SAMSON Technology Demonstrator Detailed Design Document

Phase IV SD004

Daniel Charlebois
DRDC-CSS

Prepared by: Bell Development Team,
Bell Canada 160 Elgin St. 17th Floor
Ottawa ON. K1S 5N4

CSA: W7714-08FE01

Contract Report
DRDC-RDDC-2013-C6
March 2013

IMPORTANT INFORMATIVE STATEMENTS

The information contained herein is proprietary to Her Majesty and is provided to the recipient on the understanding that it will be used for information and evaluation purposes only. Any commercial use including use for manufacture is prohibited.

Defence Research and Development Canada

SAMSON Technology Demonstrator
Detailed Design Document
Phase IV
SD004

Bell Canada
160 Elgin Street
17th Floor
Ottawa, Ontario
K1S 5N4

August 13, 2013
Revision: 3.2.2 Final

Confidentiality

This document is UNCLASSIFIED.

Disclaimer

The contents of this report do not constitute original work on behalf of the author. A number of sections of the report are comprised of material contributed by multiple authors, most notably members of the Bell SAMSON team.

Authors

Bell Development Team	Role
Glen Henderson	Lead System Architect
Brent Nordin	Intermediate Programmer
Bill Pase	Intermediate Programmer
Dan Seguin	Intermediate Programmer

Bell Q&A Team	Role
Alan Clason	Testing Specialist
William Dziadyk	Certification and Accreditation

Review

DRDC SAMSON Team	Role
Bruce Carruthers	Technical Advisor
Daniel Charlebois	Scientific Authority
Darcy Simmelink	Project Manager

Table of Contents

List of Figures

List of Tables

1.0 Introduction

The Secure Access Management for Single Operational Networks (SAMSON) Technical Demonstrator (TD) implements a new architectural approach to provide *data-centric* information protection in a multiple community network environment. Applications are enabled for SAMSON information protection through the insertion of data protection security services into the applications' data handling routines. The focus of this Defence Research and Development Canada (DRDC) Technical Demonstrator Program (TDP) was to develop a capability that provides separation of Canadian Eyes Only (CEO) and Canadian-US (CANUS) information caveats in a single network environment. However, the SAMSON TD functionality supports the use of a broader set of security attributes so as to provide the needed data-centric protection for a multiple community network environment.

The SAMSON TD system uses cryptographically bound security labels on information assets and Attribute-Based Access Control (ABAC) to enforce the required *need-to-know* security controls. Based on the policies required by the enterprise, the data is enhanced with security metadata including relevant attributes such as: classification, releasability, and membership in communities of interest. The system makes access control decisions by evaluating the attributes of the data object and the attributes of the person subject requesting access to the data. These access control decisions are then enforced at the application level, permitting or denying access to information access in accordance with policy.

The SAMSON TD system enables collapsing network-centric infrastructures and security policy based enclaves. An enterprise infrastructure, with the SAMSON TD incorporated into the architecture, will enforce adherence to policy across all applications and information assets. SAMSON provides the capability to transform an enterprise's multi-level (e.g. different sensitivity classifications, caveat-separations, compartmentalized, releasable to, community-of-interest need-to-know access controls) security architecture from a network-centric (i.e. separate "system high" network segments, complex multi-tier defence-in-depth network architectures, separate operational "enclaves") implementation to a data-centric implementation protection paradigm. The target deployment architectures can range from a small "Entry Level" user community to larger user communities with "High Availability" architectural requirements.

Data-centric security controls provide a foundation for intelligent networking and seamless unified communications using a variety of communications methods where the operational and business requirements may require that sensitive data be made broadly available to end-users in operational environments located beyond the enterprise's physical network infrastructures (e.g. cloud technologies).

The ABAC in the SAMSON TD is instantiated as a Service Oriented Architecture (SOA) and uses eXtensible Messaging and Presence Protocol (XMPP) based messaging services and Policy Enforcement Points (PEPs) to securely and reliably enforce caveat protection of data

using the eXtensible Access Control Mark-up Language (XACML) for policy based authorization.

The SAMSON TD provides the ability to use its information protection mechanisms to self-protect its own administrative interfaces. These self-protection mechanisms include the ability to prevent data assets from unauthorized access or modification. All SAMSON policy-based transactions are recorded in a tamper-resistant audit trail to provide accountability for authorized actions and establish a level of trust and integrity for the system as a whole.

The SAMSON TD relies on six independent core security services that provide:

1. Access to users' identity and associated security attributes;
2. Authorization decisions to allow or prevent transactions on information assets;
3. The creation and retrieval of security attributes on information assets;
4. Cryptographic protection of information assets;
5. Symmetric key management in support of cryptographic operations; and
6. The generation of a tamper-resistant audit trail.

A core concept for the TD is that SAMSON is not a provider of security services; rather, it connects existing security services that are present in the environment to achieve data-centric security across information assets. System architects and security officers are able to leverage their existing investment in security services by allowing them to be used in the SAMSON architectural deployment. It is possible, therefore, to add a SAMSON deployment to an existing network environment as a *security overlay*, that is, a new capability that leverages existing tools and applications.

The following external services were included in the SAMSON TD architectural deployment:

1. Authentication services (Active Directory) to establish each user's identity;
2. Endpoint security labelling of Microsoft Office information assets (Titus);
3. Key escrow services to provide reliable storage of cryptographic keys (StrongAuth);
4. Software cryptographic modules (RSA); and
5. Security information and event management (AlienVault)

It must be noted, however, that SAMSON is not dependent on any of these external services, that is, any listed service could be replaced with a service that provides equivalent functionality. The SAMSON TD is, therefore, a modular architecture.

Data applications, such as file sharing, email, instant messaging, web services, and databases, can be enabled for SAMSON TD data protection through the insertion of PEPs in the applications' data flow and data handling routines. When an information transaction is intercepted by a PEP, the transaction can be made to adhere to policy through calls to the SAMSON security interfaces, specifically, authorization, cryptographic protection and audit.

1.1 History

The Secure Access Management for Secret Operational Networks (SAMSON) technology demonstrator (TD) project originated from a research concept developed at the Network Information Operations (NIO) Section of DRDC (DRDC Ottawa). The NIO section initiated a series of studies that produced a proposal for the creation of a new security architectural approach to provide *data-centric* information protection in a multiple community network environment. Further work developed plans for, and implemented, two Secure Access Management Proof-of-Concept Demonstrators (SAMPOC I in 2002 and SAMPOC II in 2004) based on an initial architectural design.

Although the SAMPOC demonstrations were successful in proving the technical capability of the approach and generated substantial interest within the Department of National Defence (DND), these implementations were not sufficiently robust, secure, or large enough to be considered for operational deployment. The current SAMSON TD project seeks to:

- Address the deficiencies of the SAMPOC I & II demonstrators;
- Extend the capabilities of the original SAMPOC concept to encompass a complete set of information protection services;
- Raise the Technology Readiness Level (TRL) of the SAMSON concept through a more robust implementation of the extended SAMSON design; and
- Highlight, through a series of demonstrations and exercises, the technical challenges, business transformation issues and process changes that would be encountered in transitioning this technology from prototype stage to an operational deployment.

The SAMSON TD target architecture also builds upon the following research initiatives, promoted by DRDC:

- Trusted Audit Collection System (TACS) Design (2004) and Prototype (2005)
- Security Policy Engine Surveys (2002 and 2006)

1.2 Project Scope

The SAMSON TD project calls for the creation of a security enabling infrastructure that leverages, through open protocol standards, six core security services: user attribute management, security labelling of data, authorization, cryptographic services, key management and trusted auditing. A specific set of applications has been enabled for SAMSON data protection through the insertion of information protection security services

into the applications' data handling routines. This set includes representative applications that provide information protection for the following data services:

- File sharing,
- Email,
- Instant messaging,
- Web content delivery,
- Database protection, and
- Real-time command and control (C&C) information feeds.

The following aspects of the SAMSON TD are considered out of scope in terms of their inclusion in this report:

- Certification & Accreditation - Certification and accreditation is not addressed within this document. Readers interested in this subject are encouraged to consult the *SAMSON TD Certification and Accreditation Plan*.

- Complementary Security Services – The SAMSON TD outlines six core security services required for caveat separation. While these security services provide the functionality required for caveat separation, they are not intended to address all aspects of security. Consequently, there are other security services, most notably threat detection and response, which are complementary to the six core SAMSON security services. These complementary security services are considered outside the scope of the report.

The SAMSON TD requirements are defined in the specification document *SAMSON TD Functional Specifications V2.2*. Development activities for the demonstrator were defined across three phases, with a functionally complete system to be delivered at the end of the first phase. The Phase I capability target was extended in February 2010 to include specific enhancements to support participation in the Empire Challenge 2010 exercise. The Phase II capability target was delivered for and demonstrated at two military exercises:

- Empire Challenge 2011 (hosted at both DRDC Shirley's Bay and Fort Huachuca, Arizona); and

- Coalition Warrior Interoperability Demonstration 2011 (CWID).

The Phase III capability target was delivered for and integrated into the operational network for the Coalition Attack Guidance Experiment II (November 2012), again hosted at the Warfare Center at Shirley's Bay, Ottawa.

The documentation set to which this Detailed Design belongs uses as its reference architecture the SAMSON TD deployment that was utilized in the CAGE II operational

SECRET network environment[1]. The requirements set for this reference architecture was a union of:

- A subset of the capabilities defined in the SAMSON TD functional specification that defined the phase II target functionality; and
- Application support, robustness, stability and administration enhancements needed to support CAGE II participation.

This documentation set includes the following:

- SD002 - the Architectural Design Document
- SD004 - the Detailed Design Document (this document)

A complete mapping of the SAMSON technical demonstrator capabilities to the project functional specification is provided in the *Requirements Traceability Matrix* that is included as a supplement to this documentation set.

Those readers interested in a more theoretical discussion of SAMSON design principles are encouraged to consult *Authorization: An Historical Perspective* and *The Bell SAMSON Architecture & Backgrounder*.

1.3 About this Document

This document is meant to serve the basis for understanding the SAMSON architecture in support of ongoing development and deployment activities. Through an detailed description of how the SAMSON infrastructure exchanges security messages, the mechanisms by which security services are connected through the messaging infrastructure and the manner in which policy is enforced within applications, the underlying trust model by which SAMSON achieves its security objectives can be understood. Additionally, through an understanding of the content of this document, security software development teams outside of the technical demonstrator project can create new and complementary SAMSON components.

The primary goal of this document is to describe how the SAMSON TD architecture was designed, configured and deployed in its instantiation at the CAGE II SECRET environment. This instantiation formed the baseline architecture that was evaluated under the project's C&A testing and security assessment methodology. This architectural instantiation of the SAMSON TD is proposed for future deployment to the GoC CSNI SECRET operations network.

[1] Within this document, the "SAMSON TD architectural deployment" represents the instantiation of the SAMSON architecture that was deployed as part of the SAMSON Technical Demonstrator Program as of March 2013. This is separate from "SAMSON" itself, the generalized architectural specification which any COTS or third party solution provider can use to create SAMSON compliant services.

A secondary goal of this document is to specify the open protocol and messaging standards that define the SAMSON architecture and the mechanism by which SAMSON leverages those standards. With this information, independent software initiatives will be able to:

- Replace existing components in the demonstrator with equivalent services that leverage new back end services or leverage the existing back end service in a more appropriate manner for the target environment;
- Create new components that can be accessed through the existing protocol messaging infrastructure and can be leveraged in application information protection processing routines; and
- Add policy-based information protection to new applications.

The target audience for this document is security practitioners with a need to understand the proposed deployment architecture and software development teams that have a requirement to integrate security capabilities in the SAMSON information protection architecture into existing data or security services. It is anticipated that the audience is familiar with: policy-based security, service oriented architectures and security best practices.

2.0 The SAMSON Design Philosophy

The SAMSON design philosophy is best described in terms of:

1. The original stated purpose of the technical demonstrator,
2. The information protection problem that is addressed by the demonstrator and
3. The architectural approach that was taken for the demonstrator by the design team.

Each of these topics is summarized here, but it is recommended that the reader first be familiar with the more detailed section the *SAMSON TD Architectural Design Document* similarly titled: *The SAMSON Design Philosophy*.

The SAMSON concept is for a data-centric security solution. That is, SAMSON is an information protection methodology that binds security policy down to the information asset level. This idea is complementary to the similar concept of "smart data", that is, data that carries with it its own security policy in terms of access and acceptable use restrictions.

This project is examining the concept of data-centric security and its application to create the next generation of secure networks though its ability to address four basic challenges in information assurance.

1. How to enforce a unified and holistic security policy across all information assets.
2. How to restrict the operations that can be performed against information assets to specific, segregated communities.
3. How to ensure that information assets that are released are released only to individuals that have the policy right to access it.
4. How to maintain *chain-of-custody* audit records of this policy-based access control in a tamper-resistant manner.

SAMSON has been designed to address these challenges in the form of a *security overlay*, that is, a set of interconnected services that work through the exchange of messages on top of an existing network deployment. In this way, any network security or application security based environment can be enabled for data-centric protection without the need to remove or de-emphasize existing security protections. As a security overlay, SAMSON uses the pre-existing security software that is present in the environment; it is a secure communications infrastructure that allows many different security solutions to be integrated in comprehensive data-centric security architecture.

Six core security services form the basis of the technical demonstrator architecture, although the modular nature of the design allows for additional services to be added to the SAMSON information protection environment. These core services include: *identity attribute management* to manage user's security attributes, *secure labelling* to manage the security attributes on data assets, *authorization* to make access control decisions based on the security policy, *cryptographic transformation* and *key management* services to protect information assets and *trusted audit* capabilities to maintain a record of all SAMSON policy related transactions.

When viewed in this context, the SAMSON infrastructure components work together to support a defence-in-depth principle: a series of security components that provide a layered approach to protecting information assets.

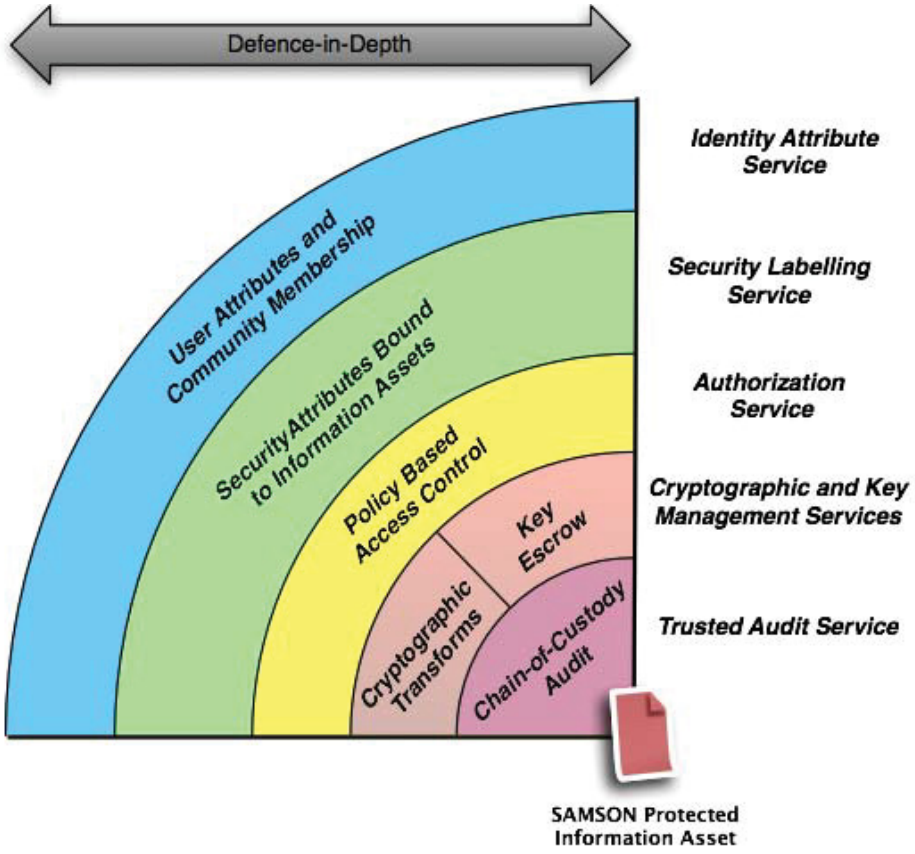


Figure 1: The SAMSON Defence-in-Depth Model

In summary, the SAMSON architecture can be examined using the NEAT model: a construct that is used to define a high assurance system.

SAMSON information protection components and processes are:

- **N**on-bypassable: SAMSON intercepts traffic between the workstation and the target data service. Information requests that comply with the security policy are allowed to proceed. While traversing the interception point, data is cryptographically transformed. Only valid requests can traverse the intercept and any attempt to access the data directly will only disclose an encrypted object.

- **E**valuatable: Each SAMSON component is implemented as a well-designed, well specified, well implemented, minimalist, low complexity module that is accessible through a well-defined, open protocol. It is possible to do assurance testing against each SAMSON interface through the use of validation and verification harnesses.

- **A**lways-invoked: Every SAMSON-relevant data request is checked by the appropriate security monitors and information protection services. The selection of what constitutes a SAMSON-relevant data request is entirely defined by the implementation. SAMSON does not place restrictions on what actions can be made subject to policy-based access control.

- **T**amperproof: The system generates an audit trail for all security relevant events that is established through a chain-of-custody. This capability detects the addition, deletion or modification of audit trail information. While this does not provide proof against tampering, it does support the detection of unauthorized modification of the audit records in support of incident handling and forensic activities.

3.0 The SAMSON Modular Architecture

This section expands upon topics presented in the section in the *Architectural Design Document* similarly titled *The SAMSON Modular Architecture*. It is recommended that the reader first be familiar with the information presented in that document in order to better understand the architectural decisions that were made during the development of the demonstrator and are described below.

3.1 Architectural Summary

The SAMSON TD is a standards-based implementation of a Service Oriented Architecture (SOA) where all security requirements are met by independent services that are accessible through open, well-defined interfaces. The information exchange formats within this architecture utilize industry accepted, open standards that are based on the eXtensible Markup Language (XML). It provides a solution in accordance with the Attribute based Access Control (ABAC) paradigm using eXtensible Access Control Mark-up Language (XACML) for the expression of security policy. Together, XACML and ABAC form a framework and a new standard for introducing data-centric information protection and assurance principles within the Government of Canada (GoC).

