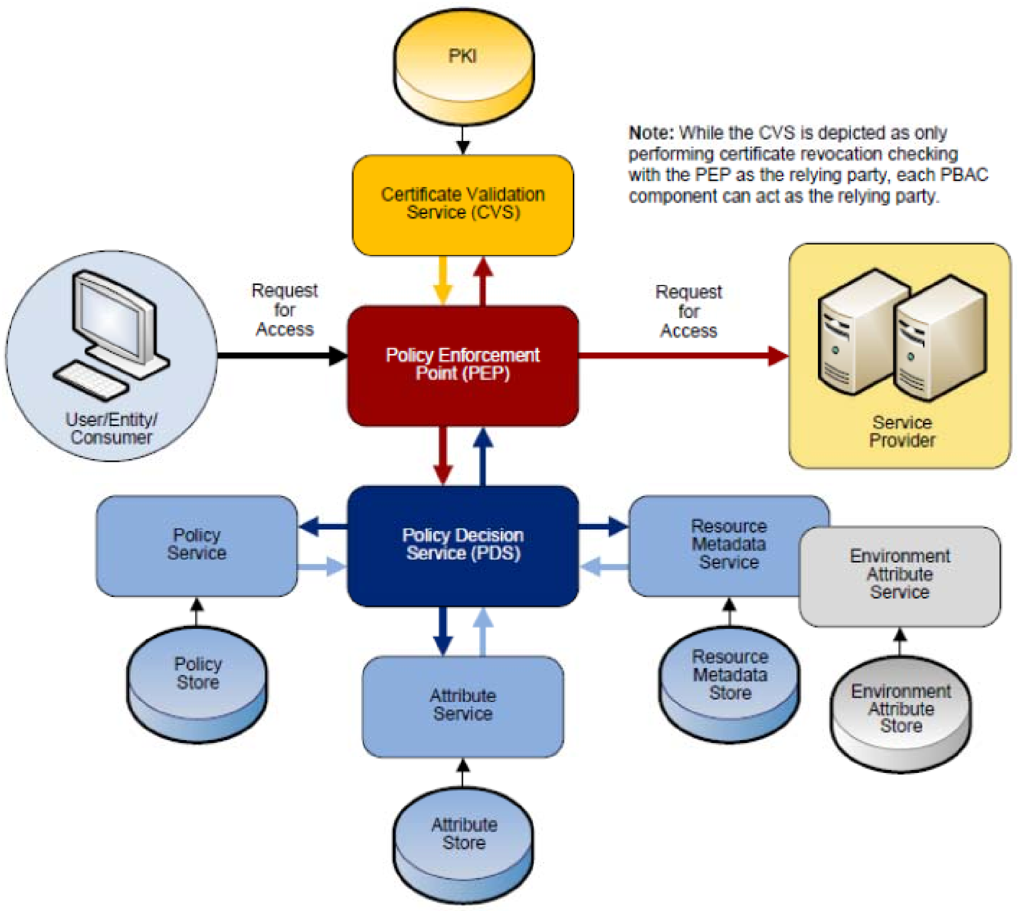

PKI
Certificate Validation Service (CVS)
Note: While the CVS is depicted as only performing certificate revocation checking with the PEP as the relying party, each PBAC component can act as the relying party.
Request for Access
Request for Access
User/Entity/ Consumer
Policy Enforcement Point (PEP)
Service Provider
Policy Service
Policy Decision Service (PDS)
Resource Metadata Service
Environment Attribute Service
Policy Store
Attribute Service
Resource Metadata Store
Environment Attribute Store
Attribute Store

Figure 22 – (Figure 2. ABAC/PBAC Architectural Components)[39]

4.10.3.GOSAC-N System Functionality

As GOSAC-N is an access control system, its primary function is to control access to the information that it is protecting. Additionally, it is intended to address to overcome some of the shortcoming of the more traditional access control systems. GOSAC-N is intended

[39] [52]Section 2.1 pg 4

to support a shift in paradigm from “need-to-know” to “need-to-share” and to eliminate “the stovepipes created by traditional access control methodologies”[40]

To implement this new paradigm, GOSAC-N replaced the traditional technology (RBAC and ACLs) for access control with newer technology (ABAC and PBAC). In so doing, GOSAC-N is able to allow for decision making at the time that the request for an object is made, rather than in advance of that request. It also allows for more responsive management of the user communities. In a traditional ACL based systems, users were added and removed from lists. These lists detailed who would have access to the information controlled by that list. If there was a change to the policy for that information, then the list would need to be updated manually. With ABAC/PBAC, users would be given attributes and based on policies, access to the object would be determined. Changes to either the user’s attributes or the policy would be reflected by access control decisions that were based on either. Entire user communities could be granted access by changing a single policy and the scope of information that a user would have access to could be modified with a single attribute change.

4.10.4.Comparison to Object Labeling

Table 7 details the difference between GOSAC-N and the Object Labeling Framework. In summary, GOSAC-N focuses less on the infrastructure and the use of access control in

[40] [53]Section 2, pg 2

a confederated environment. It is intended to address the need to improve how access control decisions are executed based on a new paradigm. The Object Labeling Framework does not develop as complete a toolset for executing access control decisions, but rather focuses on developing a confederated framework that can be deployed throughout an infrastructure to manage the distribution of and access to information.

An additional distinction between the two is how security metadata is handled. GOSAC-N relies on centrally managed user and object attributes that are provided to the PDS at the time that access is requested. This information is used to make access control decisions that are based on each request as a separate action. The Object Labeling Framework is based on the user and object attributes being transported throughout the framework and provide to the Reference Monitor in conjunction with the query. Additionally, the Object Labeling Framework provides the construction of aggregates that can utilized when the access control decision is required or stored to provide a reference for future decision.

Table 7 – Comparison of Functionality between GOSAC-N and Object Labeling

Functionality	GOSAC-N	Object Labeling
Distributed Architecture	No provisions for multiple PEPs	The use of Network Proxy allows for a distributed architecture
User and System Identification	User/System I&A are not provided as part of the query (PKI login is supported)	User and System information is included as part of each query object
Object Metadata	There are no provisions for object metadata	External object metadata is an integral part of the Object Labeling Framework
Aggregation	No provision for support object aggregation	Object Labeling supports aggregation
Aggregation Avoidance	No provision for support of aggregation avoidance	Object Labeling supports Aggregation Avoidance
History of Access	No provisions for maintaining a history of access	Object Labeling Framework details how a history of access can be supported.
Reusability	Each GOSAC-N PEP is "very resource specific" and must be tailored to the resources being protected.	Object Labeling is intended to abstract information to a common object format. Reuse should not require tailoring
Multiple Object Support	GOSAC-N focuses on information as discrete objects	Object Labeling allows for support of multiple objects through aggregation

4.10.5.Summary

In summary, both GOSAC-N and the Object Labeling Framework extend the bound of access control solutions that are available to the community. While there is some overlap in architectural functionality, this overlap provides for interfaces, where the two efforts could leverage the work being done by the other.

4.11. Cost Effectiveness

4.11.1. Introduction

Appendix F - Label Based Access Control Demonstration System contains a copy of the test plan for the reference implementation for the Object Labeling Framework.

This test plan details the test methodologies used to affirm that object and user labeling can be used as the basis for an access control system. Additionally, the report documents the results that show that this technology is viable as an access control tool. Finally, the report includes a cost assessment that shows that the functionality that was demonstrated by the system can be realized by a system that is cost effective. In summary, the reference implementation was able to support 10,000 access control requests in under 26.6 seconds. Similar levels of performance were observed for the creation and labeling of information as well as object aggregation.

The reference implementation used for testing was designed to demonstrate that object labeling could be used for access control; that an acceptable level of performance could be obtained; and that the system used for these tests could be acquired economically. The reference implementation's performance can be greatly improved through the use of software development practices commonly in use today. Many of the test results reflect a "startup" cost in time associated with the initial execution of the processes. These times could be reduced if persistent processes were used, rather than the creation and startup of each process, each time it was executed.

4.11.2.Test Results

Figure 23 shows the rate at which the system was able to create Type 1 SFF objects from raw information. From the chart, it can be seen that the system is able to handle a wide range of file sizes effectively. From the data, it appears that the "knee" in the curve only appears for file size above 2,000,000 bytes.

For object aggregation, the systems performance is equally good. Table 8 shows the results from the aggregation tests. What this data reveals is that the performance begins to taper off when more than 16 objects are in an aggregate. However, the performance when aggregating 64 objects is still 90% of that when aggregating 16 objects. One feature that Figure 24 reveals is that the average time to perform an aggregation is still less than 150 milliseconds for an aggregate with 64 objects. When coupled with the 2.5 millisecond time for making an access control decision, the total "real" time required to make an access control decision for dynamically created aggregates is still approximately 150 milliseconds.[41]

[41] Much of the time associated with creating an aggregation appears to be independent of the size of the aggregate and most like represents the processing time required to initial the aggregation process. Reduction of this time should be possible through the use of a persistent aggregation processor.

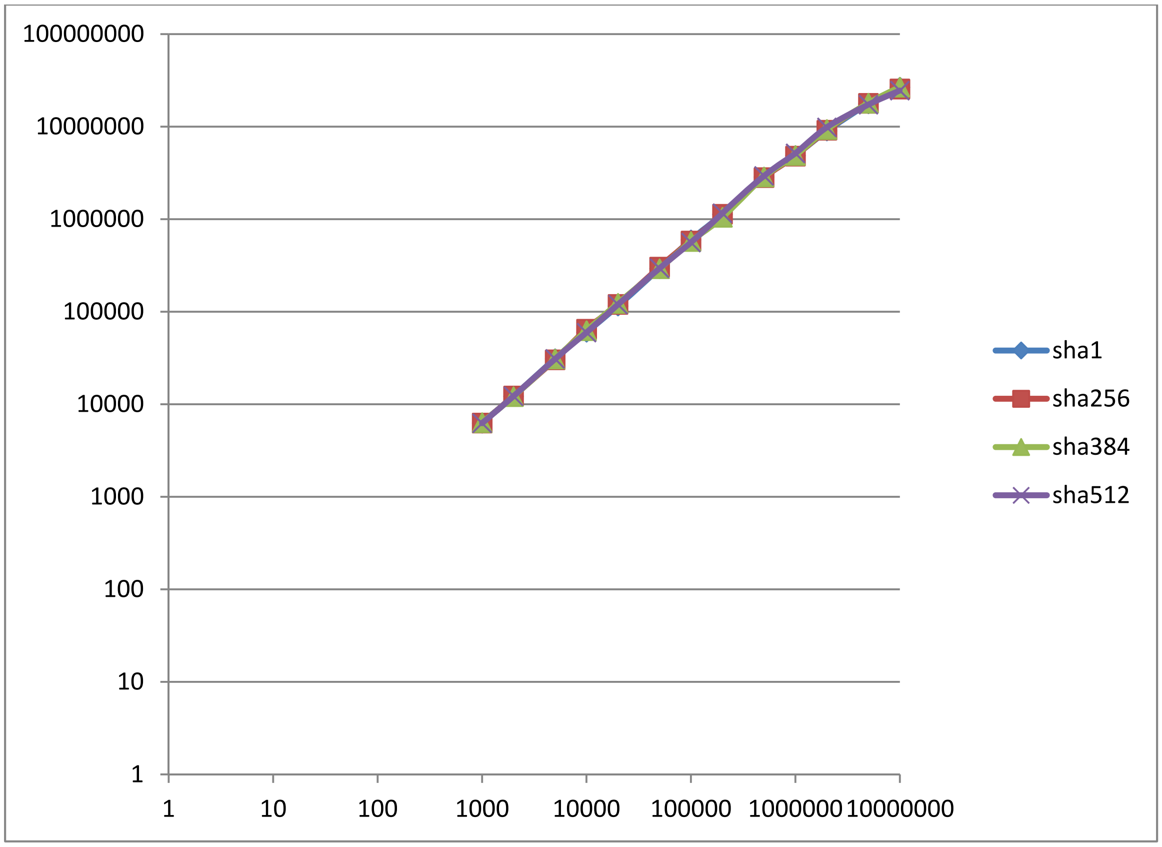


Figure 23 – Data Rate for Wrappering Objects\

Table 8 – Aggregation Rates

	Data Rate			
	sha1	***sha256***	***sha384***	***sha512***
2	15.82	15.82	15.71	15.72
4	30.89	30.83	30.82	30.90
8	61.33	61.25	61.41	61.33
16	120.77	120.49	120.86	120.81
32	223.30	223.26	223.38	222.87
64	432.00	431.29	431.49	431.80

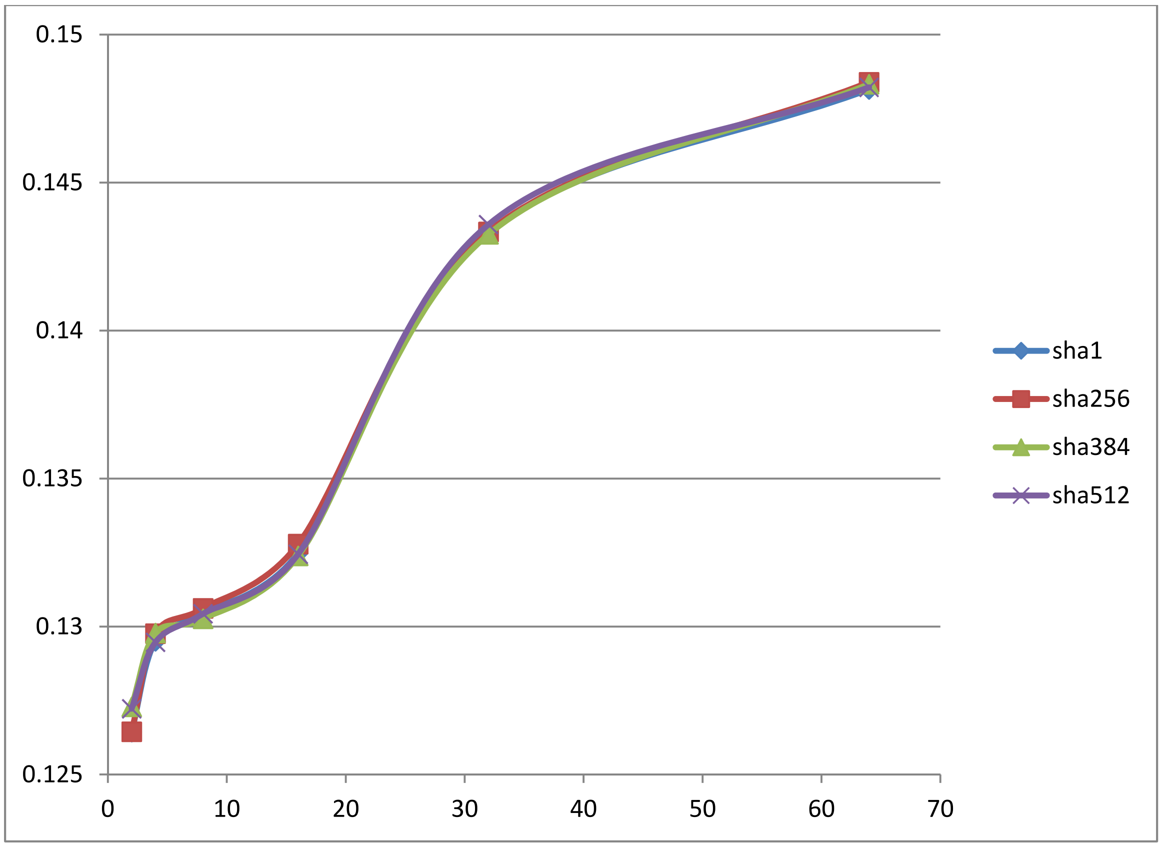


Figure 24 – Average Time to Aggregate multiple objects

5. EXTENDED USAGE OF OBJECT LABELS

5.1. Introduction

Thus far, object labels have been used as a means to grant or deny access to information. This information is contained in individual objects and the labels are evaluated on a one by one basis. By deploying Information Providers and Network Proxies throughout a network infrastructure, it is possible to secure the information that has been distributed across the network. However, as the volume of information that is immediately available because of the infrastructure increases, so does the need to expand the range of control of access to that information. Beyond evaluating object labels singularly, it is possible to provide that greater object security by using object labels. By working with multiple object labels concurrently, a wider range of security problems can be addressed. I have extended the work done thus far to address some of the problems associated with aggregation.

The first of these extensions is to consider two or more objects as a collection of objects. This is referred to as “Information Aggregation”. Information Aggregation considers the effects of grouping two or more objects together to form a single compound object. Because it does not consider time or past history, the aggregated labels can be evaluated and stored for future use.

The second extension is "Complex Aggregation". Complex Aggregation considers time or past history as a factor in determining access. For these types of controls, it is not just about preventing an unauthorized user from gaining access to information, but insuring that an authorized user is prevented from acquiring too much information too quickly. One additional requirement for employing Complex Aggregation is the need to establish a mechanism for securely maintaining the history of object use.

5.2. Information Aggregation

5.2.1. Introduction

"*Information Aggregation*" refers to the process of creating and applying additional labels that convey the information carried in the labels associated with the individual objects. The additional aggregated labels are constructed to maintain labeling information that is derived from the objects being aggregated. This construction is done by an external process called an Aggregator. Figure 25 shows how aggregated labels are formed.

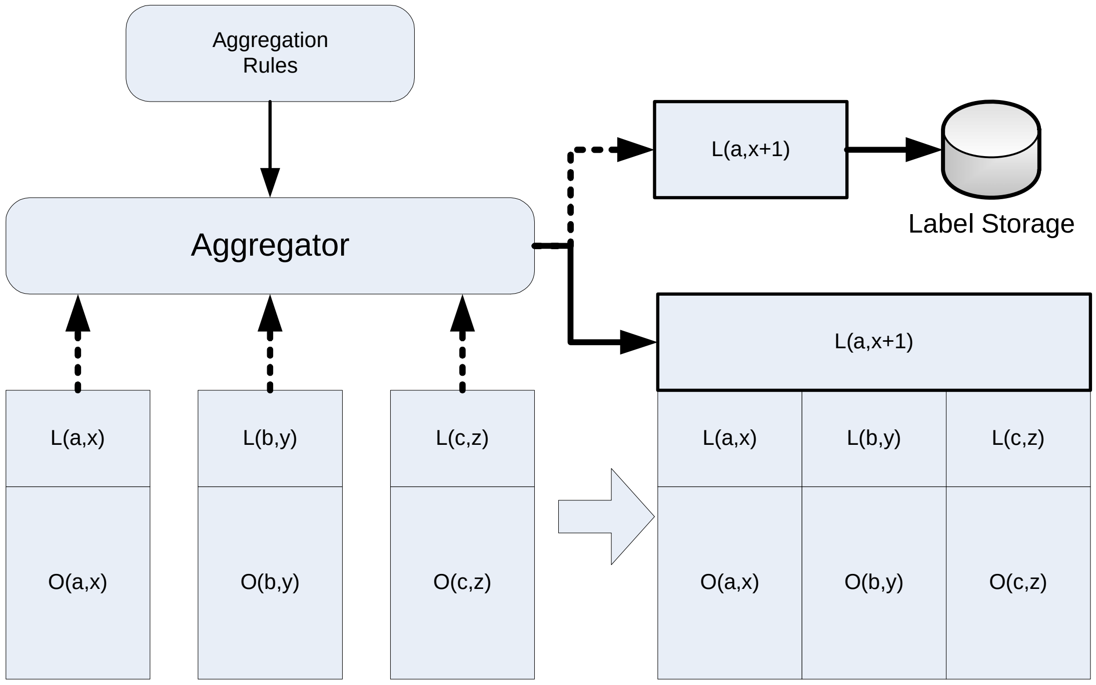


Figure 25 - An aggregated object using Simple Aggregation

The Aggregator collects the labels for the objects to be aggregated and constructs the new object label. This new label can either be applied to the new aggregated object for immediate processing or the aggregated label can be created and loaded to a data store. In the cases, where the aggregated object is to be used for immediately release to the user, it is used as the input to the Reference Monitor. Once the new object is returned to the Reference Monitor, the Reference Monitor treats the aggregated object using the simple access control rules. The key feature is that the construction of the aggregated object and determining access to the aggregated object are two separate activities. Figure 26 shows how this would be constructed.

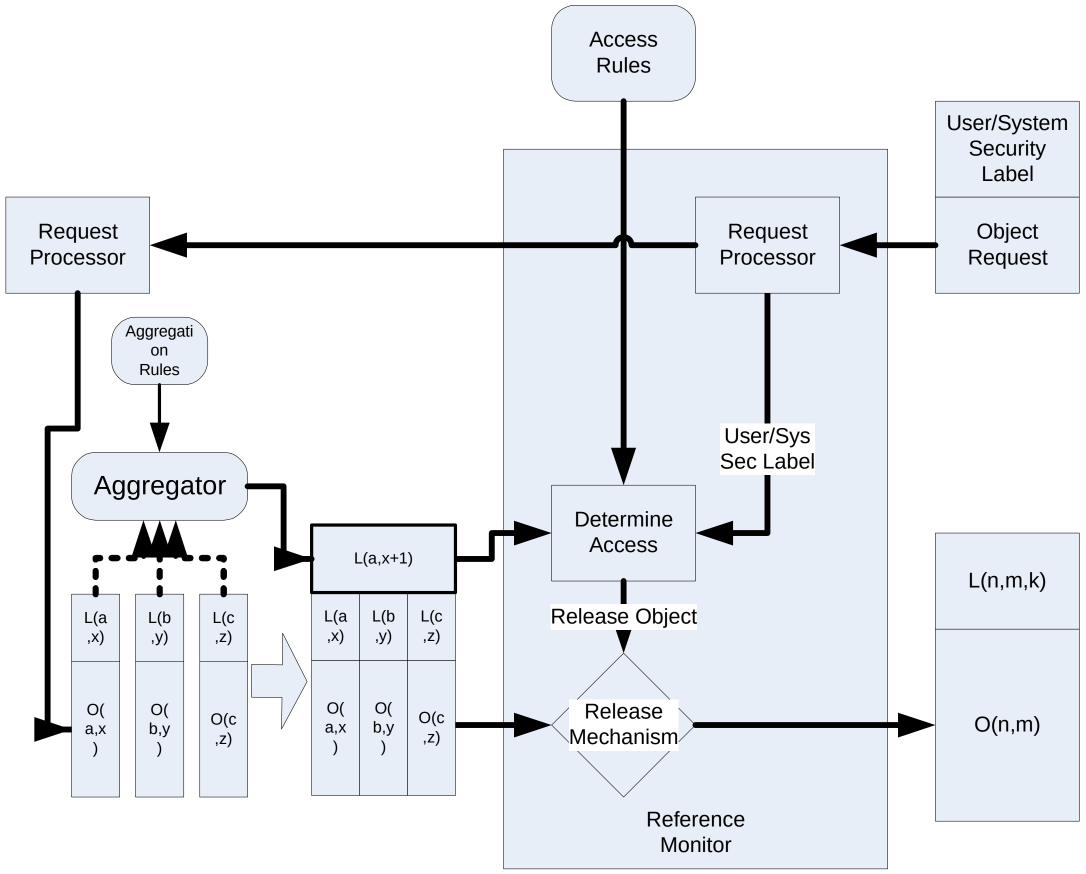


Figure 26 - Configuration of an Aggregator and Reference Monitor

Being able to store an aggregated label for future use introduces the requirement for a new set of tools for the management of information beyond the securing of the object. While I don’t discuss the use of object labels as a tool for content management and configuration control, a properly designed object store would be able to leverage this functionality. A starting point for this work would be Rozenbroek and Sibley’s work on escrow servers [54] [34] as a configuration management tool. Because the secure hash field in the aggregated label is cryptographically linked to each of the hashes for aggregated objects, only the aggregated label needs to be stored for future use. If the data store is able to associate each labeled object with its secure hash value, then the "chain of hashes" can be used to reconstruct the aggregated objects without having to store the aggregated object as separate entity.

Figure 27 shows how Secure File Format would associate each of the hashes of included objects would be secured. For the creation of aggregated headers, a Secure File Format Type 6 header is employed.

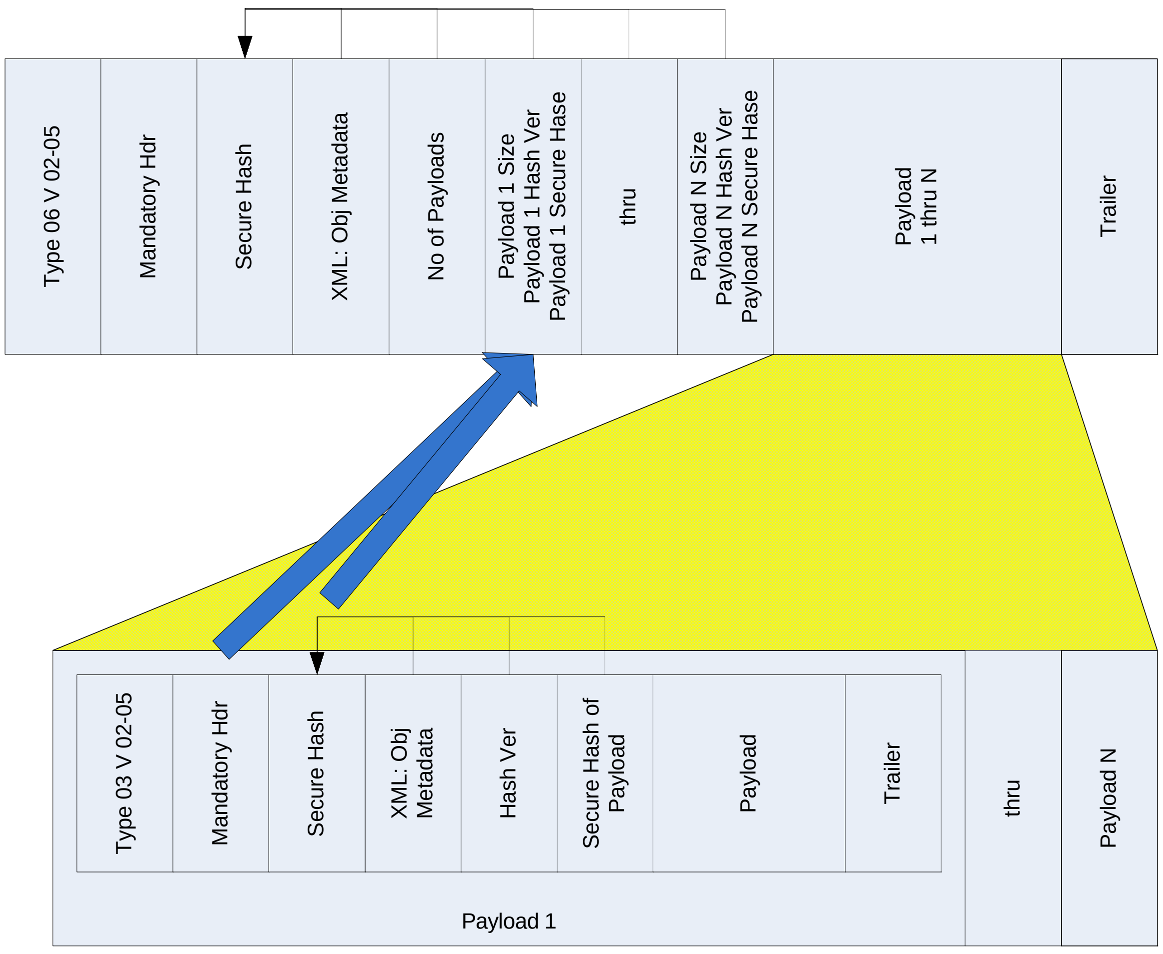


Figure 27 - Aggregated Secure File Format Headers

There are two forms of simple aggregation; “Concatenated Aggregation” and “Cumulative Aggregation”. Each aggregates information from the individual object labels based on how the aggregate will be used. The rules for executing an aggregation are stored as an XML object. The general format of which is shown in Figure 28. The format of “*Processing Rules*” is determined by the form of aggregation being applied and the type of label being aggregated. One key feature of aggregation is that each field in the object label is treated separately. Therefore, it is possible to aggregate one or more fields using Concatenated Aggregation and use Cumulative Aggregation on the remainder.

```
<Aggregate>
    <Label>
        <Name>{Name}</Name>
        <Type>{Type}</Type>
        <Form>{Form}</Form>
        {Processing Rules}
    </Label>
    …
    <Label>
    …
    </Label>
</Aggregate>
```

Figure 28 - XML Representation of an Aggregation Instruction

The <Label> and </Label> tags are used to denote the beginning and end of rules for each of the fields in the label. When aggregating an object labels, there must be one set of <Label>…</Label> tags for each field in the label. If there are fields for which there are not aggregation rules, then the aggregation fails and a new aggregated label is not created. However, it is acceptable to have aggregation rules for which there are no corresponding label fields. In these cases, no additional fields are created.

The <Name>{*Name*}</Name> tags are used to denote the name of the field upon which the aggregation is being performed.

The <Type>{*Type*}/<Type> tag is used to denote the type of field being aggregated. Table 9 shows the possible values for this tag.

Table 9 - Possible Values for the Type field in an Aggregation description

Type Value	***Description***
HIER	Used to denote a hierarchical value
CATE	Used to denote a categorical value

For the fields in the object label that have been defined as conditional, the conditional logic is executed to produce either a hierarchical or categorical type field.

The <Form>{*Form*}</Form> tag is used to denote the form of aggregation that is being performed. Table 10 shows the possible values for this tag.

Table 10 - Possible Values for {Form}

Form Value	***Description***
CONCAT	Denotes that Concatenated Aggregation should be performed
CUMULA	Denotes that Cumulative Aggregation should be performed

5.2.2.Concatenated Aggregation

For Concatenated Aggregation, the object labels are processed to create a new aggregated object label. The key feature of this type of aggregation is that the aggregated label values are generated using simple set theory principles. Figure 29 show the XML representation of a concatenated aggregation rule.

```
<Label>
    <Name>{Name}</Name>
    <Type>{Type}</Type>
    <Form>CONCAT</Form>
    <Condition>{Condition}</Condition>
</Label>
```

Figure 29 - XML Representation of a Concatenated Aggregation Rule

The <Form>{*Form*}</Form> tag is always set to "CONCAT" denoting that this is a Concatenated Aggregation.

The <Condition>{*Condition*}</Condition> tags will take on one of four values. The possible values are determined by the {*Type*} tag value. Table 11 shows what values are possible for each {*Type*} value.

Table 11 - Possible Values for {Condition}

Type Value	***Condition Value***	***Description***
HIER	MAX	Set the value of the aggregated label to that of the label with the largest label value or most precedence of the labels of the objects being aggregated
HIER	MIN	Set the value of the aggregated label to that of the label with the smallest label value or least precedence of the labels of the objects being aggregated
CATE	AND	Collects only those category values that are common to every label of the aggregated objects
CATE	OR	Collect the category values from ALL of the labels of the aggregated objects

{Condition} set to MAX

When the {*Condition*} is set to "MAX", then the aggregator collects all of the label values for the {*Name*} fields. Once collected, they would be sorted and placed in order. Next, the one with the largest value or greatest presence is selected and it will be used to create the aggregated label value. This can be expressed as:

$L_{aggregate} = max\{L_0, L_1, \ldots , L_n\}$

This type of aggregation could be used to label a compound document composed of sections at different classification level. For example, if an information request was made that returned several documents with different classification levels. This type of aggregation would be used to create the document that was marked in accordance with DoD-5200.1-PH.[23] The aggregated label would be the overall classification of the constructed document. This type of aggregation would be in compliance with a Bell-LaPadula style action. Creation of the new document would not release any information at a lower level of protection.

One point that needs to be stressed is that when the largest value or greatest precedence can’t be determined, the aggregation will fail and the aggregated object label will not be created. Figure 30 shows a lattice that may result in cases, where precedence can’t be determined. For example, if the objects with labels, L(2,1), L(2,2), and L(3,2) are being aggregated, it is not possible to determine what the resulting concatenated aggregation should be. While it is clear that L(3,2) has precedence over L(2,2), it is not possible to determine if L(2,2) or L(3,2) has greater precedence. In this case, an aggregated object label will not be created.

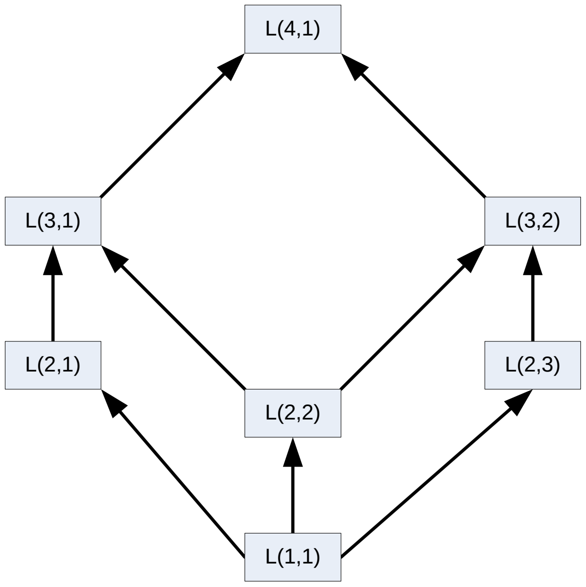


Figure 30- Ambigious Precedence

{Condition} set to MIN

When the {*Condition*} is set to "MIN", then the aggregator collects all of the label values for the {*Name*} fields. Once collected, they would be sorted and placed in order. Next, the one with the smallest value or least presence is selected and it will be used to create the aggregated label value. This can be expressed as:

$$L_{aggrgate} = min\{L_0, L_1, \dots, L_n\}$$

This type of aggregation could be used to label a compound document composed of section with different levels of integrity. For example, if several sources of information were collected to create a report, that final reported (the aggregated product) would only carry the integrity assessment of the most unreliable piece of information in the report.