It is important to recognize that the architectural goal of the SAMSON infrastructure is to provide a common set of security interfaces and the ability for data handling applications to leverage those services for a universal application of the domain's security policy. Conceptually, the SAMSON security services act as security gateways with the ability to route an internally generated SAMSON security service information request to the actual external service or process that will handle the request. SAMSON is, therefore, not tied to any specific vendor solution and can replace any vendor solution with another product that provides similar functionality.

In this way, all SAMSON security interfaces conform to a common set of design goals, including:

- They can be made to work with any external vendor solution, product or implementation;
- They can be extended to include any SAMSON-specific capabilities that are not reflected in the chosen standard;
- They are appropriately secured in order to ensure the confidentiality and integrity of these information exchanges; and
- New security services can be added to the architecture without the need to redesign or redeploy the entire security overlay.

The SAMSON architecture achieves its data protection requirements through the use of three core architectural components:

1. **Secure Messaging Service Busses (SMSBs)**: The ability for SAMSON components to exchange data over a dedicated messaging infrastructure in a manner that is secure, protocol agnostic, and reliable.

2. **Security Services Gateways (SSGs)**: The ability to bridge between the SAMSON security architecture and the back end (non-SAMSON) applications that provide the needed security functionality. These services include authorization for adherence to policy, cryptographic services for protection of data and audit for the creation of a trusted chain of evidence.

3. **Policy Enforcement Points (PEPs)**: The ability to link external application and security services to the SAMSON infrastructure in a manner that adheres to the SAMSON security protection principles.

An SMSB is the connecting infrastructure that allows PEPs to utilize the services offered by SSGs and, similarly, allows SSGs to leverage the capabilities offered by other SSGs. These components can be seen in their proper context in Figure 2: SAMSON Core Components.

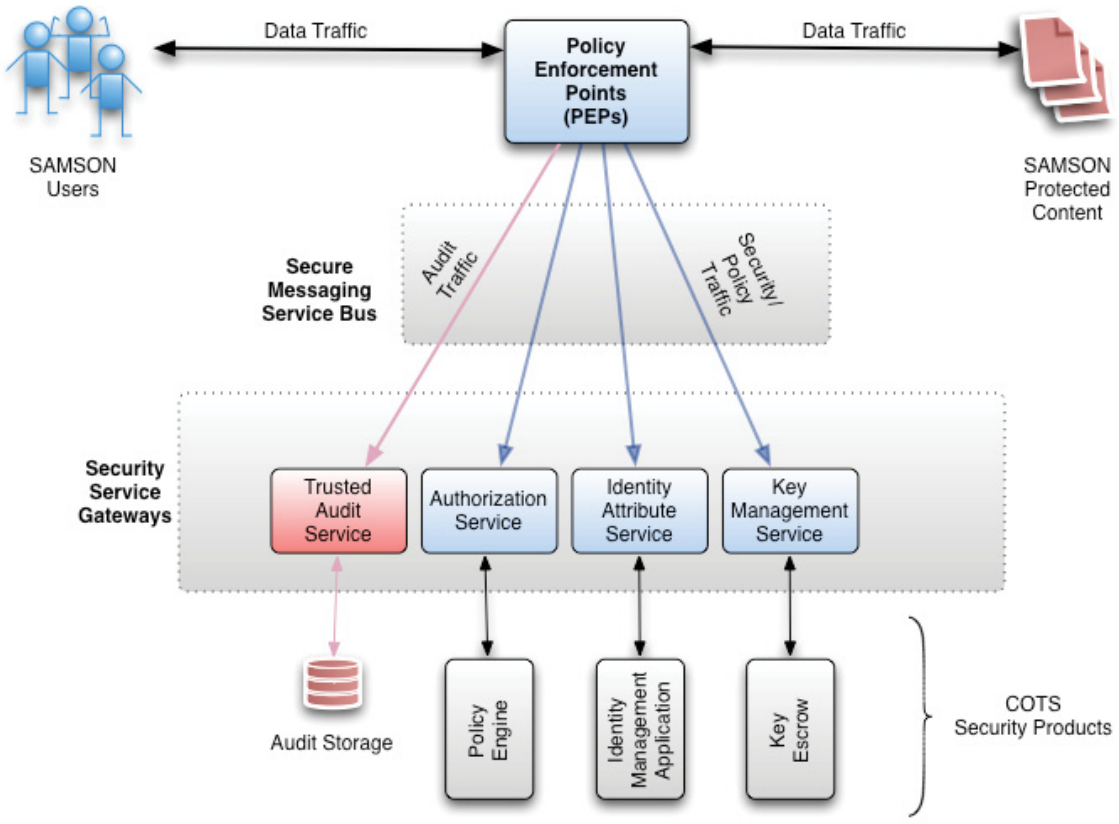


Figure 2: SAMSON Core Components

The following sections will provide the design details for the three core architectural components of SAMSON: the PEP, the SSG and the SMSB that connects these two classes of SAMSON objects.

3.2 The Secure Messaging Service Bus (SMSB)

SAMSON is implemented as an SOA and provides a set of interconnected services that work through the exchange of messages on two separate and isolated messaging infrastructures:

1. A security messaging infrastructure that carries policy data, security attributes and cryptographic information; and

2. An audit messaging infrastructure that carries audit information.

The information exchange formats in SAMSON utilize industry accepted, open standards that are based on XML. It is the responsibility of the SAMSON messaging infrastructure to provide the delivery of these messages between SAMSON components. Although the specific protocol or format of the message content will depend on the nature of the entity or service being leveraged, all messages are delivered through the same communications mechanism. With the responsibility to ensure robust, secure and trusted delivery of security messages between SAMSON components, the messaging infrastructure forms the critical core of the SAMSON architecture.

XMPP is the delivery mechanism for exchanging SAMSON security messages between the PEPs and the SSGs. Because web frameworks are the most familiar mechanism for implementing SOAs, a discussion of the parallels between HTTP-based and XMPP-based SOA frameworks is appropriate.

The following table identifies some core SOA elements, their traditional implementation mechanism via web services and the XMPP analogue.

Table 1: SAMSON XMPP Service Oriented Properties

SOA Component	Traditional Technology	XMPP Equivalent
Service discovery	WSDL/UDDI	XMPP DISCO
Messaging encoding	SOAP	XMPP content
Messages transport	HTTP	XMPP delivery
MESSAGE Security	WS-Security	Transport Layer Security (TLS)
Encryption	WS-Encryption	TLS

The XMPP DISCO protocol extension provides equivalent service discovery features to those provided by WSDL/UDDI. Information exchange is provided by XMPP messages that consist of an envelope for a delivery address and message content; much like SOAP. However while both SOAP/HTTP and XMPP provide message transport, security and encryption they diverge in a fundamental way.

HTTP systems are *stateless and connectionless*; each message, both a service request and response, is independent. In this environment, secure, authenticated messaging requires that each individual message be secured and authenticated, typically by using WS-Encryption and WS-Security. This increases message overhead substantially and also moves responsibility for secure message delivery up into higher levels of the protocol stack.

XMPP-based systems on the other hand use *persistent connections* between clients and the server. As long as the connection endpoints can be authenticated when the connection is initiated, authentication does not need to be done again for each message. If the connections are also encrypted, then there is no need to encrypt message content separately. XMPP thus offers a SOA with much lower message overhead and a better allocation of messaging delivery and protection functionality into the protocol stack.

The next section will provide more details about how SAMSON leverages XMPP as a delivery mechanism.

3.2.1 *SAMSON and XMPP*

An XMPP message delivery system is a *store-and-forward* system, similar to email, but operating in near real-time. The network is organized in a star configuration with all endpoints connecting through a central XMPP server. SAMSON services first connect to the XMPP server to set up a persistent connection or session (for message transport) and then authenticate to the server. The XMPP server provides the message routing between the SAMSON services, ensuring that messages are only delivered to their intended recipient.

SAMSON XMPP sessions leverage Transport Layer Security (TLS) to ensure that message traffic is encrypted. SAMSON also requires authentication at the session layer so that the identity of the participant in the XMPP domain is determined when the connection to the domain is established. Achieving this level of trust is required prior to any exchange of messages and offers a double layer of security:

1. Protection of the information at the transport layer connection; and

2. Authentication of the session that specifies the identity of the XMPP network participant.

In the current implementation, PEPs and SSGs are identical in that manner in which they connect to and use the XMPP messaging infrastructure. Each component's XMPP identity and credentials are specified in a local configuration file that is loaded at run time and used to connect to the XMPP domain and access the messaging services of the SMSB. Once a

SAMSON component is connected to the XMPP domain, it is able to send and receive messages to support its role within the SOA.

XMPP servers require a centralized repository to store the identity and provide authentication for the participants in the XMPP domain. In the deployed architecture, this service is an LDAP directory with a separate directory branch, or organization unit (OU), for each of the XMPP servers that provide messaging for their respective SMSB. The LDAP service is an instantiation of the OpenLDAP 4.2.3 server and is hosted on its own separate machine and used exclusively by SAMSON. Both XMPP servers access this LDAP service over the Management network using the standard LDAP protocol.

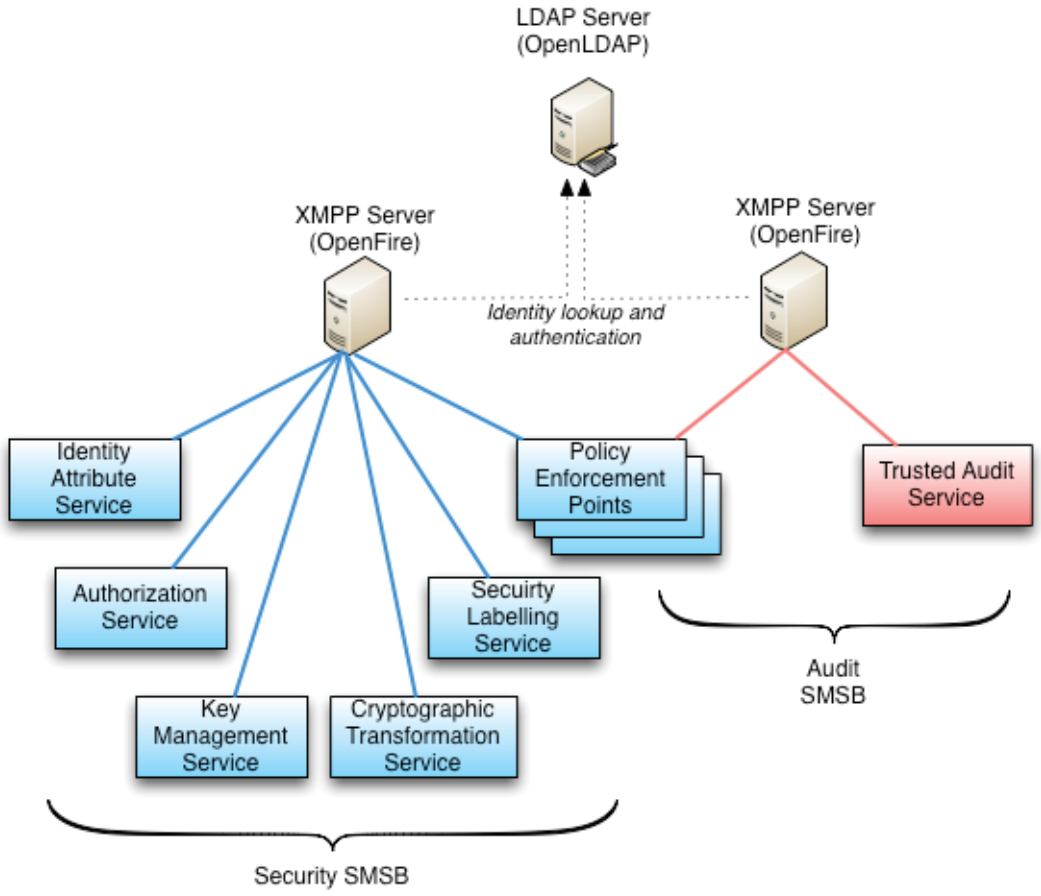


Figure 3: SAMSON SMSBs as XMPP Domains

This diagram illustrates the use of two separate XMPP domains to host security and audit messages, respectively. The security services are participants on the security SMSB whereas the PEPs exchange messages along both message busses. The SAMSON deployment hosts XMPP domains on physically separate networks and the use of TLS further protects the confidentiality and integrity of SAMSON messages.

The SAMSON administrator is responsible for creating the identities in the repository and providing the configuration information (identity and password) in the configuration file that is used by each SAMSON PEP and SSG. This configuration file also specifies the identities of all the other SSGs on in the SMSB so that each component knows where to route messages to leverage a given SAMSON service.

With the multiple layers of identity and session protection, SAMSON services have a high degree of confidence:

- That they are connected to the correct messaging server;
- That no rogue services are running to illicitly receive message traffic; and
- That there is, architecturally, built-in protection against man-in-the-middle attacks.

3.2.2 *SAMSON Messaging and XMPP*

The XMPP messages exchanged between SAMSON endpoints are simple XML documents incorporating:

1. An envelope for message addressing; and
2. A message body to carry the substance of the communication.

The following example presents a typical XMPP envelope.

```
<iq to='test@samson.org/xmpp'
    id='ex1'
    type='get'>
</iq>
```

XMPP message payloads are also XML documents and, for SAMSON services, typical message payloads are based on existing XML standards that are appropriately chosen to support the type of service they are providing. For example, a request to the SAMSON Identity Attribute Service (IAS) to request a user's community of interest membership is encoded using the XML Service Provisioning Markup Language (SPML) and Directory Services Markup Language (DSML), resulting in an XML document in the following format.

```
<spml:searchRequest
    xmlns:spml='urn:oasis:names:tc:SPML:1:0'
    xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core'
    requestID=''>
    <dsml:filter>
        <dsml:equalityMatch name='accountId'>
             <dsml:value>request_user</dsml:value>
        </dsml:equalityMatch>
    </dsml:filter>
     <spml:attributes>
         <dsml:attribute name='caveats'/>
     </spml:attributes>
</spml:searchRequest>
```

The following table lists the XML-based message formats that are used by each of the SAMSON services.

Table 2: SAMSON Service Payload Protocols

SSG	Request Message Format	Response Message Format
Identity Attribute Service	Service Provisioning markup Language (SPML)	SPML Response
	Directory Services markup Language (DSML)	DSML Response
Authorization Service	XACML Context Message	XACML Content Message
Key Management Service	XACML Context Message	SAMSON Service Response
Cryptographic Transformation Service	XACML Context Message	SAMSON Service Response
Security Label Service	XACML Context Message	SAMSON Service Response
Trusted Audit Service	AuditXML	(Not Applicable)

Both the SAMSON Service Response (SSR) and AuditXML message formats are XML documents and provide the means for exchanging necessary data between services where no standard exists.

The complete XMPP message is a simple encapsulation of the payload message within the transport (XMPP) envelope. For an IdM Service request such a combined payload and transport message would take the following form:

```
<iq to='idm@samson /xmpp'
    id='ex1'
    type='get'>
    <spml:searchRequest
        xmlns:spml='urn:oasis:names:tc:SPML:1:0'
        xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core'
        requestID=''>
        <dsml:filter>
            <dsml:equalityMatch name='accountId'>
                 <dsml:value>request_user</dsml:value>
            </dsml:equalityMatch>
        </dsml:filter>
         <spml:attributes>
             <dsml:attribute name='caveats'/>
         </spml:attributes>
    </spml:searchRequest>
</iq>
```

XML namespaces within the XMPP message body element ensure that XML element names in the message body do not collide with any XMPP envelope element names. As may be seen from this example, XMPP was designed from initial principles to support XML messages and was an enabling technology for SAMSON insofar as SAMSON is based entirely on XML messaging.

It is also worth noting that the XMPP message has minimal overhead; a fact that is a critical advantage in bandwidth constrained environments. Most significantly, neither the XMPP message envelope nor the message body needs to account for encryption or authentication as those requirements are handled by the message transport. The message transport security provided by the XMPP ensures that a consistent message payload protection mechanism is applied identically across all SAMSON security services.

3.2.3 *XMPP Infrastructure*

There are a variety of XMPP servers available. These all run like any other software service that sends and receives message over TCP/IP connections. From a SAMSON perspective, the choice of server should be immaterial. Local infrastructure considerations will help guide the server choice and the deployment architecture for the XMPP servers. When deploying an XMPP infrastructure and selecting the server software to support the messaging, the following issues should be considered:

Operating System Support: Is the XMPP server software supported on our target baseline server operating system.

High Availability: Does the software support a HA architecture, including server clustering, failover, load balancing and the leveraging of replicated databases?

User Community Sources: What technologies can be used as the repository for SAMSON component XMPP identities? For example, directory servers such as Active Directory and OpenLDAP or databases such as MSSQL or MySQL may be targeted services for supplying XMPP user community data.

Transport Layer Security: Does the XMPP server have support for TLS-based connections and mutually authenticated TLS-based session? To what degree does the server support X.509 certificate standards and certificate revocation?

Administration Support: What administrative and reporting features are provided by the server and to what degree can these integrate with local infrastructure?

As shown in Figure 3: SAMSON SMSBs as XMPP Domains, the deployed SAMSON architecture uses two OpenFire 3.7.1 open source XMPP servers as the message servers which provide the message exchange capabilities for the Security and Audit SMSBs. The deployed architecture also uses an OpenLDAP 3.2.4 directory service as the repository for the SAMSON PEP and SSG service identities.

The following section will describe the structure of the SSGs, the SAMSON components that bridge between the Secure Messaging Bus and the external security services that are to be connected to and leveraged by the SAMSON infrastructure. It is important to note that these SSGs must connect, as clients, to the XMPP messaging infrastructure.

A language appropriate XMPP client library can provide a significant part of the complexity of creating XMPP client process. At a minimum, the documentation for that library should provide advice and guidance on how to write an XMPP client that can:

1. Connect to an XMPP server,
2. Authenticate to the server, and
3. Send and receive messages with other components within that XMPP message domain.

When developing a SAMSON service, it is important to ensure that the trapping, propagation and reporting of error conditions is performed so that overall system reliability and security is not compromised.

3.3 Security Services Gateways (SSGs)

The SSGs act as security gateways with the ability to route an internally generated SAMSON security service information request to the actual external service or process that will handle the request. SAMSON is, therefore, not tied to any specific vendor solution and allows any security solution in the deployment environment with to be replaced with another product that provides similar capabilities without the need to reconfigure the entire security

overlay. In this way changes to security applications in the deployment environment are localized to the SSG that interfaces with that solution.