This type of aggregation would be in compliance with a Biba style action. Creation of the new document would not upgrade any of the information to a higher level of integrity

As with a "MAX" condition, if the smallest value or least precedence can't be determined, then the aggregation will fail and the aggregated object label will not be created.

{Condition} set to AND

When the {*Condition*} is set to "AND", then the aggregator collects all of the label values for the {*Name*} fields. For this type of aggregation, the aggregated label contains only those values that are common to all of the object labels being aggregated. This can be expressed as:

$$L_{aggregate} = \cap\ (L_0, L_1, \ldots , L_n)$$

This type of aggregation could be used to label a compound document with only those labels that will allow access to all of the component objects. For example, a company labels its engineering projects so that only members of the project and engineering management have access to the objects. If the status reports for two or more engineering projects were aggregated, then only those employees that were identified as "engineering management" would be granted access to the aggregate object.

{Condition} set to OR

When the {*Condition*} is set to “OR”, then the aggregator collects all of the label values for the {*Name*} fields. For this type of aggregation, the aggregated label contains any of the values that are found in the object labels being aggregated. This can be expressed as:

$L_{aggregate} = \cup (L_0, L_1, \ldots , L_n)$

This type of aggregation could be used to label a compound document with any of the labels that would permit access to all of the component objects. For example, if two companies are collaborating on a joint venture, then an employee with a “company” label value for either company would be granted access to that aggregated object.

5.2.3.Cumulative Aggregation

There are situations when simply concatenating the labels of the original label values does not provide enough protection for the new larger object. In these cases, aggregating individual objects generates a new larger object with label values that are not reflective of the value of the information that the new object contains.

For example, Quist[29] discusses the aggregation of location information concerning missile silos.   Taken separately, each silo’s location is unclassified. However, if the location of enough U. S. missile silos is collected in a single report, then that report becomes classified. As more location information is included, the level of classification increases. Ultimately, the report contains enough critical information that is warrants a TOP SECRET classification. From this example, it is easy to see that there are at three points in the aggregation at which the classification level changes (UNCLASSIFIED to CONFIDENTIAL, CONFIDENTIAL to SECRET, SECRET to TOP SECRET). This can be expressed as:

$$L_{aggregate} = \begin{cases} N \geq T_{TS}, TOP\ SECRET \\ N \geq T_S, SECRET \\ N \geq T_C, CONFIDENTIAL \\ N < T_C, UNCLASSIFIED \end{cases}$$

Where T_{ts} = the number of silo locations needed to produce a TOP SECRET result,
T_s = the number of silo locations needed to produce a SECRET result
T_c = the number of silo locations needed to produce a CONFIDENTIAL result

For Cumulative Aggregation, the object labels are again processed to create a new aggregated object label. Unlike Concatenated Aggregation, Cumulative Aggregation attempts to account for the fact that aggregation may produce a result that is greater than the sum of its parts.

One issue that needs to be restated, I was unable to find a much literature on how this is being handled today. Much of what I found identifies this as problem, but clear guidance or rules for implementing Cumulative Aggregation was lacking. In presenting a framework for performing this type of aggregation, I hope to foster future research.

The following shows the XML representation of a Cumulative Aggregation rule:

```
<Label>
    <Name>{Name}</Name>
    <Type>{Type}</Type>
    <Form>CUMULA</Form>
    <Case>
        <Condition>{Condition_1}</Condition>
        <Value>{Value_1}</Value>
    </Case>
    …
    <Case>
            <Condition>{Condition_1}</Condition>
            <Value>{Value_1}</Value>
    </Case>
</Label>
```

Figure 31- XML Representation of a Cululative Aggregation Rule

The <Form>{*Form*}</Form> tag is always set to “CUMULA” denoting that this is a Concatenated Aggregation.

The <Condition>{*Condition*}</Condition> tags are used to identify under what conditions the {*Name*} label will take on {*Value*}.

The <Value>{*Value*}</Value> tags are used to identify the value that the {*Name*} label will assume, if the {*Condition*} is met.

In the case of Biba type integrity problem, the logic would allow for the integrity of the aggregated object to be increased, if there were enough elements supporting the same assertion.

5.3. Complex Aggregation

5.3.1. Introduction

One of the main reasons for having a system for classifying information is so that information can be afforded an appropriate level of protection. Beyond being a security issue, there is a cost element behind this logic. There are numerous examples of safeguards being required in order adequately protecting information. In general, the more security that is required to protect something, the greater the cost. One of the ways to reduce cost is to not have information that requires the greater degree of protection.

Security label can be used to avoid creating aggregations of information that would require the greater degrees of protection.

5.3.2. Aggregation ("Critical Mass") Avoidance

One of the features presented thus far is the ability to upgrade the security of an aggregation of labeled objects based on a pre-defined security rule. That same logic can be applied in reverse. Instead of increasing the security classification, when informational critical mass has been reached, security labels can be used to prevent information critical mass from being reached by limiting the amount of labeled information that is collected in a single place.

One of the key attributes of object labeling in this framework is that they can be separated from their objects without any loss of integrity. Because of the strong linkage between the object and its label, the label can be processed without the object being involved. Earlier in my dissertation, it has been shown that this allows grant/deny access decision to be made before the object is exposed. For critical mass avoidance, a similar logic can be implemented. In this case, the aggregated security label can be created and evaluated before the individual objects are aggregated.

For critical mass avoidance, there are two implementations that were considered. The first implementation deals with the creation of individual objects that should not exceed a predetermined rating. The second implementation deals with preventing too much information from being released too quickly.

Figure 32 shows the components involved with a release process that avoids the release of information that has exceeded a critical mass values. In this case, a label aggregator is used to create the aggregated label that would have been created by the aggregator, if it has been executed. (In a real world implementation, the label aggregation function and the object aggregation functions would be performed by a common software component.) The Label Aggregator outputs an aggregated object label. This label is compared against the Critical Mass Levels to determine if aggregating these objects would produce a prohibited object. If the Critical Mass Levels would not be reached by the aggregation, then the aggregation is executed and the aggregated object is created. This object is them returned to the requester for external consumption and the new aggregated label is written to the Label Store for future use. Figure 33 shows the sequence diagram for these operations.

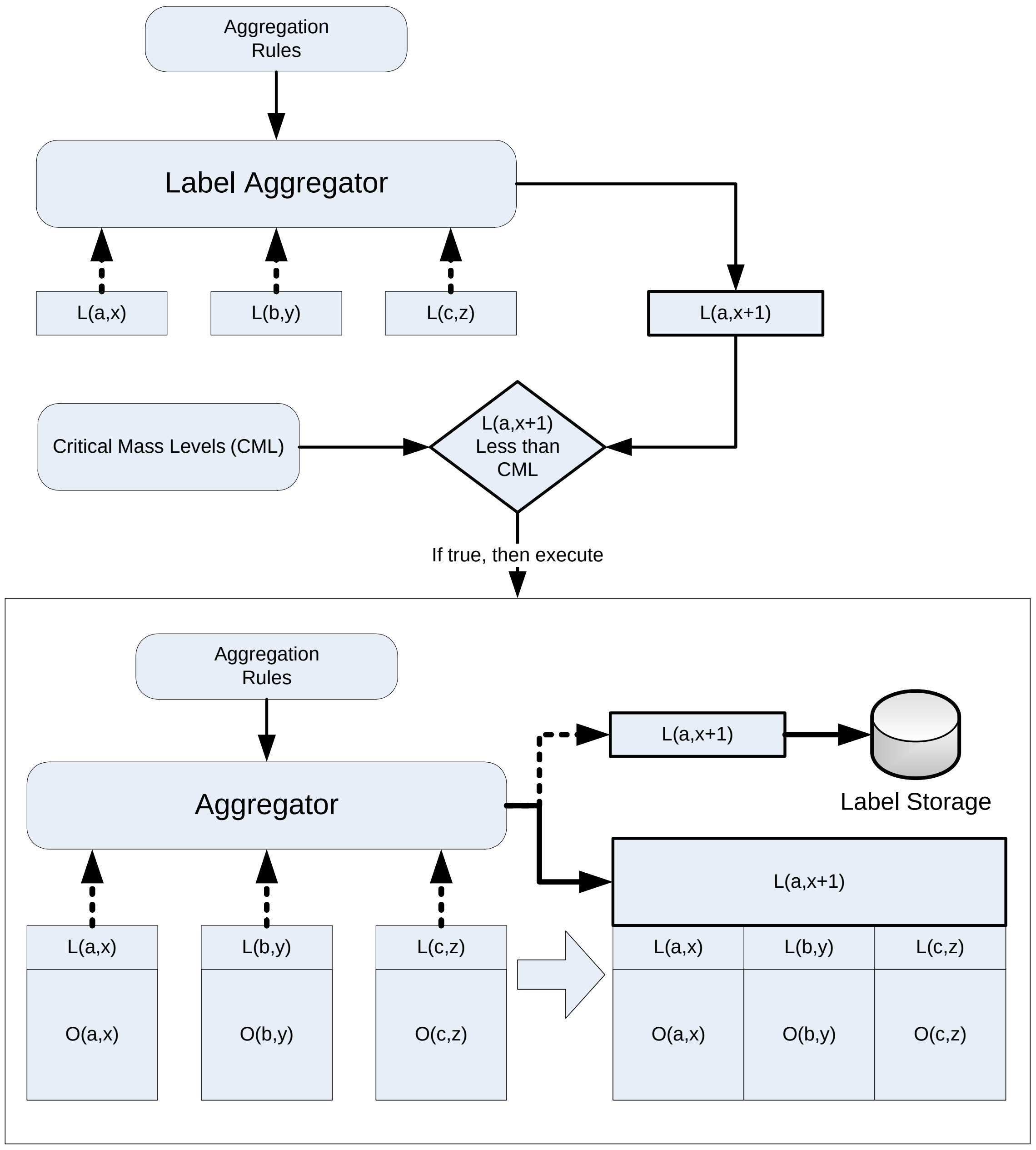


Figure 32 - Critical Mass Avoidance

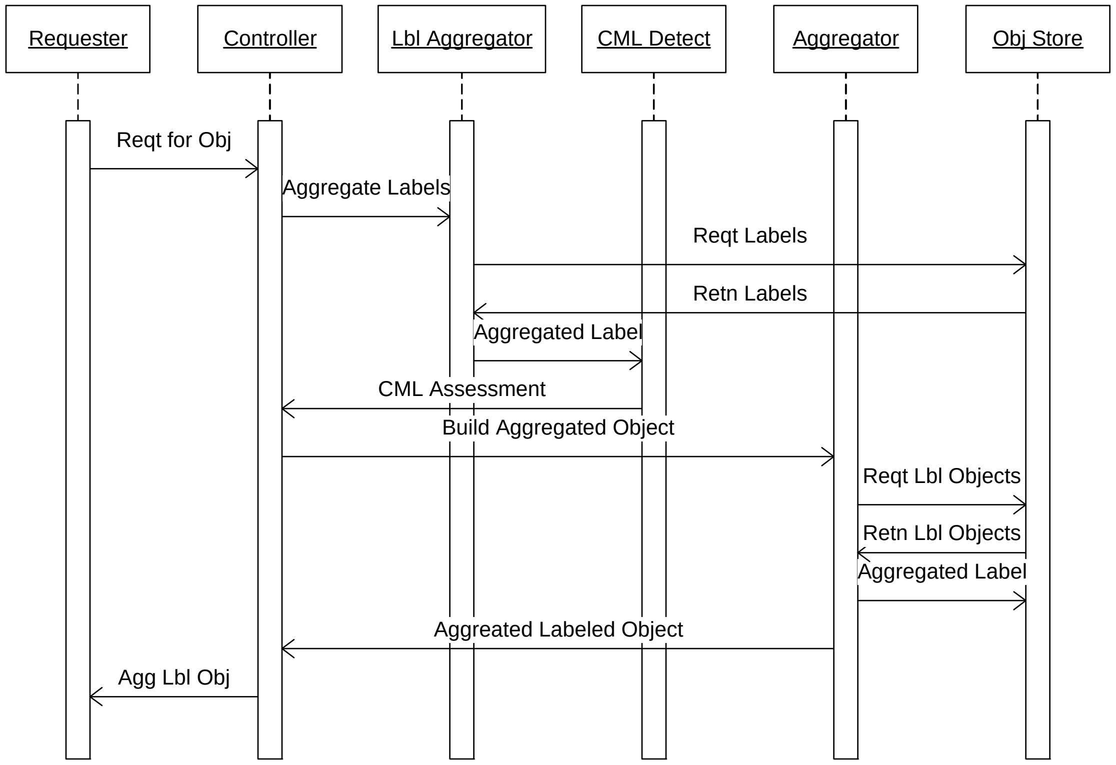


Figure 33 - Sequence Diagram for Critical Mass Avoidance

In the second case, a new aggregated object is not created. Instead, this case is used to track the information that has already been released. The external requesting process can be considered as the aggregated object. The goal of Critical Mass Avoidance in this case is to limit the amount of information that is being released. This allows for some information to be released, while insuring that too much information is not acquired too quickly.

A key addition to the new Aggregator is memory. In order to prevent Critical Mass from being reached, the aggregation process must keep track of which objects have previously been provided to the end user. Unlike the case for Cumulative Aggregation where no effort is expended to prevent the creation of aggregated objects with higher classification[42], the enhanced system requests that the labels be provided to the aggregator without their objects. The Aggregator processes the labels and determines if the aggregated object that would be created should be classified at the higher level. If the proposed object exceeds the classification threshold, then the information request is denied. In the case where all of the information is already resident in a single location, the extra processing may be considered unwarranted. However for a network connected environment where the request for information, which is distributed over more than one system or location, being able to avoid the collection of too much information can be very valuable. If the aggregation process is incorporated into the input processing for a data warehouse or other information storage system, then it can be used to insure that the data warehouse does not violate its accreditation.

Figure 34 shows the configuration of the system that would be used. In this configuration, a new Label Storage component is added. This Label Storage element is used to provide the object labels for objects that have already been released.[43] These labels along those of the new objects being requested (or loaded) are used by the Label Aggregator to create a new aggregated label. As with the Critical Mass Avoidance process, this label is compared against the Critical Mass Levels to determine if the new

[42] In the case of a single system that is hosting enough information to create an aggregated object with a higher classification, it is assumed that that system is already approved for the higher level of classification.

[43] If the aggregation process is part of the input process for a data warehouse, then the Label Storage is used to track the labels of the objects that are already being stored in the warehouse.

request can be completed. If the new label’s values don’t exceed the Critical Mass Levels, then the release process is allowed to execute. The new object and its labels are provided, along with the labels of the objects previously released by the Aggregator. The Aggregator creates a new aggregated label that is loaded into the Label Storage system for future reference. Figure 35 shows the sequence diagram for these operations.

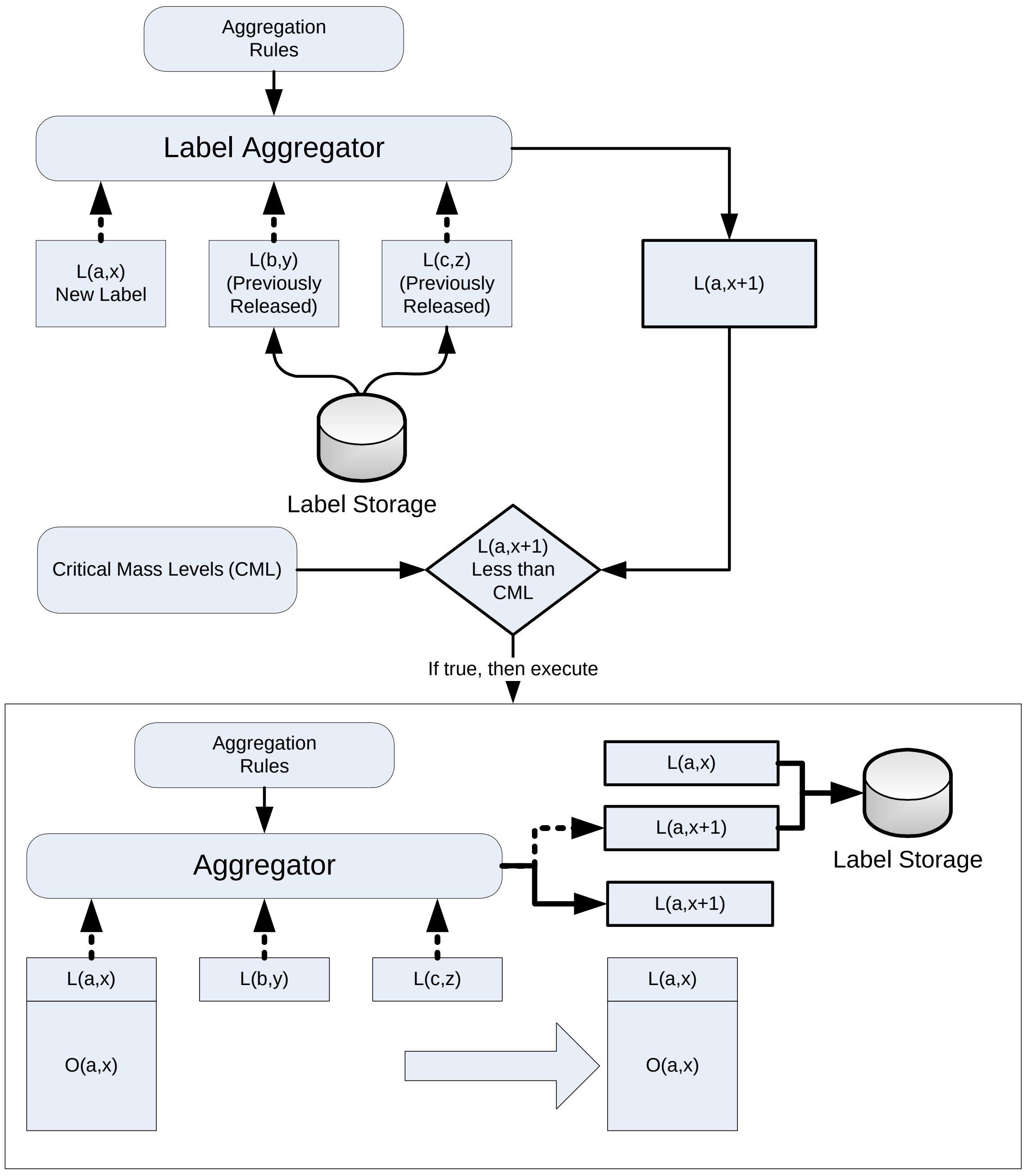


Figure 34 - Critical Mass Avoidance for Time Release

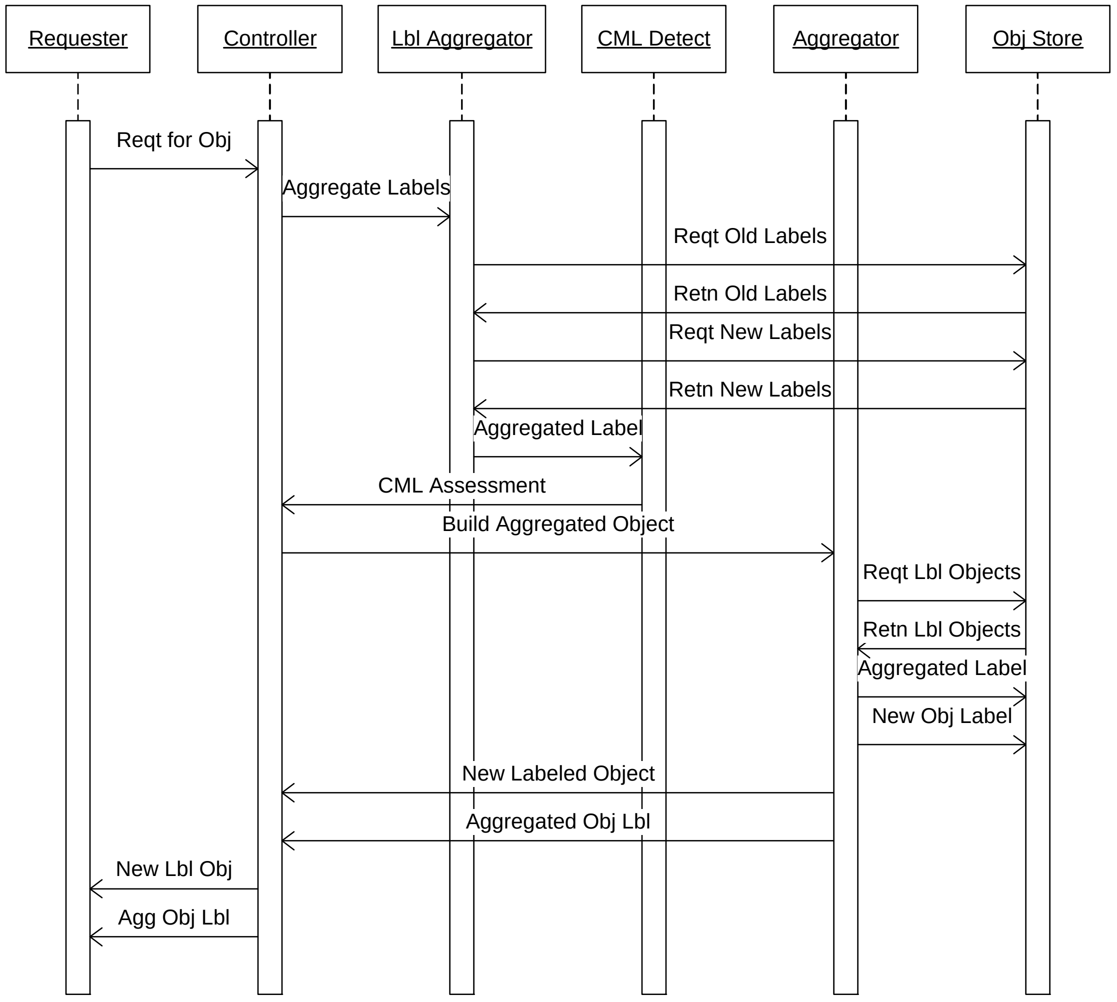


Figure 35 - Sequence Diagram for Timed Critical Mass Avoidance

5.3.3.Chinese Wall Problems (Control of Resources dynamically)

The Chinese wall security policy is one of the models used to model how information is controlled. The goal of this model is to prevent a user from having access to too much information concurrently. Brewer and Nash[8] present the example of three companies, a bank, Bank A, and two oil companies, Oil Company A and Oil Company B. In this case, the end user can have access Bank A’s information without restriction and either Oil Company A’s or Oil Company B’s information, but not both concurrently. The Chinese wall security policy presents the concept of “conflict of interest” classes. The bank would be a member of one conflict of interest class and the oil companies are members of a different conflict of interest class. In the context of object labeling, conflict of interest classes can be modeled as different category labels. Each company name would be represented as a category value.

The Chinese wall problem can be viewed as a special type of informational critical mass avoidance problem. For Chinese wall problems, the critical mass level is reached when the second object is requested. The end user can have at most one object in each ‘conflict of interest’ class. In this dissertation’s framework, conflict of interest classes are captured using category labels. If a second object from a conflict of interest class is requested, then that request will be denied. One possible extension to the Chinese wall problem is “returning information”.

If one considers the case of an analyst working for a consulting firm, it is quite possible that the consulting firm may have two or more clients in the same conflict of interest class and that an analyst for the firm may be working on more than one account. In order for the end user, the analyst, to gain access to the second company’s information, a second object, the first company’s information must “returned”. This increases the complexity of the security solution because it will need to track only which end users have been provide with objects, but also which objects the end user has returned.

In order to address this extension to the Chinese wall security problem, one final enhancement to the timed critical mass avoidance aggregator and the Reference Monitor connected to it is required. The Label Storage subsystem must be enhanced and connected to the Reference Monitor. This is done to allow the Information Provider to update the Label Store to indicate that label objects have been returned to it. The concept of returning an object is analogous to returning a book or report to a library. However, because the original object was not purged from the Information Provider, there is no actual need to transfer the object from the end user workstation back to the Information Provider. An object is considered returned when the object manager on the end user workstation asserts that the released labeled object has been deleted. Figure 36 shows the sequence diagram for returning an object.

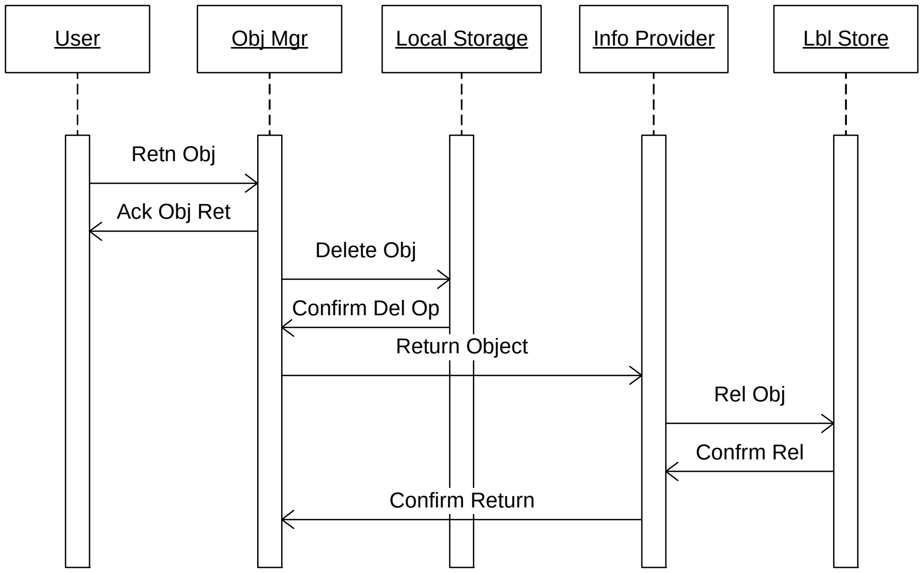


Figure 36 - Sequence Diagram for Returning an Object

In the sequence diagram, the User returns the object by instructing the local Object Manager that she is finished using it and it can be returned. The Object Manager acknowledges the request to return the object. It then deletes the object from the Local Storage system. The local storage system is a storage solution on the end user workstation that is managed by the Object Manager. The key attribute is that it is not accessible to the user except through the Object Manager. After the Local Storage confirms that the object has been deleted, the Object Manager issues a Return Object request to the Information Processor. This request can be made directly to the Information Provider or via a Network Proxy. The Information Provider releases the object from the Label Storage. The Label Storage confirms that the object has been released and the Information Provider then confirms that the object has been returned.

One additional feature of this approach is that the user can still be barred from gaining access to objects in the conflict of interest class after an object has been returned. Because access to the objects in the conflict of interest class is being managed by the object labels, any of the techniques previously discussed are available as part of this solution. One reason for delaying access to a second company's information is to provide for enough time that analyst forgets what she knows about the first company. Additionally, this prevents the analyst from hopping back and forth between two company's information thus defeating the security system. If there is a two week delay between the return of information and access to new information, the analyst's ability to work concurrently with two sets of information in the same conflict of interest class is eliminated.

6. CONCLUSION/FOLLOW UP

The use of external security labels that are strongly attached to the object that they protect is a departure from the traditional methods by which security is implemented. Traditional methods do make use of external metadata, but this metadata is not tightly coupled with the files/objects being protected. Additionally, much of that metadata is not transferred with the object as it is transported over a network.

In my dissertation, an alternative to this approach has been presented. It includes a method for strongly binding the external security metadata to the object and transporting that metadata along with the object throughout its lifecycle. Additionally, it was shown that the external security labels containing that metadata do not need to be conjoined with the object that it is protecting. In can be separated from the objects, processed, and reattached with no loss in integrity of the label or the security of the object. In fact, being able to separate the labels from the objects increases the object's security, because only the labels needed to be exposed in order to make access decisions for the object. Only after the decision to grant access to the object has been made is the object access or transferred.

With the establishment of a language for creating object labels, user and system labels, a set of rules for access the objects based on these labels, a framework for their use was presented. This framework is able to the labels and rules, not just for a single source of information, but rather can be implemented on a wide area basis. Additionally, the Object Labeling Framework includes provisions of addressing users and systems that are not label aware. This framework was realized by a reference implementation. Beyond demonstrating that the developed framework could be used for as an access control system, the cost analysis that was performed on the reference implementation confirmed that the system that would support the documented results was cost effective.

With my labeling framework development complete, I extend the language behind simple access control to address the problems of aggregation. Using the same XML representation, I have shown the same framework, with some extension, can support the creation of aggregations of labeled objects. Because these aggregates are also labeled objects with formatted security metadata, access to them is not different than for other labeled objects. The creation of aggregated objects can be done in real-time or in advance of any request. The reference implementation demonstrated that aggregation in real-time does not introduce an excessively delay is processing a request. While not investigated as part of my thesis, the use of aggregation labels as a tool for document release management warrants further investigation. A second research area that having labeled object should encourage is cumulative information aggregation. My framework provides the framework and language for creating and implementing these aggregations. The next step would be to develop the actual rules based on business practices.

The final area, I discuss is the use of my framework to avoid creating aggregations of labeled objects that would create security problems. This includes the collection of too many objects in a single locations and the collection of too many object too quickly. With the ever increasing accessibility provided by the Internet, these functionalities offer a means of "metering" access to information. Examples of this type control include a solution to the Chinese wall security problem based on object labels.

Looking forward, there are several areas of research that should be considered.

As previously stated, object labeling should be considered as a tool for document release management. Aggregation should be usable to create documentation that is based on smaller documents that are managed and concatenated at the time of publication.

Studies in the area of Cumulative Aggregation should also benefit from having a framework for realizing aggregation rules. Part of the problem with writing this dissertation was a lack of published work on aggregation from a security perspective. There is a great deal of research on aggregating information from an information management perspective, but almost nothing on security. This area of information management is not unlike the Internet in the 1980s and early 1990s, it is focused on functionality, not security. The Object Labeling Framework supports aggregation, extending this framework to capture metadata about the aggregation process should facilitate the research in business practices for cumulative aggregation

The next logical step is to construct a working implementation of the framework and assess its use in real-world networks. This would include not just the design and coding of Network Proxies, Aggregators, and interfaces to information source for production use. The current reference implementation was coded using PHP running on an Apache web server. Both the use of a scripting language and the hosting within a full-blown web server has incurred a processing overhead that could be reduced. The use of a compiled programming language, such as JAVA, that contains native support for file system and network connections would reduce this overhead, thus improving performance. Additionally, the use of persistence should reduce the response times by eliminating startup times. It should focus on the creation of standard for labeling objects and for writing the rules to govern access.

This dissertation focused on object labeling for distributed access control. It would permit access control to be realized not just on the edges of the cloud, but throughout the cloud, itself. A future area of study should consider the incorporation of GOSAC-N type functionalities that would supplement the access control rules discussed herein.

I believe that my framework can be implemented with not too much effort, it represents a major first step towards improving the security of information in large heterogeneous network environments. However, additional work will need to be done on the rules for accessing information objects and the business rules for complex problems, such as aggregation and aggregation avoidance. Key to this work will be the leveraging of existing technologies, like GOSAC-N, SecureXML[55], XACML as well as enhancement to the products presented in this dissertation.

APPENDIX A - LABELLING FRAMEWORK

Introduction

In order to make use of security labels, a common syntax and clearly defined set of rules for representing labels must be defined. In the following sections, the syntax and rules for handling objects and object labels are presented. In the first section, objects will be discussed, followed in the second section by a separate discussion on the labels that will be attached to the objects. The key distinction between “what is an object” and “what is a label” is that an object is what needs to be protected or described and a label is that entity is used for protection of or to describe the object. Figure 37 shows the relationship between the different types of objects being addressed in my dissertation.

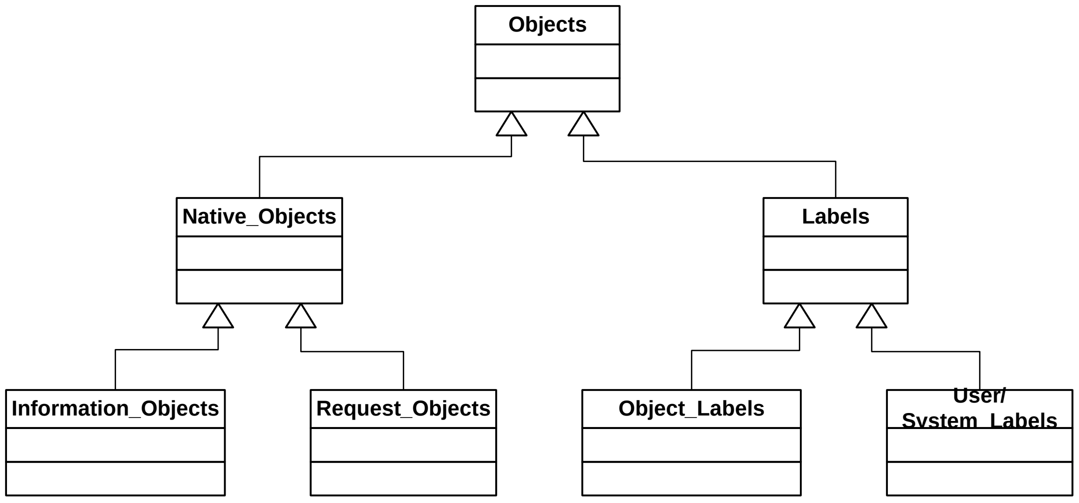


Figure 37 - Objects and Labels

<u>Objects</u>

Introduction

Objects are those items to which labels will be added to provide protection. In the real world, an object is anything to which an external set of information can be added to provide information about that object. Real world examples of objects include letters and packages, sent via the postal services and documents to which protective covers and routing sheets can be applied. In cyberspace, examples of objects include emails, file system entries (files, directories, pipes, etc), responses to queries, web pages, and transactions. In general, anything that can be encapsulated and whose contents can be selectively hidden can be considered to be an object.

For my dissertation, the exact contents of an object are not important. Instead the focus will be on how the object is used.