All SAMSON security interfaces conform to a common set of design goals, including:

- They can be made to work with any external vendor solution, product or implementation;
- They can be extended to include any SAMSON-specific capabilities that are not reflected in the chosen standard;
- They are appropriately secured in order to ensure the confidentiality and integrity of these information exchanges; and
- New security services can be added to the architecture without the need to redesign or redeploy the entire security overlay.

The SAMSON architectural approach meets the design philosophy defined above through the use of pluggable modules that interact though standard interfaces, both for application and security service protection. SAMSON is an SOA in that all of the components that support SAMSON capabilities, or leverage those capabilities for information protection, link to the core framework via those standard interfaces. While the linking and communicating interfaces provide a common mechanism by which these pluggable components join the SAMSON infrastructure, the format of the messages that are transmitted through the interface are geared specifically to the module's purpose. New modules are added, not through a programmatic interface, but through this standards-based, message-oriented pluggable architecture.

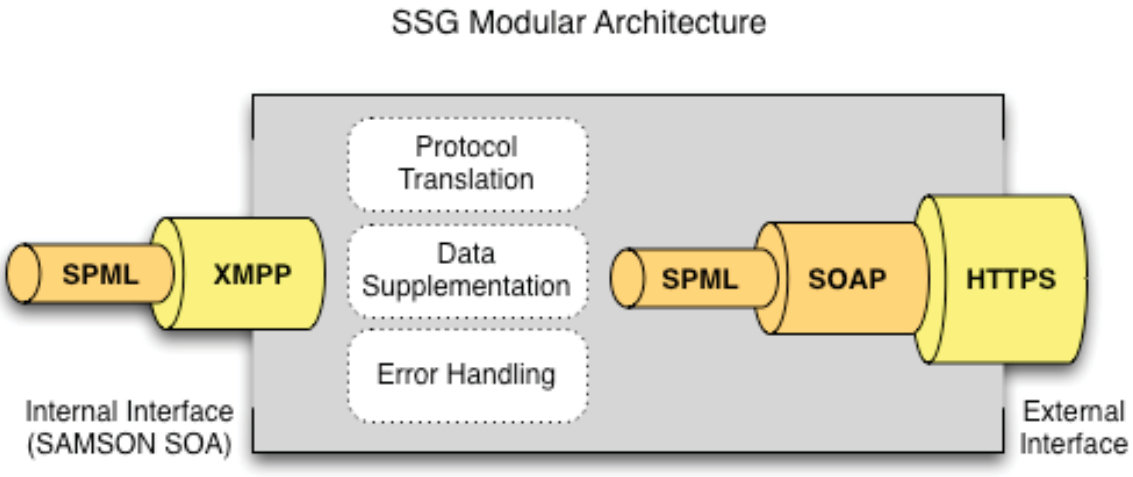


Figure 4: SAMSON Modular Interface Design

Figure 4: *SAMSON Modular Interface Design* shows a sample SSG instantiation. Extending the sample interface described in section 3.2.2: *SAMSON Messaging and XMPP*, the

internal interface for this SSG receives SPML formatted payloads delivered over the SMSB as XMPP formatted messages. On the external side of the SSG, the message is retransmitted as SPML payloads encapsulated inside SOAP envelopes and transmitted to the external security application over TLS protected HTTP.

Acting as the bridging component to link internal and external messages, the SSG performs the needed processing to ensure that the messages exchanges are properly handled, including:

- Protocol Translation of the message content from the internal SAMSON service payload format to the external format used by the security application; this external format may be of a non-standard or proprietary format;
- Data supplementation of missing information by calling other SAMSON interfaces; and
- Error handling of any error conditions raised by the external security application and mapping the external error to form that can be recorded as an audit event and, potentially, raised as a security incident.

The creation of an SSG that integrates an external security application into the deployed SAMSON architecture through the SMSB requires that certain common aspects of the component's design be addressed. The following diagram provides an abstracted view of the functions and data flows that are common across all SSG implementation.

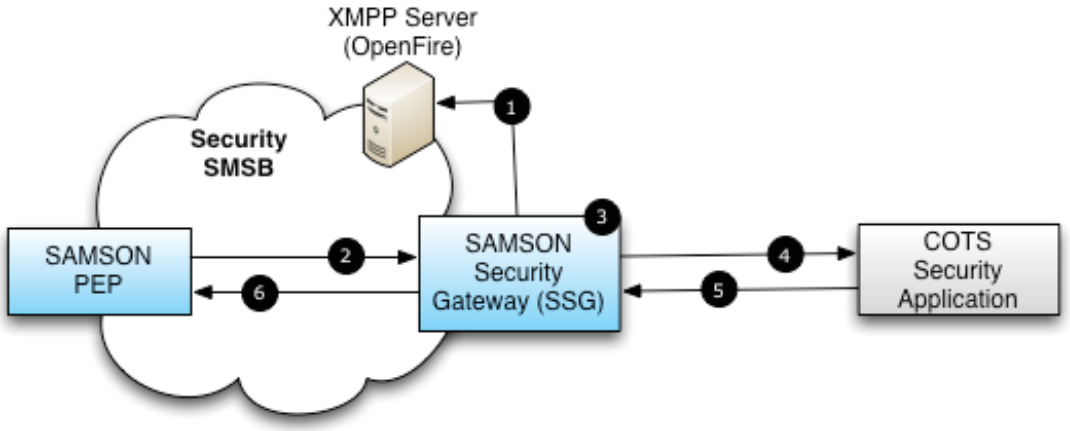


Figure 5: A Common View of the General SSG Design

Each stage in this process is described in detail below.

Step 1: Establish an XMPP connection: Typically, XMPP sessions will be established by the SSG using a language specific XMPP library such as PyXMPP[2] for Python or Gloox[3] for

[2] http://pyxmpp.jajcus.net/pyxmpp.html

C++. The XMPP connection will be established once at service start up and used through the lifetime of the service process. The nature of the persistent session with the Secure Messaging Service Bus can be leveraged to support service monitoring, presence messaging and broadcast signalling to command specific or all SAMSON components. The Security Service will need to be provided with a JID (XMPP identity and credentials) and connection information to access the XMPP domain. Once a persistent connection is made to the XMPP server, the SSG is a participant on the SMSB and can exchange messages with other SAMSON components.

Step 2: Extract the Traffic and Message Payload: When a Security Service receives a message from another SAMSON component; the first task is to extract the message content. This means that the payload must be extracted from the XMPP message envelope using a XML parser. The payload must then be interpreted, again using an XML parser, based on the type of message that is being sent. For example, continuing with the example of the Identity Attribute Service (IAS) that is responsible for querying a backend service to supply SAMSON component with user attribute information, any SAMSON component that requires user identity information will formulate an SPML/DSML message payload and transmit that payload, over the SMSB, to the IAS and then wait for the IAS to return a query response.

Step 3: Process the Message Content: When the SSG receives a message request it will extract the message payload and transform the message as needed to create the appropriate message query that can be, in turn, sent to the back end security service. For example, the IAS will accept SPML/DSML formatted request from other SAMSON component, but will transform these requests into their LDAP equivalent.
In some cases, transforming the data may require the sending and receiving of supplemental messages to other SAMSON components. The service must be able to manage these supplemental messages and ensure that the request / response cycle is mapped back to the original service request. Section 4.0: *The SAMSON Security Services* provides a more thorough description of the processing that is performed by each SSG on the message requests they receive.

Step 4: Manage the Connection to the Back End Application: The nature of this connection will depend heavily on the nature of the application being integrated into the SAMSON environment. These connections may be persistent, established once when the service is started, or created and destroyed on an as needed basis. There is no impact to the SAMSON environment which back end connection type is made, although there may be a performance impact if connections must be frequently established and are not reused.

Step 5: Submit the Request to the Back End Application: Once transformed and expanded with supplemental information from other SAMSON services, this message is submitted to the back end application. The request is encoded into the appropriate message format and wrapped in a transport envelope such as SOAP within HTTP. In this form, the message can be delivered to the target application using extended security protocols such as SSL/TLS to ensure the integrity and confidentiality of the information.

[3] http://camaya.net/gloox/

Step 6: Process Response Messages: The back end application will return a response that includes response data and status information. This information will be reformatted into the original message's payload format and wrapped into an XMPP transport message for delivery.

This general outline of ordered processing activates serves as a basis for the functioning of all SAMSON Security Gateways.

3.4 Policy Enforcement Points (PEPs)

A Policy Enforcement Point (PEP) is a component that is inserted into an application's data request/response cycle in order to add SAMSON-based information protection capabilities through the leveraging of the core security services.

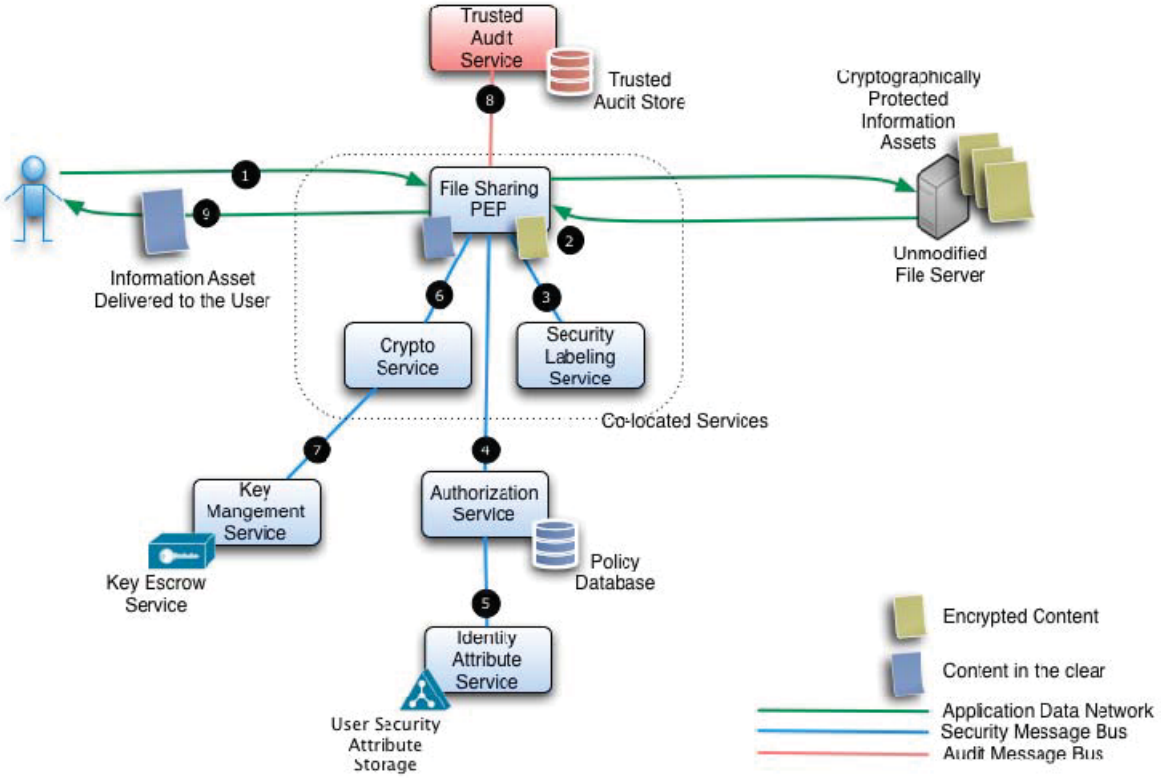


Figure 6: A SAMSON PEP Leveraging SSGs

The above diagram, illustrates how a PEP leverages SSGs in order to protect a back end file server. A sample transaction, specifically a request to retrieve a file, is shown.

1. The user sends a request to the file server to retrieve a file and the request is forwarded on by the PEP to the file server.

2. Prior to delivery, however, the PEP intercepts the file and calls upon SSGs to ensure the transaction is in compliance with the security policy. Since the PEP is, itself, a participant on the SMSB, it is able to formulate messages and transmit them to the intended SSGs over the secure messaging infrastructure.

3. The PEP queries the Security Label Service (SLS) to extract and verify the security label on the target file.

4. The PEP submits a policy decision request to the Authorization Service (AS) and receives, in response, a policy decision that conforms to the security policy.

5. The AS will itself query anther SSG, the Identity Attribute Service, to acquire security attributes for the user that are relevant to making the policy decision;

6. If the policy decision is to allow the user to access the file, a request is made to the Cryptographic Transformation Service (CTS) to decrypt the file for the user.

7. In order to obtain the necessary key fro the cryptographic operation, the CTS will query the Key Management Service (KMS) to acquire the unique key that was originally used to protect the file being retrieved.

8. Regardless of the policy decision, an audit record is created and sent to the Trusted Audit Service (TAS) so that there is a permanent and tamper resistant record of this transaction.

9. If permitted by the policy, the decrypted file is released to the user, otherwise, the data request is rejected with an appropriate error for the user (e.g. unauthorized data request).

This brief example demonstrates how the PEP leverages the six core SSGs in the processing of a typical user information request.

At a minimum, any PEP that conforms to the SAMSON architecture will consist of two sub-components or modules.

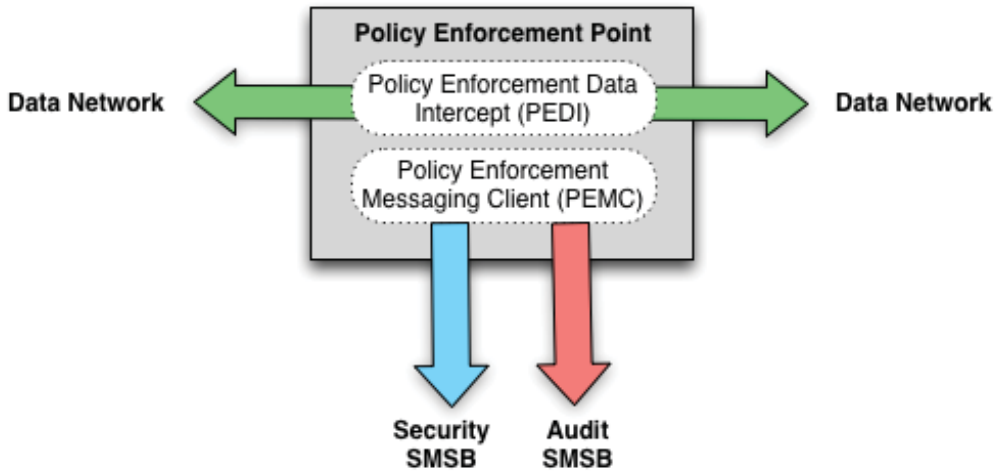


Figure 7: PEP Components

1. The Policy Enforcement Data Intercept (PEDI) is the sub-component that interrupts the data request/response cycle to extract information related to the data request in order to apply SAMSON policy enforcement; and

2. The Policy Enforcement Message Client (PEMC) is the sub-component that connects to the Secure Messaging Service Bus and leverages the Security Service Gateways to enforce Information Protection Logic (IPL).

These components are documented in detail in sections 5.1: *Generic Design of a Policy Enforcement Data Intercept* and 5.2: *Generic Design of a Policy Enforcement Message Client*, respectively.

4.0 The SAMSON Security Services

With an understanding of the SAMSON modular architecture, the messaging infrastructure, and the role of the SAMSON Service Gateways, this section describes each gateway service in detail. Whereas Section 3.3: Security Services Gateways provides an overview of the general SSG design, this section provides a description of each of the SSGs that form the core of the SAMSON architectural deployment as shown in Figure 6: A SAMSON PEP Leveraging SSGs. The list of security services deployed as part of the SAMSON architecture includes:

- The Identity Attribute Service;
- The Authorization Service;
- The Cryptographic Transformation Service;
- The Key Management Service;
- The Security Label Service; and
- The Trusted Audit Service.

This section describes the overall purpose of each service, the service's configuration for the SAMSON deployment, a list of the inputs, outputs and processing activities for each service and recommended practices for service design. The information presented in this section is meant to provide a thorough understanding of how each SAMSON service functions and also to be of benefit to security solution developers that want to provide SAMSON-based information protection of applications through the use of the SSG general design, the modular design and the common messaging infrastructure.

4.1 Identity Attribute Service (IAS)

As described in the Architectural Design Document, the Identity Attribute Service (IAS) is one of the central SAMSON Security Services based on the SSG design. This service provides supplemental security attribute information related to a SAMSON user. This supplemental information can be requested through the service's interface by any component with a need to know user attribute information.

For the SAMSON architectural deployment the Authorization Service (AS) is the only SAMSON component that leverages the IAS to retrieve security attribute information. The Authorization Service requests security attributes for a given SAMSON user in the context of evaluating a policy request. A user's membership in communities of interest (COIs) is relevant to the decision making process. For example, if a policy states that any user that is a member of the CANUS community can access CANUS information, the Authorization Service will need to query a user's COI memberships to determine if that user is a member of the CANUS community. Being a member allows that user to gain access to the requested information, but the AS must query the IAS to determine if that user is part of that community.

4.1.1 *IAS Design Considerations / Configurations*

In the SAMSON TD architectural deployment, the SAMSON *security attribute repository* stores three security attributes for each user: their *nationality*, their *clearance* level and the *communities of interest* to which the user belongs. The architectural deployment configuration includes an LDAP server (OpenLDAP version 4.2.3) using a dedicated organizational unit (OU) that is reserved for the storage of these three security attributes for each user. Within this OU, the common name (CN) matches the SAMSON user's unique name, as determined through the Window domain authentication mechanism and credential acquisition process.

This LDAP-based repository is the authoritative source for SAMSON user security attribute information. Because a user's security attributes are a critical part of the SAMSON trust model, it is important that the integrity is ensured through the solution architecture and through appropriate safeguards.

SAMSON users have a need to access their security attribute information; for example, a list of their assigned caveats should be provided when the user is applying a security label to a document. However, the user community should not be able to modify their security attributes nor should it be possible to tamper with these attributes while they are being provided to the IAS.

Within the SAMSON TD architectural deployment, therefore, there are two separate repositories for user information. The Windows Active Directory maintains account and account credentials information for all users in the domain. The SAMSON security attribute repository holds security attribute information for all SAMSON users in the domain. The security attribute repository can then be protected against tampering through technical, administrative and physical controls.

To protect the integrity of the security attribute information, the security attribute repository is the authoritative source for user's security attributes. Non-authoritative copies of these attributes are pushed to the Active Directory to provide each user with access to their attributes without the need to expose the repository to the user community. The location of and access to security attributes is shown in the following diagram.

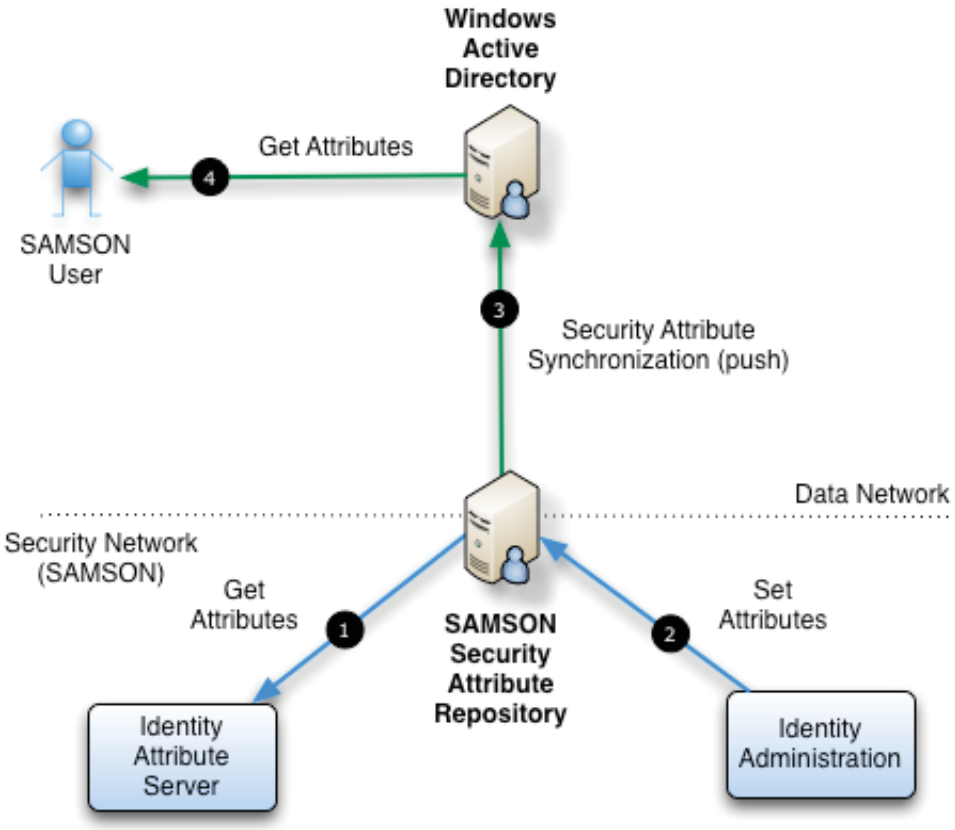


Figure 8: Security Attribute Repositories

The IAS, the SAMSON SSG that serves requests for user security attributes, accesses the user information in the repository by specifying the user for which attributes are needed and which attributes are to be returned.

1. The SAMSON IAS queries the repository (1) for user security attribute information and can forward this attribute information to any SAMSON component that requires security attribute data (e.g. the Authorization Service). Because this information is exchanged across the SAMSON security network, it is not exposed to the user community on the data network and is, therefore, not at risk to tampering by malicious users on the data network.

2. Security attribute information can only be set through the SAMSON protected Identity Administration Interface and as a result, only authorized users (e.g. the IdM administrator) has the policy right to alter security attributes. The Identity Attribute Interface is further described in section 6.2: *Identity Attribute Administration Interface*.

3. Because SAMSON users have a need to access their security attributes, security attribute information is pushed from the repository, the authoritative source, to the Active Directory. In the SAMSON TD deployment architecture, this one-way synchronization process is performed on a timed basis. The security attribute synchronization process is documented in more detail in section 6.2.1 : *Identity Attribute Synchronization*.

4. In this way, non-authoritative security attributes store in Active Directory can be retrieved by security labelling software at the user's endpoint.

In addition to being able to leverage an LDAP-based security service for identity attribute information, the current IAS implementation also has the ability to leverage a COTS-based Identity Management Solution, namely, the Sun One Identity Management Suite 8.1. Due to performance considerations and ease of deployment, the LDAP-based architecture was chosen as the prime deployment configuration.

4.1.2 *IAS Messaging and Operations*

Service Input

The SAMSON IAS Service uses two messaging standards to formulate the security attribute query request/response cycle:

1. Service Provisioning Markup Language (SPML) is an XML-based framework for exchanging user, resource and service provisioning information between cooperating organizations; and

2. Directory Services Markup Language (DSML) is a representation of directory service information in an XML syntax.

The IAS receives, over the Security SMSB, a message payload in form of an SPML/DSML formatted search request. The following table describes the information that can be provided to this service as part of an information request:

Table 3: IAS User Attribute Request Data Elements

Information Element	Description	Value
Account Name	This is the name that uniquely identities the user for which security attributes are requested.	The value provided for this field is the user's unique identity as determined through their domain credentials.
Security Attribute	This is the list of attributes that are requested from the security attribute repository for the specified user	Multiple attributes can be specified from the following list: *nationality*, *clearance*, *caveats*. Caveats will return a comma separated list of the communities of which the user is a member.

The information elements are presented to the IAS over the SMSB interface according to the formats specified in Annex A.1: Identity Attribute Service Messages.

Service Processing

The IAS follows the process flow described below.

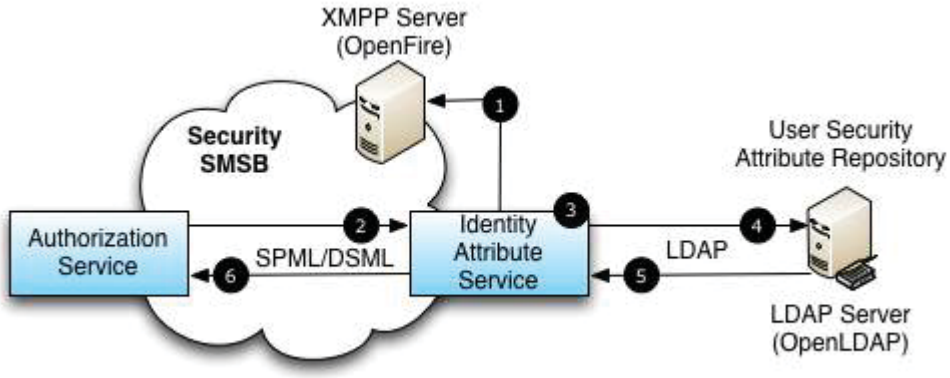


Figure 9: IAS Deployed Architecture and Information Flow

The operational configuration parameters for the IAS are described in Annex B.1:Identity Attribute Service Configuration.

1. When started, the IAS connects to the SMSB and is then ready to serve information requests for user security attribute information.

2. When the IAS receives a message from another SAMSON component such as the Authorization Service, information in the request is extracted including: the unique account name of the user and the security attributes that are to be extracted. As described in section 4.2: Authorization Service (AS), the AS receives user identity information in a policy decision request that originates at a PEP. The AS will forward on to the IAS the user identity information and a list of the security attributes required to make the policy decision.

3. This identity attribute request information is reformatted into an equivalent LDAP query format.

4. A connection is made to the LDAP server and the LDAP request is sent to the LDAP server.

5. The response from the LDAP server is parsed to extract the returned information for the queried user.

6. This information is placed into an SPML/DSML message response, encapsulated into an XMPP message and sent to the originating SAMSON component.

Service Output

When sending a response to a submitted query, the IAS sends a message payload in form of an SPML/DSML formatted search response over the Security SMSB. The SPML/DSML search response format includes the name of the requested user and individual specification for each security attribute that was requested.

The following table describes, for each requested attribute, the name of the response attribute and the format the value of that response attribute will take.

Table 4: Supported IAS User Security Attributes

Requested Attribute	Response Element	Value	Example
nationality	nationality	Text representing the user's nationality	CANADA
clearance	clearance	Text representing the user's clearance	SECRET
caveats	caveats	A comma separated list containing all the communities to which the user belongs	CEO,CANUS

The response elements and values are returned to the requesting component over the SMSB interface according to Annex A.1: Identity Attribute Service Messages. Should the user not exist or if any of the requested attributes is unspecified, a fully formed response message will be returned to the user with an empty value.

Design Notes

The set of attributes that will be accessed through the IAS will be a function of the following requirements:

- The nature of the policy expressions that will capture the security policy;
- The nature of the policy queries that will enforce the policy; and
- The information that is available through the externally provided IdM application.

The SAMSON architecture does not dictate what security attributes can or must be used in policy evaluation and, as a result, the set of attributes that are to be provided through the IAS is not bound to a specific set of attributes or data types.

The deployed SAMSON architecture only uses community of interest membership since the nationality and clearance level of all SAMSON users in known for the deployed target environment (i.e. SECRET clearance, Canadian nationality)

When designing SAMSON, a core consideration is what information uniquely identifies a SAMSON user. Where federated identity system are already in place, this choice of attribute used to specify users will be clear. Where there are heterogeneous or loosely coupled account structures, selecting the user account attribute from the information available in the environment will be more challenging. The risk from a SAMSON perspective is that it is not possible to safely make policy decision where there is ambiguity about the identity of the user. The technical demonstrator has used Windows Domain identities to uniquely identity the user community members, but this will not always be the most appropriate choice. For example, consider the scenario where there are multiple Windows domains with an account for John Smith in each domain. It is not clear if this is the same user or a different user. This is an existing problem for federated identity solutions.

When developing the IAS, the following design decisions must be considered.

Location of the Security Attributes: The repository of the security attribute information may be co-located with the user account or in a separate storage facility. For example, in accordance with the SAMSON Architectural Design, Windows Active Directory holds user accounts. The security attributes may be stored with that account through the use of custom schema fields or existing extension attribute fields. Co-locating security attribute data with account data is a simpler design since the user account and attributes can be managed through a common interface. However, when co-locating this information, it becomes more challenging to achieve separation of duties where user account management and user security attribute are maintained by separate roles. Additionally, since user security attributes are used as part of the policy decision process, it is recommended that these attributes be stored in a facility with a higher degree of physical, administrative and technical safeguards.

Provisioning: Where security attributes are stored separately from the user account as in our design, the provisioning process is more complex. The assignment of security attributes to users should be performed though interfaces that are themselves protected by SAMSON. These interfaces are described in section 6.0:SAMSON Self-Protecting SAMSON Services in the Architectural Design Document. Automated provisioning processes should be able to establish a placeholder record for the user's security attributes, but not able to assign or modify the actual security attributes.

Securing Administrative Interfaces: The interface through which security attributes are assigned to users must be, itself, a SAMSON policy-based data exchange. That is, the interface that is used to maintain user security attributes must enforce the domain's security policy. As an example, the policy right to modify user security attributes could be assigned to a specific community; only those users that belong to that community will have access to the security attribute repository administrative interface. Use of this interface is subject to all SAMSON protection from access control to auditing.

As previously described, the deployed SAMSON IAS can also leverage a Sun One IdM service as a user security attribute repository; however, this configuration has not been used in the current target architecture. The Sun IdM Suite supports the SPML protocol; the SAMSON IAS forwards the SPML messages without modification. The IAS retransmits messages between the SAMSON XMPP transport protocol and the SOAP over HTTP transmission protocol supported by the Sun solution.

4.2 Authorization Service (AS)

As described in the *Architectural Design Document*, the Authorization Service (AS) is one of the central SAMSON security services and is based on the general SSG design. This service provides policy decisions, as determined by an evaluation of the security domain's access control policy, to SAMSON PEPs that, in turn, enforce the application of the policy decision.

Each of the PEPs in the deployed architecture, including the PEPs that protect information assets such as: files, email messages, instant messages and web services, formulate a policy request for each user information access transaction, such as: retrieving a file or sending an email message. The resulting decision that is supplied by the AS is enforced by the PEP to permit the transaction to take place or deny the user's activity.

The SAMSON AS is a bridging component that will link policy requests from the PEPs to a Policy Decision Point (PDP) in the target environment. The PDP is a processing function that interprets a policy request and evaluates that request against the currently stated security policy. The security policy is an expression of the access control rules for the security domain and the policy engine, or PDP, is the processing component that evaluates an access request in the context of the security policy to derive the correct access control decision.

The nature of the policy requests that are submitted to the AS is a function of the following requirements:

- The security policy specification, that is, the range of attributes that are represented in the policy and/or available in the environment;
- The ability of the policy engine to process complex policy logic; and
- The granularity of access control that is needed at the application-centric PEPs

When viewed as a general architecture, SAMSON does not dictate the format of the policy requests so long as the request/response cycle is well-formed XACML expressions. The type of policy decisions that can be provided through the AS is not tied to a specific set of policy query attributes or conditions.

In the deployed SAMSON architecture, PEPs submit policy requests that contain: the requesting user's identity, security attribute information of the desired information asset and a statement of the operation that is being performed on the resource. In the XACML specification, these three data elements are referred to as the subject, resource and action,

respectively. The AS also adheres to the XACML specification for the expression of policy decision; typically “Permit” for a request that is allowed under the policy or “Deny” for requests that cannot be allowed to take place.

4.2.1 *AS Design Considerations / Configurations*

Although the general SAMSON architecture supports the use of COTS-based PDPs that exist as separate, pre-existing components in the target environment, the SAMSON architectural deployment integrated the PDP into the AS as a software sub-component. In this sense, the deployed AS not only has the ability to act as a policy decision interface to the PEPs, it also contains, within its process space, the ability to evaluate policy requests and make policy decisions.

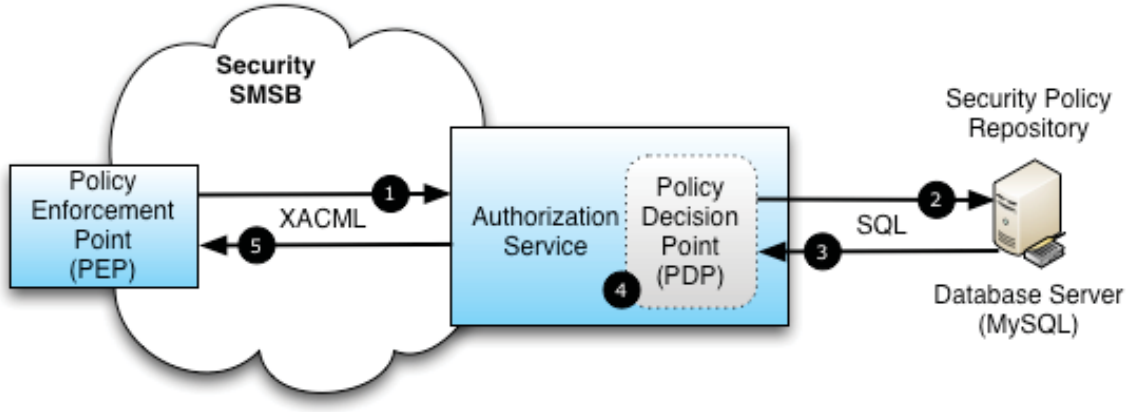


Figure 10: AS Deployed Architecture

In this architecture, security policies are stored in a database that is queried by the PDP to access the policies that are relevant in the evaluation of a policy request. Policies are stored in the database as XACML 2.0 formatted policy expressions. Queries to the security policy repository are made using SQL over the security network. While the security network is the same network that hosts the SMSB, these SQL calls are not made using the XMPP messaging infrastructure; they are made directly to the database service. The security network remains, however, a segregated network that is isolated from the user community and operational data services.

4.2.2 *AS Messaging and Operation*

Service Input

The AS expects all PEPs to express policy queries in the form of XACML 2.0 context request messages and will, in turn, supply XAMCL context response messages when returning a policy decision. All traffic between the PEPs and the AS is made over the SMSB such that the XACML context messages are encapsulated within XMPP message structures for delivery to the intended SAMSON component recipient.

The following table describes the information that can be provided to the AS within an XACML context request message:

Table 5: AS Policy Request Data Elements

Information Element	Description	Value
Subject	This is the unique identifier of the originator of the policy request. This is the user that initiated the data transaction that requires a policy check. The PEP determines the unique identity of the user and encodes this information within a *Subject* element of the XACML expression	For the SAMSON deployment, the user's Windows unique domain account is used to identify the user in both security policies and policy requests.
Resource	This is the caveat in the security label on the information asset that is being requested by the user. The PEP, often leveraging the Security Label Service, acquires the caveat from the asset and encodes this information within a *Resource* element of the XACML expression.	The source of the caveat will depend on the data asset being requested. For example, a file will house caveat information within a security label whereas an IM chat room will have the caveat stored as a property of the room itself.
Action	This is the action that is being performed against the data asset. The action is encoded within an *Action* element of the XACML expression.	The action value can take any alphanumeric textual form (e.g. READ,WRITE).

It is possible to overload the application of the action policy, that is, it is possible to interpret an action across multiple applications and, in this way, reduce the number of policies that are required. For example, if the PEP that protects file shares interprets a "READ" action as the retrieval of a file form a SAMSON protected file share and the PEP that protects email message interprets a "READ" action the act of receiving an email message, then one policy rule can be used to cover both file and email transactions. Overloading the use of actions can help keep the list of policies more manageable. This topic is covered in more detail in section: 5.1.2:PEP Actions on Data.

Service Processing

The AS messaging handling process is shown in Figure 10: AS Deployed Architecture and can be described in terms of the following operations.

The operational configuration parameters for the AS are described in Annex B.2:Authorization Service Configuration. When started, the AS connects to the SMSB and is then ready to service requests for policy decisions.

1. SAMSON PEPs submit policy requests to the AS as XACML context messages.

2. The AS extracts the policy request from the payload and sends the request to its PDP processing module. The PDP requests relevant policies from the security policy repository over SQL.

3. The repository returns these policies to the PDP and the PDP evaluates the policy request in the context of the retrieved policies.

4. Based on the submitted policy request, the relevant security policies and the PDP decision logic, a policy decision is made.

5. This decision is returned to the PEP as a XACML context response message and is enforced by the PEP.

For the SAMSON TD architectural deployment, the PDP decision logic evaluates policy requests in two separate stages:

1. *Policy Evaluation*: The retrieval of policies from the policy storage repository and the matching of the policy request against each policy to get the correct rule that applies to that policy. After evaluation, each policy will have an associated rule to apply for this request that is carried over to the second stage of the decision making process.