Native Objects

A native object represents the information being protected by security labels. For any object that is going to be protected by external labels, it can be represented by a box with an object designation in the box. Figure 38 shows how an unlabeled or "native" object can be represented. An object in its 'native' for can be anything that can be represented or modeled as an object. For example, the individual files in a directory structure are objects. Additional examples include entries in a database, objects that are transported over an Internet connection, and uniquely identifiable fields in an XML data stream. Each object can be uniquely identified by "N(n,m)". The first index, n, is used to designate the n^{th} object and any labels that are associated with that object. The second index, m, is used to indicate the "order" of the object. The order of the object refers to the number of labels that have been attached to the object. If m equals 0 then the object is native. If m is greater than or equal to 1 then the object contains 1 or more security labels as well as the object.

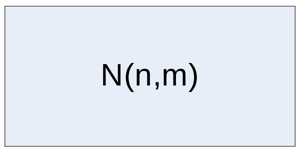


Figure 38- Native Object

Information Object

The first class of native objects is the “information objects.” Information objects are used by a system or system component to provide information in an objectized form. Information Objects are represented by a box with a unique identifier, “O(n,m)”. Figure 39 shows how an information object is presented.

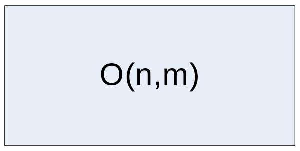


Figure 39- Information Object

Request Objects

The second class of native objects is “request objects”. Request objects are used by the user/system to request information, the response to the request is returned as a labeled information object. Like other native objects, they are labeled by the information systems as a means of transferring security metadata about the request. The attached labels will carry information about the user making the request and infrastructure over which the request object traverses. Figure 40 shows a request object.

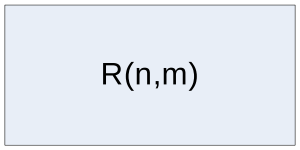


Figure 40- Request Object

Request object are distinguished from other native objects in that the security labels that are applied to them document the security attributes of the user making the request and network path (systems traversed) used to deliver the request object to the system that will process the request. Rather the security associated with them is determined by the user and systems involved in making the request. In the cases where the request object needs to be protected, an object label should be applied prior to having other labels applied.

Labeled Objects

As the name implies, a labeled object is any object to which one or more labels has been applied. As detailed earlier, a labeled object is one for which the value of m is greater than or equal to 1. The depiction of a labeled object is identical to that of either a native or request object.[44]

[44] A second depiction for labeled objects will be presented as part of the introduction of object labels. Either depiction is acceptable.

<u>Labels</u>

Introduction

Labels are the core mechanism for conveying security information about objects. They are external to the objects that they are designed to protect. NIST's FIPS 188 [56] defines the role of security labels as follows:

> *"Security labels convey information used by protocol entities to determine how to handle data communicated between open systems. Information on a security label can be used to control access, specify protective measures, and determine additional handling restrictions required by a communications security policy."*[45]

The use of an external mechanism, such as labels, to secure and manage objects must adhere to certain principles in order to have value as a means of providing security to object.

First, any information that is secured using object labels must be able to be restored to its original condition. The rationale behind this principle goes back to the basic information security requirement for data integrity. NIST Special Publication 800-33 defined data integrity as "the property that data has not been altered in an unauthorized manner"[46]. Adding additional information, externally to the original data must not alter the original data. If adding the object labels does is allowed to alter the original information in a non-

45 [56] pg 1

46 [57] pg 21

recoverable manner, then an “unauthorized” alteration has occurred and the object label has failed to protect the original object.

Second, any labels attached to an object must be able to be separated and reattached without any loss of linkage between the label and its object. Additionally, any processing performed with the label while separated from the object must not invalidate the integrity of the label or weaken the security, it is provides to the object. One of the key goals of using labels to secure object is protecting the object until after the appropriateness of its release has been determined. If the object must be exposed along with the object label in order to determine if the object can be exposed, then the use of object labels is invalid.

Object Labels

For any label that is used to protect an object, either native or labeled, it can be represented by a box with a label designation in the box. Figure 41 shows how a label can be represented. A label can carries 2 indexes. The first variable, ‘n’, indicates to which object the label is applicable. The second index, m, indicates the level or tier at which the label is applied. Lower values of m indicate that the label is closer to the object.

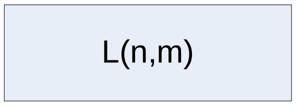


Figure 41 - Basic Label Representation

Because an object can be labeled numerous times, this second index is important for uniquely identifying which tier of labels is being discussed. Figure 42 shows the relationship between a tier 0 labeled object and a tier 1 object.

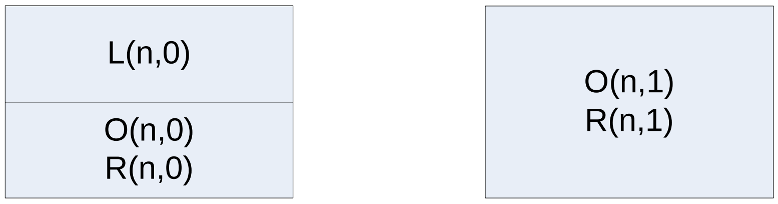


Figure 42 - Representation of a Labeled Object

Figure 43 shows the more general relationship between object and labels at different tiers.

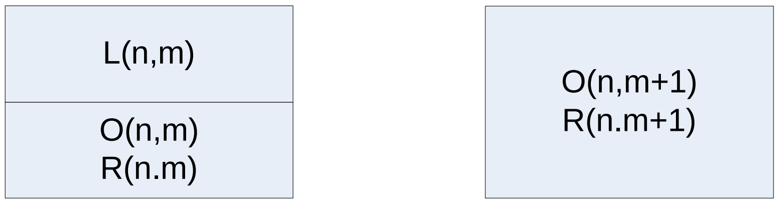


Figure 43 - General Representation of a Labeled Object

In general, a labeled object is not a native object to which a single label has been applied. While the use of single label is applicable for many problems, the use of a security framework that allows for the use of multiple layers of labeling is more desirable.

User Labels

Just as request objects are a class of native object, User Labels are a class of object labels. User labels are used to convey the security attributes of the user that has made the request. It is worth noting that a user is not limited to the end-user (human) making the ultimate request, rather a user is any end-user or application that is making a request. The treatment of applications as end-users is important in that it allows for greater information control. By treating applications in this manner, control of the information can be executed on any element of the network over which the information would be carried. Examples of applications would include end-user applications, such as word processors and spreadsheets, as well as network/server components, such as proxies (including Cross Domain Solutions), web servers, middleware applications, search engines, and database servers. Their inclusion permits security to be implemented not just on the “edges” of the network, but also within the network, itself. Since there is no difference between an end-user and an application, end-users, automated applications, and proxies can all be modeled in a common manner. Figure 44 shows how a user label is represented.

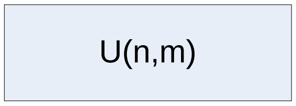


Figure 44 - User Security Label

System Labels

A second type of label that is applied to request objects is the system label. System labels like user labels convey security attributes. For system labels, the attributes are applicable to the system from which the request is being made. Figure 45 shows how system labels are represented.

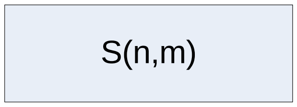


Figure 45- System Security Label

Labeled Request Objects

Request object are a class of native objects and as such follow the same rules as native objects for labeling. The key difference is that both user and system labels will normally be applied to request objects. Because there are two different types of labels that can be applied to a request object, the application of the labels is slightly different.

For non-request object, the security labels are added and the indexing incremented as the number of labels increases. For request objects, the order in which labels are added becomes somewhat more important.

In general, the first label added to a request object is the user label. This label conveys the security attributes of the user and may be used to identify and authenticate the user. Figure 46 depicts the request object with a user label attached. The same tiering convention is used. With the first (user) label applied the request object can be referenced as R(n,1).

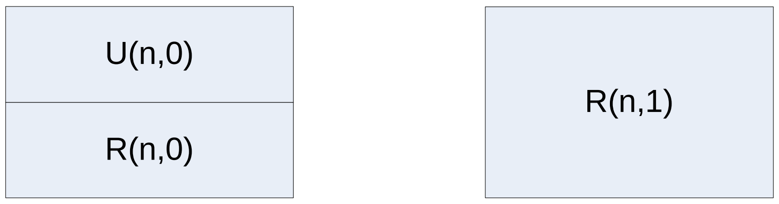


Figure 46 - Request Object with a User Label attached

After the user label is applied, the security label for the system that hosted the user is added. As the request object is transported from intermediate system to intermediate system, each of these systems appends its security label to the request object. For each intermediate system that handles the labeled request object, an additional system label is added. Figure 47 depicts how an object is represented, when a user and system are attached. It is worth noting that in this case, the resulting labeled object is denoted

R(n,2). This is to denote that the request object is carrying two labels, despite the fact that the first system label was added.

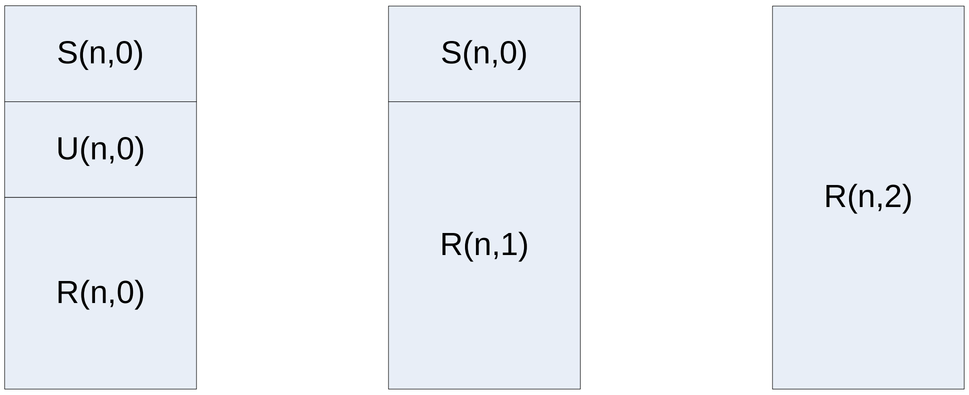


Figure 47 - Request Object with a User and System Label attached

Figure 48 shows the more general representation of an object after a user label has been applied.

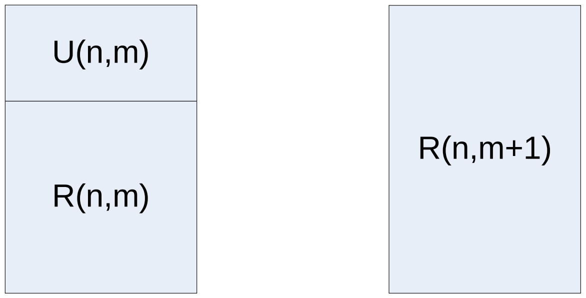


Figure 48 - Generic Request Object with a User Label attached

Figure 49 shows the more general representation of an object after a system label has been applied.

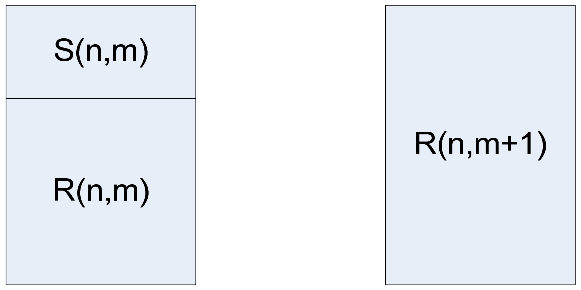


Figure 49 - Generic Request Object with a System Label Attached

APPENDIX B - SECURE FILE FORMAT

Introduction

In *An Architecture for Managing Access to and Permission for Multiple Versions of Objects in a Distributed Environment* [34], I introduced the "Secure File Format"[47] and offer a structure for building multi-tiered labeled objects. The design objectives for Secure File Format included:

> Development of a securable means for transferring information over unsecured environments;
> Provide an operating system agnostic framework for handling information that can be treated as objects;
> Be able to work with information in any format; and
> Allow for additional controls and information (such as audit entries and digital signatures), in the form of external labels to incorporated into the object over its lifecycle.

Because Secure File Format meets these objectives, which some of which are requirements of my dissertation's framework, it was chosen as the foundation for much of the work. Secure File Format offers a simple structured framework for defining external labels that could be used to manage the objects to which they were attached. This framework is lightweight and expandable. Three key features of the Secure File Format framework are:

[47] I'm currently updating a RFC for submission to the IETF for consideration as an Internet standard. As this version has not been submitted, a copy is included as an appendix in my dissertation.

that is treats all “secured data files” without regard to their content or format;
the use of multiple types of labels, each with a defined function; and
the use of a single format for all header types.

I decided on this approach rather than trying to create a single label format that would address all of the management and security needs of the labeled object for several reasons.

As stated, one of the design requirements for Secure File Format was to be agnostic of the contents of the information being labeled. This is important for several reasons. First, it allows for any type of object to be protected by the addition of one or more external wrappers without having to alter the object’s contents. (Encryption is not considered altering the contents of the object, as it neither adds nor removes any information from the object. An encrypted information object can be returned to its original state by the use of decryption.) Second, the contents and format of the object have no affect on the external security mechanism.

By using separate defined header for each management/security operation, expanding the Secure File Format to support additional functionality can be accomplished without having to change any existing headers. The inclusion of a version number in each header allows for new version of existing headers to be added, again without changing existing versions of the header. For example, adding new hash sizes to the header supporting the Secure Hash Algorithms (SHA) was done without changing the headers supporting the smaller hash sizes.[48] Any software making use of the shorter hash sizes was unaffected by the inclusion of the new hash size.

By treating management/security operations on an individual basis, the order in which they are applied and processed is based on the task which that wrapper supports. This means that externally labeled information that conforms to the Secure File Format standard can easily be introduced into a more diverse set of environments than labeling schemes that assert a pre-defined order for their application. This flexibility allows labeled information object to be a more viable means of managing and securing information.

One of the design objectives of Secure File Format was the ability to augment an existing object with new labels throughout its lifecycle. Smaller lightweight headers or “wrappers” support this functionality more easily. Existing wrappers don’t need to be changed, altered or re-applied as the wrapped object is augmented with new information. If previously attached headers contain “signoff” or “release authority” that make use of digital signatures and secure hashes, then the integrity of these signoff is not affected by the addition of new wrappers.

[48] In an earlier version of the Secure File Format RFC, only message digest sizes of 160 and 256 bits were defined. Additional digest sizes of 384 and 512 bits were added later. Their inclusion in the Secure File Format standard required only that version 04h and 05h be added to the secure hashing algorithm. For fielded applications that are currently using versions 01h, 02h, and 03h of the secure hashing algorithm and don’t have a need for the larger message digests, no changes or alterations are required.

Finally, having a single format for all headers, the design and construction of tools for working with the headers could be greatly simplified. (This will be discussed in more detail later in this section.)

A second reason for using Secure File Format is that Secure File Format defines a mechanism by which labels (in the form of wrappers) are securely 'attached' to objects. This secure attachment is done, while still allowing the labels to be utilized by external applications after being separated from the wrappered objects. This feature is critical because it allows the information in the labels to be processed without the object being required. For access control purposes, this means that the decisions concerning access to the object or objects can be made without exposing the protected object, until after access to the object has been granted. In addition to access control decisions, independent processing of the information in the labels contained in the wrappers can be used for network congestion management, metering of information release, as well as business process decisions, such as limiting access to resources based on subscription levels.

Internal Structure of a Secure File Format Object

As stated, a key functionality associated with Secure File Format is the ability to add new layers of functionality by including additional labels. In Secure File Format terminology, this is referred to as adding additional wrappers. A wrapper consists of a header that pre-pended to the object and an optional trailer. This functionality is a critical component in preparing the security labels associated with users and systems as well as processing aggregation. Figure 50 shows the general layout of a Secure File Format object including the relationship between headers, trailers and wrappers.

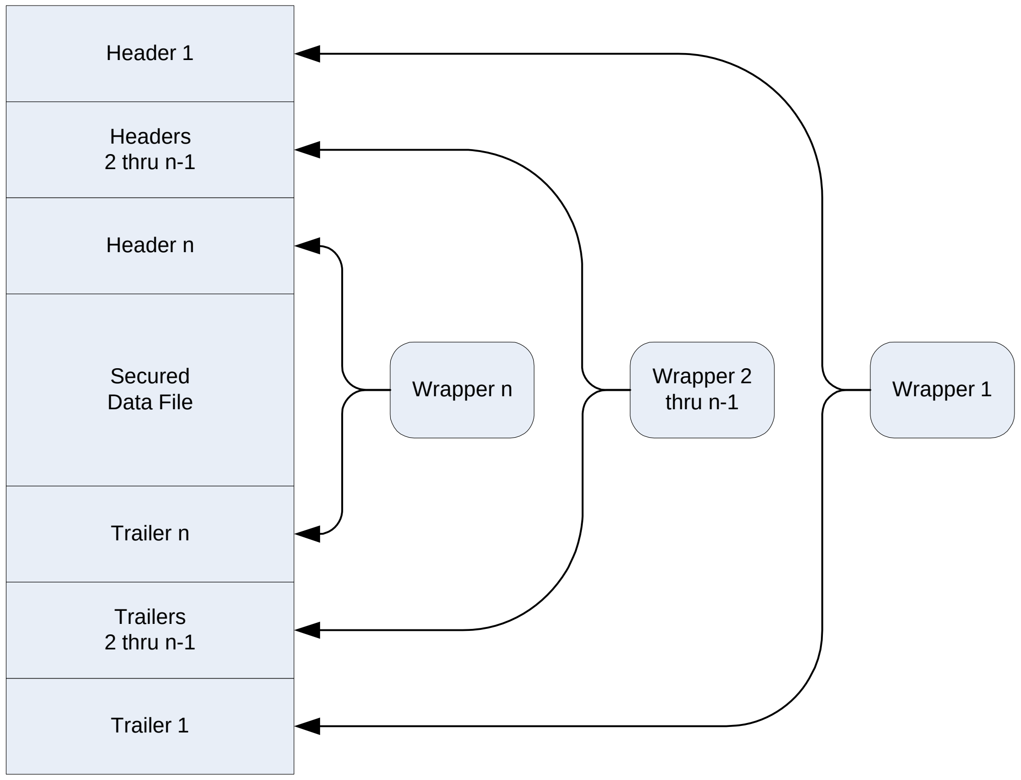


Figure 50 - Secure File Format object layout[49]

The final functionality that Secure File Format offers is a defined mechanism for grouping multiple objects together to form a single aggregated object. This single

[49] [34] Figure 1

aggregated object will carries one or more additional labels that provide the security and management information for the entire object. The information in this new label is either in addition to the information contained in the individual object labels or was calculated based on the information contained in the individual labels. An example, where additional information is applied, would be when this functionality is used to concatenate a directory structure into a single Secure File Format object, each file or object would have its own label with metadata (filename, owner, groups, size, creation date, etc) about that object. The "group" label would contain information that was common to all of the individual objects. This might include the directory path, and directory permissions.

The case when the information in the new "group" label is based on the information in the object labels is discussed in much greater detail and is one of the pillars of the material that I present. Beyond this single label, additional wrappers or tiers of labels can be added to provide for additional security.

Within each wrapper, Secure File Format defines a common format for all wrapper types. This format, depicted in Figure 51, contains 4 fields that are common to every Secure File Format wrapper and payload. The fields are the:

The Mandatory Header;
The Optional Header;
The Payload; and
The Trailer.

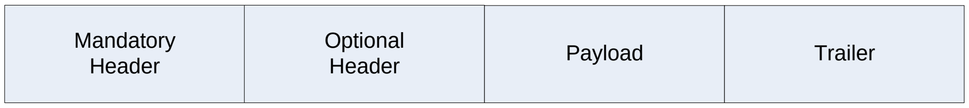


Figure 51- Format of a Secure File Format Wrapper[50]

The Mandatory Header is used to define the format of the rest of the header and has a structure that is identical in every type of wrapper. This header's function is to define the format wrapper type, the wrapper version, and the size of the three remaining fields.

The Optional Header follows immediately after the Mandatory Header. This header is used to carry information about the payload and should be considered informational, the external object label that is being applied the object carried in the payload. The actual format of this header varies based on what information or label, it is supporting. The order and placement of the fields in this header are defined by the "Type" field that is carried in the Mandatory Header.

The Payload follows immediately after the Optional Header; its starting location is determined by the "Hdr Size" field in the Mandatory Header. The Payload contains the object that is being secured or managed by the information contained in the Optional Header. With exception of the inner most wrapper (always a Type 01h wrapper), the Payload is another Secure File Format object. This is done to simplify the development of tools for constructing and processing of Secure File Format object and the information

[50] [34] Figure 2

contained therein. Processing of a Secure File Format object is executed by recursively "unwrapping" the file until the native file is exposed. The mandatory use of a Type 01h wrapper on the native file allows for all future processing to be performed on objects that follow the same wrapper structure.

The Trailer follows immediately after the Payload; its starting location is determined by adding the "Hdr Size" and the "Payload Size" fields in the Mandatory Header. The Trailer is responsible for providing any information required secure the rest of the header fields and Payload.

Strong Interaction between Wrappers

Secure File Format strongly attached wrappers (containing labels) to the underlying objects (carried in the Payload) through the use of secure hashes. The defined hashes identified for use by the Secure File Format include the Secure Hash Algorithms (SHA) [58]. Secure File Format allows for different length SHA hashes to be generated based on the need for security. The use of secure hashes for strongly attaching labels to objects is ideal for several reasons.

First, a hash value, called a 'message digest', is mathematically related to the object that has been hashed. Any changes to the object being hashed will produce a new message digest. Beyond this, the contents of the original files can't be derived the produced hash.

Second, the number of possible message digest values, called the space, is exceptionally large. For SHA, there are currently four message digest sizes that are defined. The smallest of these is 160 bits; the largest is 512 bits. This equates to a space of 2^{160} and 2^{512} possible values, respectively. Within a space this large, the probability that two objects will produce the same message digest is exceptional small. Hash Collision Analysis details a hash collision analysis for different number of unique objects against different message digest spaces. Hash Collision Analysis also conservatively estimates the number of possible objects that would need to be hashed in a very large network environment. This analysis supports the use of hashes as a means to bind labels to labeled objects. An additional benefit for using hashes to secure objects is that the message digests can be used as a unique object identifier.

In the rare cases when a hash collision occurs, altering the padding to generate a new message digest can be executed without affecting the payload or the validity of the message used to secure it.

Within the Optional Header fields, the message digest for the header of the next lower wrapper is stored. This message digest is identical to the message digest stored in the next lower wrapper and thus provides the strong linkage. For the innermost wrapper, the message digest is the message digest of the object being secured. Integrity for each wrapper is maintained by the message digest for that wrapper. This insures that message digests for the lower wrapper or the access labels can't be changed without being detected. It also insures that the object (lower wrapper) being protected remains unchanged as well. Should the lower wrapper change, then the linkage is broken because the new wrapper will have a different hash. These message digests would be carried as "informational" labels.[51] No access decisions would be based on informational labels or their contents, but they are an integral part of the label. Figure 52 shows how two Secure File Format tiers are strongly linked. Because each wrapper carries enough information to connect it to the object that it protects internally, it can be removed from the object without fear of breaking the linkage. It can also be "reattached" securely to the object in the future.

[51] Informational Labels are used to convey information as part of the object label.

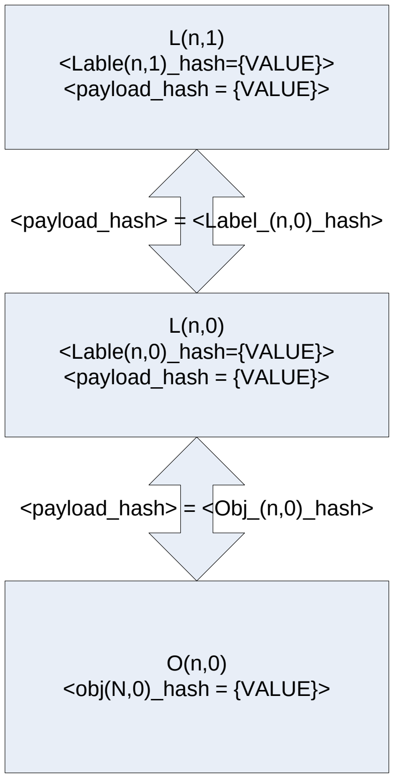


Figure 52- Secure File Format Wrapper Linkage

In my Secure File Format draft RFC, I include provisions for protecting multiple objects using under a single wrapper or security label. To do this, the hashes for each of the objects to be protected are included in the optional header. As in the case, where a single hash is carried in the optional header field, the chain of hashes provides linkage between the wrappers. Figure 53 shows how multiple objects can be protected in this matter.

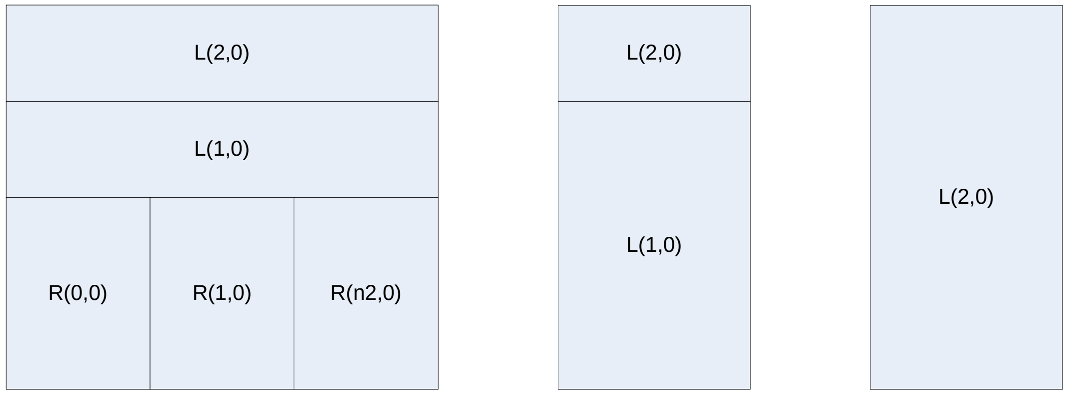


Figure 53- Secure File Formatted Object with Multiple Objects

An additional feature that this approach provides is the ability to separate objects without breaking the security of the entire object structure. Figure 54 shows how a Secure File Format protecting a single object from a collection would be presented. The overall labeled object (far left) can be decomposed into the L(0,2) security label and the L(0,1) header and secured object. At this point, the L(0,1) labeled object still contains all of the native objects and their header. After the next decomposition, only the desired object R (1, 0) is still wrappered by L(0, 1). For separation of labeled object, the higher order labels, L(0, 1) and higher, are not altered. This is required in order maintain the chain of hashes that are securing the objects. If the extra payload message digests were removed from the L(0, 1), then its hash would be changed and the chain of hashes would be broken. The removal of the unwanted objects does not break the chain of hashes at the higher level, L(0,2) because L(0,1) has not been altered.

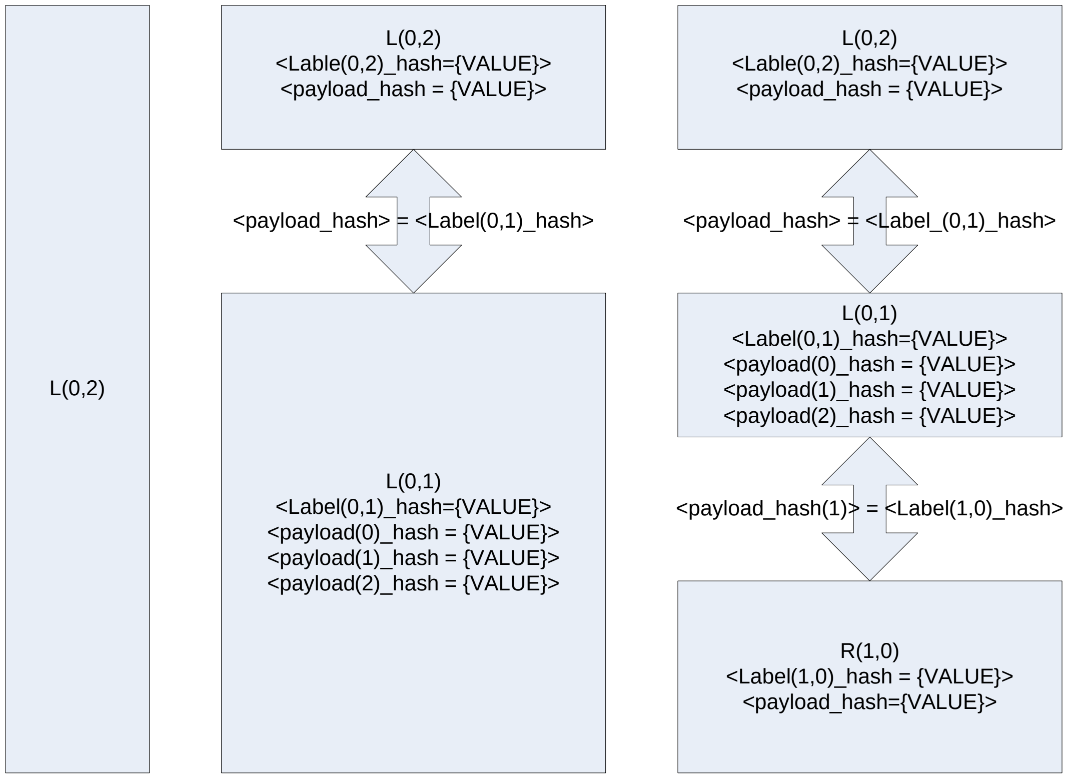

L(0,2)
L(0,2)
<Lable(0,2)_hash={VALUE}>
<payload_hash = {VALUE}>
<payload_hash> = <Label(0,1)_hash>
L(0,1)
<Label(0,1)_hash={VALUE}>
<payload(0)_hash = {VALUE}>
<payload(1)_hash = {VALUE}>
<payload(2)_hash = {VALUE}>
L(0,2)
<Lable(0,2)_hash={VALUE}>
<payload_hash = {VALUE}>
<payload_hash> = <Label_(0,1)_hash>
L(0,1)
<Label(0,1)_hash={VALUE}>
<payload(0)_hash = {VALUE}>
<payload(1)_hash = {VALUE}>
<payload(2)_hash = {VALUE}>
<payload_hash(1)> = <Label(1,0)_hash>
R(1,0)
<Label(1,0)_hash = {VALUE}>
<payload_hash={VALUE}>

Figure 54- Separation of a Protected Secure Files Format Object

Figure 55 provides an alternative representation of the interrelationships between the fields in different wrappers.

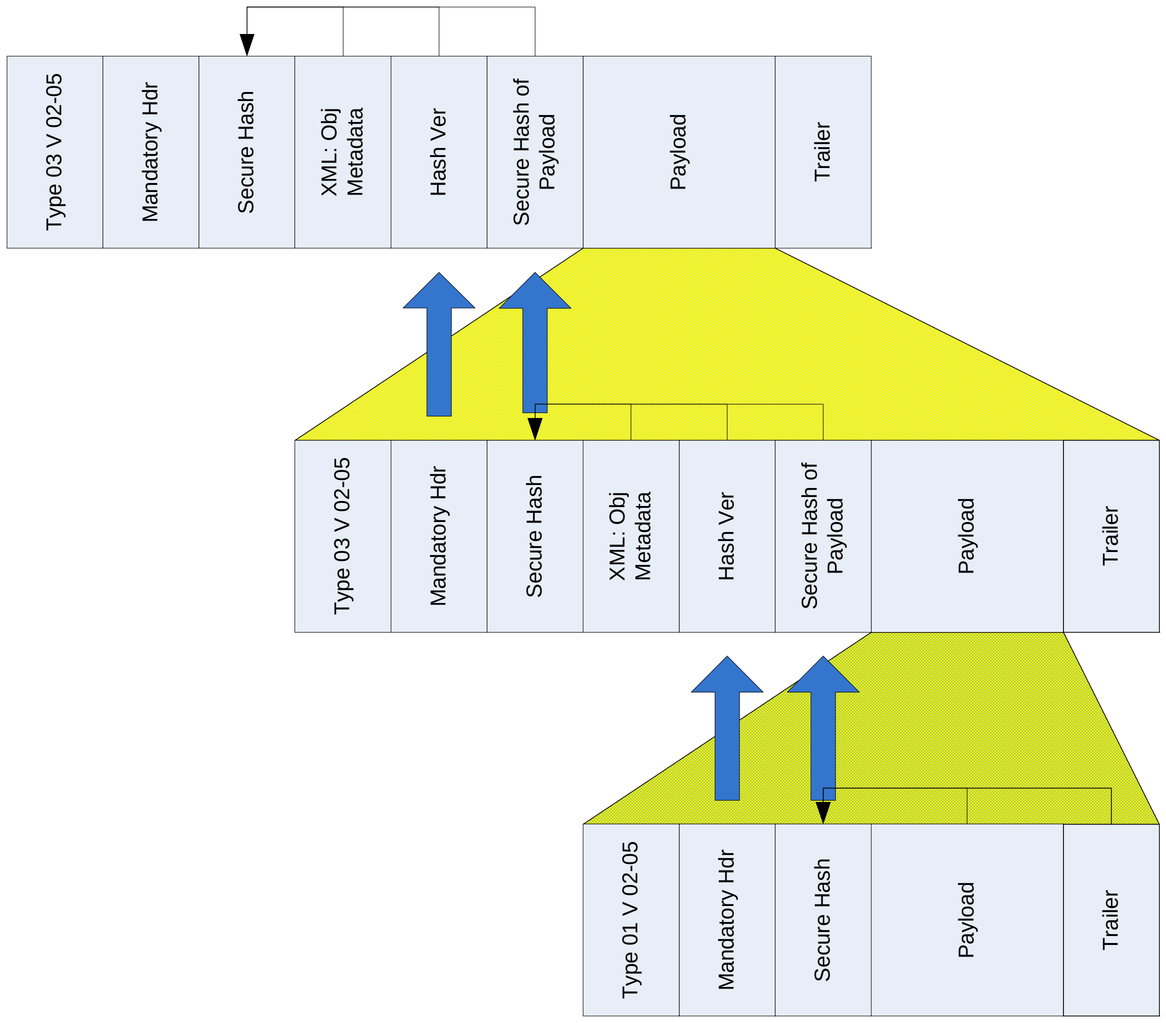


Figure 55 - Relationship between fields in different SFF Wrappers

Strong Bonding

Borrowing from the notations presented in "Modelling and Analysis of Security Protocols" [59], a Secure File Format wrapper can be modeled as:

$W_n = [MH_n \cdot OH_n \cdot PL_n \cdot T_n]$ *[B.1]*

Where W_n - the n^th^ wrapper in the Secure File Format Object
MH_n − Mandatory Header for Wrapper n,
OH_n − Optional Header for Wrapper n,
PL_n − Payload for Wrapper n (PL_1 is the object being secured or managed
T_n − Trailer for Wrapper n

OH_n can be further decomposed as;

$OH_n = [BH_n \cdot MD_{n-1} \cdot AH_n]$ *[B.2]*

Where BH_n – the portion of the Optional Header before the hash (message digest)
MD_{n-1} – the message digest of the Optional Header fields of the Payload (MD_1 is the message digest for the object being protected)
AH_n – the portion of the Optional Header after the hash (message digest)

In the fields before the message digest, the security and management information placed. This is done in the form of an XML object. Because of this, BH_n can be decomposed to:

$BH_n = [SHA_L([LBL_n \cdot AL_n \cdot MD_{n-1} \cdot AH_n]) \cdot LBL_n \cdot AL_n]$ *[B.3]*

Where $SHA_L(X)$ – indicates the Secure Hash Algorithm with message digest length L has been performed on X
LBL_n – the security or management data attached by that wrapper
AL_n – the remainder of the Optional Header between LBL_n and MD_{n-1}
MD_n is message digest for the most of the fields in the Optional Header of the Payload and can be written as:

$MD_n = SHA_L([LBL_n \cdot AL_n \cdot MD_{n-1} \cdot AH_n])$ *[B.4]*

The fields AL_n and AH_n are included in the message digest for the Optional Header. However, they don't factor into the use of external labels to secure the payload, but their inclusion allows for the integrity of their content to be assured.