2. *Policy Rule Evaluation*: All the rules that were determined, based on the per-policy evaluation in the first stage, are assessed to arrive at the final decision that is returned to the PEP.

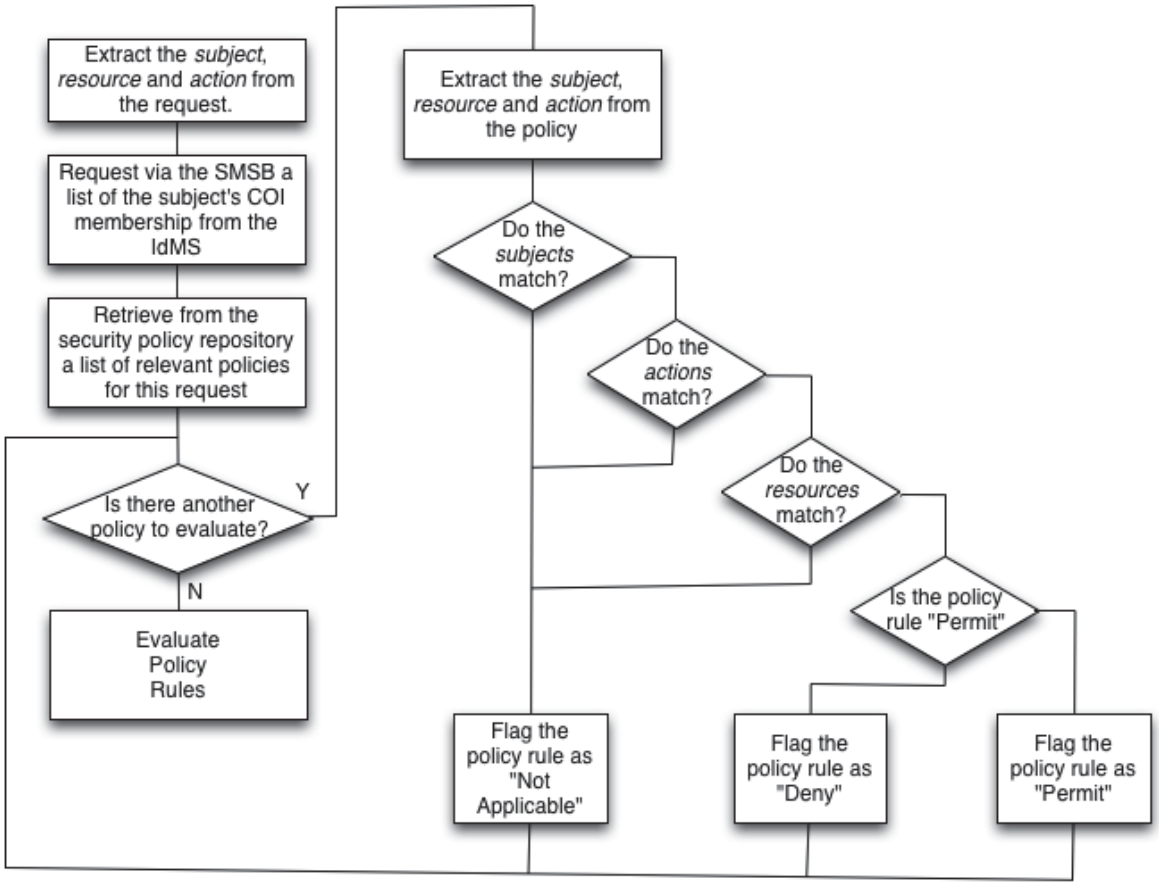


Figure 11: PDP Policy Evaluation Logic (Stage 1)

This logic is described in detail below.

1. When the AS receives an XACML context request message, the AS extracts the relevant policy request data from the message: the *requesting subject* (the user's unique identity), the *requested resource* and the *requested action*.

2. The AS provides the request information to its integrated PDP.

3. The PDP acquires the user's security attributes through a call to the IAS over the SMSB. Specifically, the PDP requests, for the requested subject, a list of the user's community of interest memberships. The IAS will respond with a list of COIs. For a user with no memberships, this will be an empty list.

4. The PDP will retrieve from the policy database, via SQL over the security network, the policies that are relevant to this decision-making process. Policies are deemed relevant if they apply to the user explicitly or any of the COIs to which the user belongs.

5. The PDP will extract from the list of relevant policies: the *policy subject*, the *policy resource* and the *policy action* and attempt to match the each policy to the original policy request. This evaluation is performed, for each policy, in the following manner:

    a. The PDP evaluation process first ensures that the policy's *subject* matches the requested user or any of the COIs to which the user belongs.
    b. If there is a *subject* match, the PDP will then match the *action* in the policy to the *action* in the request.
    c. If there is an *action* match, the PDP will then proceed to match the *resource* in the policy to the *resource* in the request.
    d. If there is a *resource* match, the PDP will then extract the policy rule that applies to this policy (e.g. “permit” or “deny”).
    e. If any of the matching activities fail, this policy is not relevant to the decision making process and the rule is deemed to be “not applicable”

6. Once all policies are reviewed, the result of each matching process is evaluated:

    a. The presence of a single “deny” result will cause the policy decision to be “deny”
    b. If there are no “deny” results, and there is at least one “permit” result, the policy decision is to “permit”
    c. If there are no “permit” results and no “deny” results, the policy decision is to “deny” (no applicable policies results in an implicit deny).

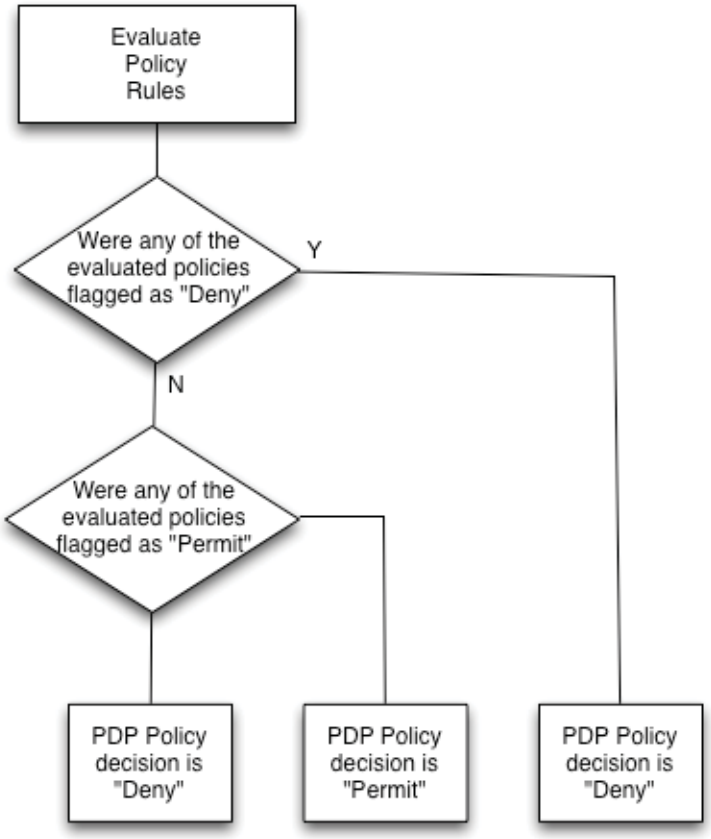


Figure 12: PDP Policy Rule Evaluation Logic (Stage 2)

It should be noted that in this PDP implementation, a security policy or policy request *resource* field might be blank. These are treated as wildcards and will always be considered a match. This also applies to the *action* entry in both the policy request and security policy.

Service Output

When providing a policy decision, the AS sends, over the Security SMSB, a message payload in form of an XACML context response message. This message includes an XACML context *decision* element that can take the following values.

Table 6: AS Policy Response Data Elements

Information Element	Description	Value
Decision	This is the result of the policy evaluation process based on the submitted policy request by the user and the current security policy. This decision is encoded into a *decision* element in the XACML response message.	As per the XACML 2.0 standard, this field can take one of three values: "Permit" (action is permitted), "Deny" (action is denied) or "Error". When any error is encountered by the AS (e.g. policy database is offline) an "Error" decision is returned and it is the responsibility of the PEP to handle this situation. In the SAMSON deployment, the PEP will deny the action and audit the error condition.

The XACML context response message format is described in Annex A.2: Authorization Service Messages.

In the course of serving its function, any SAMSON PEP that has requirement to perform a policy-based access control check will obtain this information through a query to the AS. This applies across a wide range of potential application data security handling; a sample list is provided below:

- Checks against file operations, such as: listing, opening, saving;
- Checks on the sending and receiving of email messages and attachments;
- Checks on instant message / chat room activities including: joining a chat room, sending covert messages; and
- Checks on accessing web-based services and content.

4.3 Key Management Service (KMS)

As described in the *Architectural Design Document*, the Key Management Service (KMS) is one of the central SAMSON security services and is based on the general SSG design. This service provides cryptographic key generation and connects to a COTS key storage facility in order to support the cryptographic transformations for the protection of information assets.

The PEP, as part of its role to enforce policy, directs that data be cryptographically transformed: encrypting data if it is being introduced to the protected environment and decrypting the data if data is being released to a SAMSON user. The PEP ensures that these transformations take place by calling upon the SAMSON core cryptographic service as described in section 4.4: Cryptographic Transformation Service (CTS). The PEP will direct the CTS to encrypt or decrypt a particular data asset as appropriate for the requested action on the data. The CTS requires, in turn, access to appropriate keys in order to perform the cryptographic operations.

In the SAMSON architecture, each protected data asset is individually encrypted with its own unique symmetric key. This approach to information protection means that protected information content is not disclosed to a user if the data asset has not been requested though the PEP; the PEP being the only component that is able to direct the decryption of previously encrypted data.

The CTS is the only client of the KMS since the CTS is the only service that requires cryptographic keys for its operation. When encrypting an asset, a unique key must be generated for use in the encryption process. The CTS also requires a *key token*, a unique identifier that can be used to subsequently retrieve the key from the escrow system when decrypting content.

The KMS is a bridging component that will link cryptographic key requests from the CTS to two separate subcomponents in the target environment:

1. A key generation service that will create the needed keys; and

2. A key escrow system that will store the keys and allow them to be retrieved using the key token to specify the key that is to be retrieved. The key token is supplied by the escrow system and the format of the token is, therefore, specific to the key escrow solution.

4.3.1 *KMS Design Considerations / Configurations*

In the deployed SAMSON architecture, the key generation is performed by a FIPS-compliant cryptographic toolkit supplied by RSA, the BSAFE® Crypto-C Micro Edition 4.0.1, that operates as a software module inside the KMS process space. The RSA libraries are used

in FIPS mode so that keys that are generated through the libraries adhere to the FIPS standards for key creation.

The deployed architecture also uses a COTS key escrow system supplied by StrongAuth. This system supports key storage and retrieval over a TLS protected connection.

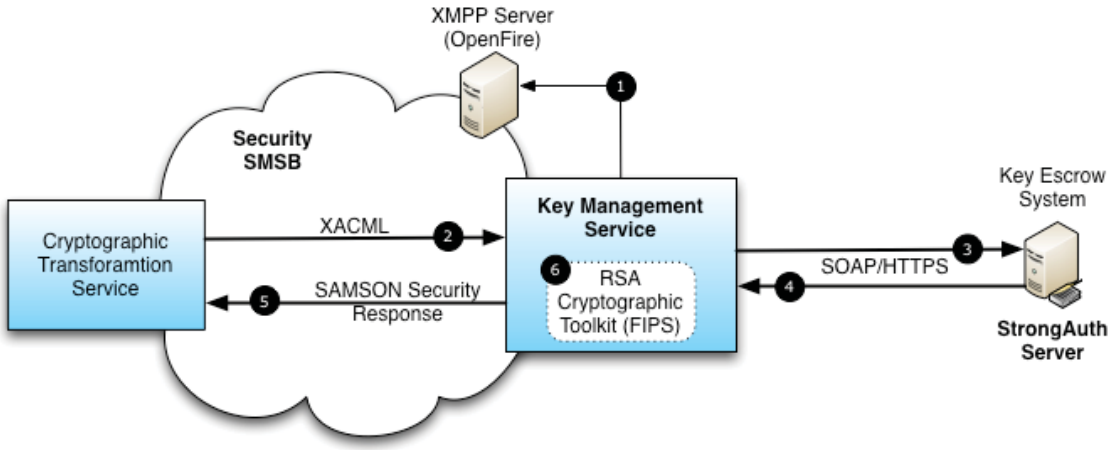


Figure 13: KMS Deployed Architecture

1. As an SSG, the KMS connects and authenticates to the Security XMPP server in order to become a participant on the Security SMSB.

2. Once connected to the SMSB, the KMS waits for key request messages. In the deployed architecture, the sole client of the KMS is the CTS; when a key is needed, the CTS sends an XACML context request message that contains the key operation request (2). The XACML message format was re-purposed from its existing role of expressing security policy requests to serve as a key request protocol since the *action* and *resource* XACML elements can suitably be used to express the key operation commands and key tokens, respectively.

3. Key storage and retrieval operations are performed by the KMS by reformulating the key request as an XML-based query expression as required by StrongAuth (3). This query is encapsulated in a SOAP envelope and is sent to the StrongAuth system over the security network. While this security network is the same network that hosts the SMSB, these key storage calls are not made using the XMPP messaging infrastructure; they are made directly to the escrow system. The security network remains, however, a segregated network that is isolated from the user community and operational data services.

4. The response from the key escrow system is received by the KMS.

5. The key and key token information is reformulated by the KMS as SAMSON Security Response (SSR) message. The format of these context request and services response messages is described in Annex A.3: Key Management Service Messages.

6. If the key operation includes the generation of a new key, the RSA cryptographic software module is leveraged to create the new key. When a new key is created, the escrow system is called to store the key for subsequent retrieval.

4.3.2 *KMS Messaging and Operations*

Service Input

The KMS expects the CTS to express key commands in the form of XACML 2.0 context request messages and will, in turn, supply the requested key information in SSR format. All traffic between the CTS and the KMS is made over the SMSB such that the XACML context messages are encapsulated within XMPP message structures for delivery.

The following table describes the information that can be provided to the KMS within an XACML context request message:

Table 7: KMS Key Request Data Elements

Information Element	Description	Value
Action	This is the action that is being requested by the CTS for the KMS.	Two actions are defined: GENERATE_STORE which will create a key and store that key in the key escrow system and RETRIEVE_KEY that will retrieve an existing key from the key escrow system. In both cases, a key is returned to the CTS.
Resource	When the action is RETRIVE_KEY, the resource field holds the key token for the requested key.	A key token. For StrongAuth, a key token is a 16 digit unique identifier.

The format of the key token is implementation specific since it is the key escrow system that creates the token and supplies it to the KMS. For the StrongAuth implementation, the key token is a 64-bit pseudo random value that bears no commonality with the key to which it is linked.

Service Processing

When the KMS is started, it reads the configuration information elements, defined in Annex B.3:Key Management Service Configuration, to determine the location and access privileges to connect to the key escrow system.

The KMS calls the RSA supplied start-up function to initialize the cryptographic module. The RSA start-up function will perform a self-check, including the verification of the module software license. Only once the library has been initialized can it be used for cryptographic functions like key generation.

Once the RSA library is successfully initialized, the KMS connects to the SMSB and waits for key request messages from the CTS. When a key request message is received, the content is extracted to determine the request type, as specified in the action element in the XACML context message format.

If the message is a GENERATE_STORE command, the KMS performs the following actions:

1. The KMS calls the RSA cryptographic library, to request a 256-bit key. Since the key generation is performed by the RSA toolkit in FIPS mode, the key will have been created according to the standards defined in the FIPS-140-2 specification. This is a pseudo-random key based on entropic sources polled by the RSA library.

2. The KMS formulates a key storage request, encapsulates it within a SOAP envelope and sends it to the key escrow system (StrongAuth) over a TLS protected session.

3. The key escrow system returns a response that contains a key token. The key token generated by the StrongAuth service is a 16 digit unique identifier.

4. The KMS formulates an SSR message with the key token and the key and sends the payload to the CTS over the SMSB.

If the message is a RETRIEVE_KEY command, the KMS performs the following actions:

1. The KMS extracts the supplied key token that was specified in the resource element of the XACML context message.

2. The KMS formulates a key retrieval request that includes the key token, encapsulates it within a SOAP envelope and sends it to the key escrow system (StrongAuth) over a TLS protected session.

3. The key escrow system returns a response that contains the requested key..

4. The KMS formulates an SSR message with the key token and the key and sends the payload to the CTS over the SMSB.

Service Response

While KMS key requests use an XACML context message format, the KMS uses the SSR message format for key responses. The SSR format is an XML-based message format with name-value pairs to specify the returned values. All SSR messages are comprised of a list element that contains the name-value pairs and a status element that reports the completion state of the response: success or error.

Each SSG encodes the list element of the SSR message with a unique name to distinguish between message usages. For the KMS, all SSR list messages are defined as "keyOP" responses.

The following table describes the name-value pairs that are returned for each key command

Table 8: KMS Key Response Message Format

Key Command	Named Value	Value
GENERATE_STORE	key	The newly generated key that was created by the FIPS CME library.
	token	The key token that was provided by the key escrow system when the key was stored.
RETRIEVE_KEY	key	The key that was retrieved from the key escrow system for the key token that was supplied in the RETRIEVE_KEY call.

The KMS will return an error code to the caller if any of the following conditions are encountered:

- Message cannot be parsed due to a malformed request;
- No action element was specified in the request or the action is unsupported;
- The key or token could not be extracted from the request; or
- An error occurred in one of the external components (RSA or StrongAuth).

Implementation Notes

An alternate implementation of the KMS was developed that uses a MySQL database backend to store keys (rather than StrongAuth Key Escrow) and a locally deployed SOAP based service to broker access to the database. This alternate implementation uses a locally resident PRNG service for key creation rather than the RSA FIPS-compliant library. The alternate KMS allows SAMSON to be used in environments that do not have a 3rd party key escrow system or FIPS cryptographic modules, but is not recommended for operational environments.

4.4 Cryptographic Transformation Service (CTS)

As described in the *Architectural Design Document*, the Cryptographic Transformation Service (CTS) is one of the central SAMSON security services and is based on the general SSG design. This service provides:

- The encryption of information assets when they are protected by the SAMSON environment; and
- The decryption of information assets when they are to be delivered to an authorized SAMSON user.

The interpretation of what is means for an asset to be protected by SAMSON is dependent on the type of asset being protected. The following table identifies, for each data type, when and how information is stored in its encrypted form and the actions the CTS performs on the data in order to protect it.

Table 9: Protected Data states and CTS actions on Data

Data Type	Protected Form	CTS Action
Files	All files that reside on a SAMSON protected file share are stored in an encrypted form.	When a user attempts to upload a file to a SAMSON protected file share and that action is authorized according to the security policy, the PEP will call the CTS to request that the file be encrypted. When a user attempts to retrieve a file and that action is authorized, the PEP will call the CTS to request that the file be decrypted prior to delivery to the user.
Email Messages	All email messages are stored as a SAMSON encrypted attachment to an email in the message recipient's mailbox while awaiting delivery.	When a user sends an email message and the message can be delivered according to the security policy, the PEP will call the CTS to encrypt the message (body and attachments). A copy of the encrypted message will reside in each recipient's mailbox. When a user retrieves messages via the PEP, each authorized message is decrypted by the CTS prior to delivery.

Data Type	Protected Form	CTS Action
Instant Messages	All persistent chat room messages are stored as encrypted, base64 encoded messages at the IM Message Server. Each message is uniquely encrypted using a common key for the chat room as a whole	When a user joins a chat room through a SAMSON PEP, the IM server will deliver a chat room history. The IM server delivers each message separately and the PEP will, in turn, send each message to the CTS for decryption. When user sends a message to the chat room, the PEP calls the CTS to encrypt the message. That message stays permanently encrypted at the IM server, but any user that access the chat room will receive a copy of the message that has been decrypted by the CTS.

As described in section 4.3: Key Management Service, the CTS must acquire keys for the cryptographic operations it provides.

In the course of serving its function, any SAMSON component that has requirement to protect or unprotect an information asset will do so through a call to the CTS. This is primarily seen in the PEP architecture where intercepted data must be protected for storage or unprotected for delivery to an authorized SAMSON user:

- Files are encrypted when they are transferred from the user's workstation to the SAMSON protected file repository and decrypted when they are retrieved from the repository for delivery to the authorized user.

- Email messages, including attachments, are encrypted when they are stored for delivery and decrypted when a user connects to his mail service.

- IM messages are encrypted as they reside in the IM server chat room message repository. Only when a user is permitted to enter a chat room are these message unprotected and delivered. Note that the IM PEP also has a facility that allows SAMSON users to assign separate caveats to individual messages. Each message will have its own policy check and there is a unique key used for each chat room/caveat level combination.

The decryption and delivery of SAMSON protected information is always predicated on the successful policy call to the AS that grants the user access to the data

4.4.1 *CTS Design Considerations / Configurations*

Co-location of the CTS with the PEP

One important decision, when designing a CTS service is the mechanism by which data assets that have been intercepted by the PEP will be transmitted to the CTS for protection. There are two options: to transmit the file to the CTS over some trusted path or to move the CTS service to the data. The SAMSON TD deployment architecture uses the latter design.

By moving the service to the data, both the PEP and the CTS can refer to absolute local paths for the protection of data. However, this architecture means a separate CTS must be deployed with every PEP that requires cryptographic transformations. Under the XMPP-based architecture, however, it is a simple matter to create multiple instantiation of SAMSON security services by assigning each CTS service a unique identity on the SMSB.

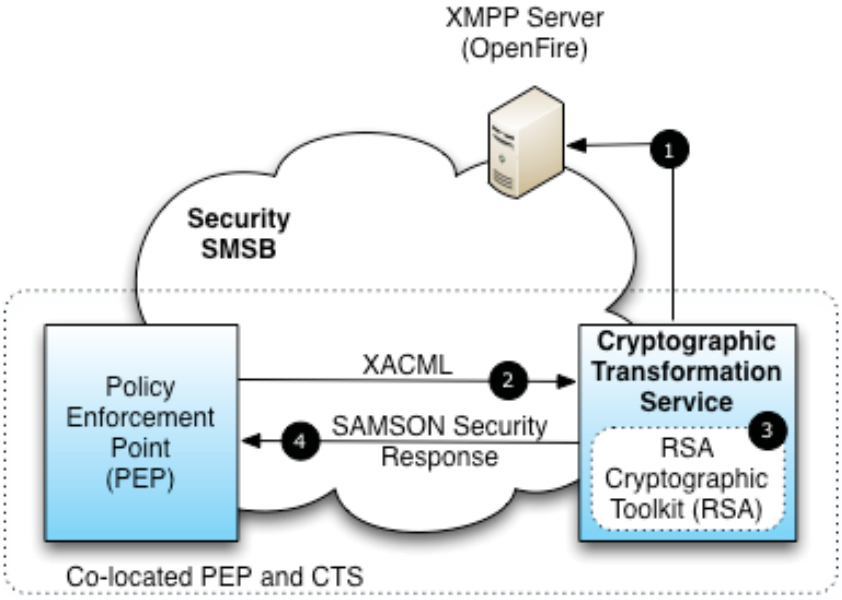


Figure 14: CTS Deployed Architecture for each PEP

The CTS is a bridging component that will link cryptographic action requests from the PEPs to a FIPS-complaint software module for execution. In the deployed SAMSON architecture, the cryptographic operations are performed by a FIPS-compliant cryptographic toolkit supplied by RSA, the BSAFE® Crypto-C Micro Edition 4.0.1, that operates as a software module inside the CTS process space. The RSA libraries are used in FIPS mode so that keys that are generated through the libraries adhere to the FIPS standards for key creation.

1. As an SSG, the CTS will connect and authenticate to the Security XMPP server in order to become a participant on the Security SMSB. Once connected to the SMSB, the CTS will wait for cryptographic operation requests.

2. In the deployed architecture, the sole client of the CTS is the co-located PEP; when a cryptographic operation is needed against a data asset, the local PEP sends an XACML context request message that contains the cryptographic operation request to the CTS.

3. The CTS performs the action against the data asset referenced in the request.

4. The CTS sends a status response back to the calling PEP in a SAMSON Security Response message. The formats of the context request and security response messages are described in Annex A.4: Cryptographic Transformation Service Messages.

Container versus Non-Container Assets

The SAMSON CTS operates on two types of assets: container assets and non-container assets.

When the CTS protects a file, it has the ability to wrap the encrypted result inside a *SAMSON container*: an envelope that allows some unprotected information to exist outside of the protected (encrypted) payload. The format of the SAMSON container is a ZIP archive and, in addition to the encrypted payload, includes:

- The key token that is used as a reference to the actual key (stored in the key escrow system) that was used to protect the payload;

- A hash digest to detect tampering of the container contents;

- The original file name of the information asset; and

- A copy of the security label on the information asset as it existed when SAMSON originally protected the asset: a copy of the original asset security label.

The SAMSON containers are protected against tampering in that:

- Any attempt to alter the contents will likely break the container formatting rendering the archive invalid to SAMSON users;

- Any attempt to alter the contents where the attacker is able to reformat the container will result in a modified digest that can be detected and flagged as a security event;

- Any attempt to alter the contents of the payload where the attacker can recalculate the digest and reformat the envelope will result in a payload that will fail decryption; and

- Any attempt to access the decrypted payload by altering the security label will fail since the external (container) security label will not match the internal security label that is attached to the original unprotected file.

SAMSON containers are used by the PEPs that are responsible for protecting files and email messages. These PEPs work on individual assets: a separate policy check is done on each file or message prior to delivery and, therefore, each asset needs its own security label. When an asset need its own security label, a container form of the asset is necessary.

Other PEPs do not require separately labelled assets. For example, the instant message PEP sets a security level at the chat room; the chat room itself has a security label for all messages it contains. Without the need for security labels on each message, the IM PEP does not need to work on container assets and it can use simple encrypted objects. These are *non-container assets*. While there is no digest of the encrypted file that can be used to detect tampering, any encrypted messages that are illicitly modified will not produce the original plaintext on decryption.

There is an implicit protection against tampering since re-encrypting a modified message (or introducing a new illicit message to the chat room) would require the cryptographic key for that chat room. Access to that key it not possible from IM server: the IM server is on the DATA network whereas the key can only be acquired from the KMS on the Security SMSB on the Security network.

Container Digests

All Samson containers include a digest that is calculated at creation time that is used to detect tampering of the container upon decryption. The digest is calculated using the SHA-512 hash algorithm. Digest calculation is dependent on the order in which the source material is submitted for the calculation. For Samson containers, the digest is calculated as follows:

1. The contents of the encrypted file;

2. The caveat for the original file;

3. The token for the key;

4. The filename of the original file; and

5. The key used to encrypt the original file.

When a request to decrypt the file is received, the container is opened and the token is extracted. This is then used to retrieve the key from the KMS. Once the key is received, the crypto service will recalculate the digest, and compare it to the digest in the container. If these don't match, the original file is not decrypted and a message is sent back to the requesting service. If the container was modified, either it's contents modified, or the order of its members, the digest calculation will show a variance, and fail the comparison.

CTS Operations

As previously stated, the CTS must be able to acquire a key in order to execute the specified operation. The acquisition of the key is achieved through a call to the KMS over the Security SMSB. The nature of this key acquisition activity is dependent on whether this is an encryption call or a decryption call and whether the process that is calling the CTS has supplied the token. Four possible conditions are possible:

Table 10: Types of Cryptographic Operations on SAMSON data

Asset Type / Crypto operation	CTS Encryption Function	CTS Decryption Function
Non-container asset **(Token is supplied in the call to the CTS)**	CTS uses the token to lookup the key using a RETRIEVE_KEY call to the KMS. The key returned from the KMS is used in the encryption function.	CTS uses the token to lookup the key using a RETRIEVE_KEY call to the KMS. The key returned from the KMS is used in the encryption function
SAMSON container asset **(Token is not supplied in the call to the CTS)**	CTS acquires a new key from the KMS using a GENERATE_STORE call to acquire the key and the associated token. The key is used in the encryption function and a container is build that includes the ciphertext and token.	CTS extracts the token from the data asset's container. CTS uses this token to lookup the key using a RETRIEVE_KEY call to the KMS. The key returned from the KMS is used in the encryption function

PEPs that have a data asset granularity at the file level expect those files to be in a container, that is, to have security labels embedded within the file's structure. For PEPs that do not assign security labels to files, such as IM chat messages; the encryption of a data asset does not require a container for the message to be built. Depending on the request from the PEP, the CTS will place the encrypted asset into a container or not. This provides the PEP with a high degree of flexibility on how it can manage the information assets it is protecting. However, when the PEP requests a simple encryption operation with no container being built, it is the responsibility of the PEP to specify an appropriate token so the CTS can obtain the correct key from the KMS in order to perform the crypto operation.

4.4.2 *CTS Messaging and Operation*

Service Input

The CTS expects the PEP to express cryptographic transformation commands in the form of XACML 2.0 context request messages and will, in turn, supply the requested status response information in SAMSON Service Response (SSR) format. All traffic between the PEP and the CTS is made over the SMSB such that the XACML context messages are encapsulated within XMPP message structures for delivery.

The following table describes the four types of operations that a PEP can request of the CTS and the information that must be provided within the XACML context request protocol for each operation type:

Table 11: CTS Request Message Content by Message Type

Operation Type	Resource Field 1	Resource Field 2	Action Field	Environment Field
Encrypt to get a container	Plaintext file	Container file	COPY_ENCRYPT	The label on the data.
Decrypt a container	Container file	Plaintext file	COPY_DECRYPT	The label on the data.
Encrypt a file	Plaintext file	Encrypted file	FILE_ENCRYPT_TOKEN	The token for the key to use.
Decrypt a file	Encrypted file	Plaintext file	FILE_DECRYPT_TOKEN	The token for the key to use.

In the XACML context request message format, multiple *resource* elements are used to express the absolute path to the plaintext and ciphertext. The order in which the resources are specified depends on the operation to be executed: resource field 1 is always the file on which the cryptographic operation is taking place and resource field 2 is always the resulting object that is produced by the operation.

The *environment* element is used in two ways:

- When a PEP requests that a container be built for the encrypted object, the *environment* element is the label information that should be placed as the security label in the envelope.
- When a non-container asset is being requested, the *environment* element provides the PEP-supplied token that can be used by the CTS to retrieve the key for the operation.

Service Processing

When a CTS is started (recall that there is a separate CTS co-located with each PEP) it calls the RSA supplied start-up function to initialize the cryptographic module. The RSA start-up function will perform a self-check, including the verification of the module software license. Only once the library has been initialized can it be used for cryptographic functions like encryption and decryption.

Once the RSA library is successfully initialized, the CTS connects to the SMSB and waits for cryptographic request messages from the local PEPs. When a cryptographic request message is received, the content is extracted to determine the request type, as specified in the *action* element in the XACML context message format. Additionally, the CTS extracts the *resource* elements where the first element is the cryptographic operation *source* and the second element is the cryptographic operation *target*.

If the requested *action* is COPY_ENCRYPT the CTS will perform the following operations:

1. Extract the security label from the *Environment* element of the XACML message.

2. Formulate and send a GENERATE_STORE message to the KMS and receive a key and token in response. Since the CTS is being requested to create a new encrypted asset, a new unused key is required from the KMS.

3. Use this new key in a call to the RSA library to transform the source file into an encrypted version of that file.

4. Generate a hash digest of the encrypted file using the methodology described in the previous section.

5. Create a new container file (ZIP archive) with the name of the target file and add the following data to the container: the encrypted payload, the hash digest, the original filename of the file (taken from source) and the label.

Once the container has been successfully created, the CTS can return a status message to the PEP indicating that the operation if complete. Any errors encountered in the encryption process will be reflected in the response message to the PEP.

If the requested action is COPY_DECRYPT the CTS will perform the following operations:

1. Perform a digest hash of the contents of the container and compare the result to the digest that is stored in the container to ensure that the container has not been tampered with.

2. Retrieve the token from the container.

3. Use this token in a RETRIEVE_KEY message and send this message to the KMS. The KMS will return the associated key for this token.

4. Use this retrieve key in a call to the RSA library to transform the source file into a decrypted version of that file.

Once the original file has been successfully decrypted, the CTS can return a status message to the PEP indicating that the operation if complete. Any errors encountered in the decryption process will be reflected in the response message to the PEP.

If the requested action is FILE_ENCRYPT_TOKEN the CTS will perform the following operations:

1. Extract the token from the *Environment* element of the XAMCL message.

2. Use this token in a RETRIEVE_KEY message and send this message to the KMS. The KMS will return the associated key for this token.

3. Use this retrieved key in a call to the RSA library to transform the source file into an encrypted version of that file.

Once the encrypted file has been successfully created, the CTS can return a status message to the PEP indicating that the operation if complete. Any errors encountered in the encryption process will be reflected in the response message to the PEP.

If the requested action is FILE_DECRYPT_TOKEN the CTS will perform the following operations:

1. Extract the token from the *Environment* element of the XAMCL message.

2. Use this token in a RETRIEVE_KEY message and send this message to the KMS. The KMS will return the associated key for this token.

3. Use this retrieved key in a call to the RSA library to transform the source file into a decrypted version of that file.

Once the decrypted file has been successfully created, the CTS can return a status message to the PEP indicating that the operation if complete. Any errors encountered in the decryption process will be reflected in the response message to the PEP.

Service Output

While CTS cryptographic operation requests uses an XACML context message format, the CTS uses the SSR message format for cryptographic responses. The SSR format is an XML-based message format with name-value pairs to specify the returned values. All SSR messages are comprised of a list element that contains the name-value pairs and a status element that reports the completion state of the response: success or error.

Each SSG encodes the list element of the SSR message with a unique name to distinguish between message usages. For the CTS, all SSR list messages are defined as “cryptoOP” responses.

The following table describes the name-value pairs that are returned for each key cryptographic operation:

Table 12: CTS Response Messages by Cryptographic Operation

Cryptographic Operation	Named Value	Value
COPY_ENCRYPT	target	Path to the Container File
COPY_DECRYPT	target	Path to the Plaintext File
FILE_ENCRYPT_TOKEN	target	Path to the Ciphertext File
FILE_DECRYPT_TOKEN	target	Path to the Plaintext File

The CTS will return an error code to the caller if any of the following conditions are encountered:

- Message cannot be parsed due to a malformed request;
- No action element was specified in the request or the action is unsupported;
- The source file for the operation cannot be accessed; or
- An error occurred in the external RSA library.

4.5 Secure Labelling Service (SLS)

As described in the *Architectural Design Document*, the Security Label Service (SLS) is one of the central SAMSON security services and is based on the general SSG design. At an architectural level, the SLS is responsible for:

1. Extracting the security label from data assets; and
2. Validating the contents of a secured object to determine if the label is correct for the asset it references.

When evaluating a user’s request for data, the PEPs must be able to submit a policy decision request to the AS. This decision request must include the security label on the

asset being requested. The PEP, therefore, makes a call to the SLS that will extract security label information from the information asset.

In the SAMSON TD deployment architecture, the SLS only operates on file assets. The extraction of label information from other asset types, including email messages, chat rooms and web sessions, is performed by the PEP for those assets. This topic is discussed in more detail in section 5.1: Generic Design of a Policy Enforcement Data Intercept and in the sections that describe the individual PEP architectures.

The SLS Service is a bridging component that will link SAMSON Security Label request messages to software modules or external services that are able to interpret the security attribute information that is bound with information assets. For the SAMSON TD deployed architecture, the SLS includes, as part of its processing capabilities, file handling routines that are embedded with the SLS and can interpret file formats and extract security label information for the supported file types.

The SLS must be able to interpret two general classes of files:

- Files that are not currently protected by SAMSON and, therefore, are labelled using a structure that it not specified by the SAMSON architecture; and

- Files that have been protected by SAMSON and have security attribute information stored in a format that is defined by the SAMSON architecture.

SAMSON is able to process information assets that have been labelled by the user at the endpoint using a COTS security labelling solution, but it is the role of the SLS to understand and interpret that label information. Separate SLS implementations may therefore be needed to handle different labelling formats. The SAMSON TD deployment architecture includes file handling routines that can interpret the format of files that have been labelled using the Titus suite of security labelling products. Other labelling solutions can be enabled for SAMSON, but would require the creation of file handling routines that can interpret that solution's file label format.