In the cases, where the payload contains multiple wrappered objects, the MD_{n-1} field is replaced by a concatenation of all of the message digests from the wrappers in the payload.[52] This means that MD_{n-1} can be generalized without any loss of integrity to:

$$MD_{n-1} = \left[MD_{n-1,1} \cdot MD_{n-1,2} \cdot MD_{n-1,3} \cdot \ldots \cdot MD_{n-1,m}\right] \qquad [B.5]$$

Where MDn-1,i – the message digest of the ith wrappered object in the Payload that contains m wrappered objects in total.

Therefore MD_n can be written as:

$$MD_n = SHA_L\left(\left[LBL_n \cdot AL_n \cdot \left[MD_{n-1,1} \cdot MD_{n-1,2} \cdot MD_{n-1,3} \cdot \ldots \cdot MD_{n-1,m}\right] \cdot AH_n\right]\right) \qquad [B.6]$$

This equation shows that any change to either the security/management information (LBL_n) or the message digest(s) of the objects being protected ($MD_{n-1.i}$) will result in change to MD_n. Any change in the message digest would indicate that either the label or the message digests of the objects in the payload have been altered. Changes in a message digests of an object in the payload is an indication that either its labels or the message digest of a lower wrapper have been altered. This process is recursive and will continue until the message digests of the protected object as used.

A second property shown by this equation is a lack of dependence on the payload in calculating and maintaining the message digests for the payload. It is only when the wrapper is being applied to the object that the payload and wrapper must be treated

[52] The Secure File Format mandates that the first wrapper applied to an object must be a Type 1 wrapper. This mandate insures that a message digest unique to that object has been calculated.

collectively. Once the wrapper has been created, the payload can be separated from the wrapper without any loss of integrity or security.

APPENDIX C - HASH COLLISION ANALYSIS

Introduction

This appendix contains two investigations. The first deals with assessing the probability of a hash collision for a different size message digests and number of objects being hashed. The second investigates the number of objects that would need to being hashed based on the environment, where labeling is being used.

Probability of a Hash Collision Analysis

In order for cryptographic hashes to have any value as a means of associating security labels with objects, the probability that two objects or labels will generate the same message digest must be exceptional small. When two different objects produce the same message digest, a "hash collision" is said to have occurred. Should a hash collision occur, the system must be able to respond with a secondary association mechanism.

What follows is the analysis to determine the probability that two objects or labels will produce a collision. This analysis is a modification of the "General Birthday Problem." [60] Instead of the days in a year being the space of possible values, the possible message digests, S, are used. N denotes the number of objects that must be uniquely hashed in the hash space. $P_{hc}(N, S)$ is defined as the probability that a collision will occur for N objects when the message digest can be one of S possible values. This is represented as:

$$P_{hc}(N,S) = 1 - Q_{hc}(N,S) \qquad [C.1]$$

Where $Q_{hc}(N,S)$ – the probability that a collision will not occur

$$Q_{hc}(N,S) = 1 * ((S-1)/S) * ((S-2)/S) * ((S-3)/S) * \ldots * \left(\left(S-(N-1)\right)/S\right) \qquad [C.2]$$

This can be rewritten as:

$$Q_{hc}(N,S) = 1 * (1 - 1/S) * (1 - 2/S) * (1 - 2/S) * \ldots * (1 - (N-1)/S) \qquad [C.3]$$

Recalling that the Taylor Series expansion for e^x is:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots \qquad [C.4]$$

And can be simplified for small values of x to:

$$e^x \approx 1 + x \qquad [C.5]$$

Substituting [C.5] into [C.3],

$$Q_{hc}(N,S) = 1 * e^{-1/S} * e^{-2/S} * e^{-3/S} * \ldots e^{-(N-1)/S} \qquad [C.6]$$

This can be simplified to:

$$Q_{hc}(N,S) = e^{-N(N-1)/2*S} \qquad [C.7]$$

For large values of N, this can be simplified to:

$$Q_{hc}(N,S) = e^{-N^2/2*S} \qquad [C.8]$$

Substituting back into [C.1]

$$P_{hc}(N,S) = 1 - e^{-N^2/2*S} \qquad [C.9]$$

Table 12 shows the value for $P_{hc}(N,S)$ for different values of N and S. The values for S were chosen to correspond to the hash space associated with different versions of the Secure Hash Algorithm (SHA) [58].

Table 12 - Probability of Hash Collision

P(n,s) as a function of n and s				
	Space (s)			
	SHA-1	SHA-256	SHA-384	SHA-512
	2^{160}	2^{256}	2^{384}	2^{512}
Number of Entries (n)	1.462E+48	1.158E+77	3.94E+115	1.34E+154
1E+18	3.421E-13	0	0	0
1E+21	3.421E-07	0	0	0
1E+24	0.2897327	0	0	0
1E+33	1	4.318E-12	0	0
1E+36	1	4.318E-06	0	0
1E+39	1	0.9866746	0	0
1E+51	1	1	1.266E-14	0
1E+54	1	1	1.269E-08	0
1E+57	1	1	0.0126095	0
1E+72	1	1	1	3.729E-11
1E+75	1	1	1	3.729E-05
1E+78	1	1	1	1

Analysis of the Number of Labeled Objects that can be supported

A quick review of Table 12 shows that using available hashing solutions that large numbers of labels can generated with a very low probability of a hash collision. However, additional analysis is required to determine how large an environment can be supported for a given message digest size.

Equation [C.9] can be rewritten as:

$1 - P_{hc}(N,S) = e^{-N^2/2*S}$ [C.10]

Using the Taylor Series Approximation from [C.5]

$e^{-Phc(N,S)} = e^{-N^2/2*S}$ [C.11]

Taking the natural Log of [C.11]

$- P_{hc}(N,S) = -N^2/2 * S$ [C.12]

Solving [C.12] for N yields:

$N = \sqrt{2 * P_{hc}(N,S) * S}$ [C.13]

It is now possible to determine N, given that the size of the message digest and the acceptable probability of a hash collision are known.

Solving [C.12] for S yields:

$S = N^2/(2 * P_{hc}(N,S))$ [C.14]

It is now possible to determine how many objects can be supported for a message digest of a given size, given number of the labels per object and an acceptable probability that a hash collision are known.

At this point in order to prove the viability of using message digest to associate labels with labeled objects, I made several assumptions about the network environment where message digests would be used.

The first assumption was that this system would be used for an “Internet” sized network. This is a worst case assumption, as the current Internet is the largest networking environment. Currently, the number of Internet addressable hosts is bound by the IPv4 32-bit address fields in the IP header.[61] Under the emerging IPv6 standard [62], address fields will be 128-bits in size. For my analysis, I set the number of hosts in the network at 2^{128} hosts.

The second assumption was that each object in the system, no more than 2^{32} labels would be required. This value takes into account not only the labeling of objects, but also the use of labels as each user’s request traverses the network. This large value should reduce the risk of “replay” style attacks.

The third assumption was an acceptable level of risk of a hash collision. I chose a value of 1 in 2^{40} as an acceptable level. In English, this equates to “the probability that the message digests generated by two random objects is less than one is a trillion (10^{12}).

Equation [C.14] show the relationship between the number of host, the number of labels per object and the number of labeled objects per host

$$N = N_{hosts} * N_{obj/host} * N_{lbl/obj} \qquad [C.15]$$

Substituting [C.15] into [C.13] and solving for $N_{obj/host}$ yields

$$N_{obj/host} = \frac{\sqrt{2 * P_{hc}(N,S) * S}}{N_{hosts} * N_{lbl/obj}} \qquad [C.16]$$

Applying values for $P_{hc}(N,S)$, S, N_{hosts} , and $N_{lbl/obj}$, yields

$$N_{obj/host} = \frac{\sqrt{2 * 2^{12} * 2^{384}}}{2^{128} * 2^{32}} = \sqrt{2^{77}} \approx 3.89 * 10^{11}$$ objects per host

Thus with a message digest size of 384 bits, it is possible to use message digests to associate objects labels with objects in an Internet size network with an acceptable small probability of a hash collision.

APPENDIX D - TYPES OF LABELS SUPPORTED

Introduction

As previously stated, my dissertation is not trying to define a “one size fits all” security solution based on object labels. But rather, it presents the tools and concepts that can be used to build label based frameworks that are tailored to the needs of the community.

For both object and user/system labels, there is a common format for labels that can be bound to an object. The syntax for each element in an object or user/system labels[53] is:

```
<Security_Label>
    <Security_ID>{Security_ID}</Security_ID>
    <Label>
        <Name>{Name}</Name>
        <Type>{Type}</Type>
        <Value>{Value}</Value>
    </Label>
    …
    <Label>
        <Name>{Name}</Name>
        <Type>{Type}</Type>
        <Value>{Value}</Value>
    </Label>
</Security_Label>
```

Figure 56 - XML representation of a Security Label

[53] The term “Security” will be used to represent either “Object” or “User/System”

The Security_Label is composed of a Security_ID tag and one or more <Labels>

The <Security_ID>{*Security_ID*}</Security_ID> tag defines the name of the Security Label. It should contain the message digest of the object to which it is attached. However, this is not a mandatory requirement as the strong bonding between the Security_Label and the secured object is insured with cryptographic hashes.

The <Label>{*Label Information*}</Label> tags contain the information that will be used by the Reference Monitor to determine if access is Granted. The <Label> tag defines the name of the label using the <Name>{*Name*}</Name> tag, the type of tag that the label returns using the <Type>{*Type*}</Type> tag, and the value assigned by the tag using a combination of <Value>{*Value*}</Value> tag.

The <Type>{*Type*}</Type> tag defines what type of label is being represented. Table 13 details the four values that are defined for {*Type*} in my dissertation. This list is not exhaustive and should be expanded over time, if the use of external labels becomes a common practice in the security realm.

Table 13 - Possible Values for {Type}

{Type} Value	***Type of Label***
HIER	Defines a Hierarchal Label
CATE	Defines a Category Label
COND	Defines an Conditional Label
INFO	Defines a Informational Label

Each of these type values will be discussed in detail in the following paragraphs.

The <Value>{*Value*}</Value> tag defines the value for the label. The nature of this field is determined by the {*Type*} field value and will be discussed in more detailed for each defined {*Type*} value.

Types of Labels

NIST's FIPS 188 [56] (and others) defines labels as being either "hierarchical" or "categorical". My dissertation builds on this work and adds additional labeling types. These additional labeling types are "conditional" and "informational". When attached to either a native or labeled object[54], a label is used to convey some information about the object. A key feature about labeling an object is that the information conveyed by the label is external to the object and that the use of the label does not require that the object is be accessed as part of using the label. The ability to work with labels independently of the object is a key point and required to address the requirement that the object and label can be separated without any loss of security. However, it does impose requirements on labeling. These requirements include:

> That the label can be treated as an object;
> That there is a strong linkage between the label and its object;
> That this linkage is not altered by separating the label from its object; and

[54] A Secure File Format Type 1 object is considered a native object. Any other Secure File Format Type object is considered a labeled object.

That neither the label nor the object can be altered without destroying this linkage.

Each of the different types of labels is further detailed in the following sections.

Hierarchical Labels

These labels are used to define a quality or characteristic of the object being labeled that hierarchical in nature. The information carried by a hierarchical label can be either discrete or linear. In both case, the general format of a hierarchical label is:

```
<Label>
    <Name>{Name}</Name>
    <Type>HIER</Type>
    <Value>{Value}</Value>
</Label>
```

Figure 57 - XML Representation of a Hierarchical Label

The <Name> field carries the name of the label and is used to link the label value with the access rules against which it will be evaluated.

For a hierarchical label, the <Type> field is assigned a value of “HIER”.

These <Value> field will contain either a discrete or linear value.

For discrete values, there is a defined order that determines the precedence for evaluating the label value. Discrete values can be either numeric or alphanumeric. In the case of numeric values, the precedence is automatically established; a higher value has higher precedence. In the case of alphanumeric values, the order of precedence must be defined and understood by all parties. An example of this is the U. S. Government classification system enacted by Executive Order 12958[21]. Under this system, there are four levels of classification which form a linear lattice. This lattice is shown in Figure 58.

$$\begin{tikzcd}
\text{TOP SECRET} \\
\text{SECRET} \arrow[u] \\
\text{CONFIDENTIAL} \arrow[u] \\
\text{UNCLASSIFIED} \arrow[u]
\end{tikzcd}$$

Figure 58- Classification Precedence

Figure 58 shows that TOP SECRET has a higher precedence than SECRET, SECRET has a higher precedence than CONFIDENTIAL, and CONFIDENTIAL has a higher precedence than UNCLASSIFIED. Without a common understanding, the order of these words does not implicitly carry the underlying precedence.

In a FIPS 188 label environment, the hierarchical labels are represented by an integer value between 0 and 255. For a FIPS 188 label, the precedence is implied. 0 is lower than 1, 1 is lower than 2, up through 254 which is lower than 255. It is a trivial matter to implement the U. S. Government classification systems using FIPS 188 labels. All that is required is for the classification levels to be mapped to an integer value, such that the value used to represent ‘TOP SECRET’ is greater than the value used to represent ‘SECRET’. The ‘SECRET’ value must be greater than the ‘CONFIDENTIAL’ value and the ‘CONFIDENTIAL’ values must be greater than the ‘UNCLASSIFIED’ value. Table 14 shows one possible mapping that supports the U. S. Government classification system using FIPS 188 formatted labels.

Table 14 - Possible mapping of U. S. Gov’t Classifications to FIPS 188 values.

Classification Level	***FIPS 188 Value***
TOP SECRET	128
SECRET	96
CONFIDENTIAL	64
UNCLASSIFIED	32

For DoD classification system (a discrete alphanumeric label), the label can be written as:

<Value>{TOP SECRET | SECRET | CONFIDENTIAL |UNCLASSIFIED}</Value>

For FIPS 188 labels (a discrete numeric label), the label can be written as:

<Value>{I is an integer and 0<= I <= 255}</Value>

For linear values, only properly formatted numeric values are allowed. This automatically establishes the order of precedence, larger numeric values have higher precedence. Linear values are when the information being carried by the label is not a discrete value. An example of linear or non-discrete information that could be carried by this type of security label is time. In the Unix/Linux/Posix operating systems, time is measured as the number of seconds since the EPOCH (00:00:00 UTC, January 1, 1970)[55].

For a liner label, the general form is:

<Value>{R is a real number and $R_{\text{Lowest Possible Value}}$ <= R <= $R_{\text{Largest Possible Value}}$}</Value>

Categorical Labels

These labels are used to define a quality or characteristic of the object being labeled that is not hierarchical in nature. The information carried by a category label does not include the concept of precedence; all label values are of equal ranking. The general form of a category label is:

```
<Label>
    <Name>{Name}</Name>
    <Type>CATE</Type>
    <Value>{Value_1}</Value>
    …
    <Value>{Value_N}</Value>
</Label>
```

Figure 59 - XML Representation of a Category Label with Multiple Values

An example of categorical labels is a list of possible payment options for a retailer. The retailer might accept cash, credit cards or debit cards as payment for goods. Possible

[55] [63] Pg 484

payment options (categories) would include cash, credit cards, checks, and debit cards. For a customer payment, a single value for the payment label would be applied to the payment object. While for the vendor, the acceptable payment type label might have multiple values.

For the acceptable payment type, the value of label would be represented as:

```
<Label>
    <Name>Acceptable_Payment</Name>
    <Type>CATE</Type>
    <Value>Cash</Value>
    <Value>Credit_Card</Value>
    <Value>Debit_Card</Value>
</Label>
```

Figure 60 - XML Representation of a Category Label with Multiple Values

When considering category labels, there are two ways to assess how the labels will be processed. The first is where possessing any of the values of C_i is consider sufficient and the second where all values of C_i must be met. This will be discussed in detail under the labeling algebra for categorical labels.

Conditional Labels

A third type of label, not identified by FIPS 188, is the conditional label. Conditional labels differ from hierarchical and categorical labels in that their value is static or pre-determined, but rather is calculated at the time that the label is processed. The resulting label can be either a hierarchical or categorical. Determination of the label's value is based either on other fields within the label as well as external attributes that are provided to the Reference Monitor. Because the label values used for determining access are controlled by these external attributes, safeguards must be in place to insure that they are not compromised.

```
<Label>
    <Name>{Name}</Name>
    <Type>COND</Type>
    <Result>{Type}</Result>
    <Case>
            <Condition>DEFAULT</Condition>
            <Value>{Default Value}</Value>
    </Case>
    <Case>
            <Condition>{Condition_1}</Condition>
            <Value>{Value_1}</Value>
    </Case>
    …
    <Case>
            <Condition>{Condition_N} </Condition>
            <Value>{Value_N}</Value>
    </Case>
</Label>
```

Figure 61 - XML Representation of Conditional Label

The <Result>{*Type*}</Result> identifies the format of the value that is returned. Possible values for the result type are either "HIER" or "CATE" For a conditional label, there is a default label value and one or more conditional values. Any information that is required to determine which conditions has been satisfied must be available to the Reference Monitor at time when access is being requested.

The <Case>…</Case> is used to designate a single condition and the result if that condition is satisfied. The first case statement is always the "DEFAULT" condition. This case is always invoked, and if any of the subsequent cases is application then the "DEFAULT" value is replaced. Figure 62 and Figure 63 depicts the Conditional Label formally and as a traditional logic statement.

$$\neg\Big(\big((Cond_1 \Leftrightarrow Value_1) \Rightarrow \neg(Cond_2 \Leftrightarrow Value_2)\big) \Rightarrow \neg(Cond_3 \Leftrightarrow Value_3)\Big) \Leftrightarrow Value_{Default}$$

Figure 62 - Formal Model of a Conditional Label

$$Label\_Value = \begin{cases} Default \rightarrow Value_{Default} \\ Condition_1 \rightarrow Value_1 \\ Condition_2 \rightarrow Value_2 \\ \quad\quad\quad \dots \\ Condition_N \rightarrow Value_N \end{cases}$$

Figure 63 - Traditional Representation of a Conditional Label

The <Condition>{*Condition*}</Condition> tag is used to define the condition that must be satisfied for the <Value>{*Value*}</Value to be taken on by {*Name*}. The general format for the {*Condition*} statement is shown in Figure 64.

$$Operator(Variable_1, Variable_2, \ldots, Variable_n)$$

Figure 64- Genrel Format of a Conditional tag

The possible *Operator* values are listed in Table 15.

Table 15 - Possible Conditional Operators

Operation	***Language***	***Syntax***
(EQ)	Equals	$(Variable_1 = \mathrm{Variable}_2) \leftrightarrow TRUE$
(GT)	Greater Than	$(Variable_1 > \mathrm{Variable}_2) \leftrightarrow \mathrm{TRUE}$
(GE)	Greater Than or Equal	$(Variable_1 \geq \mathrm{Variable}_2) \leftrightarrow \mathrm{TRUE}$
(LT)	Less Than	$(Variable_1 < \mathrm{Variable}_2) \leftrightarrow \mathrm{TRUE}$
(LE)	Less Than or Equal	$(Variable_1 \leq \mathrm{Variable}_2) \leftrightarrow \mathrm{TRUE}$
(NE)	Not Equal	$(Variable_1 \neq \mathrm{Variable}_2) \leftrightarrow \mathrm{TRUE}$

An example of conditional labeling is the automatic declassification of a document after a given date. The default value would be the higher classification and the conditional value would be the lower classification with a date value after which this lower classification value would be applicable. This would be denoted as:

$$\text{Classification} = \begin{cases} \text{Default} \Rightarrow \text{Higher Classification Value} \\ \text{Time} >= DATE \Rightarrow \text{Lower Classification Value} \end{cases}$$

The label would be represented as:

```
<Label>
    <Name>CLASSIFICATION</Name>
    <Type>COND</Type>
    <Result>HIER</Result>
    <Case>
        <Condition>DEFAULT</Condition>
        <Value>SECRET</Value>
    </Case>
    <Case>
        <Condition>(GT)(${DATE_TIME},"Future_Date)")</Condition>
        <Value>UNCLASSIFIED</Value>
    </Case>
</Label>
```

Figure 65 - XML Representation of a Conditional Label that declassifies its object after a given date

For this example, any attempts to access this object prior to the *Future_Date* would require the requester to have a SECRET clearance. Any request make after the *Future_Date* would not be subject to this restriction. The ${DATE_TIME} value would be provide to the Reference Monitor or Network Proxy when access to the object is being requested from a Trusted Server.

Informational Labels

```
<Label>
    <Name>INFO</Name>
    <Type>INFO</Type>
    <Security_Tag>
        <Name>INFO</Name>
        <Value>INFO</Value>
    </Security_Tag>
    <Security_Tag>
        <Name>INFO</Name>
        <Value>INFO</Value>
    </Security_Tag>
    <Value>{Non-access related information}</Value>
</Label>
```

Figure 66 - XML Representation of an Informational Label

The final type of labels that can be carried by object labels is informational labels. As the name implies these labels carry static information about the object. This information can be divided into two categories; Security and General. Securing information is attributes about the object that are available to the Access Processor for evaluating conditional labels. They are stored in the <Value>{Value}/<Value> tag in an XML format. Figure 66 shows how <Security_Tag> tag. An information label can contain any number of <Security_Tag> tags. When these values are being used, the can be accessed by name. The naming convention is "INFO:{*Security Tab* Name. These labels can also carry general information that will be used by search engines and external applications. General Information that would typically be conveyed by this type of label would include searchable keywords and tags, audit logs and history, the filename of the native object, the creation/ modification date of the native object, the owner and group memberships of the native object, and other metadata about the native object. These labels would most often be used to load the native object into a local file system or database. Additionally, this information could be used by navigational systems, such as web servers or search engines. Having this information externally accessible increases the security of the protected objects, as the objects don’t need to be exposed before the access determination is made. This will be discussed further later in my dissertation.

APPENDIX E - EXAMPLE PROBLEMS THAT CAN BE SOLVED WITH LABELED OBJECTS

Introduction

In this appendix, I present some examples of the use of object labels to address some security problems. The examples that will be presented are:

Simple Security Clearance Access;
Simple Aggregation; and
Automatic Declassification of a document;

For each of the examples, one or more sample labels for the objects, users, and systems along with the rules for evaluating the labels.

Simple Security Clearance Access

For simple security clearance access, access to the labeled object is based on the user and system's clearance labels when compared against the object's classification label. The rule for this type of access control is shown is Figure 67.

```
        <Access_Rules>
            <Test>
                <Testname>Simple_Access_Control</Testname>
                <Rule>
                    <Name>Classification</Name>
                    <Type>HIER</Type>
                    <Operator>(GE)</Operator>
                </Rule>
</Access_Rules>
```

Figure 67 – Sample Rule for Simple Access Control

There are three objects that are considered. The first object is classified as SECRET, the second object is classified as TOP SECRET, and the third object is UNCLASSIFIED. Figure 68, Figure 69, and Figure 70 show the XML label for each document, respectively.

```
        <Object_Label>
            <Object_ID>Document_001</Object_ID>
            <Label>
                <Name>Classification</Name>
                <Type>HIER</Type>
                <Value>SECRET</Value>
            </Label>
</Object_Label>
```

Figure 68- Sample Object Lables for Document_0001

```
        <Object_Label>
            <Object_ID>Document_002</Object_ID>
            <Label>
                <Name>Classification</Name>
                <Type>HIER</Type>
                <Value>TOP_SECRET</Value>
            </Label>
</Object_Label>
```

Figure 69- Sample Object Lables for Document_0002

```
        <Object_Label>
            <Object_ID>Document_003</Object_ID>
            <Label>
                <Name>Classification</Name>
                <Type>HIER</Type>
                <Value>UNCLASSIFIED</Value>
            </Label>
</Object_Label>
```

Figure 70- Sample Object Lables for Document_0003

There are three users that are considered. The first user has a TOP SECRET clearance, the second user has a SECRET clearance, and the third user is not cleared to have access to any classified information. Figure 71, Figure 72, and Figure 73 show the XML label for each user, respectively.

```
        <User_Label>
            <User_ID>USER_001</User_ID>
            <Label>
                <Name>Classification</Name>
                <Type>HEIR</Type>
                <Value>TOP_SECRET</Value>
            </Label>
</User_Label>
```

Figure 71 - Sample User Labels for User_0001

```
        <User_Label>
            <User_ID>USER_002</User_ID>
            <Label>
                <Name>Classification</Name>
                <Type>HEIR</Type>
                <Value>SECRET</Value>
            </Label>
</User_Label>
```

Figure 72 - Sample User Labels for User_0002

```
        <User_Label>
            <User_ID>USER_003</User_ID>
            <Label>
                <Name>Classification</Name>
                <Type>HEIR</Type>
                <Value>UNCLASSIFIED</Value>
            </Label>
</User_Label>
```

Figure 73 - Sample User Labels for User_0003

There are three users that are considered. The first user has a TOP SECRET clearance, the second user has a SECRET clearance, and the third user is not cleared to have access to any classified information. Figure 74 and Figure 75 show the XML label for each system, respectively.

```
        <System_Label>
            <User_ID>System_001</User_ID>
            <Label>
                <Name>Classification</Name>
                <Type>HEIR</Type>
                <Value>TOP_SECRET</Value>
            </Label>
</System_Label>
```

Figure 74 - Sample System Labels for System_0001

```
        <System_Label>
            <User_ID>System_002</User_ID>
            <Label>
                <Name>Classification</Name>
                <Type>HEIR</Type>
                <Value>UNCLASSIFED</Value>
            </Label>
</System_Label>
```

Figure 75 - Sample System Labels for System_0002

Table 16 - Access Results for Simple Acces Control

(GE)	***System_001 (TOP SECRET)***			***System_002 (UNCLASSIFIED)***		
	User_001 (TS)	***User_002 (S)***	***User_003 (U)***	***User_001 (TS)***	***User_002 (S)***	***User_003 (U)***
Doc_001 (S)	Access Granted	Access Granted	Access Denied	Access Denied	Access Denied	Access Denied
Doc_002 (TS)	Access Granted	Access Denied	Access Denied	Access Denied	Access Denied	Access Denied
Doc_003 (U)	Access Granted	Access Granted	Access Granted	Access Granted	Access Granted	Access Granted

Simple Aggregation

For Simple Aggregation, a new label is created that captures the labels of the objects that are being aggregated. In this example, three labels are being aggregated to form a new object. Figure 76 shows the rules for aggregation.

```
        <Aggregate>
            <Label>
                <Name>Classification</Name>
                <Type>HIER</Type>
                <Form> CONCAT </Form>
                <Condition>MAX</Condition>
            </Label>
            <Label>
                <Name>Category</Name>
                <Type>CATE</Type>
                <Form>CONCAT</Form>
                <Condition>AND</Condition>
            </Label>
            <Label>
                <Name>Company</Name>
                <Type>CATE</Type>
                <Form>CONCAT</Form>
                <Condition>OR</Condition>
            </Label>
</Aggregate>
```

Figure 76 – Sample Rule for Simple Aggregation

Figure 76 shows that there are three rules for aggregation, one for each of the label fields in the object label. The first rule deals with the aggregation of a hierarchical label. The <Condition> tag asserts that the value with the highest precedence or greatest value should be used to populate the resulting object label. The second and third rules deal with the aggregation of category labels. The second rule asserts that only those values that are common all of the object labels will be included in the resulting object label. The third rule asserts that any value from any of the object labels should be included in the resulting object label. Figure 77, Figure 78, and Figure 79 show the details the labels for the three objects being aggregated.

```
<Object_Label>
    <Object_ID>Object_001</Object_ID>
    <Label>
        <Name>Classification</Name>
        <Type>HIER</Type>
        <Value>CONFIDENTIAL</Value>
    </Label>
    <Label>
        <Name>Category</Name>
        <Type>CATE</Type>
        <Value>ALPHA</Value>
        <Value>BETA</Value>
        <Value>GAMMA</Value>
    </Label>
    <Label>
        <Name>Company</Name>
        <Type>CATE</Type>
        <Value>ABC</Value>
    </Label>
</Object_Label>
```

Figure 77- Sample Object Lables for Simple Access Control

```
<Object_Label>
    <Object_ID>Object_002</Object_ID>
    <Label>
        <Name>Classification</Name>
        <Type>HIER</Type>
        <Value>SECRET</Value>
    </Label>
    <Label>
        <Name>Category</Name>
        <Type>CATE</Type>
        <Value>ALPHA</Value>
        <Value>BETA</Value>
    </Label>
    <Label>
        <Name>Company</Name>
        <Type>CATE</Type>
        <Value>DEF</Value>
    </Label>
</Object_Label>
```

Figure 78- Sample Object Lables for Simple Access Control

```
<Object_Label>
    <Object_ID>Object_003</Object_ID>
    <Label>
        <Name>Classification</Name>
        <Type>HIER</Type>
        <Value>UNCLASSIFIED</Value>
    </Label>
    <Label>
        <Name>Category</Name>
        <Type>CATE</Type>
        <Value>BETA</Value>
        <Value>GAMMA</Value>
    </Label>
    <Label>
        <Name>Company</Name>
        <Type>CATE</Type>
        <Value>DEF</Value>
    </Label>
</Object_Label>
```

Figure 79- Sample Object Lables for Simple Access Control

There are three classification values, one from each of the original objects. Because “SECRET” has higher precedence that either “CONFIDENTIAL” or “UNCLASSIFIED”, it is captured by the resulting label. The category label will contain on “BETA” because it is the only value that is common to all three labels. The Company label will contain both “ABC”, and “DEF” because each value is found in one or more of the original labels.

Each of the label tags can be represented as a logic statement. For the Classification tag, the statement would be:

MAX(CONFIDENTIAL, SECRET, UNCLASSIFIED) = SECRET

For the Category tag, the statement would be:

$$\cap\ (\{ALPHA, BETA, GAMMA\}, \{ALPHA, BETA\}, \{BETA, GAMMA\}) = \ BETA$$

For the Company tag, the statement would be:

$$\cup\ (\{ABC\}, \{DEF\}, \{DEF\}) = \ ABC, DEF$$

Figure 80 show the labels for the resulting object.

```
<Object_Label>
    <Object_ID>Resulting_Object</Object_ID>
    <Label>
        <Name>Classification</Name>
        <Type>HIER</Type>
        <Value>SECRET</Value>
    </Label>
    <Label>
        <Name>Category</Name>
        <Type>CATE</Type>
        <Value>BETA</Value>
    </Label>
    <Label>
        <Name>Company</Name>
        <Type>CATE</Type>
        <Value>ABC</Value>
        <Value>DEF</Value>
    </Label>
</Object_Label>
```

Figure 80- Sample Object Lables for an Aggregated Object

Automatic Declassification of a Document

Several Executive Orders mandate that classified documents are declassified after a given period of time unless certain conditions applied. Under these conditions, some types of classified information can remain classified for extended periods of time. However for the vast majority of information, declassification will occur at either 10 or 25 years after the document was originally classified. Object labeling can provide for this automatic declassification. Using object labels, automatic declassification can be execute without additional relabeling.

For object label based labeling, the declassification rules are included in the object's label. Because these rules are included in the object's label, the access rules are simple and require that the classification (clearance) of the user/system is greater than or equal to that of the object at the time of access. Figure 81 shows what the access rules would look like.

```
<Access_Rules>
    <Test>

    <Testname>Object_Label_Based_Automatic_Declassification</Testname>
        <Rule>
            <Name>Classification</Name>
            <Type>HIER</Type>
            <Operator>(GE)</Operator>
        </Rule>
</Access_Rules>
```

Figure 81 – Sample Rule for Simple Access Control

Figure 82 shows the object label for an object that will be automatically declassified after 00:00 June 30 2015[56]. For Document_001, the default classification level is SECRET. After June 30, 2015, the default condition is not applicable and the classification is UNCLASSIFIED. One issue to remember is structuring the conditional statements is the need for external trusted attributes. In this case, the current date and time ${DATE_TIME} is provided by a trusted server.