While it is the role of the SAMSON SLS Service to retrieve the security label on a data asset, the SLS also is the logical place where validation of the security label can be performed. Validation, in the case, is an evaluation performed on the asset to ensure that the security label properties accurately reflect the information content of the asset itself. For example, a file with a community membership that included Canadian non-citizens should not include information in the national interest. It is important to note that verification, the process of ensuring that the security has not been altered, must always be done prior to releasing information to the SAMSON user. Validation may also be a batch, or offline, procedure that is performed against all information assets according to a defined schedule.

4.5.1 *SLS Design Considerations / Configuration*

Co-location of the SLS with the PEP

Similar to the deployment architecture of the CTS, when designing an SLS service a decision must be made to either move the data from the PEP to the SLS for processing or move the SLS to the PEP so that both services are co-located and information can be passed between these components as local absolute file references. The SAMSON TD SLS deployment architecture uses the same deployment design as the CTS, namely, separate SLS services are co-located with each PEP.

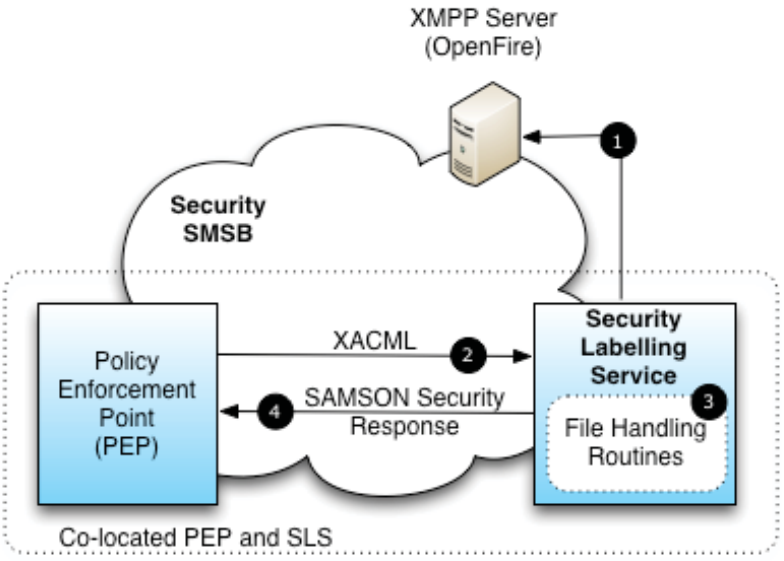


Figure 15: SLS Deployed Architecture for Each PEP

1. As an SSG, the SLS will connect and authenticate to the Security XMPP server in order to become a participant on the Security SMSB. Once connected to the SMSB, the SLS waits for security label operations requests.

2. In the deployed architecture, the sole client of each SLS is the co-located PEP; when a security label operation is needed against a file, the local PEP sends an XACML context request message that contains the security label operation request to the SLS (2).

3. The SLS performs the action against the data asset referenced in the request.

4. The SLS sends a status response back to the calling PEP in a SAMSON Security Response message (4). The format of the context request and security response messages is described in Annex A.5: Security Label Service Messages.

Based on this design, therefore, an individual PEP may be deployed with:

- A co-located SLS if the PEP requires security labels extracted from files; and
- A co-located CTS if the PEP must do cryptographic operations on data.

In the case of the SAMSON TD deployment architecture:

- File Share PEPs require labelling and cryptographic services and are deployed with a co-located SLS and CTS;
- Instant Message PEPs require no labelling services (label extraction from the chat room is performed within the PEP itself) but do require cryptographic services and are deployed with a co-located CTS; and
- Web Session PEPs require no labelling services (again the PEP extracts the label) and no cryptographic services; this class of PEP is deployed without co-located supplemental services.

External Label File Format

When a PEP receives a file for storage in a SAMSON protected file share, the PEP must make a policy decision to determine if the transaction can take place according to the security policy. To formulate this policy decision request, the PEP must call the SLS to extract the label from that file. This file must be properly labelled using the **external label file format**.

For the SAMSON TD deployment, a file using the external label format is a ZIP archive that includes a directory named *docProps* and a file in that directory called *custom.xml.* Within this XML formatted file, the following properties are defined:

```
<Properties xmlns=http://schemas.openxmlformats.org/officeDocument/2006/custom-properties
xmlns:vt="http://schemas.openxmlformats.org/officeDocument/2006/docPropsVTypes">
<property fmtid="{D5CDD505-2E9C-101B-9397-08002B2CF9AE}" pid="2" name="TitusGUID">
<vt:lpwstr>432af8d0-5dc5-4165-ae00-b66b25221490</vt:lpwstr></property>
<property fmtid="{D5CDD505-2E9C-101B-9397-08002B2CF9AE}" pid="3"
name="SAMSONDEMOCLASSIFICATION"><vt:lpwstr>CLASSIFICATION</vt:lpwstr></property>
<property fmtid="{D5CDD505-2E9C-101B-9397-08002B2CF9AE}" pid="4"
name="SAMSONDEMOCAVEATS"><vt:lpwstr>CAVEATS</vt:lpwstr></property>
</Properties>
```

The SAMSON deployed architecture, specifically the SLS, uses values stored in the property with the name *SAMSONDEMOCAVEATS* as the caveats to be associated with the file.

The Titus document classification plug-in for the Microsoft Office suite of products allows the creation of labelled: Word, Excel and PowerPoint files that adhere to this file label format.

The Titus file labelling software, therefore, is a facilitating component that should be available to SAMSON users at the endpoint.

It is significant to note that this section describes files as having security labels. While this label format includes classification information, the currently deployed SLS architecture only uses caveat information on files. It is anticipated that SAMSON will reside on a SECRET network where all labelled information will be at the SECRET level.

Internal Label File Format

While stored within the protected SAMSON space (e.g. stored on a file server protected by a SAMSON PEP) files have the **internal label file format**. This file format is the format that the CTS uses when it creates a container object. As specified in section 4.4:Cryptographic Transformation Service, these containers are a ZIP archive and contain, in addition to the encrypted object, a file called *caveats* that stores the caveat that was present on the file when it was first encrypted. The contents of the caveats file is extracted by the SLS when working on an internal labelled file.

Verification

In the current deployed architecture, it is the CTS that performs the verification activity. The CTS ensures, through the creation of and subsequent testing of the hash digest on the file to ensure that the container and label has not been tampered with. The SLS will be expanded to include the ability to check the integrity of a security label, however, this capability is not present in the SLS in the deployed architecture.

Validation

The SLS in the deployed architecture has the ability to plug-in additional processing modules that can perform advanced reasoning on documents to ensure that the label on the data is correct or to recommend a label for a document that is not labelled. One such plug-in, a Naïve Bayesian Classifier, has been developed for this architecture, however, no plug-ins are currently included in the SAMSON TD architectural deployment.

4.5.2 *SLS Messaging and Operation*

Service Input

The SLS expects the PEP to express security label request commands in the form of XACML 2.0 context request messages and will, in turn, supply the requested security label in SAMSON Service Response (SSR) format. All traffic between the PEP and the SLS is made over the SMSB such that the XACML context messages are encapsulated within XMPP message structures for delivery.

Only one request message is currently supported, namely, a request to retrieve a label from a specified file. The context message takes the following form:

Table 13: SLS Request Message Content by Message Type

Message Type	Subject Field	Resource Field	Action Field	Environment Field
Extract the security label from the file	N/A	The target file being queried	FILE_GET_LABEL	N/A

Within this XACML context message structure, the *resource* element is the absolute path to the file whose label is being requested and the *action* element is FILE_GET_LABEL. When received, this message instructs the SLS to retrieve the security label from the file. It is the responsibility of the SLS to determine the file type and to engage the correct label parsing routines to extract the label from the information asset. It is anticipated that additional actions for the verification and validation of labels will be added in subsequent revisions of the SLS.

Service Processing

When an SLS is started, it connects to the SMSB and waits for file label request messages from the local PEPs. When a file label request message is received, the content of the message is extracted to determine the request type, as specified in the *action* element in the XACML context message format, and the target file for the operation as specified in the resource element.

If the requested *action* is FILE_GET_LABEL the SLS will perform the following operations:

1. Determine the type of the target file.
2. If the file is an external file (Microsoft Office file with a security label) the SLS will open the custom.xml file in the file's archive format and retrieve the caveat list from the SAMSONDEMOCAVEATS property.
3. If the file is an internal file (SAMSON protected file) the SLS will open the container (archive file) and retrieve the caveat list from the caveats file.

Once the caveats have been successfully retrieved, the SLS can return a status message to the PEP indicating that the operation if complete. Any errors encountered in the retrieval process will be reflected in the response message to the PEP.

Service Output

While SLS label retrieval requests use an XACML context message format, the SLS uses the SSR message format for the response. The SSR format is an XML-based message format with name-value pairs to specify the returned values. All SSR messages are comprised of a list element that contains the name-value pairs and a status element that reports the completion state of the response: success or error.

Each SSG encodes the list element of the SSR message with a unique name to distinguish between message usages. For the SLS, all SSR list messages are defined as "assignedlabel" responses.

The following table describes the name-value pairs that are returned for each key label retrieval operation:

Table 14: SLS Response Message Content by Message Type

Response to Request Message	Response Type	Response
GET_FILE_LABEL	caveat	Comma separated list of caveats on a file.

The SLS will return an error code to the PEP if any of the following conditions are encountered:

- Message cannot be parsed due to a malformed request;
- No action element was specified in the request or the action is unsupported;
- The source file for the operation cannot be accessed; or
- There is no label present in the file.

4.6 Trusted Audit Service (TAS)

As described in the Architectural Design Document, the Trusted Audit Service (TAS) is one of the central SAMSON Security Services based on the SSG design and is the key service for maintaining and demonstrating the integrity of SAMSON information protection processes. The TAS supports the integrity of the SAMSON trust model through the creation of audit records that are linked via a chain-of-custody to ensure tampering has not occurred within the audit trail. The audit records that are protected and stored though the TAS keep a transactional history of the policy decisions and access control enforcement across all SAMSON protected resources. Since all policy enforcement activities are recorded within the TAS and the integrity of the TAS can be demonstrated, the TAS can be used to track information access requests and the rationale for why information was disclosed to SAMSON users.

The TAS extends to other SAMSON components the ability to receive and store transaction data associated with information requests. The information that is audited is intended to be a trusted and tamper-resistant record of auditable actions that have taken place within the SAMSON protected environment. Processing the incoming audit information and augmenting individual records with supplemental information that allows the integrity of the record to be asserted and the integrity of the stream of audit records to be similarly maintained achieve the SAMSON design goals for a tamper-resistant audit trail. Tampering of audit information, including the insertion, modification or deletion of audit records, can be detected by integrity verification routines that provide:

- Real-time or scheduled checks of the audit store integrity;
- Complex audit store analysis to detect suspicious patterns of activity; and
- Incident management procedures, including notification capabilities when security events are detected in the SAMSON environment.

The clients of the TAS are the SAMSON PEPs. The PEPs intercept data traffic and call SAMSON security services to provide data-centric information protection. In this light, the PEPs are the central coordinating point to drive the information protection logic and are the best source for audit record information. A PEP that receives an information request will generate an audit record to state whether the transaction was permitted or denied as per the security policy, this transaction includes the ancillary meta data to support a complete audit record (Annex A.6 : AuditXML Schema). The PEP will also create audit records for error conditions that are reported from SAMSON security services. For example, if a file's security label cannot be extracted, SAMSON cannot make a policy decision regarding this file and the transaction should be denied. Such a transaction is submitted to the TAS with the error condition reported within the record.

All audit traffic is transmitted over the Audit SMSB. Since the PEPs are the only components that submit audit records, the PEPs are the only components, other than the TAS itself, that are participants on the Audit XMPP domain. This is shown in Figure 3: SAMSON SMSBs as XMPP Domains.

It is significant to note that each PEP will connect to the Audit SMSB with its own unique session; each PEP, therefore, becomes a trusted audit client (TAC) of the TAS. Each TAC generates and submits audit records that can be distinguished from those of other TACs. The sub-module code required to act as a TAC is embedded within the general PEP design.

Architecturally, the PEPs submit audit records to the TAS and the TAS stores the processed (integrity protected) audit records in an independent repository. This architecture can be seen in the following diagram:

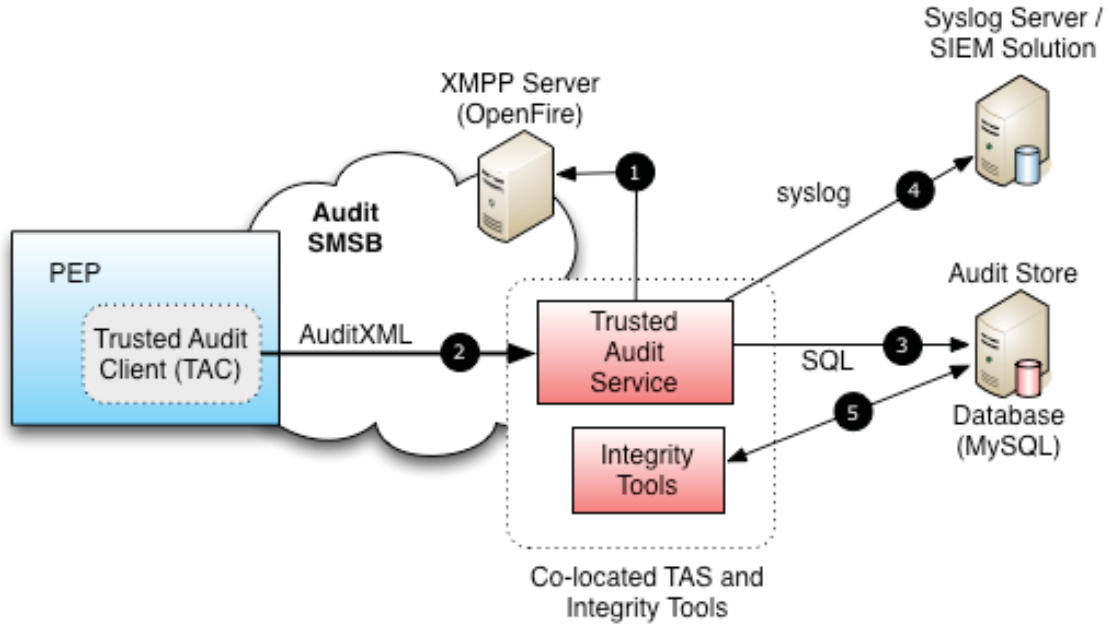


Figure 16: TAS Deployed Architecture

The TAS leverages a database for the storage of audit records. The SAMSON TD architectural deployment uses a MySQL based audit storage facility and audit records are submitted to the audit store using SQL on the audit network. While this audit network is the same network that hosts the audit SMSB, these SQL insertions are not made using the XMPP messaging infrastructure; they are made directly to the database itself. The audit network remains, however, a segregated network that is isolated from the user community and operational data services.

1. As an SSG, the SLS will connect and authenticate to the Audit XMPP server in order to become a participant on the Audit SMSB.

2. Once connected to the SMSB, the TAS waits for AuditXML formatted audit records that are sent from the PEPs.

3. The TAS processes the audit record: extracting the audit record content, extending the audit data to include chain-of-custody integrity protections and submits the record to the audit repository for storage.

4. If the record contains information that should raise a security event (e.g. exceptional policy violation, error conditions), a syslog record is created and sent to a syslog service. In the SAMSON TD architectural deployment, AlienVault, an open source SIEM solution, which will raise a security event from this syslog records and send a notification email to the security officer.

5. Audit integrity tools can then called on-demand or according to a scheduled basis to periodically verify that the integrity of the audit chain has not been broken and that audit records have not been illicitly modified.

Audit records are reviewed using an administrative interface documented in section 6.3: Audit Review Interface (ARI). Similarly, the design of the integrity verification tools is described in section 6.4: Audit Integrity Checker (AIC).

4.6.1 *TAS Design Considerations / Configuration*

The message flow from the PEP to the TAS leverages the trusted delivery mechanism provided by the XMPP architecture. When a PEP sends a message on the Audit SMSB, the XMPP server ensures that the message reaches the TAS so long as the TAS is present on the XMPP domain. As with the Security SMSB, participants on the Audit SMSB use TLS protection to ensure the confidentiality and integrity of their sessions. The use of TLS between the PEP and the TAS adds to the chain of custody protection and integrity of the audit records created within the SAMSON architecture.

The exchange between the PEP and the TAS is one-way; audit records are submitted to the TAS, but the TAS does not send any receipt message. This “fire and forget” model for the TAS is an intentional design decision for the following reasons:

1. Adding a response message will double the traffic on the audit network;

2. If a transaction cannot be audited, the only thing a PEP could do is deny the transaction; this is seen as an unacceptable point of failure for SAMSON (if the TAS is disabled, all SAMSON activity is disabled); and

3. The XMPP server can queue messages for delivery, holding those records until the TAS is available without the loss of audit data.

If there is a need to disable all PEPs if the TAS is not available, a better approach would be to have the PEPs acts on XMPP presence messages: announcements from the XMPP server that the TAS is no longer present. The SAMSON TD currently does not provide this behaviour.

Event Notification: The TAS supports security event notification. When an error condition is encountered, the TAS will create and submit a TAS message to an appropriate facility within the deployment environment. Error conditions are raised whenever a SAMSON component fails to service a request (e.g. a back end database is offline) or when there is a violation of implicit security logic. For example, the policy denial of the retrieval of a file from a SAMSON file share should not happen and may be indicative of malicious activity. In such a case, the TAS would submit a syslog message in addition to an audit record. The TAS configuration specifies the target syslog service. In the SAMSON deployment architecture, syslog messages are directed to an AlienVault 4.1 Open Source SIEM solution that accepts

TAS error conditions raises and alert about this condition and send an email message to the security officer. These solution components are described in section 6.5: SAMSON Security Event Management.

Protection of the Audit Storage: The TAS design does not place any restrictions on what storage facility is used to house the audit records. Any chosen storage facility will need to be configured before it can be used with TAS, for example, an audit repository database will need a schema that maps to the AuditXML structure and the associated extended security attributes. The selection of the facility and the security protections that SAMSON places on the audit records, industry-best practices should be applied to the configuration, deployment and operation of the storage repository to further ensure the confidentiality, integrity and availability of the audit records.

Audit Administration: The interface through which audit records are review must be, itself, a SAMSON policy-based data exchange. That is, the interface that is used to perform audit analysis, audit review and forensic activities must enforce the domain's security policy. As an example, the policy right to access audit records could be assigned to a specific community; only those users that belong to that community will have access to the audit analysis interface. Use of this interface would then be subject to all SAMSON protection from access control to auditing[4]. For the SAMSON TD deployment architecture, audit records are reviewed using an administrative interface documented in section 6.3:Audit Review Interface.