If this server is unavailable or can't provide a trusted date and time, then the conditional statement can't be evaluated. For this example, the document would remain SECRET. It is usually desirable to deny a legitimate request for access, rather than grant an illegitimate request for access.

If the default condition for the document was UNCLASSIFIED with the condition that the document was SECRET before the June 30, 2015 declassification date, then not having a trusted date and time would have resulted in the default condition being realized before the declassification date and the document being prematurely released.

[56] The exact format for specify features like dates and time is not defined, but should be inline with what is otherwise available.

```
        <Object_Label>
            <Object_ID>Document_001</Object_ID>
            <Label>
                <Name>Classification</Name>
                <Type>COND</Type>
                <Result>HIER</Result>
                <Case>
                    <Condition>DEFAULT</Condition>
                    <Value>SECRET</Value>
                </Case>
                <Case>
                    <Condition>(GT)(${DATE_TIME},”20150630000”)</Condition>
                    <Value>UNCLASSIFIED</Value>
                </Case>
            </Label>
</Object_Label>
```

Figure 82- Sample Object Lables with a single Conditional Case

Figure 83 shows an object label that includes two stages of declassification. The default condition is SECRET. The document is reclassified as CONFIDENTIAL on August 15, 2016 and then later declassified on December 31, 2020.

The classification level of the document is set to SECRET. The conditional statements are then evaluated until one is satisfied. Once a condition is met, then the processing of conditions stop. For this example, the first conditional processed checks if the current time and date is after when the document should be completely declassified. If this is true, then the document’s label is set to UNCLASSIFIED and processing stops. If this is not true, then the second conditional is evaluated. If the current time and date are past when the document should be reclassified, then the document label is set to

CONFIDENTIAL. If neither of the conditional is satisfied, then the document's label remains as SECRET.

```
        <Object_Label>
            <Object_ID>Document_002</Object_ID>
            <Label>
                <Name>Classification</Name>
                <Type>COND</Type>
                <Result>HIER</Result>
                <Case>
                    <Condition>DEFAULT</Condition>
                    <Value>SECRET</Value>
                </Case>
                <Case>
                    <Condition>(GT)(${DATE_TIME},"202012310000")</Condition>
                    <Value>UNCLASSIFIED</Value>
                </Case>
                <Case>
                    <Condition>(GT)(${DATE_TIME},"201608150000")</Condition>
                    <Value>CONFIDENTIAL</Value>
                </Case>
            </Label>
</Object_Label>
```

Figure 83- Sample Object Lables with Multiple Conditional Cases

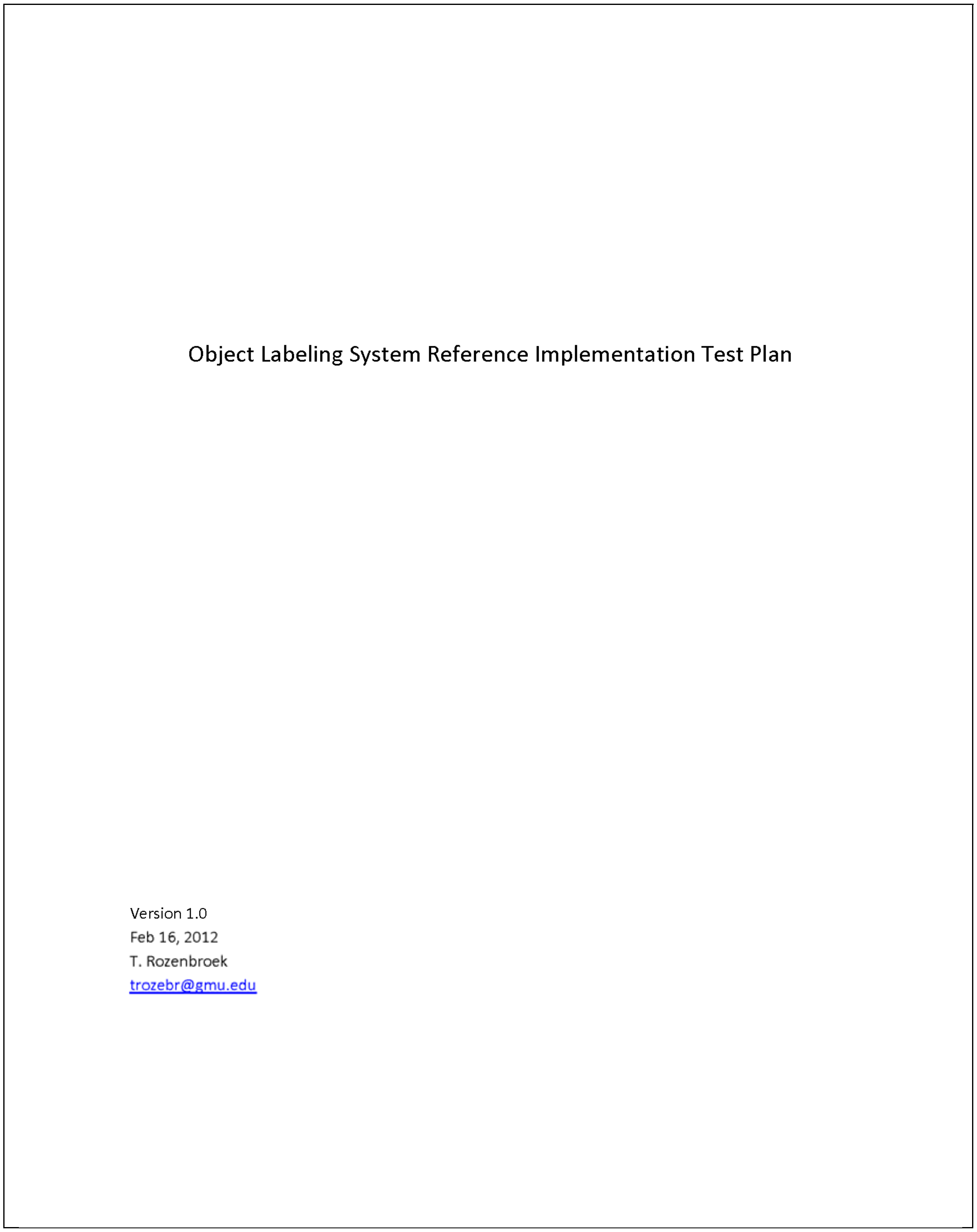
A key point to remember is that conditionals only apply to the Label of which they are a part. If a second <Label> tag were part of the object's label, it would be unaffected by any of the conditional cases in the Classification tag.

A similar set of labels can be applied to either user or system labels as well. In this way, the user or system would be granted temporary access to a higher classification of information. However, that access would automatically be removed after a given date or once another condition has been satisfied.

APPENDIX F - LABEL BASED ACCESS CONTROL DEMONSTRATION SYSTEM

This appendix contains a copy of the Object Labeling System Reference Implementation Test Plan. This test plan represents the testing that was performed on a developed system that was developed to insure that the object labeling functionality that was presented in this dissertation could be implemented in a cost effective manner.

In addition to documenting the reference implementation, there is a cost analysis. One of the key tenets of this research is that object labeling can be implemented so as to be cost effective. For the purposes of addressing this tenet, an equivalent server implementation was defined and priced out. The results that are documented in this appendix can be replicated on a server with a total cost of less than $2000.00



Object Labeling System Reference Implementation Test Plan

Version 1.0

Feb 16, 2012

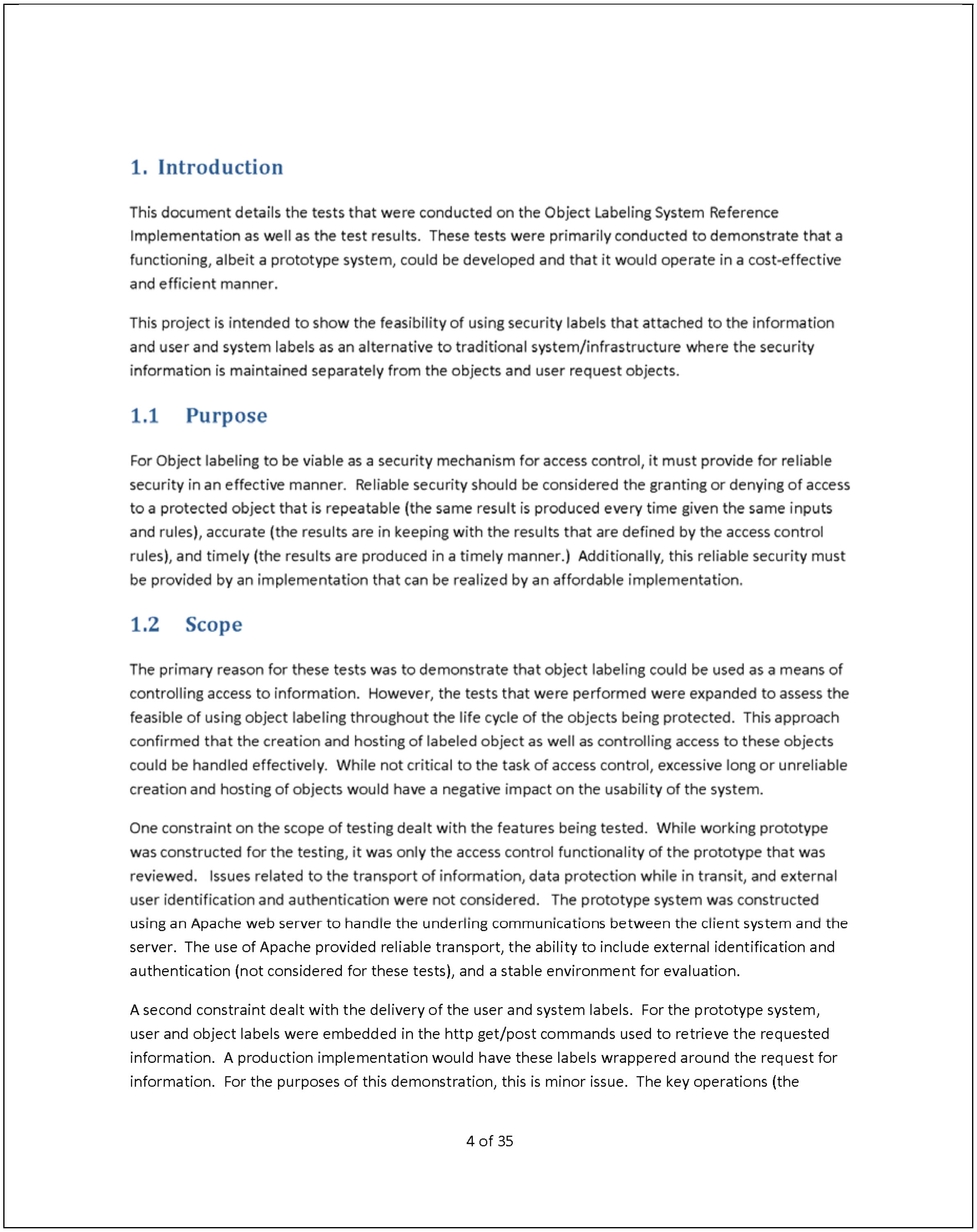
T. Rozenbroek

trozebr@gmu.edu

Table of Contents

Table of Tables

Table of Figures



1. Introduction

This document details the tests that were conducted on the Object Labeling System Reference Implementation as well as the test results. These tests were primarily conducted to demonstrate that a functioning, albeit a prototype system, could be developed and that it would operate in a cost-effective and efficient manner.

This project is intended to show the feasibility of using security labels that attached to the information and user and system labels as an alternative to traditional system/infrastructure where the security information is maintained separately from the objects and user request objects.

1.1 Purpose

For Object labeling to be viable as a security mechanism for access control, it must provide for reliable security in an effective manner. Reliable security should be considered the granting or denying of access to a protected object that is repeatable (the same result is produced every time given the same inputs and rules), accurate (the results are in keeping with the results that are defined by the access control rules), and timely (the results are produced in a timely manner.) Additionally, this reliable security must be provided by an implementation that can be realized by an affordable implementation.

1.2 Scope

The primary reason for these tests was to demonstrate that object labeling could be used as a means of controlling access to information. However, the tests that were performed were expanded to assess the feasible of using object labeling throughout the life cycle of the objects being protected. This approach confirmed that the creation and hosting of labeled object as well as controlling access to these objects could be handled effectively. While not critical to the task of access control, excessive long or unreliable creation and hosting of objects would have a negative impact on the usability of the system.

One constraint on the scope of testing dealt with the features being tested. While working prototype was constructed for the testing, it was only the access control functionality of the prototype that was reviewed. Issues related to the transport of information, data protection while in transit, and external user identification and authentication were not considered. The prototype system was constructed using an Apache web server to handle the underling communications between the client system and the server. The use of Apache provided reliable transport, the ability to include external identification and authentication (not considered for these tests), and a stable environment for evaluation.

A second constraint dealt with the delivery of the user and system labels. For the prototype system, user and object labels were embedded in the http get/post commands used to retrieve the requested information. A production implementation would have these labels wrappered around the request for information. For the purposes of this demonstration, this is minor issue. The key operations (the

4 of 35

parsing of the user and system labels, and the access control determination) are still being performed in the same manner as they would be performed in a production environment.

The final constraint dealt with the returning of the labeled object. For the performance testing, not labeled objects were returned to the client. The purpose of access control is to determine if the requesting user should be granted access to the object. Once the determination has been made, the process for returning the object is academic. Additionally, the size of the object will impact the number of access determination that are made, not because of the performance of the reference monitor, but of the size of the file being sent. Queries that return larger files will take longer to complete and will only affirm that it takes longer to return greater amounts of data. During the functional testing, the labeled objects were included in the response. For these tests, the focus was on affirming that the labeled object was being properly processed.

2. Test Environment

The Object Labeling System test environment consists of the labeling server, client workstation, test data, and test scenarios. Each of which will be discussed in the next few sections.

2.1 System Architecture

Figure 1 shows the hardware architecture for the test environment. All of the label processing is performed by the Labeling Server. All user interactions and test script control is performed by the Client Workstation. All communications between the Client Workstation and Labeling Server was done via either HTTP on TCP port 80 or SSH/SFTP on TCP port 22.

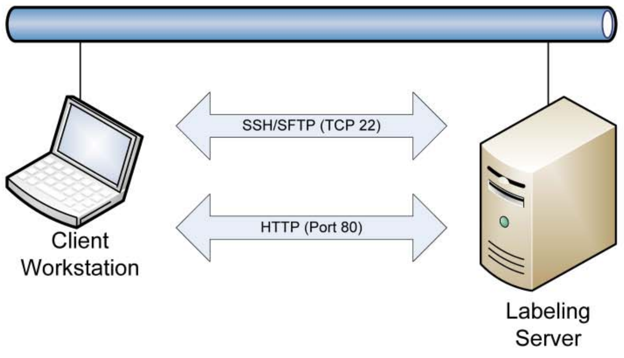


Figure 1 - Hardware Architecture

Figure 2 shows the internal architecture of the web server and its internal components. The Reference Implementation of the labeling system is coded in PHP and hosted internally using an Apache Web Server. As the focus of the testing was on the reference monitor functionality and not the communications between the server and client, the use of Apache is appropriate and provides a high quality thoroughly testing environment for evaluating the labeling system. For a production ready system, Apache could be either replaced by a smaller dedicated communications package or augmented to support protocols other than HTTP and HTTPS. In either case, the reference monitor software could be reused.

The Reference Monitor implementation and the PHP common libraries are coded using the PHP programming language. PHP is a scripting language that is commonly used to implement web services and building web pages. By using PHP, the reference monitor can easily interfaced with functionality to protect a wide range of information services.

The PHP Common Libraries offer functionality that is used by two or more components. By reusing common software libraries, the cost of maintaining functionality is reduced and the chances for implementation problems is reduced.

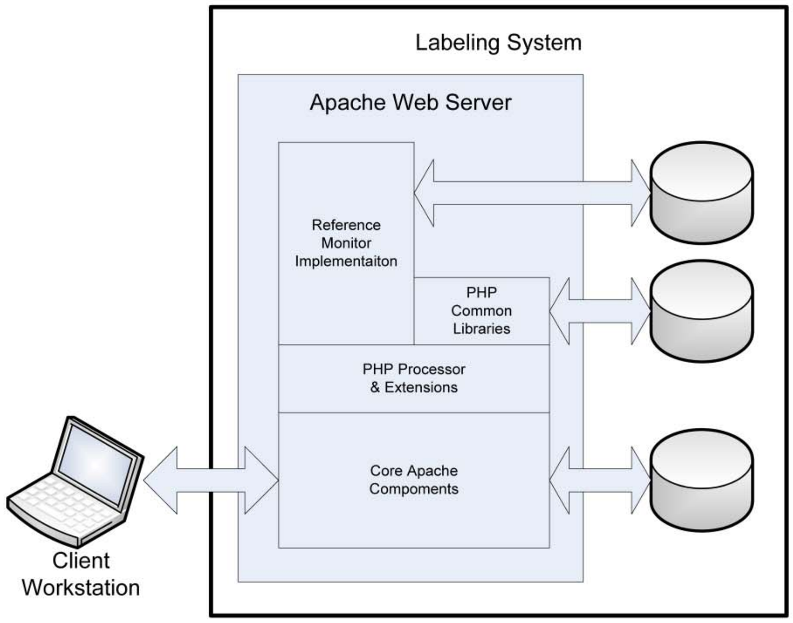


Figure 2- Server Architecture

Figure 3 shows the architecture for the interactive utilities that were tested. These utilities were used to create, manage, and aggregate both labeled and unlabeled objects. As with the Reference Monitor, the working utilities were written in PHP. The key value in writing both in PHP is re-usability and smaller code base to maintain and port to another language. For both the reference monitor and utilities, there is a single set of PHP common libraries. Additional functionality was provided by using shell scripts. Shell scripting was used for software development, application deployment, data installation, and when a PHP utility needed to be executed repeatedly.

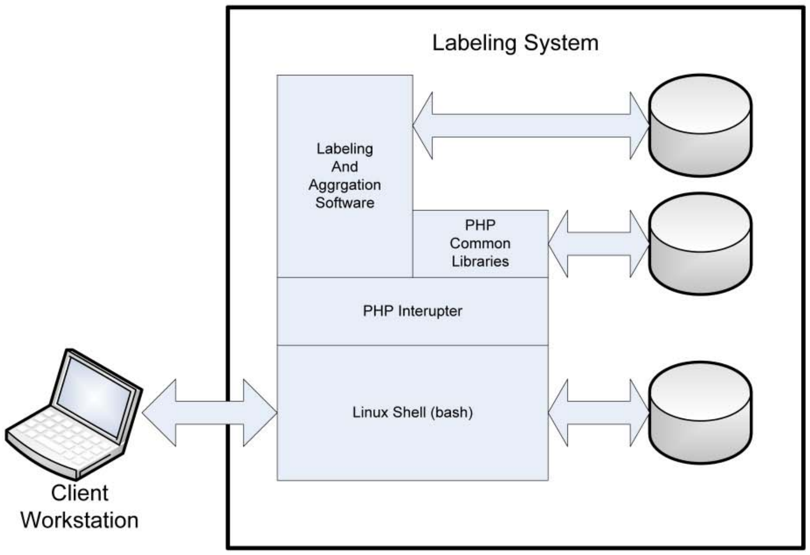


Figure 3 - Label Utility Architecture

2.2 Labeling Server

All of the labeling operations were performed on the Labeling Server. Table 1 lists the specification for the hardware supporting the server.

Table 1 – Server Hardware Specifications

Server Hardware	
Model No	Hewlett Packard TX2-1275dx
Processor	2.20 GHz AMD Turion X2 RM-74 Dual-Core Mobile Processor
RAM	8 GBytes
Disk Space	100 GByte Internal IDE Hard Drive (Partitioned to support multiple operating systems)
Network	10/100/1000 MBps Ethernet

The hardware used for the server is a mid-range notebook computer running the Ubuntu x64 server operating systems. In addition to the Ubuntu server operating system, the Ubuntu desktop was installed for testing. This desktop GUI was not used during the testing, but was installed for troubleshooting purposes. While the use of a notebook to collect performance metrics provided an assessment of the system performance, it needed to be compared against a working server to assess the affordability of the labeling system. For this comparison, an entry level server as identified that would provide performance that was greater than that of the hardware used during testing.

A Hewlett Packard Proliant DL320 G6 Server was identified as an entry-level server that offered performance that was greater than the notebook, but still roughly about the same. This determination was based on a specification review, where each of the key system components (CPU performance, amount of memory, disk speed and network speed) was compared. In every case, the Proliant's performance was greater than or equal to that of the notebook. Table 2 shows the side by side comparison of the two systems as well as the cost of the configured server.

Table 2 - System Comparison

	HP TX2-1275dx	HP Proliant DL320	
CPU (Passmark	1,104[1]	1,787[2]	
Memory	8 Gbytes	Up to 96 Gbytes	
Disk Type	IDE	SATA	
Network Interface Speed	10/100/1000 MBps	10/100/1000 MBps	
Price		Base System	$1395.00
		Add 4 GBytes Ram	$ 105.00
		300 Gbyte Disk	$ 439.00
		Total Cost	**$ 1939.00**

The second component of the labeling server is the software that is installed on the server. Table 3 lists all of the open source software that was installed on the labeling server. The initial server software was either downloaded directly from the Ubuntu web site (www.ubuntu.com) or installed using the "apt-get" utilities included with the operating system. The initial server installation included the installation of the Apache web server, the PHP language processor, and the user shells. The apt-get utility was used in install the additional PHP modules.

[1] http://www.cpubenchmark.net/cpu_lookup.php?cpu=AMD+Turion+X2+Dual-Core+Mobile+RM-74(add citiation after being loaded to thesis)
[2] http://www.cpubenchmark.net/cpu.php?cpu=Intel+Xeon+E5503+%40+2.00GHz

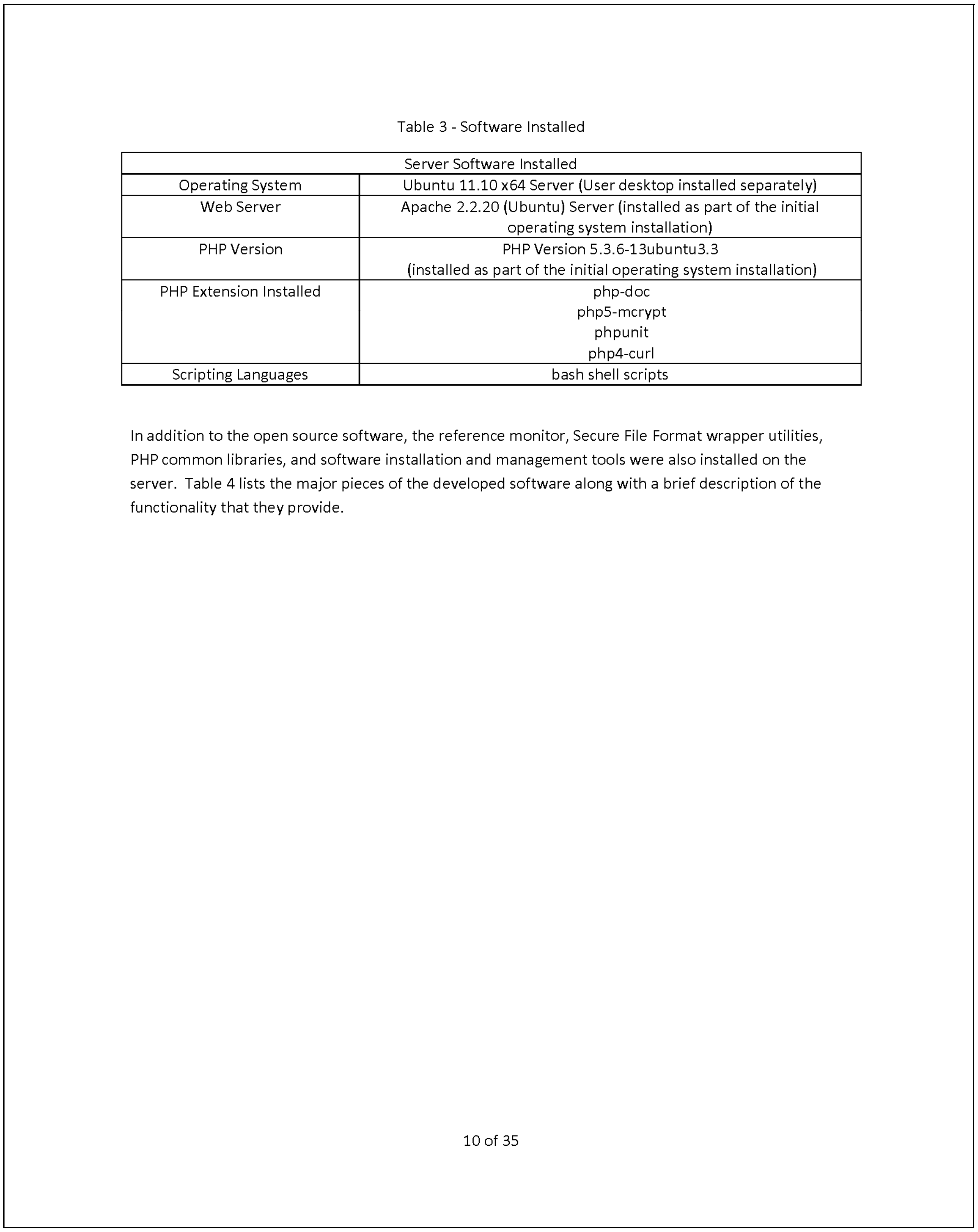


Table 3 - Software Installed

Server Software Installed	
Operating System	Ubuntu 11.10 x64 Server (User desktop installed separately)
Web Server	Apache 2.2.20 (Ubuntu) Server (installed as part of the initial operating system installation)
PHP Version	PHP Version 5.3.6-13ubuntu3.3 (installed as part of the initial operating system installation)
PHP Extension Installed	php-doc php5-mcrypt phpunit php4-curl
Scripting Languages	bash shell scripts

In addition to the open source software, the reference monitor, Secure File Format wrapper utilities, PHP common libraries, and software installation and management tools were also installed on the server. Table 4 lists the major pieces of the developed software along with a brief description of the functionality that they provide.

Table 4 - Developed Software

Program Name	Functionality provided
Index.php	Main website navigation tools. Dynamically builds a list of all SFF files in the web server's DATA directory
SFF_Bulk_Retriever.php	Main web server test program. SFF_Bulk_Retreiver.php when executed, processes every file in the web server's DATA directory 100 times. The output of this program is returned to the web client and contains time information as well as the object, user, and system labels and how they were processed. SFF_Bulk_Retriever generates the user labels based on the 9th and 10th characters in the object name and the system label based on the 11th and 12th characters in the object name. SFF_Bulk_Retriever.php is a wrapper for the SFF_Handler.php which does the actual assessment of the labels.
SFF_Retriever.php	Interactive test program. SFF_Retriever.php is initialed from the client workstation by clicking on the link returned by index.php. SFF_Retriever.php accepts as inputs the user label, the system label, and the name of the object being requested. These three values are embedded in the link displayed on the index page. SFF_Retriever.php is a wrapper for SFF_Handler.php, which does the actual assessment of the labels.
SFF_Handler.php	Core Reference Monitor program. SFF_Handler.php accepts the user and system labels from the invoking program, extracts the object label from the object file, compares the three labels and based on the rules defined in the RULES.XML file determines if the user/system should be granted access to the object.
SFF_Wrapper.php	Main wrappered object creation program. SFF_Wrapper.php accepts numerous input parameters, which it uses to create Secure File Formattted objects. SFF_Wrapper.php calls a second program SFF_Type_xxxx.php based on the type of wrapper being applied to the object.
SFF_Functions.php	Collection of standard PHP functions that are common to two or more SFF programs. SFF_Functions.php contains functions for parsing XML objects, and processing both hierarchy and category labels.

2.3 Client Workstation

The second component in the testing environment is the client workstation. Its primary function during testing is executing functionality on the labeling server. For the majority of the testing, the client workstation is used to "kick-off" the functionality and display the results returned by the server. For a small portion of the testing, the client workstation is actively involved with the execution of the testing. For the interactive testing, either a user at a keyboard or the Selenium test suite[3] will be used. Table 5 lists the specifications for the client workstation hardware.

[3] http://www.seleniumhq.org

Table 5 - Client Hardware Specifications

Client Hardware Specifications	
Model No	Hewlett Packard Pavilion dv7
Processor	2.10 Ghz AMD Pheonom II N830 Triple-Core Processor
RAM	4 Gbytes
Disk Space	150 GByte Internal Hard Drive (Partitioned to support Multiple Operating Systems) 700 GByte External Hard Drive connected via USB-2 interface
Network	10/100/1000 MBps Ethernet

The hardware for the client workstation is an off-the-shelf Hewlett Packard notebook computer. This system is running Windows 7 x64 Home Edition. Additionally, several software packages and tools have been installed on the workstation. Table 6 shows a partial list of the software that is installed on the client workstation. This table identifies only the software that was actively used for testing.

Table 6 - Client Software Installed

Client Software Installed	
Operating System	Windows 7 x64 Home Premium
Web Browser	Firefox V 9.0.1
Firefox Plug-ins	Selenium Test Suite (Ver 1.6.0)
SSH Terminal Software	PuTTY 0.61

2.4 Test Data

There are four types of test data that were used during the testing.

The first type of data was used to represent the data objects to be protected. This data, stored in files of differing length consisted of random values that would be labeled and later used in the access control and aggregation tests. Table 7 shows the breakout of files by size and number of each size file that were used during testing. The use of decimal based file size insured that each data object would require "padding" as part of the wrappering process.

Table 7 - Breakout of Test Data Sets

File Size	Number of Files	Total Size (Bytes)
1000 Bytes	400	400,000
2000 Bytes	400	800,000
5000 Bytes	400	2,000,000
10,000 Bytes	400	4,000,000
20,000 Bytes	400	8,000,000
50,000 Bytes	400	20,000,000
100,000 Bytes	400	40,000,000
200,000 Bytes	300	60,000,000
500,000 Bytes	200	100,000,000
1,000,000 Bytes	200	200,000,000
2,000,000 Bytes	200	400,000,000
5,000,000 Bytes	200	1,000,000,000
10,000,000 Bytes	200	2,000,000,000
Total Number of Files/Size	**4100**	**3,835,200,000**

The second type of data used was the object, user, and system labels. The object labels were used as part of the testing of the wrappering process. A total of 256 different object labels were used. The object labels that was attached to an object was determined by the first two bytes in the name of the Type 0001 wrappered object. The name of the Type 0001 wrappered object was the message digest for the data object (with padding) with a ".SFF" file extension. Since the distribution of the message digest values is random, this allows a good selection of object labels to be applied. Figure 4 shows a typical Object label that was attached to an object as part of a Type 0003 wrapper. This is also the same format that was generated during the aggregation of multiple objects.

```
<Object_Label>
    <Object_ID>Document_0b5</Object_ID>
    <Label>
         <Name>Classification</Name>
         <Type>HIER</Type>
        <Value>SECRET</Value>
     </Label>
     <Label>
         <Name>Integrity</Name>
         <Type>HIER</Type>
         <Value>CERTIFIED</Value>
     </Label>
     <Label>
         <Name>Group_001</Name>
         <Type>CATE</Type>
         <Value>A</Value>
     </Label>
     <Label>
         <Name>Group_002</Name>
         <Type>CATE</Type>
         <Value>C</Value>
     </Label>
</Object_Label>
```

Figure 4 – Sample Object Label

The user and system labels were not used in the creation of the labeled object, but were used during the access control tests. For testing purposes, these XML objects were included in the HTTP GET and HTTP POST commands that were used to represent a information request. Figure 5 shows a typical User Label. Like the Object Labels, a total of 256 User Labels were generated and used during testing. For the interactive testing, the actual user labels were embedded in the HTML link. For the performance testing, the user labels were selected based on the 9th and 10th characters in the filename.

```
<User_Label>
    <User_ID>Document_0c9</User_ID>
    <Label>
        <Name>Classification</Name>
        <Type>HIER</Type>
        <Value>TOP_SECRET</Value>
    </Label>
    <Label>
        <Name>Integrity</Name>
        <Type>HIER</Type>
        <Value>LOW</Value>
    </Label>
    <Label>
        <Name>Group_001</Name>
        <Type>CATE</Type>
        <Value>B</Value>
    </Label>
    <Label>
        <Name>Group_002</Name>
        <Type>CATE</Type>
        <Value>C</Value>
    </Label>
</User_Label>
```

Figure 5 – Sample User Label

Figure 6 shows a typical System Label. Like the Object and User Labels, a total of 256 System Labels were generated and used during testing. For the interactive testing, the actual system labels were embedded in the HTML link. For the performance testing, the System Labels were selected based on the 11th and 12th characters in the filename.

```
<System_Label>
    <System_ID>Document_063</System_ID>
    <Label>
        <Name>Classification</Name>
        <Type>HIER</Type>
        <Value>CONFIDENTIAL</Value>
    </Label>
    <Label>
        <Name>Integrity</Name>
        <Type>HIER</Type>
        <Value>HIGH</Value>
    </Label>
    <Label>
        <Name>Group_001</Name>
        <Type>CATE</Type>
        <Value></Value>
    </Label>
    <Label>
        <Name>Group_002</Name>
        <Type>CATE</Type>
        <Value>C,D</Value>
    </Label>
</System_Label>
```

Figure 6 – Sample System Label

The determination of access is controlled by rules. These rules are stored on the Labeling Server in a rules file. This file details which fields will be involved in making the access control decision and what operation will be performed on that field. Figure 7 shows a typical rules file. It is worth noting that every rule in the rules file must be satisfied for access to be granted, but not every label field in the object, user, and system labels need be involved with that decision.

```
<Access_Rules>
    <Test>
        <Testname>Simple_Access_Control</Testname>
        <Rule>
            <Name>Classification</Name>
            <Type>HIER</Type>
            <Operator>(GE)</Operator>
        </Rule>
            <Name>Integrity</Name>
            <Type>HIER</Type>
            <Operator>(LE)</Operator>
        </Rule>
        <Rule>
            <Name>Group_001</Name>
            <Type>CATE</Type>
            <Operator>(ALL)</Operator>
        </Rule>
        <Rule>
            <Name>Group_002</Name>
            <Type>CATE</Type>
            <Operator>(ANY)</Operator>
        </Rule>
    </Test>
</Access_Rules>
```

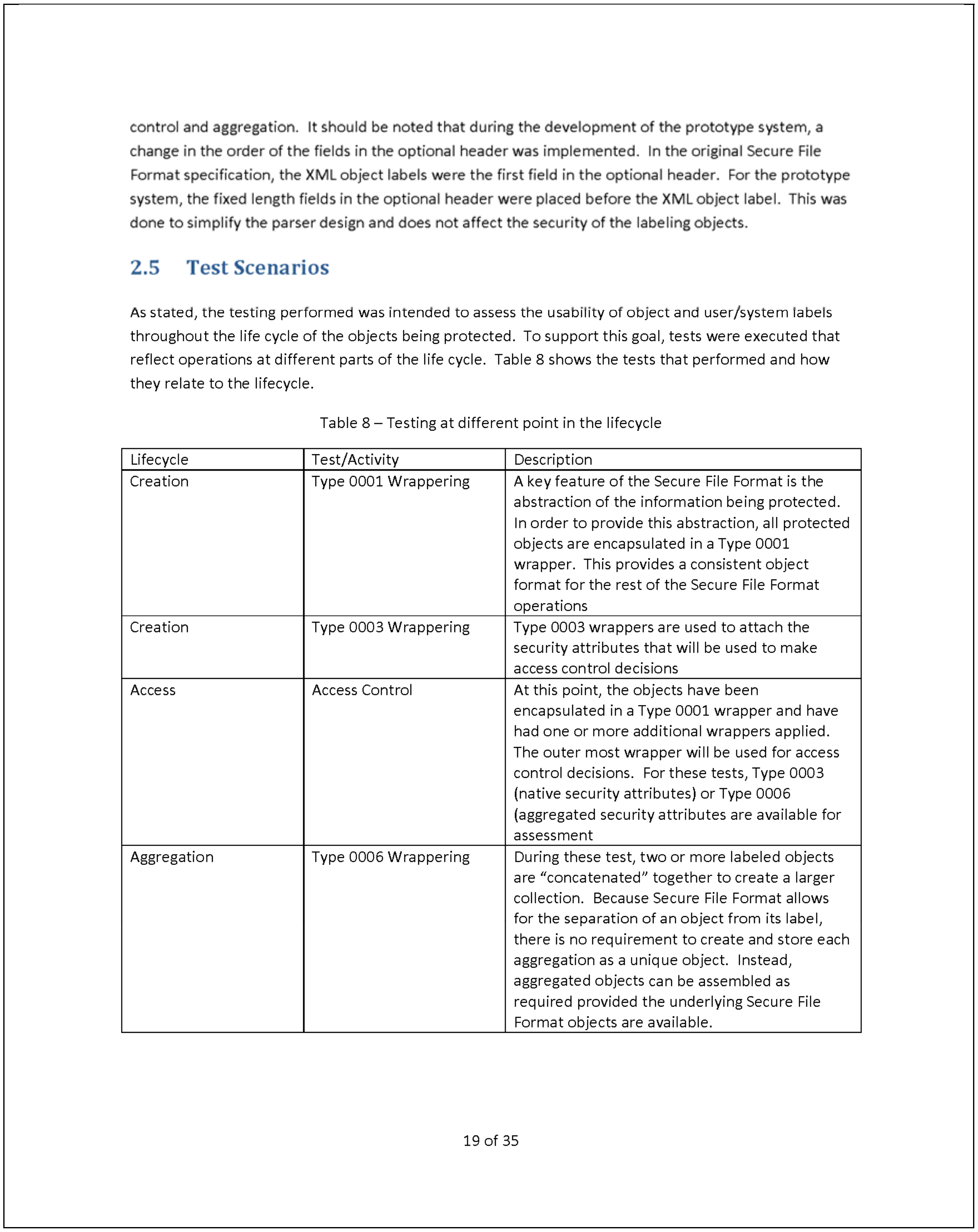
Figure 7 – Sample Set of Access Rules

The last type of data that was used for these tests was hierarchy mapping data. This data was used to translate human readable values, such as SECRET or HIGH, into a defined hierarchy. For each human readable value, a numerical value was assigned. It is these values that are actual assessed during the access control determination. It is worth nothing that value assignment hierarchy is only appropriate for each label. Actual values can be reused for different labels, but have no relationship between the labels.

```
<Mapping>
    <Map>
        <Name>UNCLASSIFIED</Name>
        <Value>1</Value>
    </Map>
    <Map>
        <Name>CONFIDENTIAL</Name>
        <Value>2</Value>
    </Map>
    <Map>
        <Name>SECRET</Name>
        <Value>3</Value>
    </Map>
    <Map>
        <Name>TOP_SECRET</Name>
        <Value>4</Value>
    </Map>
    <Map>
        <Name>LOW</Name>
        <Value>1</Value>
    </Map>
    <Map>
        <Name>MEDIUM</Name>
        <Value>2</Value>
    </Map>
    <Map>
        <Name>HIGH</Name>
        <Value>3</Value>
    </Map>
    <Map>
        <Name>CERTIFIED</Name>
        <Value>4</Value>
    </Map>
    <Map>
        <Name>TEMPLATE</Name>
        <Value>1</Value>
    </Map>
</Mapping>
```

Figure 8 – Sample Mapping File

During the first performance tests, labeled objects are created. These objects are wrappered with a Type 0001 wrapper, followed by a Type 0003 wrapper. These first tests demonstrate that the labeling process can be executed in an acceptable timeframe. In a real world environment, these labeled objects would hosted on a server and made available to customers, subject to access control restrictions. As such, their use in later test confirms that labeled objects created can be used by the system for access



control and aggregation. It should be noted that during the development of the prototype system, a change in the order of the fields in the optional header was implemented. In the original Secure File Format specification, the XML object labels were the first field in the optional header. For the prototype system, the fixed length fields in the optional header were placed before the XML object label. This was done to simplify the parser design and does not affect the security of the labeling objects.

2.5 Test Scenarios

As stated, the testing performed was intended to assess the usability of object and user/system labels throughout the life cycle of the objects being protected. To support this goal, tests were executed that reflect operations at different parts of the life cycle. Table 8 shows the tests that performed and how they relate to the lifecycle.

Table 8 – Testing at different point in the lifecycle

Lifecycle	Test/Activity	Description
Creation	Type 0001 Wrappering	A key feature of the Secure File Format is the abstraction of the information being protected. In order to provide this abstraction, all protected objects are encapsulated in a Type 0001 wrapper. This provides a consistent object format for the rest of the Secure File Format operations
Creation	Type 0003 Wrappering	Type 0003 wrappers are used to attach the security attributes that will be used to make access control decisions
Access	Access Control	At this point, the objects have been encapsulated in a Type 0001 wrapper and have had one or more additional wrappers applied. The outer most wrapper will be used for access control decisions. For these tests, Type 0003 (native security attributes) or Type 0006 (aggregated security attributes are available for assessment
Aggregation	Type 0006 Wrappering	During these test, two or more labeled objects are “concatenated” together to create a larger collection. Because Secure File Format allows for the separation of an object from its label, there is no requirement to create and store each aggregation as a unique object. Instead, aggregated objects can be assembled as required provided the underlying Secure File Format objects are available.

2.5.1 Creation

During the creation phase, Type 0001 and Type 0003 wrappers were applied to each object using the SFF_wrapper.php program. Based on the inputs provided to SFF_wrapper.php either a Type 0001 wrapper was applied to the object or a Type 0003 wrapper was applied to an existing Secure File Format object. SFF_wrapper.php is intended to be run from either a linux command prompt or included in a shell script. The formal method was used to validate SFF_wrapper.php's functionality. Incorporation into shell script s were used for the performance tests.

When used to create Type 0001 wrappers, SFF_wrapper.php accepted the filename, and hashing algorithm as inputs. Optionally, an additional filename could be provided for the output. If an output filename was not provided, SFF_wrapper.php defaults to using the object's message digest with a .SFF extension for the output file.

When used to create and attach Type 0003 wrappers, SFF_wrapper.php accepts the filename of the Type 0001 wrappered object, the hashing algorithm, and either a text string with XML formatted security label or the filename of a file containing the XML formatted security label. Optionally, an additional filename can be provided for the output. If an output filename was not provided, SFF_wrapper.php defaulted to the message digest for the Type 0003 wrapper with a .SFF extension.

2.5.2 Access

During the part of the object's lifecycle where access to the object is being controlled by the labeling server, a single program, SFF_Handler.php, is used to make access control assessments. SFF_Handler.php is called by different programs, depending on the mode of access. For the interactive testing, (a single object at a time), SFF_Handler.php is called by SFF_Retriever.php. For the performance testing (multiple objects per run), SFF_Handler.php is called by SFF_Bulk_Retriever.php.

SFF_Retriever.php is invoked by the Apache web server and accesses the query and its attributes as inputs. It parses the returned values for the user label, system label and filename for the object and passes them to SFF_Handler.php for processing.

SFF_Bulk_Retriever.php is also invoked by the Apache web server, but does not accept any attributes. Instead, it generates a list of all of the labeled objects in the web server's data directory. From the filename, it selects one of 256 user labels and one of 256 system labels to be used as inputs for SFF_Handler.php. SFF_Bulk_Retriever.php then passes the object's filename, the user label and system label to SFF_Handler.php for processing. SFF_Bulk_Retriever.php repeats this process for each object in the data directory, thus forming a "RUN". SFF_Bulk_Retriever.php cycles through the RUN 100 times. For this test, this results in 10,000 access control determination being made per invocation of SFF_Bulk_Retriever.php. SFF_Bulk_Retriever.php provides the output back to the web client from which it was evoked. Along with the output from SFF_Handler.php for each decision, SFF_Bulk_Retriever.php returns the start and stop time for each invocation.

Using a single instant of SFF_Handler.php for both interactive and performance tests was required, not just to confirm the functionality of the program, but also to measure the program's performance in an interactive setting. Measuring the response time via just the interactive web client/web server would not yield results that were accurate. For the user's perspective, the time required for a page to be displayed included not just the time required for the access control decision to be made, but also the time required for transmission to the client and the time for the client to render the returned web page. These latter two activities are not germain to the SFF_Handler.php's performance.

2.5.3 Aggregation

During this part of the objects' lifecycles, access to the labeled object is not a concern. Instead, the focus is on creating collections of related objects and insuring that the new collection is properly protected. This collection process is called "aggregation" and includes not just the concatenation of the objects. In addition to the concatenation, the metadata carried in the object labels must be considered as well. For example, if two (or more) classified objects are put into a single collection, then the classification level of the collection is that of the object will the highest classification. Similar issues are present for most types of metadata.

Aggregation of information is handled on the labeling server in either an interactive manner (creating aggregated objects via a command prompt) or via a batch process, where object aggregates without direct user involvement. In both cases, SFF_wrapper.php is used to create the aggregates. For aggregation, a Type 0006 wrapper is created. Type 0006 wrappers differ from the other wrappers discussed deals with the inclusion of the payload. Type 0006 wrappers can be created with either the payload included or the payload not included. The determination of whether to include the payload is based on how the Type 0006 wrapper will be used and if the objects that would be included the payload will be available after the object is created. For example, if the aggregation process is being used to create a "CD" that will be provided to a customer, then including the aggregated objects in the payload is appropriate. If the type 0006 wrapper was created as part of creating a collection that will be served up by the web server, then inclusion of the aggregated object with the Type 0006 wrapper is unwarranted.

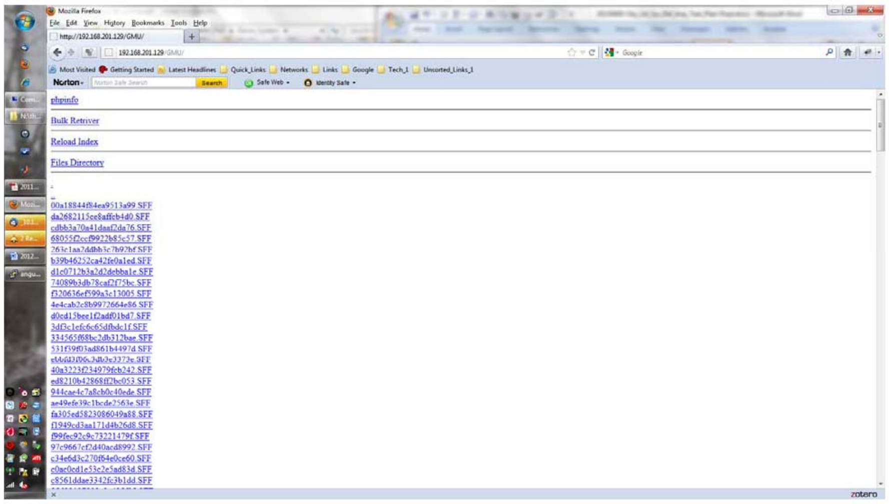
For this test, Type 0006 Secure File Format objects were created with a small (~1000 character) payloads. Building aggregates with larger objects does not affect the aggregation process, but rather increases the time to create the object. For a Type 0006 wrapper, SFF_wrapper.php accepts the hashing algorithm, and list of filenames for the objects to be aggregated. Optionally, an additional filename could be provided for the output. If an output filename is not provided, SFF_wrapper.php defaults to using the type 0006 wrapper's message digest with a .SFF extension for the filename.

Aggregation testing was conducted using test scripts. These scripts created aggregates that included 2, 4, 8, 16, 32, and 64 Type 0003 objects. Sha1, Sha256, Sha384, and Sha512 message digests were created for each of the aggregates. For each of the 24 possible combinations, 1000 aggregates were created. In total 24,000 type 0006 aggregated objects were generated.

3. Interactive Access Test Cases

Because of the need to collect timing metrics, interactive access testing was used only to validate that SFF_Handler.php was processing the labels properly. While SFF_Bulk_Retriever.php was started via the web interface, it is not considered an interactive test as the user does not interact with it once, it is started.

Figure 9 show the top of the main navigation screen for the labeling server's web page. From this screen, the user is able to start the SFF_Buik_Retriever.php script by clicking on the "Buik_Retriever" link. At the bottom of the screen are the links to the individual labeled objects stored on the server. Click in any of the links starts the process of requesting access to the object.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://192.168.201.129/GMU/
192.168.201.129/GMU/
Most Visited Getting Started Latest Headlines Quick_Links Networks Links Google Tech_1 Unsorted_Links_1
Norton Search Safe Web Identity Safe

phpinfo

Bulk Retriver

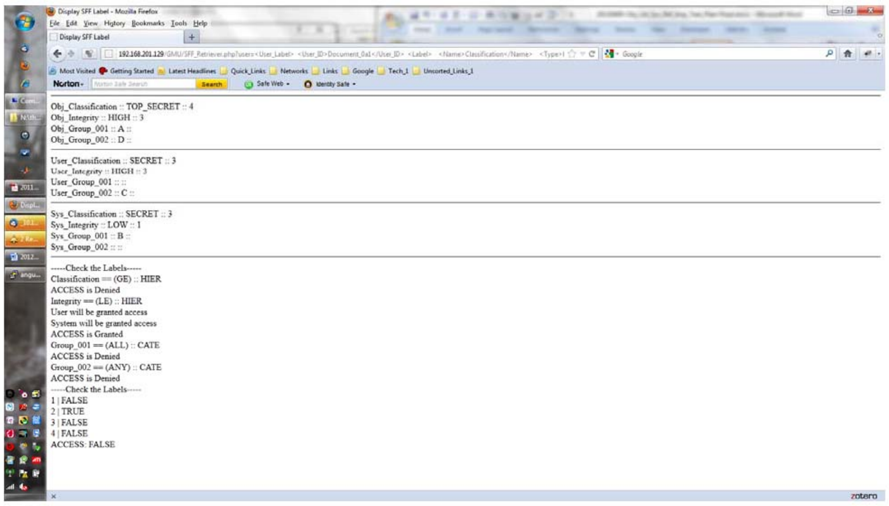
Reload Index

Files Directory

.
..
00a18844f84ea9513a99.SFF
da2682115ee8affcb4d0.SFF
cdbb3a70a41daaf2da76.SFF
68055f2ccf9922b85c57.SFF
263c1aa2ddbb3c7b92bf.SFF
b39b46252ca42fe0a1ed.SFF
d1c0712b3a2d2debba1e.SFF
74089b3db78caf2f75bc.SFF
f320636ef599a3c13005.SFF
4e4cab2c8b9972664e86.SFF
d0cd15bee1f2adf01bd7.SFF
3df3c1efc6c65dfbdc1f.SFF
334565f68bc2db312bae.SFF
531f39f03ad861b4497d.SFF
[illegible]
40a3223f234979fcb242.SFF
ed8210b42868ff2bc053.SFF
944cae4c7a8cb0c40ede.SFF
ae49efe39c1bcde2563e.SFF
fa305ed5823086049a88.SFF
f1949cd3aa171d4b26d8.SFF
f99fec92c9c73221479f.SFF
97c9667cf2d40acd8992.SFF
c34e6d3c270f64e0ce60.SFF
c0ac0cd1e53c2e5ad83d.SFF
c8561ddae3342fc3b1dd.SFF

Figure 9 – Main Navigation Screen

Figure 10 – Sample Unsuccessful Access AttemptFigure 10 shows the results of the access control determination. The selected object carries four labels that were used in access control determination. The four labels used where specified by the RULES files. In this case, the object carries a classification of "TOP_SECRET", an integrity value of "HIGH", a Group_001 value of "A", and a Group_002 value of "D".

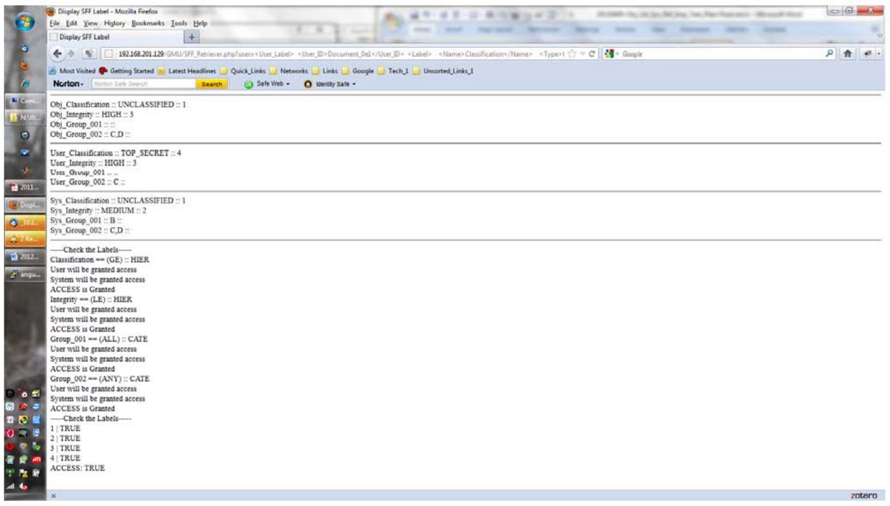


```
Obj_Classification :: TOP_SECRET :: 4
Obj_Integrity :: HIGH :: 3
Obj_Group_001 :: A ::
Obj_Group_002 :: D ::

User_Classification :: SECRET :: 3
User_Integrity :: HIGH :: 3
User_Group_001 :: ::
User_Group_002 :: C ::

Sys_Classification :: SECRET :: 3
Sys_Integrity :: LOW :: 1
Sys_Group_001 :: B ::
Sys_Group_002 :: ::

-----Check the Labels-----
Classification == (GE) :: HIER
ACCESS is Denied
Integrity == (LE) :: HIER
User will be granted access
System will be granted access
ACCESS is Granted
Group_001 == (ALL) :: CATE
ACCESS is Denied
Group_002 == (ANY) :: CATE
ACCESS is Denied
-----Check the Labels-----
1 | FALSE
2 | TRUE
3 | FALSE
4 | FALSE
ACCESS: FALSE
```

Figure 10 – Sample Unsuccessful Access Attempt

The next two sections show the User Label values and System Label values. Following the User and System Labels, are the results of the access control comparison.

In the "Check the Labels" Section , the results of the four comparisons is summarized. For the classification, access is denied. In this case, neither the user nor the system has classification value that is greater than or equal to that of the object. For the integrity, access is granted. In this case, both the user and system requirements for integrity are less than or equal to that of the object. For Group_001, the user and system must be a member of the every group of which the object is a member. The user is not a member of any group and the system is only a member of a group to which the object does not belong. Therefore, access is again denied. For Group_002, the user and system are not members of any groups to which the object belongs. Access is denied. The final section shows the final determination. Because access was denied for one or more of the individual labels, access to the object is denied.



```
Obj_Classification :: UNCLASSIFIED :: 1
Obj_Integrity :: HIGH :: 3
Obj_Group_001 :: ::
Obj_Group_002 :: C,D ::

User_Classification :: TOP_SECRET :: 4
User_Integrity :: HIGH :: 3
User_Group_001 :: ::
User_Group_002 :: C ::

Sys_Classification :: UNCLASSIFIED :: 1
Sys_Integrity :: MEDIUM :: 2
Sys_Group_001 :: B ::
Sys_Group_002 :: C,D ::

-----Check the Labels-----
Classification == (GE) :: HIER
User will be granted access
System will be granted access
ACCESS is Granted
Integrity == (LE) :: HIER
User will be granted access
System will be granted access
ACCESS is Granted
Group_001 == (ALL) :: CATE
User will be granted access
System will be granted access
ACCESS is Granted
Group_002 == (ANY) :: CATE
User will be granted access
System will be granted access
ACCESS is Granted
-----Check the Labels-----
1 | TRUE
2 | TRUE
3 | TRUE
4 | TRUE
ACCESS: TRUE
```

Figure 11 – Sample Successful Access Attempt

Figure 11 shows another results screen. For this request, access has been granted. Checking the summary section shows that all 4 label comparison result in the both the user and system being granted access. For the classification, the object is UNCLASSIFIED and can access by the user, who has a 'TOP_SECRET" classification and the system that has an "UNCLASSIFIED" classification. For the Integrity, the user and system requirements are less than that of the object, so access is granted. For Group_001, the object is not part of any group, so there is no group_001 requirement and access is granted. For Group_002, the user is a member of one of the two groups of which the object is a member, and the system is member of both. Therefore, access is granted.

4. Performance Test Cases

As stated previously the interactive tests were used to visual confirm that SFF_Handler.php was processing the object, user, and systems properly. With this interactive inspection completed, testing shifted to scripted testing. Scripted testing removed the user from the testing processes and allows the system to include timing information based on the system clock.

Four sets of performance tests were conducted. The first set of tests measured the system's performance while initially creating the labeled objects. The second set of tests measured the system's performance while adding security metadata to labeled objects. The third set of tests measured the system's performance while making access control determinations. The fourth set of tests measured the system's performance will aggregating objects.

4.1 Initial Object Labeling

The test data for this test case consisted of 4100 files. For each of the 4100 files, the wrapper program, SFF_wrapper.php, was run a total of four times. This was done to create a Type 0001 wrappered object for each of the supported message digests, sha1, sha256, sha384, and sha512. A total of 16400 wrappered files were created and used throughout the rest of the testing. In total, this equates to over 16 Gbytes for working labeled data that was loaded to the server for the use in the remaining tests.

SFF_wrapper.php was called repeatedly by a script program that was used to create 16,400 Type 0001 Secure File Formatted objects. In addition to controlling the labeler, this script program capture the timing information along with the parameters provided to SFF_wrapper.php. For these results, the average time to create a Type 0001 wrapper object was calculated. Table 9 shows the average time. Additionally, Figure 12 shows the average time graphically.

Table 9 - Time to wrapper an Object

	Average Time (second)			
	sha1	sha256	sha384	sha512
1000	0.12037	0.121804	0.120339	0.123657
2000	0.119747	0.119195	0.119716	0.119847
5000	0.119764	0.119524	0.11936	0.119585
10000	0.120213	0.119527	0.119494	0.119165
20000	0.119903	0.119793	0.11955	0.119814
50000	0.120057	0.119292	0.11946	0.12026
100000	0.121177	0.121044	0.122016	0.119165
200000	0.122548	0.123823	0.122763	0.123194
500000	0.129645	0.132979	0.130967	0.133941
1000000	0.134704	0.1459	0.141748	0.142606
2000000	0.153164	0.162812	0.15997	0.157014
5000000	0.201638	0.250973	0.220422	0.215108
10000000	0.410876	0.407461	0.38784	0.398927

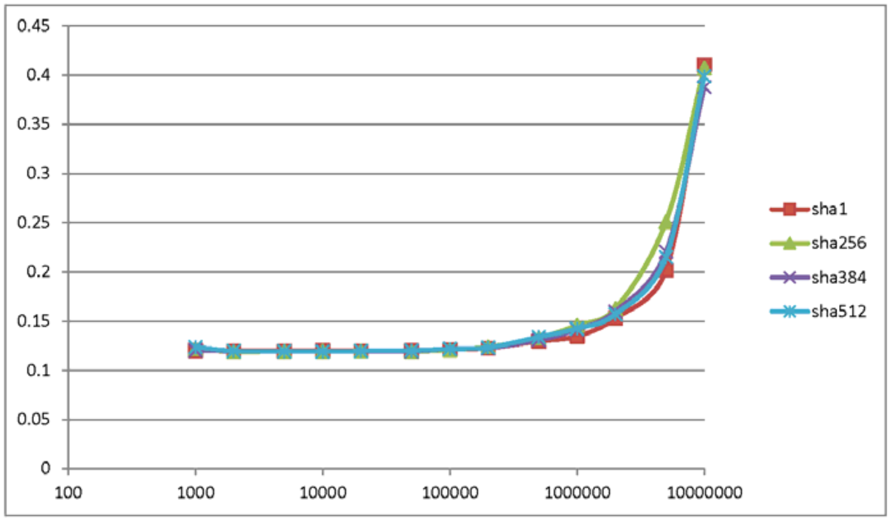


Figure 12 - Average Time to Wrapper an Object

From Figure 12, it can be seen that the time required to label larger files is larger. This is an expected result. However, this result is not linear; larger files are wrappered at a greater rate than smaller files.

Table 10 revisits the average time to wrapper an object, but takes into account the size of the object being wrapppered. Figure 13 shows the same information graphically. It can be seen that the speed with which SFF_wrappers.php creates Type 0001 objects increases with the size of the file. This continues until a file size of about 5,000,000 is reached. It is at this point that a knee in the curve appears to have been reached. However, work with larger file sizes would be needed to confirm this. From a performance perspective, this increase in speed as file size increases allows SFF_wrapper.php to create about 6 labeled objects per second for file sizes that are less than 5,000,000. Equally important, this means that SFF_wrapper.php can process over 1 Gigabyte of data in about 40 minutes for 50,000 character files. For larger files, the time is even shorter.

Table 10 – Data Rate for Wrappering Objects

	Data Rate (characters/second)			
	sha1	sha256	sha384	sha512
1000	8307.69	8209.945	8309.839	8086.893
2000	16701.92	16779.29	16706.2	16687.96
5000	41748.94	41832.58	41890.11	41811.32
10000	83185.68	83663.01	83685.9	83917.54
20000	166801	166954.4	167293.3	166924.7
50000	416470	419139.3	418549.9	415764.3
100000	825240.6	826142.6	819566	821191.5
200000	1632008	1615211	1629162	1623453
500000	3856693	3759982	3817758	3732984
1000000	7423710	6854031	7054753	7012333
2000000	13057921	12284102	12502376	12737734
5000000	24796896	19922438	22683802	23244142
10000000	24338235	24542199	25783806	25067259

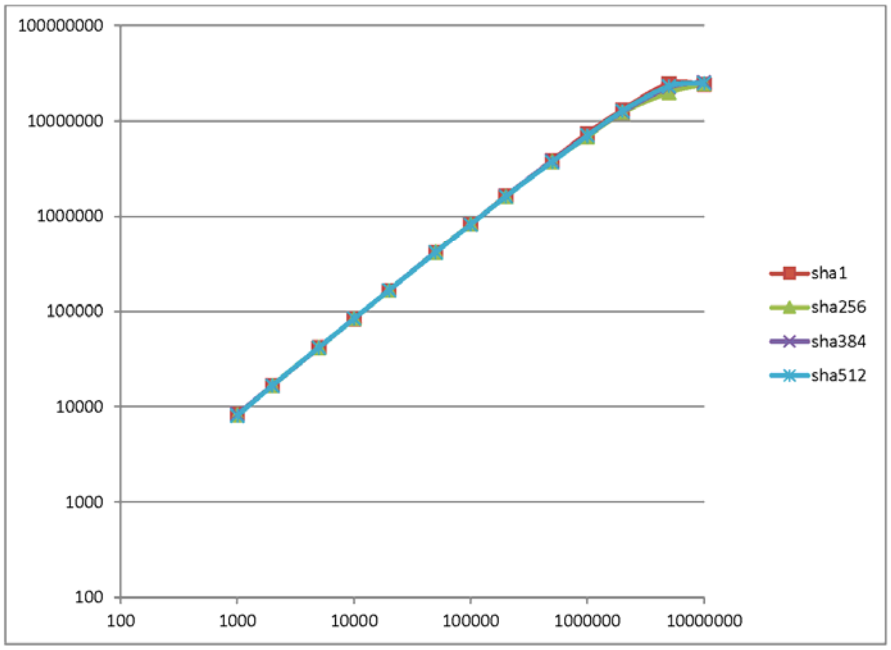


Figure 13 – Data Rate for Wrappering Objects

4.2 Applying Security Metadata

The second part of creating Secure File Formatted objects is the application of security metadata. This metadata is captured and applied through the use of a Type 0003 wrapper. The contents of this wrapper are used to identify the security attributes of the wrappered object and to cryptographically bind these attributes to the underlying object. For this test, the 16400 Type 0001 wrappered objects would serve as the payload. Each of these type 0001 wrappered objects was augmented with a Type 0003 wrapper containing one of 256 object labels. The specific label that was applied was determined by two characters in the message digest for that object. As with the Type 0001 wrappers, SFF_wrapper.php was used to apply the additional layer.

SFF_wrapper.php was called repeatedly by a script program that provided the filename of the Type 0001 object, the filename of the object label to be used, and the message digest algorithm. Message digest algorithm selection was based on the message digest used for the Type 0001 wrapper, itself. At the end of the script's operation, a total of 16400 new Type 0003 wrappered objects had been created. These objects included the new wrapper and the original Type 0001 wrappered object. In addition to controlling SFF_wrapper.php, this script captured the time metrics for the operation. Table 11 shows the average time required to applied a Type 0003 wrapper. Figure 14 shows the average times graphically.

Table 11 – Time to Label an Object

	Average Time (second)			
	sha1	sha256	sha384	sha512
1000	0.159735	0.159481	0.157642	0.159979
2000	0.163542	0.162795	0.16527	0.163543
5000	0.160651	0.165042	0.161613	0.161605
10000	0.166401	0.154078	0.159852	0.167859
20000	0.173134	0.166151	0.162167	0.166976
50000	0.170834	0.16455	0.171404	0.169289
100000	0.166934	0.171809	0.174076	0.167859
200000	0.177222	0.176471	0.19064	0.172026
500000	0.175228	0.177451	0.174632	0.169887
1000000	0.205134	0.207852	0.204475	0.193928
2000000	0.222568	0.218338	0.214365	0.203296
5000000	0.28267	0.279383	0.278033	0.288995
10000000	0.374797	0.391362	0.373463	0.407301

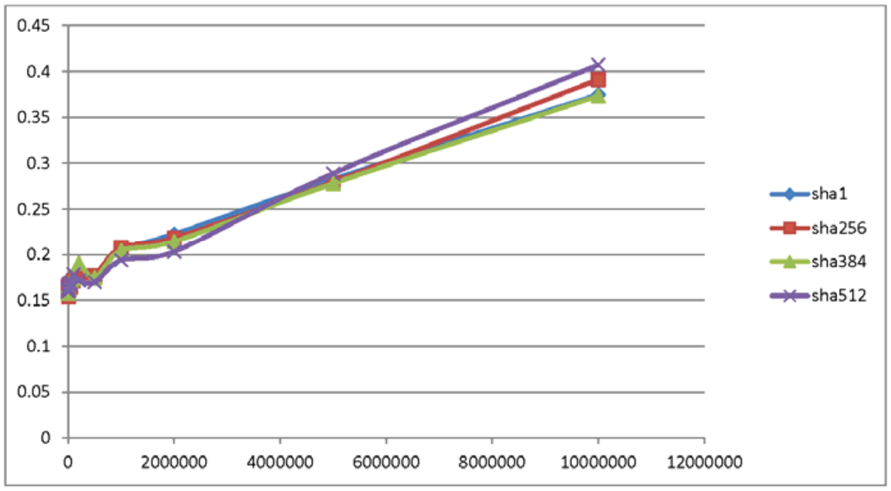


Figure 14 – Time to Label an Object

From Figure 14, it can be seen that the time to label an object increases as a function of the size of the size of the object being labeled. This is expected given that the original object is copied into the new object. Larger files will take longer to copy. As with the addition of a Type 0001 wrapper to the raw data file, the time required to label an object increased more slowly than expected as file size grew. Again, it was seen that the larger file size the faster the object could be created. Table 12 shows the rate at which an object could be labeled as a function of the size of the initial object. Figure 15 shows this data graphically. At around 500,000 characters, a knee starts to appear in the curve. However, as with wrappering an object, labeling an object is efficient enough to support labeling 1 Gbyte of data in around 40 minutes.

Table 12 – Date Rate for Labeled Objects

	Data Rate (characters/second)			
	sha1	sha256	sha384	sha512
1000	6260.371	6270.355	6343.488	6250.822
2000	12229	12285	12101	12229
5000	31123	30295	30938	30940
10000	60096	64902	62558	59574
20000	115518	120373	123330	119778
50000	292683	303860	291708	295352
100000	599039	582042	574463	560715
200000	1128528	1133334	1049097	1162613
500000	2853428	2817676	2863172	2943135
1000000	4874861	4811109	4890572	5156554

2000000	8986010	9160113	9329898	9837871
5000000	17688458	17896576	17983471	17301340
10000000	26681117	25551820	26776430	24551846

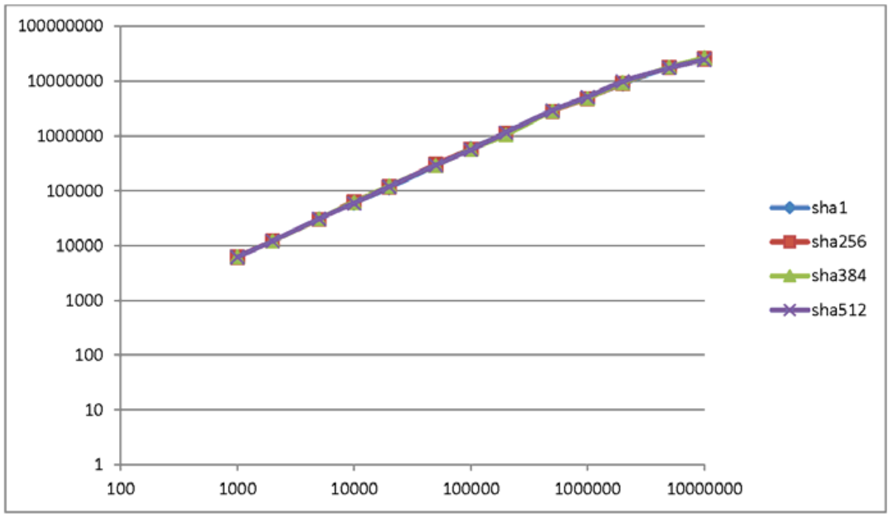


Figure 15 – Data Rate for Labeled Objects

4.3 Access Control

While loading data should only need to be performed once, an object may be accessed any number of times after it is hosted. The third set of tests assess the labeling server's ability to support a large number of requests for access control decisions.

The task of making access control decision was handled by SFF_Handler.php. This one file was used for both the interactive queries as well as the bulk requests. Because the interactive queries involved elements that were not part of the access control decision, the data for this test was gathered using SFF_Bulk_Retriever.php. SFF_Bulk_Retriever.php executes 10000 access control requests by invoking SFF_Handler.php 10000 times. Additionally, SFF_Bulk_Retriever.php captures the start and end times for each run. In assessing SFF_Handler.php's performance 250.000 access control requests were issued. Table 13 details the key results from these tests.

Table 13 – Access Control Statistics

SFF_Handler.php Performance	
Average Time (per 10000 requests)	26.6 seconds
Average Time to process a single request	2656 microseconds.

This test demonstrated that the labeling server was able to provide over 20000 access control decision in under one minute. As mentioned before, these values do not account for the effort associated with returning the requested object. However, the performance penalty for returning data is no greater than that of a web server returning non-labeled information. The other key fact that these test demonstrated was that the process of processing an access control request, using 4 security metadata fields, on average added less than 3 milliseconds to the overall time required to process a request.

4.4 Aggregation

The final test performed focused on the labeling server's performance while performing aggregation. Analysis of the results that the more objects that are included in an aggregate, the longer that it takes. Table 14 shows the average times for aggregation based on the number of objects in the aggregate and the message digest length used. Figure 16 shows the same information graphically.

Table 14 – Average Time to Aggregate multiple objects

	Average Time (second)			
	sha1	sha256	sha384	sha512
2	0.126437	0.126444	0.127289	0.12722
4	0.129499	0.129761	0.129783	0.129467
8	0.130449	0.130615	0.130262	0.130432
16	0.132482	0.132788	0.132388	0.132444
32	0.143302	0.143333	0.143253	0.143583
64	0.148147	0.148391	0.148324	0.148218

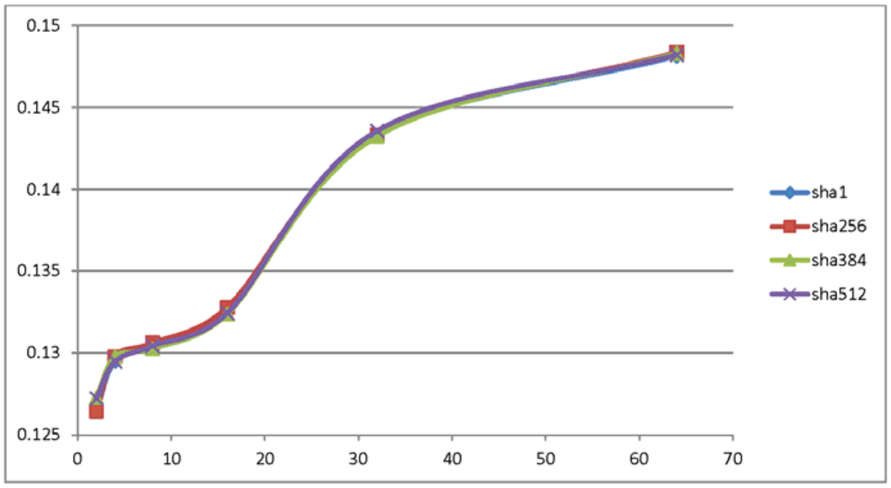


Figure 16 – Average Time to Aggregate multiple objects

Further analysis was conducted to determine what impact the number of objects being aggregated had on the number of aggregation that could be performed per unit time. Table 15 shows the number of objects that could be aggregated per second as a function of the number of objects in the aggregate. Figure 17 shows the same information graphically. What this data reveals is that the performance begins to taper off when there are more than 16 objects in the aggregate. The other factor that these test showed was that there is an initial "startup" time that appears to be independent of the number of objects being aggregated,

Table 15 – Aggregation Rates

	Aggregation Rate (objects/second)			
	sha1	sha256	sha384	sha512
2	15.82	15.82	15.71	15.72
4	30.89	30.83	30.82	30.90
8	61.33	61.25	61.41	61.33
16	120.77	120.49	120.86	120.81
32	223.30	223.26	223.38	222.87
64	432.00	431.29	431.49	431.80

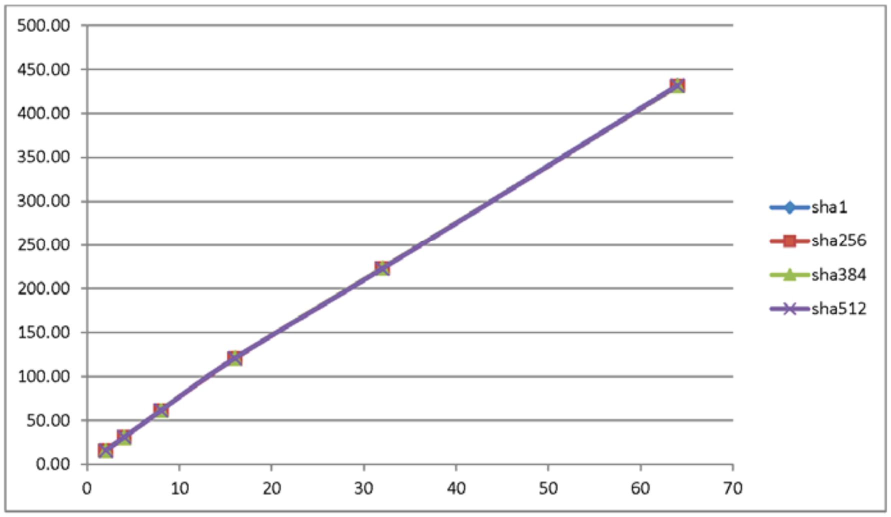


Figure 17 – Aggregation Rates

5. Conclusions and Follow Up Work

In conclusion, these tests were intended to show that an access control system that was based on the use of object labels could be developed and be cost effective.

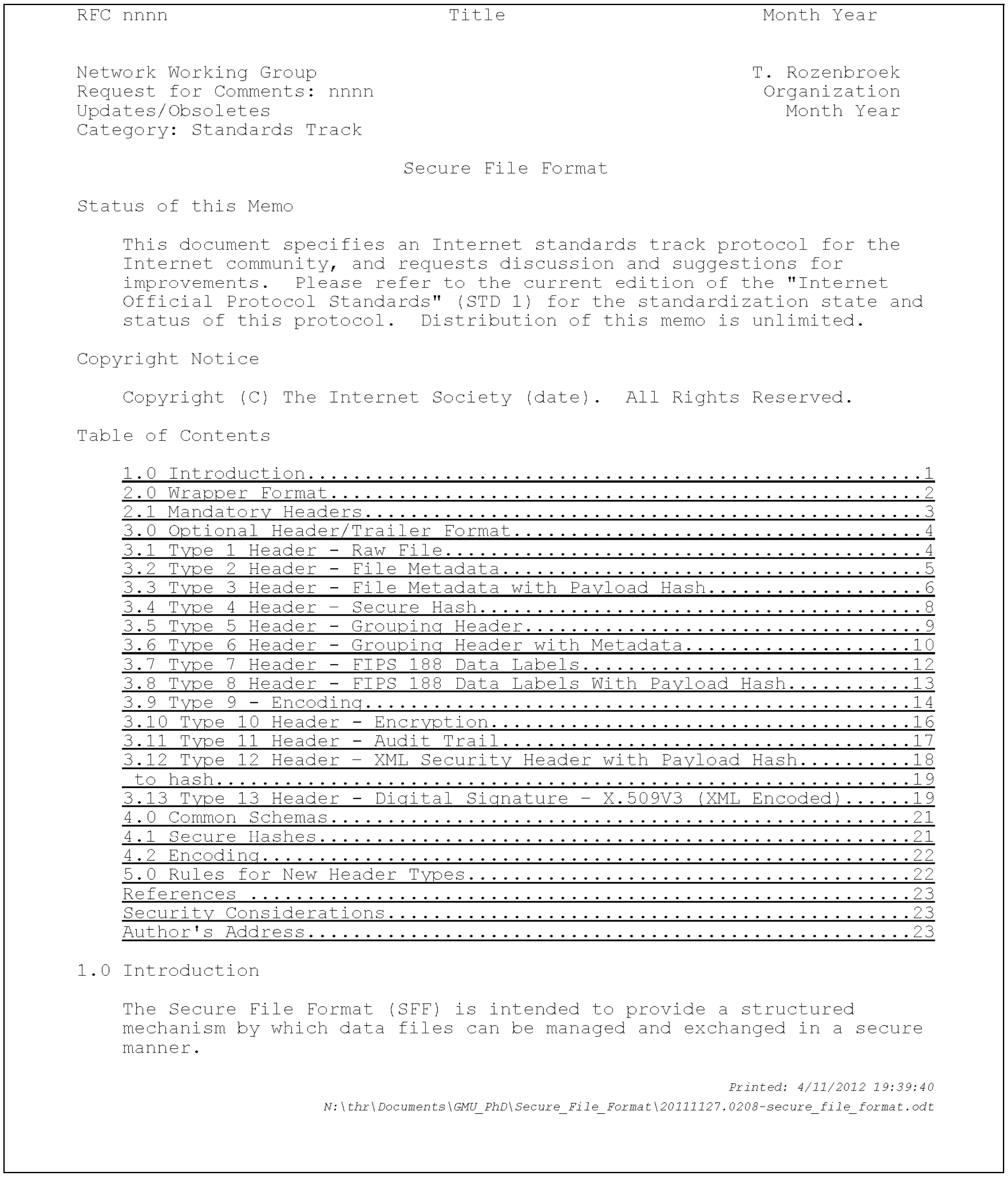
Based on the performance metrics that were gathered, the labeling server was able to handle the major access control and information management activities in a time and effective manner. Most of the activities were performed in under 0.20 seconds. However, the collected data suggests that much of this time is associated with starting each of the operations. Much of this startup time could be reduced with the use of a persistence labeling application.

The second issue was the cost factor. Part of this investigation investigated the cost for a server. Based on the published specification for the hardware used for test and the product literature available from the Internet, it was calculated that the same or better functionality could be gotten from a Hewlett Packard DL320 Proliant server. This server offered enough disk space to maintain 250,000 1 Mbyte files, a processor that was rated almost 60% faster, and an equal amount of memory. The cost for this system was less than $2000.

In short, these tests and the market investigation showed that a $2000 system should be able to provide the functionality required to implement an object label based access control solution.

APPENDIX H – SECURE FILE FORMAT RFC (DRAFT)

This appendix contains a copy of the text of the Secure File Format RFC. At the time that this dissertation was being written, it was also being rewritten to include additional functionalities not covered in the original version, which was published at part of *An Architecture for Managing Access to and Permission for Multiple Versions of Objects in a Distributed Environment* [34].



Network Working Group T. Rozenbroek
Request for Comments: nnnn Organization
Updates/Obsoletes Month Year
Category: Standards Track

Secure File Format

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Table of Contents

1.0 Introduction

The Secure File Format (SFF) is intended to provide a structured mechanism by which data files can be managed and exchanged in a secure manner.

Printed: 4/11/2012 19:39:40
N:\thr\Documents\GMU_PhD\Secure_File_Format\20111127.0208-secure_file_format.odt

The Secure File Format provides security and management functions by applying one or more “wrappers” to the original data file or data files. There are different types of wrappers, each of which provides a different security and/or management service. Examples of the security services provided include Mandatory Access Control using data labels, Encryption, Digital Sign-offs, and Audit Trails. Examples of management services include the addition of meta-data and grouping of several related files.

2.0 Wrapper Format

The general structure of a Secure File Format file is depicted in Figure 1. This figure shows the original data file (Secured Data File) enclosed in one or more sets of headers/trailers. Each header and associated trailer form a wrapper or layer. The headers contain the administrative and security related information. The trailers are typically used to provide 'padding' that is required in order to provide properly sized objects for checksumming and secure hashing. The use of the trailer in this manner allows the payload to be secured while still maintaining its integrity.

```
      +-----------+
      | Header 1  |
      +-----------+
      /           /
      /           /
      /           /
      +-----------+
      | Header n  |
      +-----------+
      |  Secured  |
      | Data File |
      +-----------+
      | Trailer n |
      +-----------+
      /           /
      /           /
      /           /
      +-----------+
      | Trailer 1 |
      +-----------+
```
Figure 1 - SFF File Format

The general structure of each SFF layer is depicted in Figure 2. Each layer consists of four different parts. The first two parts compose the header for that layer. The first part is the “Mandatory Header”. This part is carried by all SFF layers, and carries information that needed to resolve the “Optional Header”. The “Optional Header” is the second part of the header. The Optional Header contains information for the functions that the layer provides. Section 2.1 describes the structure of the Mandatory Headers. Section 3 describes the structures of the Optional Headers. The next part is the payload. The payload consists of either the secured data file or another SFF layer. The final part is the trailer. The function of the trailer is to insure that integrity of the rest of parts of that layer.

```
+------------------+-----------------+---------+---------+
| Mandatory Header | Optional Header | Payload | Trailer |
+------------------+-----------------+---------+---------+
```
Figure 2 - SSF Layer Format

All fields in the Mandatory Header, the Optional Header and the Trailer are multiple of 8-bits in length. Common field sizes include 16, 32, and 64 Bit fields which are stored as 4, 8, and 16 character hexadecimal values, respectively. However, any field size that is a multiple of 8-bits is valid. Additionally, all numerical fields in the Mandatory Header, Optional Header, and Trailer fields are stored as Big-Endian (MSB first) values. The Payload does not carry the same format limitation.

2.1 Mandatory Headers

The format of the Mandatory Headers is shown in Figure 3. A Mandatory Header is composed of five numerical fields. These fields are:

Type,
Version,
Header Size,
Payload Size, and
Trailer Size.

The 'Type' field is a 32-bit (4 Byte) field that denotes the type of Optional Header. Each 4-bits in the Type field can be represented as a hexadecimal value. Section 3 details the defined values for this field. A Type Value of 00000000h is prohibited. Any Type Value with the field Hexadecimal digit value of Fh in the most significant digit (F0000000h - FFFFFFFFh) can be used for internal/development purposes.

```
+------+---------+-------------+--------------+--------------+
| Type | Version | Header Size | Payload Size | Trailer Size |
+------+---------+-------------+--------------+--------------+
```
Figure 3 - Mandatory Header Format

The 'Version' field is a 32-bit (4 Byte) field that denotes the version of that type of Optional Header. Each 4-bits in the Version field can be represented as a hexadecimal value. The Version field is included to allow for easier parsing of SSF files. For example, a hashing algorithm, which can produce different length hashes would only have a single Type Value and multiple Version. There would be one version for each hash length.

The 'Header Size' field is a 64-bit (8 Byte) field that stores the size in bytes of both of the two headers (Mandatory and Optional). The Header Size value is represented as a hexadecimal value. The minimum value of the Header Size field is 32 (00000020h). The Optional Header fields, if present, begin at the thirty-third (33) byte.

The Header Size field can be used as the Payload Offset.

The 'Payload Size' field is a 64-bit (8 Byte) field that stores the size of the payload. The Payload Size value is represented as a hexadecimal value.

The sum of the Header Size and Payload Size fields can be used as the Trailer Offset.

The 'Trailer Size' field is a 64-bit (8 Byte) field that contains the size of the Trailer. The Trailer Size value is represented as a hexadecimal value. When the three size fields are added together, the result equals the total size of the Secure File Format file.

3.0 Optional Header/Trailer Format

Each Header/Trailer pair has an unique set of fields that address the requirement for that wrapper. However, each pair contains the mandatory standard fields defined above, and zero or more optional fields that contain the management and security information for the payload. The information contained in the optional header fields is determined by the 'Type' and 'Version' fields in the Mandatory Header. These types are global defined by a RFC, but the ability to create locally unique optional headers is retained. For locally unique optional headers, the first four bits in the Version field are set to '1' (F0000000h through FFFFFFFFh).

3.1 Type 1 Header - Raw File

Type = 00000001h
Title: Raw File
Version = 00000001h|00000002h|00000003h|00000004h|00000005h

Description: This unit wraps a raw file within a base SFF wrapper. The wrapper has two basic formats. The first contains only the mandatory header fields; there are no optional header fields or trailer.

The second format contains the optional fields required to support the secure hashing of the payload. This unit must be the inner most wrapper of any secure file format file.

Format (Version = 00000001h):
```
+-----------+------------+-----------+----------+-----------+
| Type      | Version    | Hdr Size  | Pay Size | Trl Size  \
    | 00000001h | 00000001h  | 00000020h |  Varies  | 00000000h /
    +-----------+------------+-----------+----------+-----------+

    +---------+---------+
    \ Payload | Trailer |
    / Varies  |   N/A   |
    +---------+---------+

    Format (Version = 00000002h|00000003h|00000004h|00000005h):
    +-----------+------------+----------+----------+----------+
    | Type      | Version    | Hdr Size | Pay Size | Trl Size \
    | 00000001h | see Note A | Varies   |  Varies  | Varies   /
    +-----------+------------+----------+----------+----------+

    +-------------+---------+------------+
    \ Secure Hash | Payload | Trailer    |
    / see Note A  | Varies  | see Note B |
    +-------------+---------+------------+

    Purpose:  This header is used to create a generic SFF file from any
    file without regard of the file’s type or structure.  This header
    should be used anytime an object is handled using the Secure File
    Format.

    Its secondary purpose is to provide a cryptographically verifiable
    means of assuring that the Payload's contents have not been altered.
    The use of the version of the wrapper that provide a secure hash value
    of 'message digest' is critical to cryptographically binding the
    information on the optional headers to the contents of the Payload.

    Notes:
     A - The application of a secure hash of the payload is defined in
         Section 4.1 of this RFC.

     B - The Trailer will be filled with '0' as padding to supply the
         secure hashing solution with the correct amount of information
         to hash.

3.2 Type 2 Header - File Metadata

    Type = 00000002h
    Title: File Metadata
    Version = 00000001h|00000002h|00000003h|00000004h|00000005h

    Description: This wrapper captures the metadata, which is related to
    the raw object that is contained in the payload section of this
    header.  The metadata, itself, is stored as an XML object.  XML
    objects have been defined by several other RFCs [RFC2396] and the W3C.
    The current are defined by  [W3C.REC-xml]. This RFC does not impose
    any restrictions of the format or content of the XML Object.  This is
    done to allow for the widest range of formats of metadata without
    having to define a new header for each format of metadata.  When a
    Secure Hash is used, it is only applicable to the Object Metadata
    field.  For those instances, where the Metadata and the Payload need
    to be securely bound, a Type 3 (File Metadata with Payload Hash)
    should be used.  For those instances, where the Metadata and the
    Payload need to be securely hashed using a single hash, a Type 4
    (Secure Hash) wrapper should be applied.

    Format (Version = 00000001h):
    +-----------+-----------+----------+----------+-----------+
    | Type      | Version   | Hdr Size | Pay Size | Trl Size  \
    | 00000002h | 00000001h |  Varies  |  Varies  | 00000000h /
    +-----------+-----------+----------+----------+-----------+

    +-----------------+---------+---------+
    \ Object Metadata | Payload | Trailer |
    /  XML formatted  |  Varies |   N/A   |
    +-----------------+---------+---------+

    Format (Version = 00000002h|00000003h|00000004h|00000005h):
    +-----------+------------+----------+----------+----------+
    | Type      | Version    | Hdr Size | Pay Size | Trl Size \
    | 00000002h | see Note A | Varies   |  Varies  | Varies   /
    +-----------+------------+----------+----------+----------+

    +-------------+-----------------+---------+------------+
    \ Secure Hash | Object Metadata | Payload | Trailer    |
    / see Note A  |  XML formatted  | Varies  | see Note B |
    +-------------+-----------------+---------+------------+

    Purpose:  This header is used to capture the metadata for the object
    or objects contained in the payload section.

    Notes:
     A - The application of a secure hash of the payload is defined in
         Section 4.1 of this RFC.

     B - The Trailer will be filled with '0' as padding to supply the
         secure hashing solution with the correct amount of information
         to hash.

3.3 Type 3 Header - File Metadata with Payload Hash

    Type = 00000003h
    Title: File Metadata with Payload Hash
    Version = 00000001h|00000002h|00000003h|00000004h|00000005h

    Description: This wrapper captures the metadata, which is related to
    the raw object that is contained in the payload section of this
    header.  As with a Type 2 (File Metadata) header, the metadata is
    stored as an XML object.  XML objects have been defined by several
    other RFCs [RFC2396] and the W3C.  The current are defined by
    [W3C.REC-xml]. This RFC does not impose any restrictions of the format
```

```
or content of the XML Object.  This is done to allow for the widest
range of formats of metadata without having to define a new header for
each format of metadata.  The wrapper differs from a Type 2 (File
Metadata) wrapper in that the Version from the Mandatory header and
Secure Hash field from the Optional Header of the Payload are included
in the Optional Header of the wrapper.  This wrapper should only be
used with payloads that contain a secure hash field that conform to
the specification in section 4.1 of this RFC.  If the payload does not
a Secure Hash field as part of the Optional header, then the Secure
Hash of the Payload field is omitted.  When a Secure Hash is part of
the Optional Header, it is only applicable to the Object Metadata
field, the Version field (from the Payload), and the Secure Hash field
(from the Payload).  For those instances, where the Metadata and the
Object need to be securely hashed using a single hash, a Type 4
(Secure Hash) wrapper should be applied.  Type 3 Headers are the
preferred approach for securely attaching metadata to a Payload when
the wrappers are being attached throughout the secured data file's
lifecycle.

Format (Version = 00000001h):
+-----------+-----------+----------+----------+-----------+
| Type      | Version   | Hdr Size | Pay Size | Trl Size  \
| 00000003h | 00000001h |  Varies  |  Varies  | 00000000h /
+-----------+-----------+----------+----------+-----------+

+-----------------+--------------+-------------+---------+---------+
\ Object Metadata | Hash Version | Secure Hash | Payload | Trailer |
/  XML formatted  | of Payload   | of Payload  |  Varies |   N/A   |
+-----------------+--------------+-------------+---------+---------+

Format (Version = 00000002h|00000003h|00000004h|00000005h):
+-----------+------------+----------+----------+----------+
| Type      | Version    | Hdr Size | Pay Size | Trl Size \
| 00000003h | see Note A | Varies   |  Varies  | Varies   /
+-----------+------------+----------+----------+----------+

+-------------+-----------------+--------------+--------------+
\ Secure Hash | Object Metadata | Hash Version | Secure Hash  \
/ see Note A  |  XML formatted  | of Payload   | of Payload   /
+-------------+-----------------+--------------+--------------+

+---------+------------+
\ Payload | Trailer    |
/ Varies  | see Note B |
+---------+------------+

Purpose:  This header is used to capture the metadata for the object
or objects contained in the payload section.  Additionally, by
including the Secure Hash from the Payload, the secured contents of
the Payload are cryptographically linked with the Object's Metadata.

Notes:
      A - The application of a secure hash of the payload is defined in
          Section 4.1 of this RFC.

      B - The Trailer will be filled with '0' as padding to supply the
          secure hashing solution with the correct amount of information
          to hash.

3.4 Type 4 Header - Secure Hash
   Type = 00000004h
   Title: Secure Hash

    Version = 00000002h|00000003h|00000004h|00000005h

    Description:  This wrapper is used to secure both the Metadata in the
    Optional Header and the Payload with a single secure hash.  This
    wrapper should be used when there is no need to interact with the
    metadata independent of the Payload.  The metadata is stored as an XML
    object.  XML objects have been defined by several other RFCs [RFC2396]
    and the W3C.  The current are defined by  [W3C.REC-xml]. This RFC does
    not impose any restrictions of the format or content of the XML
    Object.  This is done to allow for the widest range of formats of
    metadata without having to define a new header for each format of
    metadata.  Because the Secure Hash field is a required part of the
    Optional Header, there is no defined format where the Version =
    00000001h.  Additionally, the Optional Header does not contain Hash
    Version and Secure Hash of Payload fields.

    Format (Version = 00000002h|00000003h|00000004h|00000005h):
    +-----------+------------+----------+----------+----------+
    | Type      | Version    | Hdr Size | Pay Size | Trl Size \
    | 00000004h | see Note A | Varies   |  Varies  | Varies   /
    +-----------+------------+----------+----------+----------+

    +-------------+-----------------+---------+-------------+
    \ Secure Hash | Object Metadata | Payload | Trailer     |
    / see Note A  |  XML formatted  | Varies  | see Notes B |
    +-------------+-----------------+---------+-------------+

    Purpose:  This header is used to secure both the metadata for the
    object or objects contained in the payload section using a single
    Secure Hash.  This Header Type offers an alternative method for
    securing the payload and its metadata.  This is the preferred header
    to be used if no additional wrappers are going to be applied to this
    object.

    Notes:
     A - The application of a secure hash of the payload is defined in
         Section 4.1 of this RFC.

     B - The Trailer will be filled with '0' as padding to supply the
         secure hashing solution with the correct amount of information
         to hash.
```

```
3.5 Type 5 Header - Grouping Header

    Type = 00000005h
    Title: Grouping
    Version = 00000001h|00000002h|00000003h|00000004h|00000005h

    Description: This wrapper is used to collect a group of 2 or more
    objects that can be or need to be treated as a single object.  This
    wrapper does not allow for metadata to be applied as part of the
    grouping.  In cases where a single set of metadata can be applied to
    every object in the payload, a Type 6 wrapper should be applied.  Type
    5 wrappers should not be used to concatenate multiple raw objects.
    Rather, each object should first be wrappered with a separate Type 1
    wrapper.

    Format: (Version = 00000001h)
    +-----------+-----------+------------+------------+----------+
    | Type      | Version   | Hdr Size   | Pay Size   | Trl Size \
    | 00000005h | 00000001h | see Note A | see Note B | 00000000 /
    +-----------+-----------+------------+------------+----------+

    +---------------+
    \ No of Payload \
    / see Note C    /
    +---------------+

    +-------------+-------------+ ~ +-------------+
    \ Payl 1 Size | Payl 2 Size | ~ | Payl n Size \
    / see Note D  | see Note D  | ~ | see Note D  /
    +-------------+-------------+ ~ +-------------+

    +-----------+-----------+ ~ +-----------+---------+
    \ Payload 1 | Payload 2 | ~ | Payload n | Trailer |
    /  Varies   |  Varies   | ~ |  Varies   |   N/A   |
    +-----------+-----------+ ~ +-----------+---------+

    Format: (Version = 00000002h|00000003h|00000004h|00000005h)
    +-----------+-------------+------------+------------+----------+
    | Type      | Version     | Hdr Size   | Pay Size   | Trl Size \
    | 00000005h | see Notes A | see Note A | see Note B | 00000000 /
    +-----------+-------------+------------+------------+----------+

    +-------------+---------------+
    \ Secure Hash | No of Payload \
    / see Note A  | see Note C    /
    +-------------+---------------+

    +----------------+--------------+--------------+
    \ Payload 1 Size | Hash Version | Secure Hash  \
    / see Note D     | of Payload 1 | of Payload 1 /
```

```
+----------------+--------------+--------------+

+----------------+--------------+--------------+
\ Payload 2 Size | Hash Version | Secure Hash  \
/ see Note D     | of Payload 2 | of Payload 2 /
+----------------+--------------+--------------+

(repeat through Payload n)

+----------------+--------------+--------------+
\ Payload n Size | Hash Version | Secure Hash  \
/ see Note D     | of Payload n | of Payload n /
+----------------+--------------+--------------+


+-----------+-----------+ ~ +-----------+------------+
\ Payload 1 | Payload 2 | ~ | Payload n | Trailer    |
/  Varies   |  Varies   | ~ |  Varies   | see Note E |
+-----------+-----------+ ~ +-----------+------------+
```

Purpose: This wrapper is used to collect two or more objects and have them treated as a single entity by the outer most SFF wrappers. While this wrapper is used to capture multiple objects in its payload, it does not distunish between them. All objects are treated identically.

Notes:

A - The application of a secure hash of the payload is defined in Section 4.1 of this RFC.

B - The Payload Size represents the summation of all Payload subfields. (The Payload subfields include the No of Payload field, Payload x Size fields, and the Payload x fields.)

C - 32 Bit field which contains the number of separate payloads

D - 64 Bit field denoting the size of the payload

E - The Trailer will be filled with '0' as padding to supply the secure hashing solution with the correct amount of information to hash.

3.6 Type 6 Header - Grouping Header with Metadata

Type = 00000006h
Title: Grouping
Version = 00000001h|00000002h|00000003h|00000004h|00000005h

Description: This wrapper is used to collect a group of 2 or more objects that can be or need to be treated as a single object. This wrapper allows a single set of metadata to be applied as part of the grouping. Like Type 5 wrappers, Type 6 wrappers should not be used to concatenate multiple raw objects. Rather, each object should first be

```
wrappered with a separate Type 1 wrapper.  As with a Type 2 (File
Metadata) header, the metadata is stored as an XML object.  XML
objects have been defined by several other RFCs [RFC2396] and the W3C.
The current are defined by  [W3C.REC-xml]. This RFC does not impose
any restrictions of the format or content of the XML Object.  This is
done to allow for the widest range of formats of metadata without
having to define a new header for each format of metadata.

Format: (Version = 00000001h)
+-----------+-----------+------------+------------+----------+
| Type      | Version   | Hdr Size   | Pay Size   | Trl Size \
| 00000006h | 00000001h | see Note A | see Note B | 00000000 /
+-----------+-----------+------------+------------+----------+

+-----------------+---------------+
\ Object Metadata | No of Payload \
/  XML formatted  | see Note C    /
+-------------+-------------------+

+-------------+-------------+ ~ +-------------+
\ Payl 1 Size | Payl 2 Size | ~ | Payl n Size \
/ see Note D  | see Note D  | ~ | see Note D  /
+-------------+-------------+ ~ +-------------+

+-----------+-----------+ ~ +-----------+---------+
\ Payload 1 | Payload 2 | ~ | Payload n | Trailer |
/  Varies   |  Varies   | ~ |  Varies   |   N/A   |
+-----------+-----------+ ~ +-----------+---------+

Format: (Version = 00000002h|00000003h|00000004h|00000005h)
+-----------+-------------+------------+------------+----------+
| Type      | Version     | Hdr Size   | Pay Size   | Trl Size \
| 00000006h | see Notes A | see Note A | see Note B | 00000000 /
+-----------+-------------+------------+------------+----------+

+-------------+-----------------+---------------+
\ Secure Hash | Object Metadata | No of Payload \
/ see Note A  | XML Formatted   | see Note C    /
+-------------+-----------------+---------------+

+----------------+--------------+--------------+
\ Payload 1 Size | Hash Version | Secure Hash  \
/ see Note D     | of Payload 1 | of Payload 1 /
+----------------+--------------+--------------+

+----------------+--------------+--------------+
\ Payload 2 Size | Hash Version | Secure Hash  \
/ see Note D     | of Payload 2 | of Payload 2 /
+----------------+--------------+--------------+

(repeat through Payload n)
```

```
+----------------+-------------+--------------+
\ Payload n Size | Hash Version | Secure Hash  \
/ see Note D     | of Payload n | of Payload n /
+----------------+-------------+--------------+

+-----------+-----------+ ~ +-----------+---------+
\ Payload 1 | Payload 2 | ~ | Payload n | Trailer |
/  Varies   |  Varies   | ~ |  Varies   |   N/A   |
+-----------+-----------+ ~ +-----------+---------+
```

```
Purpose:  Purpose:  This wrapper is used to capture information that
is common to all objects in the payload.  Complex directory structure
can be constructed by using multiple Directory and Directory with
Payload Hash wrappers along with Grouping and Grouping with Payload
Hash wrappers.  While this wrapper is used to capture multiple objects
in its payload, it does not distunish between them.  All objects are
treated identically.

Notes:
  A - The application of a secure hash of the payload is defined in
      Section 4.1 of this RFC.

Notes:
 A - The application of a secure hash of the payload is defined in
      Section 4.1 of this RFC.

 B - The Trailer will be filled with '0' as padding to supply the
      secure hashing solution with the correct amount of information
      to hash.

  B - The Payload Size represents the summation of all Payload
subfields.  (The Payload subfields include the No of Payload
field, Payload x Size fields, and the Payload x fields.)

  C - 32 Bit field which contains the number of separate payloads

  D - 64 Bit field denoting the size of the payload
```

3.7 Type 7 Header - FIPS 188 Data Labels

```
The National Institute of Standards and Technology (NIST) has
published a standard for providing data labeling for Internet
connections.

Type = 00000007h
Title: FIPS 188 Header
Version = 00000001h|00000002h|00000003h|00000004h|00000005h

Description: This wrapper adds a NIST FIPS 188 Header to the included
payload [ FIPS 188].  FIPS 188 or CIPSO labels are used in providing
Network level Mandatory Access Control.  In an IPv4 network, the FIPS
188 is inserted into the IP Options field of the IP Header [RFC 791]
```

```
   For this wrapper, a simple CRC-32/64 checksum field is used to bind
   the FIPS 188 header field to the Payload.

   Format: (Version = 00000001h)
   +-----------+-----------+----------+----------+----------+
   | Type      | Version   | Hdr Size | Pay Size | Trl Size \
   | 00000007h | 00000001h | Varies   |  Varies  | Varies   /
   +-----------+-----------+----------+----------+----------+

   +-------------+---------+------------+
   \ FIPS 188    | Payload | Trailer    |
   / see Note C  | Varies  | see Note B |
   +-------------+---------+------------+

   Format: (Version = 00000002h|00000003h|00000004h|00000005h)
   +-----------+-------------+----------+----------+----------+
   | Type      | Vers        | Hdr Size | Pay Size | Trl Size \
   | 00000008h | see Notes A | Varies   |  Varies  | Varies   /
   +-----------+-------------+----------+----------+----------+

   +-------------+-------------+---------+------------+
   \ Secure Hash | FIPS 188    | Payload | Trailer    |
   / see Note A  | see Note C  | Varies  | see Note B |
   +-------------+-------------+---------+------------+

   Purpose:  This header is used to attach a NIST FIPS 188 formatted
   label to an object.

   Notes:
   A - The application of a secure hash of the payload is defined in
        Section 4.1 of this RFC.

   B - The Trailer will be filled with '0' as padding to supply the
        secure hashing solution with the correct amount of information
        to hash.

   C - Format Defined by NIST FIPS 188 Documentation

3.8 Type 8 Header - FIPS 188 Data Labels With Payload Hash

   The National Institute of Standards and Technology (NIST) has
   published a standard for providing data labeling for Internet
   connections.

   Type = 00000008h
   Title: FIPS 188 Header
   Version = 00000001h|00000002h|00000003h|00000004h|00000005h

   Description: This wrapper adds a NIST FIPS 188 Header to the included
   payload [FIPS 188].  FIPS 188 or CIPSO labels are used in providing
   Network level Mandatory Access Control.  In an IPv4 network, the FIPS
   188 is inserted into the IP Options field of the IP Header [RFC 791]
```

```
    For this wrapper, a simple CRC-32/64 checksum field is used to bind
    the FIPS 188 header field to the Payload.

    Format: (Version = 00000001h)
    +-----------+-------------+----------+----------+----------+
    | Type      | Vers        | Hdr Size | Pay Size | Trl Size \
    | 00000008h | see Notes A | Varies   |  Varies  | Varies   /
    +-----------+-------------+----------+----------+----------+

    +-------------+-------------+-------------+
    \ FIPS 188    | Hash Version | Secure Hash \
    / see Note C  | of Payload   | of Payload  /
    +-------------+-------------+-------------+

    +---------+-----------+
    \ Payload | Trailer    |
    / Varies  | see Note B |
    +---------+-----------+

    Format: (Version = 00000002h|00000003h|00000004h|00000005h)
    +-----------+-------------+----------+----------+----------+
    | Type      | Vers        | Hdr Size | Pay Size | Trl Size \
    | 00000008h | see Notes A | Varies   |  Varies  | Varies   /
    +-----------+-------------+----------+----------+----------+

    +-------------+------------+--------------+-------------+
    \ Secure Hash | FIPS 188    |  Hash Version | Secure Hash \
    / see Note A  | see Note C  | of Payload    | of Payload  /
    +-------------+------------+--------------+-------------+

    +---------+-----------+
    \ Payload | Trailer    |
    / Varies  | see Note B |
    +---------+-----------+

    Purpose:  This header is used to attach a NIST FIPS 188 formatted
    label to an object.

    Notes:
     A - The application of a secure hash of the payload is defined in
         Section 4.1 of this RFC.

     B - The Trailer will be filled with '0' as padding to supply the
         secure hashing solution with the correct amount of information
         to hash.

     C - Format Defined by NIST FIPS 188 Documentation

3.9 Type 9 - Encoding

    Type = 00000009h
    Title: Encoding
```

```
Version = 00000001h

Description:  This unit wraps the payload, which has been encoded.

Format: (Version = 00000001h)
+-----------+-------------+----------+----------+----------+
| Type      | Vers        | Hdr Size | Pay Size | Trl Size \
| 00000008h | see Notes A | Varies   |  Varies  | Varies   /
+-----------+-------------+----------+----------+----------+

+-------------+--------------+-------------+
\ FIPS 188    | Hash Version | Secure Hash \
/ see Note C  | of Payload   | of Payload  /
+-------------+--------------+-------------+

+---------+------------+
\ Payload | Trailer    |
/ Varies  | see Note B |
+---------+------------+

Format: (Version = 00000002h|00000003h|00000004h|00000005h)
+-----------+-------------+----------+----------+----------+
| Type      | Vers        | Hdr Size | Pay Size | Trl Size \
| 00000008h | see Notes A | Varies   |  Varies  | Varies   /
+-----------+-------------+----------+----------+----------+

+-------------+-------------+---------------+-------------+
\ Secure Hash | FIPS 188    |  Hash Version | Secure Hash \
/ see Note A  | see Note C  | of Payload    | of Payload  /
+-------------+-------------+---------------+-------------+

+---------+------------+
\ Payload | Trailer    |
/ Varies  | see Note B |
+---------+------------+

Purpose:  This header is used to attach a NIST FIPS 188 formatted
label to an object.

Notes:
 A - The application of a secure hash of the payload is defined in
     Section 4.1 of this RFC.

 B - The Trailer will be filled with '0' as padding to supply the
     secure hashing solution with the correct amount of information
     to hash.

 C - Format Defined by NIST FIPS 188 Documentation
Purpose:  This header is used to encode the payload.  By encoding the
payload using the defined encoding schema, limitations in transport
mechanisms can be overcome (For example, SMTP can only handle 7-bit
    ASCII.  By using base64 encoding, secure file format objects can be
    sent by an SMTP compliant email handler without loss of functionality.)


    Notes:
     A - The encoding of the payload is defined in Section 4.2 of this
         RFC.

    Notes:
     A - The application of a secure hash of the payload is defined in
         Section 4.1 of this RFC.

     B - The Trailer will be filled with '0' as padding to supply the
         secure hashing solution with the correct amount of information
         to hash.

      B - The format of the Trailer section of this Type wrapper is
    defined         in Section 4.2 of this RFC.

3.10 Type 10 Header - Encryption

    Type = 0000000Ah
    Title: Mime-Type
    Version = 00000001h

    Description:  This wrapper encrypts the payload using the encryption
    algorithm defined in the Version field.

    Format: (Version = 00000001h)
    +-----------+-------------+----------+----------+----------+
    | Type      | Vers        | Hdr Size | Pay Size | Trl Size \
    | 00000008h | see Notes A | Varies   |  Varies  | Varies   /
    +-----------+-------------+----------+----------+----------+

    +-------------+-------------+-------------+
    \ FIPS 188    | Hash Version | Secure Hash \
    / see Note C  | of Payload   | of Payload  /
    +-------------+-------------+-------------+

    +---------+------------+
    \ Payload | Trailer    |
    / Varies  | see Note B |
    +---------+------------+

    Format: (Version = 00000002h|00000003h|00000004h|00000005h)
    +-----------+-------------+----------+----------+----------+
    | Type      | Vers        | Hdr Size | Pay Size | Trl Size \
    | 00000008h | see Notes A | Varies   |  Varies  | Varies   /
    +-----------+-------------+----------+----------+----------+

    +-------------+-------------+---------------+-------------+
    \ Secure Hash | FIPS 188    |  Hash Version | Secure Hash \
    / see Note A  | see Note C  | of Payload    | of Payload  /
    +-------------+-------------+---------------+-------------+

    +---------+------------+
    \ Payload | Trailer    |
    / Varies  | see Note B |
    +---------+------------+

    Purpose:  This header is used to attach a NIST FIPS 188 formatted
    label to an object.

    Notes:
     A - The application of a secure hash of the payload is defined in
         Section 4.1 of this RFC.

     B - The Trailer will be filled with '0' as padding to supply the
         secure hashing solution with the correct amount of information
         to hash.

     C - Format Defined by NIST FIPS 188 Documentation
    Purpose:  This header is used to encrypt its payload.

3.11 Type 11 Header - Audit Trail

    Type = 0000000Bh
    Title: Audit_Trail
    Version = 00000001h

    Description:

    Format: (Version = 00000001h)
    +-----------+-------------+----------+----------+----------+
    | Type      | Vers        | Hdr Size | Pay Size | Trl Size \
    | 00000008h | see Notes A | Varies   |  Varies  | Varies   /
    +-----------+-------------+----------+----------+----------+

    +-------------+--------------+-------------+
    \ FIPS 188    | Hash Version | Secure Hash \
    / see Note C  | of Payload   | of Payload  /
    +-------------+--------------+-------------+

    +---------+------------+
    \ Payload | Trailer    |
    / Varies  | see Note B |
    +---------+------------+

    Format: (Version = 00000002h|00000003h|00000004h|00000005h)
    +-----------+-------------+----------+----------+----------+
    | Type      | Vers        | Hdr Size | Pay Size | Trl Size \
    | 00000008h | see Notes A | Varies   |  Varies  | Varies   /
    +-----------+-------------+----------+----------+----------+
    +-------------+-------------+---------------+-------------+
    \ Secure Hash | FIPS 188    |  Hash Version | Secure Hash \
    / see Note A  | see Note C  | of Payload    | of Payload  /
    +-------------+-------------+---------------+-------------+

    +---------+------------+
    \ Payload | Trailer    |
    / Varies  | see Note B |
    +---------+------------+
```

    Purpose: This header is used to attach a NIST FIPS 188 formatted
    label to an object.

    Notes:
     A - The application of a secure hash of the payload is defined in
         Section 4.1 of this RFC.

     B - The Trailer will be filled with '0' as padding to supply the
         secure hashing solution with the correct amount of information
         to hash.

     C - Format Defined by NIST FIPS 188 Documentation
    Purpose:

3.12 Type 12 Header - XML Security Header with Payload Hash

Type = 0000000Ch
Title: XML Security with Payload Hash
Version = 00000001h|00000002h|00000003h|00000004h|00000005h

Description: This wrapper captures the metadata, which is related to the raw object that is contained in the payload section of this header. As with a Type 2 (File Metadata) header, the metadata is stored as an XML object. XML objects have been defined by several other RFCs [RFC2396] and the W3C. The current are defined by [W3C.REC-xml]. This RFC does not impose any restrictions of the format or content of the XML Object. This is done to allow for the widest range of formats of metadata without having to define a new header for each format of metadata. The wrapper differs from a Type 2 (File Metadata) wrapper in that the Version from the Mandatory header and Secure Hash field from the Optional Header of the Payload are included in the Optional Header of the wrapper. This wrapper should only be used with payloads that contain a secure hash field that conform to the specification in section 4.1 of this RFC. If the payload does not a Secure Hash field as part of the Optional header, then the Secure Hash of the Payload field is omitted. When a Secure Hash is part of the Optional Header, it is only applicable to the Object Metadata field, the Version field (from the Payload), and the Secure Hash field (from the Payload). For those instances, where the Metadata and the Object need to be securely hashed using a single hash, a Type 4 (Secure Hash) wrapper should be applied. Type 3 Headers are the

```
    preferred approach for securely attaching metadata to a Payload when
    the wrappers are being attached throughout the secured data file's
    lifecycle.

    Format (Version = 00000001h):
    +-----------+-----------+----------+----------+-----------+
    | Type      | Version   | Hdr Size | Pay Size | Trl Size  \
    | 0000000Ch | 00000001h |  Varies  |  Varies  | 00000000h /
    +-----------+-----------+----------+----------+-----------+

    +-----------------+--------------+-------------+---------+---------+
    \ Object Metadata | Hash Version | Secure Hash | Payload | Trailer |
    /  XML formatted  | of Payload   | of Payload  |  Varies |   N/A   |
    +-----------------+--------------+-------------+---------+---------+

    Format (Version = 00000002h|00000003h|00000004h|00000005h):
    +-----------+------------+----------+----------+----------+
    | Type      | Version    | Hdr Size | Pay Size | Trl Size \
    | 00000003h | see Note A | Varies   |  Varies  | Varies   /
    +-----------+------------+----------+----------+----------+

    +-------------+-----------------+--------------+--------------+
    \ Secure Hash | Object Metadata | Hash Version | Secure Hash  \
    / see Note A  |  XML formatted  | of Payload   | of Payload   /
    +-------------+-----------------+--------------+--------------+

    +---------+------------+
    \ Payload | Trailer    |
    / Varies  | see Note B |
    +---------+------------+

    Purpose:  This header is used to capture the metadata for the object
    or objects contained in the payload section.  Additionally, by
    including the Secure Hash from the Payload, the secured contents of
    the Payload are cryptographically linked with the Object's Metadata.

    Notes:
     A - The application of a secure hash of the payload is defined in
         Section 4.1 of this RFC.

     B - The Trailer will be filled with '0' as padding to supply the
         secure hashing solution with the correct amount of information
         to hash.

3.13 Type 13 Header - Digital Signature - X.509V3 (XML Encoded)

    Type = 0000000Dh
    Title: Digital Signature - X.509V3
    Version = 00000001h|00000002h|00000003h|00000004h|00000005h
```

```
Description:  The X509Certificate element is used to validate the
Signature.

Format: (Version = 00000001h)
+-----------+-------------+----------+----------+----------+
| Type      | Vers        | Hdr Size | Pay Size | Trl Size \
| 00000008h | see Notes A | Varies   |  Varies  | Varies   /
+-----------+-------------+----------+----------+----------+

+-------------+--------------+-------------+
\ FIPS 188    | Hash Version | Secure Hash \
/ see Note C  | of Payload   | of Payload  /
+-------------+--------------+-------------+

+---------+------------+
\ Payload | Trailer    |
/ Varies  | see Note B |
+---------+------------+

Format: (Version = 00000002h|00000003h|00000004h|00000005h)
+-----------+-------------+----------+----------+----------+
| Type      | Vers        | Hdr Size | Pay Size | Trl Size \
| 00000008h | see Notes A | Varies   |  Varies  | Varies   /
+-----------+-------------+----------+----------+----------+

+-------------+------------+---------------+-------------+
\ Secure Hash | FIPS 188    |  Hash Version | Secure Hash \
/ see Note A  | see Note C  | of Payload    | of Payload  /
+-------------+------------+---------------+-------------+

+---------+------------+
\ Payload | Trailer    |
/ Varies  | see Note B |
+---------+------------+

Purpose:  This header is used to attach a NIST FIPS 188 formatted
label to an object.

Notes:
 A - The application of a secure hash of the payload is defined in
     Section 4.1 of this RFC.

 B - The Trailer will be filled with '0' as padding to supply the
     secure hashing solution with the correct amount of information
     to hash.

 C - Format Defined by NIST FIPS 188 Documentation


Purpose:  This header is used to digitally sign an object with a
X.509V3 formatted certificate.  This header is used when the identity
of the signer can be validated using an Public Key Infrastructure.
```

Notes:
 A – The application of a secure hash of the payload is defined in Section 4.1 of this RFC.

 B – The format of the X509V3 Cert is defined by RFC 3075.

4.0 Common Schemas

The following section details the schemas, which have been incorporated into more than one Wrapper Type.

4.1 Secure Hashes

The Secure File Format allows for different hashing solutions and different length hashes or “message digests”, to be used to secure object metadata carried in the Optional Header fields, the Payload, and combinations of these two. There are four versions of secure hashing allowed by the Secure File Format standard. All of the four are utilize the Secure Hash Algorithm (SHA). The Secure Hash Algorithm is defined by the Federal Information Processing Standard (FIPS) 180-3, which is published by the National Institute of Standards and Technology (NIST). Table 4.1 details the defined version values for wrapper types that use secure hashing. These version values are also used by the Hash Version for Payload field in the Optional Header. The Secure Hash field contains a 'numerical' values and is subject to the 8-bit 'rule” defined in section 2.0. For algorithms that require padding to meet the field size requirements defined in Section 2.0, prepended with bit values of '0' followed by the Message Digest.

Version	Algorithm	S/H	M/D	PAD	Comments
00000001h	N/A	N/A	N/A	N/A	No Hashing is used
00000002h	SHA-1	20	160	0	see NIST FIPS 180-3
00000003h	SHA-256	32	256	0	see NIST FIPS 180-3
00000004h	SHA-384	48	384	0	see NIST FIPS 180-3
00000005h	SHA-512	64	512	0	see NIST FIPS 180-3

Table 4.1 – Valid Secure Hashing Schemas

where S/H – Secure Hash Field Size in bytes
      M/D – Message Digest Size in bits
      PAD – Number of '0' bits prepended to the Message Digest

The Secure Hash header composed of a required field and an additional optional field. The first (required) field, called the "Checksum Schema Field" is a 4-byte ASCII field representing a 16-bit hexadecimal field. Each value in this field denotes a different secure hashing schema. If the secure hashing schema identified by this first field requires additional information, that additional information is included in a second (optional) field. The second field, which is present only when needed, is managed as an XML object. Table 3.12.1 lists the acceptable values for the Checksum Schema field and the XML DTD for the second field, if present. Additionally, this table identifies the value to be used for populating the "Trailer Size" field for each secure hashing schema.

4.2 Encoding

There is currently one version for encoding. Encoding Schemas are not defined in the RFC, but rather are referenced. The encoding type has two formats. The first does not include option header fields, the second includes provisions for optional header fields, if required by the encoding schema. For those versions that require additional information, that information is stored in a XML object defined by a DTD. Table 4.2 details the possible values for the Version field.

Version	DTD	Comments
00000000h	N/A	Invalid
00000001h	N/A	Base64 Encoding (defined by RFC 2045)

Table 4.2 – Possible Encoding Schemas

4.3 Encryption Schemas

Encryption Schemas are not defined at this time.

5.0 Rules for New Header Types

Type = xxxxxxxxh
Title: Mime-Type
Version = 00000001h

Description:
A textual description of how this type of header is added here.

Format:

```
+-----------+-----------+----------+----------+----------+
    | Type      | Vers      | Hdr Size | Pay Size | Trl Size \
    | xxxxxxxxh | 000000xxh |  Varies  |  Varies  |  Varies  /
    +-----------+-----------+----------+----------+----------+

    +-----------------------------------------------------+
    \ Optinal Header Fields are added here.               \
    / When A Secure Hash is used, it is the first field /
    +-----------------------------------------------------+

    +---------+---------+
    \ Payload | Trailer |
    / Varies  | Varies  |
    +---------+---------+

    Purpose:
    A textual description of this type of header's purpose is added here.
```

References

```
    [FIPS 180] NIST FIPS 180-3 Secure Hash Standard (SHS)
               <http://csrc.nist.gov/publications/fips/fips180-3/
               fips180-3_final.pdf>

    [FIPS 186] NIST FIPS 186-3 Digital Signature Standard (DSS)
               <http://csrc.nist.gov/publications/fips/fips186-3/
               fips_186-3.pdf>

    [FIPS 188] NIST FIPS 188 Standard Security Label for Information
               Transfer
               <http://csrc.nist.gov/publications/fips/fips188/
               fips188.pdf>

    [RFC791]   Internet Protocol
               <http://www.ietf.org/rfc/rfc791.txt>

    [RFC2045]  Multipurpose Internet Mail Extensions (MIME) Part One:
               Format of Internet Message Bodies
               <http://www.ietf.org/rfc/rfc2045.txt>

    [RFC3075] Uniform Resource Identifiers (URI): Generic Syntax
              <http://www.ietf.org/rfc/rfc2396.txt>

    [RFC2396]  Berners-Lee, T., Fielding, R. and L. Masinter,
               "Uniform Resource Identifiers (URI): Generic
               Syntax", RFC 2396, August 1998.

    [W3C.REC-xml] Bray, T., Paoli, J., Sperberg-McQueen, C. and
                  E. Maler, "Extensible Markup Language (XML) 1.0
                  (2nd ed)", W3C REC-xml, October 2000,
                  <http://www.w3.org/TR/REC-xml>.
```

Security Considerations

Author's Address

Thomas Rozenbroek
7370 Quaking Drive
Sunerland, MD 20689

Phone: 703 625-2985
Email: trozenbr@gmu.edu

Full Copyright Statement

REFERENCES

REFERENCES

[1] Harris, Shon, *CISSP All-in-One Exam Guide*, 5th ed. New York: McGraw-Hill Books, 2010.

[2] ASD(P), “DoDD 3600.01 Information Operations.” [Online]. Available: http://www.dtic.mil/whs/directives/corres/pdf/360001p.pdf. [Accessed: 02-Oct-2011].

[3] R. Farrow, *Unix System Security*, First ed. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc, 1991.

[4] J. Postel and J. Reynolds, “RFC 959 - File Transfer Protocol (FTP).” [Online]. Available: http://www.ietf.org/rfc/rfc959.txt. [Accessed: 21-Nov-2011].

[5] Denning, Dorothy E., “A Lattice Model of Secure Information Flow.” [Online]. Available: http://www.cs.georgetown.edu/~denning/infosec/lattice76.pdf. [Accessed: 25-Jun-2011].

[6] Bell, D. E.; La Padula, L. J., “Secure Comptuer System: Unified Exposition and Multics Interpretation,” Mar-1976. [Online]. Available: http://csrc.nist.gov/publications/history/bell76.pdf. [Accessed: 23-Oct-2011].

[7] Biba, K.J., “Integrity Considerations for Secure Computer Systems,” Apr-1997. [Online]. Available: http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA039324&Location=U2&doc=GetTRDoc.pdf. [Accessed: 16-Oct-2011].

[8] D. F. C. Brewer and M. J. Nash, “The Chinese Wall Security Policy.” [Online]. Available: http://www.cs.purdue.edu/homes/ninghui/readings/AccessControl/brewer_nash_89.pdf. [Accessed: 31-Oct-2011].

[9] “DoD 5200.28-STD Department of Defense Trusted Computer System Evaluation Criteria.” [Online]. Available: http://csrc.nist.gov/publications/history/dod85.pdf. [Accessed: 29-Mar-2011].

[10] NCSC, “Trusted Network Interpretation,” 31-Jul-1987. [Online]. Available: http://csrc.nist.gov/publications/secpubs/rainbow/tg005.txt. [Accessed: 24-Oct-2011].

[11]“RFC 1108 - U.S. Department of Defense Security Options for the Internet Protocol.” [Online]. Available: http://tools.ietf.org/html/rfc1108. [Accessed: 26-Jun-2011].

[12]“RFC 1038 - Draft revised IP security option.” [Online]. Available: http://tools.ietf.org/html/rfc1038. [Accessed: 31-Oct-2011].

[13] Mager, Alan, “Investigation of Technologies and Techniques for Labelling Information Objects to Support Access Management.” [Online]. Available: http://pubs.drdc.gc.ca/PDFS/unc43/p524601.pdf. [Accessed: 25-Jun-2011].

[14]“RFC 1296 - Internet Growth (1981-1991).” [Online]. Available: http://tools.ietf.org/html/rfc1296. [Accessed: 26-Jun-2011].

[15] Defense Communications Agency, *DDN Directory*. DDN Network Information Center, 1984.

[16] Comer, Douglas, *Internetworking with TCP/IP*, First ed. Englewood Cliffs, New Jersey: Prentice Hall, 1988.

[17] Defense Communications Agency, “MIL STD 1777,” in *DDN Protocol Handbook*, vol. 1, 3 vols., Menlo Park, California: DDN Network Information Center, 1985.

[18] Kahn, David, *The Codebreakers*. New York: The MacMillan Company, 1967.

[19]“Security Classification of Information, volume 1 (Quist), Chapter Two.” [Online]. Available: http://www.fas.org/sgp/library/quist/chap_2.html. [Accessed: 26-Jun-2011].

[20]“Security Classification of Information, volume 1 (Quist), Chapter One.” [Online]. Available: http://www.fas.org/sgp/library/quist/chap_1.html. [Accessed: 26-Jun-2011].

[21]“Executive Order 12958 - Classified National Security Information,” 20-Apr-1995. [Online]. Available: http://www.dtic.mil/dtic/pdf/customer/STINFOdata/EO_12958.pdf. [Accessed: 20-Oct-2011].

[22] G. W. Bush, “Executive Order 13292—Further Amendment to Executive Order 12958, as Amended, Classified National Security Information,” 28-Mar-2003. [Online]. Available: http://edocket.access.gpo.gov/2003/pdf/03-7736.pdf. [Accessed: 21-Nov-2011].

[23]“DoD 5200.1-PH DoD Guide to Marking Classified Documents.” [Online]. Available: http://www.dtic.mil/dtic/pdf/customer/STINFOdata/DoD5200_1ph.pdf. [Accessed: 28-Feb-2011].

[24] ASD(C4I), “DOD 5200.1-R Information Security Program,” Jan-1997. [Online]. Available: http://www.fas.org/irp/doddir/dod/5200-1r/. [Accessed: 26-Jun-2011].

[25]“Re aggregation | Define Re aggregation at Dictionary.com.” [Online]. Available: http://dictionary.reference.com/browse/re+aggregation. [Accessed: 16-Oct-2011].

[26] Quist, Arvin S., “Classification of Compilations of Information,” Jun-1991. [Online]. Available: http://www.fas.org/sgp/library/compilations.pdf. [Accessed: 26-Jun-2011].

[27] DOE OCIO Director’s Office, “DOE OCIO May 2005 Comunique,” May-2005. [Online]. Available: http://www.hss.doe.gov/classification/news/CommuniQue200505.pdf. [Accessed: 16-Oct-2011].

[28]“DoDI 8500.2 Information Assurance (IA) Implementation.” [Online]. Available: http://www.dtic.mil/whs/directives/corres/pdf/850002p.pdf. [Accessed: 14-Oct-2011].

[29]“Security Classification of Information, volume 2 (Quist), Chapter Ten.” [Online]. Available: http://www.fas.org/sgp/library/quist2/chap_10.html. [Accessed: 26-Jun-2011].

[30]“NRC: Glossary -- Critical mass.” [Online]. Available: http://www.nrc.gov/reading-rm/basic-ref/glossary/critical-mass.html. [Accessed: 13-Jul-2011].

[31] JTF - Transformation Initiative, “NIST Special Publication 800-53 Rev 3 - Recommended Security Controls for Federal Information Systems and Organizations,” Aug-2009. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-53-Rev3/sp800-53-rev3-final_updated-errata_05-01-2010.pdf. [Accessed: 20-Oct-2011].

[32] ASD(NII)/DoD CIO, “DoDD 8500.01E Information Assurance (IA).” [Online]. Available: http://www.dtic.mil/whs/directives/corres/pdf/850001p.pdf. [Accessed: 14-Oct-2011].

[33] ASD(NII)/DoD CIO, “DODD 8570.01 Information Assurance Training, Certification, and Workforce Management,” 15-Aug-2004. [Online]. Available: http://www.dtic.mil/whs/directives/corres/pdf/857001p.pdf. [Accessed: 14-Oct-2011].

[34] Rozenbroek, Thomas, “An Architecture for Managing Access to and Permission for Multiple Versions of Objects in a Distributed Environment: modeling and analysis,” in *Proceedings of the Twelfth International Conference on Telecommunications Systems*, Monterey, California, 2004.

[35] W3C, “Extensible Markup Language (XML) 1.0 (Fifth Edition).” [Online]. Available: http://www.w3.org/TR/xml/. [Accessed: 09-Mar-2011].

[36]“RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1.” [Online]. Available: http://tools.ietf.org/html/rfc2616. [Accessed: 11-Mar-2011].

[37]“THE PRIVACY ACT OF 1974, 5 U.S.C. § 552a -- As Amended.” [Online]. Available: http://www.justice.gov/opcl/privstat.htm. [Accessed: 20-Oct-2011].

[38] Adams, C, “RFC 2510 - Internet X.509 Public Key Infrastrcture,” Mar-1999. [Online]. Available: http://www.ietf.org/rfc/rfc2510.txt. [Accessed: 20-Oct-2011].

[39]“RFC 4510 - Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map.” [Online]. Available: http://tools.ietf.org/html/rfc4510. [Accessed: 20-Oct-2011].

[40] Carter, Gerald, *LDAP System Adminstration*, First ed. Sebastopol, CA: O’Reilly & Associates, Inc, 2003.

[41] Bovet, Danial P. & Cesati, Marco, *Understanding the Linux Kernel*. Sebastopol, CA: O’Reilly Media, Inc, 2006.

[42]“RFC 1457 - Security Label Framework for the Internet.” [Online]. Available: http://tools.ietf.org/html/rfc1457. [Accessed: 04-Mar-2011].

[43] Anderson, James P., “Computer Security Technology Planning Study ESD-TR-73-51 Vol 1,” Oct-1972. [Online]. Available: http://nob.cs.ucdavis.edu/history/papers/ande72a.pdf. [Accessed: 19-Oct-2011].

[44] ASD C3I, *Department of Defense Trusted Computer Systems Evaluation Criteria*. U. S. Department of Defense, 1985.

[45]“NCSC-TG-004 [Aqua Book] Glossary of Computer Security Terms [Version 1, 10/21/88].” [Online]. Available: http://www.fas.org/irp/nsa/rainbow/tg004.htm. [Accessed: 23-Oct-2011].

[46] Ames, Stanley R.; Gasser, Morrie; Schnell, Roger R., “Security Kernal Design and Implementation: An Introduction,” 1983. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1654439. [Accessed: 23-Oct-2011].

[47]“World Internet Usage Statistics News and World Population Stats.” [Online]. Available: http://www.internetworldstats.com/stats.htm. [Accessed: 24-Oct-2011].

[48]“Internet Systems Consortium | January, 2011 Domain Survey.” [Online]. Available: http://ftp.isc.org/www/survey/reports/2011/01/. [Accessed: 24-Oct-2011].

[49]“RFC 4120 - The Kerberos Network Authentication Service (V5).” [Online]. Available: http://tools.ietf.org/html/rfc4120. [Accessed: 04-Mar-2011].

[50]“RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.” [Online]. Available: http://tools.ietf.org/html/rfc5280. [Accessed: 21-Nov-2011].

[51]“RFC 4880 - OpenPGP Message Format.” [Online]. Available: http://tools.ietf.org/html/rfc4880. [Accessed: 04-Mar-2011].

[52]“GOSAC-N Whitepaper_v2.0_5.25.11.pdf.” [Online]. Available: http://www.gosac-n.org/sites/default/files/GOSAC-N%20Whitepaper_v2.0_5.25.11.pdf. [Accessed: 08-Dec-2011].

[53]“GOSAC-N System Architecture and Design,” *Forge.mil*. [Online]. Available: https://software.forge.mil/sf/docman/do/downloadDocument/projects.gosac_n/docman.root.design/doc11323. [Accessed: 02-Mar-2012].

[54] T. Rozenbroek and E. H. Sibley, “An architecture for Propagating Modifications to Mobile Polices,” in *Proceedings of the Tenth International Conference on Telecommunication Systems*, Monterey, Cal, 2002.

[55] D. E. Eastlake and K. Niles, *Secure XML*. Boston, MA: Pearson Education, Inc, 2003.

[56] National Institute of Standards and Technology (NIST), “NIST FIPS Pub 188 Standard Security Label for Information Transfer,” 06-Sep-1994. [Online]. Available: http://csrc.nist.gov/publications/fips/fips188/fips188.pdf. [Accessed: 26-Jun-2011].

[57] Gary Stoneburner, “NIST Special Publication 800-33 - Underlying Technical Models for Information Technology Security.” [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-33/sp800-33.pdf. [Accessed: 04-Apr-2012].

[58] National Institute of Standards and Technology (NIST), “NIST FIPS PUB 180-3 Secure Hash Standard (SHS),” Oct-2008. [Online]. Available: http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf. [Accessed: 11-Mar-2011].

[59] Ryan, Peter; Schneider, Steve, *Modelling and Analysis of Security Protocols*, First ed. Great Britian: Pearson Education Limited, 2001.

[60]"JSTOR: The American Mathematical Monthly, Vol. 73, No. 4 (Apr., 1966), pp. 385-387." [Online]. Available: http://www.jstor.org.mutex.gmu.edu/stable/2315408. [Accessed: 16-Oct-2011].

[61]"RFC 791 - Internet Protocol." [Online]. Available: http://tools.ietf.org/html/rfc791. [Accessed: 21-Nov-2011].

[62]"RFC 2460 - Internet Protocol, Version 6 (IPv6) Specification." [Online]. Available: http://tools.ietf.org/html/rfc2460. [Accessed: 21-Nov-2011].

[63] Lewine, Donald, *POSIX Programmer's Guide*. Sebastopol, CA: O'Reilly & Associates, Inc, 1991.

CURRICULUM VITAE

Thomas H. Rozenbroek received her Bachelor of Engineering from Stevens Institute of Technology in 1983. He received a Master of Science in Information Systems from Strayer University in 1996 and a Master of Science in Systems Engineering from the Naval Postgraduate School in 2012.