4.6.2 *TAS Messaging and Operation*

Service Input

Audit records are sent from the PEP to the TAS as AuditXML records. AuditXML is an XML-based message format that has been designed for the submission of SAMSON audit records. The full specification of the AuditXML schema is listed in Annex A.6: AuditXML Schema, however, the following table provides a description of the significant data elements that a PEP must populate into an AuditXML record before it is submitted to the TAS.

[4] Note that in this case, access the audit interface is an auditable event, a clear case of SAMSON protecting SAMSON that is part of the technical demonstrator architectural objectives.

Table 15: AuditXML Elements

AuditXML Element	AuditXML SubElement	Description
Principal	userId	The identity of the SAMSON user that initiated the data request associated with this audit record. Naturally, this is the identity that was used in the policy check to the AS.
	ipAddress	The address of the PEP that submitted the audit record.
	programName	An identifier of the type of PEP that submitted the audit record. For example 'fileshare' for all PEPs that provide file-based information protection.
Action	Operation	What action was performed on the data. In this context, the action is the policy action (e.g. READ,WRITE) that was submitted to the AS for a policy decision.
	Target	The resource against which the policy decision was made. In this context, the target is the security label of the asset against which the policy decision was made.
tacOrigin		A identifier that unique differentiates the audit record created by this TAC from those of other TACs.
Notes		A free form section in which per operation or per file type data can be inserted into an audit record (see below)
Timestamp		A timestamp on the audit record, referencing the time when the audit record was created.

Of significant interest is the notes element. As new PEP are created and introduced to the environment or new actions are made policy enforceable, new auditable information is generated. For example, in a file share PEP it is desirable to audit the name of the file that has been released to the user. This is a piece of audit information that is only relevant to file sharing. Similarly, a decision to allow a user to join a chat room created audit information that is only relevant to that action on that data type. The chain-of-custody audit protection approach requires, however, that there be only one audit repository (database table) rather than a separate repository for each type of audit record. To resolve this challenge, a free form (name-pair) field was added to the AuditXML schema. Each audit record contains, therefore, a section of mandatory fields that must be populated for each audit event and an element comprised of freeform audit notes that contain audit information that is relevant to that record type only. In this way, a balance between flexibility and standardization is achieved for audit record expression.

Note that in the SAMSON TD deployment environment, the virtualization infrastructure is time synchronized (all virtualization server use a common time source) and all virtual machines inherit the time from their virtualization host. In this way, all components in the SAMSON TD are time synchronized and the timestamp generated by the PEPs reflect the time that the auditable transaction took place. In a deployment environment, the virtualization components would leverage a trusted time source for time synchronization.

Service Processing

Upon receiving an AuditXML record from a TAC (the auditing software module within a PEP), a TAS will perform the following actions.

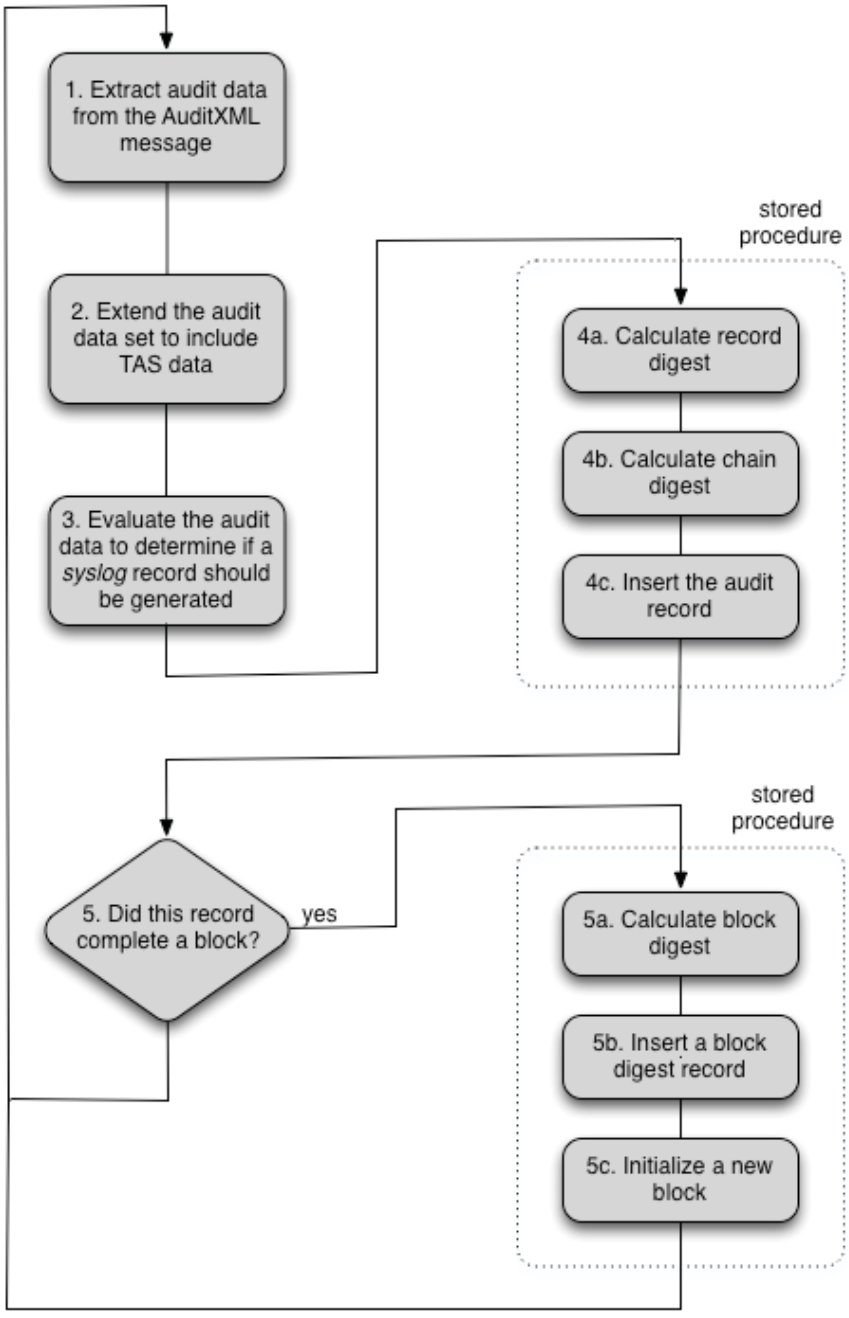


Figure 17: Audit Processing Logic

Step 1. Extract the audit information: The AuditXML structure is extracted and placed into a local data structure that will be used to create a database record that can be inserted into the audit store.

Step 2. Extend the local audit data to include the TAS relevant audit information: The audit record that is inserted into the audit store is a combination of client information and audit server information. The TAS data that is added to the audit record includes:

- An identifier for the TAS that accepted the audit record; the SAMSON TD design supports the structure of many TACs submitting records to a TAS and multiple TAS that submit audit records to the audit store. By adding the TAS identity to the audit record it is clear which TAS submitted the audit record.

- A TAS timestamp is also added to the record to show when then record was added to the store. If there are processing delays or buffering, the TAC timestamp and TAS timestamp may be different.

- The TAS also enumerates the audit record with the currently active block sequence and provides a unique block number for the record. Audit records are placed in blocks and integrity is maintained on a per-block basis. Tampering of an audit record will violate the integrity of a block, but not the entire audit trail.

Step 3. Evaluate the audit record to determine if it should be logged for notification: Currently, the TAS logs for notification all audit records that reflect an error condition. If an audit record contains an *errorcode* element within its notes element, a syslog message is generated and sent to the syslog server as defined in the TAS configuration. The syslog message is a comma-separate message that contains:

1. The SAMSON user that initiated the transaction that raised a notification.

2. The SAMSON protected resource that was requested in the transaction.

3. The action on the resource that was requested in the transaction.

4. The IP address of the PEP that submitted the audit record for this notification.

5. The PEP application that submitted the audit record for this notification.

6. The PEP command that was requested by the data intercept.

7. The error code that was sent in this audit record.

8. The error text associated with this audit record.

The submission of a syslog message is a *fire and forget* protocol. Syslog records are sent, but no response is expected from the syslog server. For the SAMSON TD architectural deployment, the processing performed by the syslog service, the SIEM solution and the generation of notification messages is described in section 6.5: SAMSON Security Event Management.

Step 4. Create the digests and insert the audit record: The calculation of digest values to add integrity to the record and the submission of the record to the audit store are performed by a stored procedure. Messages are grouped into blocks so that integrity violation can be contained. That is, if a record is tampered with the TAS is able to re-assert integrity by restarting the chain after a certain number of records. The number of records in a block is configurable. If the most recently added record completes a block, a block digest is created and stored and a new audit block is initialized.

Each audit record contains a hash over the concatenation of its own fields following those of the audit record before it (or "0" if there is no antecedent). The first record of an audit block (after the first block) chains the last record of its preceding block. This approach protects the integrity of each record, since any alteration of a record would break own hash and the hash on the following record.

Each audit record is numbered in sequence, per audit block, and each audit block is numbered in sequence, per TAS instance. This approach provides evidence of any insertion of spurious records or deletion of audit records. Furthermore, any such malicious activity would break the hash of the first record in the next block. This is shown in Figure 18: Audit Record Digests.

Each audit record in the database contains two SHA-160 message integrity digests.

1. The TAS calculates the record digest over the concatenation of its own fields.

2. The TAS calculates the chain digest over the concatenation of its record digest and the chain digest of the preceding record. The first record of an audit block uses "0" for its chain digest: this design limits the cascade of any tampering event to (the record and all subsequent records of) its own block. This approach protects the integrity of each record, since any alteration of a record would break own record digest and the chain digest on the following record.

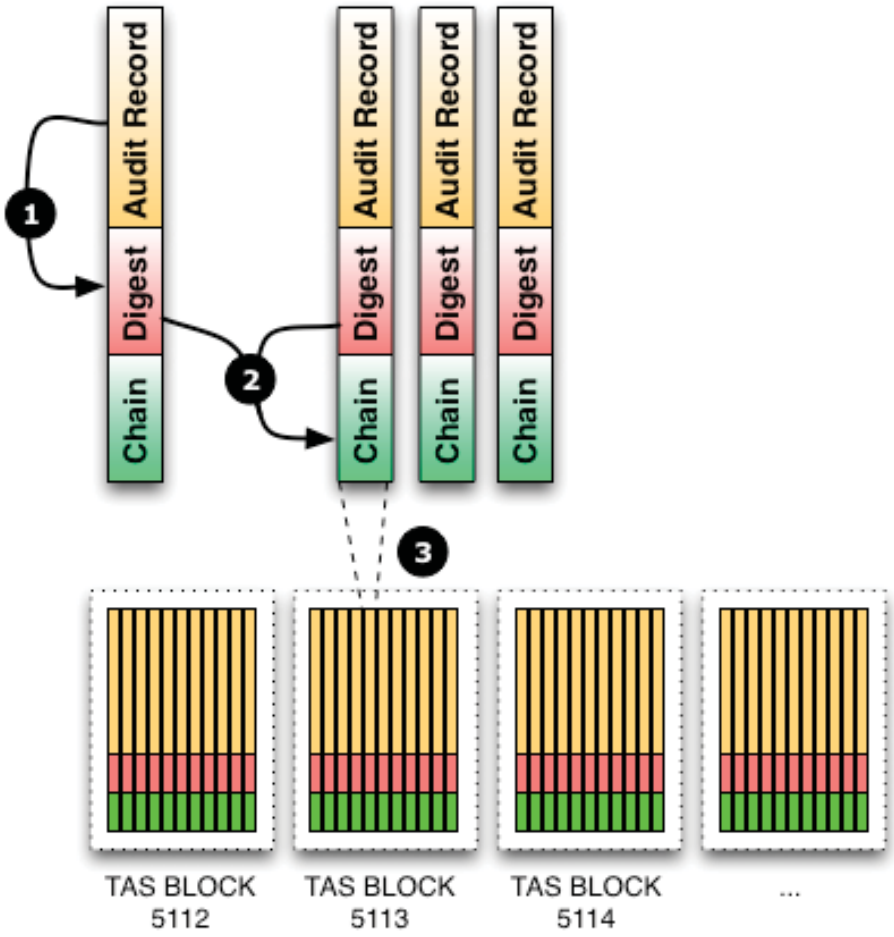


Figure 18: Audit Record Digests

3. Once the digests have been calculated, the record is inserted into the audit store (3). The store procedure will return a status message that will indicate if a block has been fully populated.

Step 5. Processing of a Completed Block: If, after the insertion of the most recent record, a block contains its maximum number of records, the block is locked down a new block is started. Creating a digest across all records in the block locks a block. The intent is to provide a digital signature on the block digest; however, this is not currently in place within the SAMSON TD architectural deployment. Once a block is locked, a new block is initiated and audit records are added to this block.

5.0 The SAMSON Data Intercept Strategy

The components previously described in this document form the core of the SAMSON messaging infrastructure. This list of components includes:

1. **A Secure Messaging Service Busses (SMSBs)**: The ability for SAMSON components to exchange data in a manner that is secure, protocol agnostic, and reliable.

2. **Security Services Gateways (SSGs)**: The ability to bridge between the SAMSON security architecture and the back end (non-SAMSON) applications that provide the needed security functionality.

3. **Policy Enforcement Points (PEPs)**: The ability to link external application and security services to the SAMSON infrastructure in a manner that adheres to the SAMSON security protection principles.

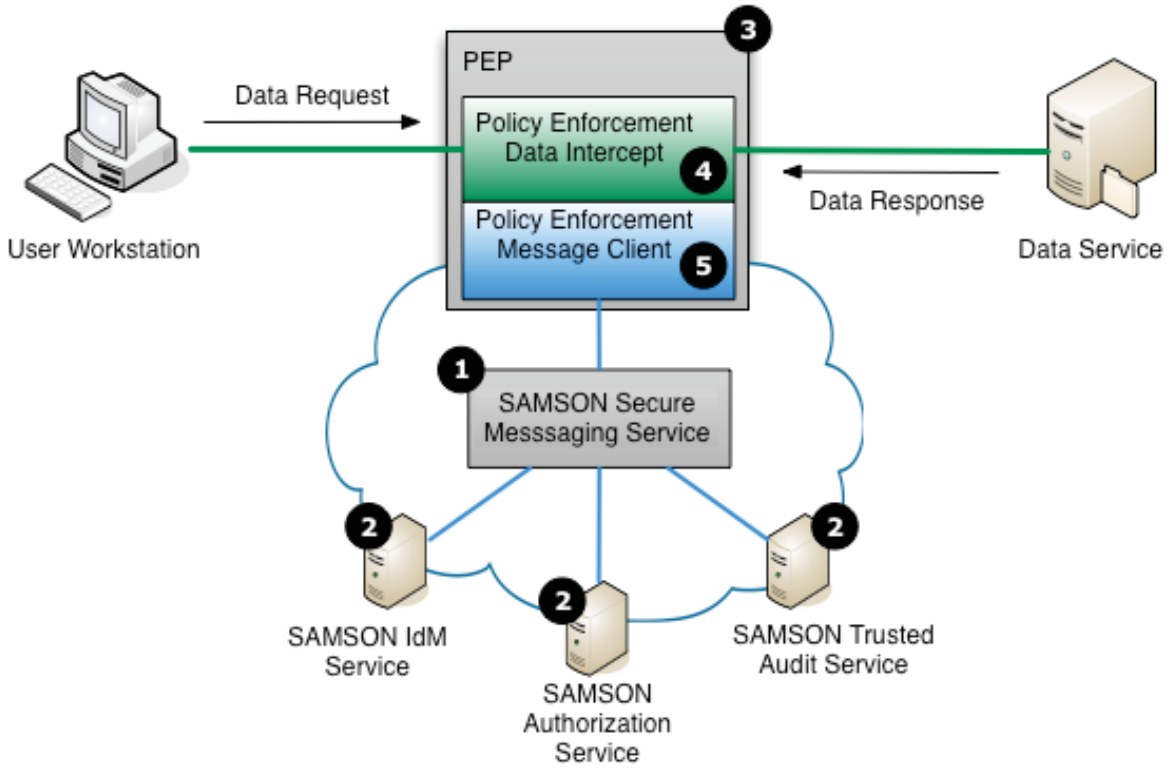


Figure 19: The Policy Enforcement Data Intercept

This section describes the PEP component architecture and provides design details for the PEPs that have been included in the SAMSON TD architectural deployment. As a preliminary observation on PEP design it should be noted that PEPs are defined by two significant sub-components, also shown in Figure 19: The Policy Enforcement Data Intercept:

4. The **Policy Enforcement Data Intercept (PEDI)** that intercepts traffic between the user's endpoint workstation application and the target data service; and

5. The **Policy Enforcement Message Client (PEMC)** that performs the exchange of SAMSON security messages with other components over the SMSB.

These two sub-components, the components that are most closely linked to the user's session, user data requests and information assets to be protected, are described detail in the following sections.

5.1 Generic Design of a Policy Enforcement Data Intercept

This section describes the generic design for a PEDI, identifying the required capabilities that a PEDI would need in order to protect information assets by leveraging the SAMSON security services via the PEMC.

The PEDI performs the following functions:

- Intercepts information requests that are sent from the workstation to the target data server;

- Collects information that is needed to formulate a policy request;

- Sends the policy request data to the PEMC. The PEMC, as previously discussed, implements the information protection logic to ensure that the data request is handled correctly and exchanges the necessary messages with other components to leverage SAMSON information protection services; and

- Based on the response from the PEMC, the PEDI will allow the information request to proceed: allowing data to be sent to the back end server (i.e. a file upload) or returning data back to the user (e.g. a directory listing).

From this description of PEDI operations, it is obvious that it is the sub-component that works most closely with the data service itself. The nature of the PEDI architecture can be described in terms of: the data asset profile, the data service security functions, security operations on the data and the intercept network architecture. Each of these aspects of the PEDI's design is described in detail in the following sections.

5.1.1 *PEP Data Assets*

In order to be able to protect an application with SAMSON services, the granularity of the data asset on which the security operations are performed must be determined. For file sharing services, this granularity was chosen to be at the individual file level, but other file sharing constructs, such as directories and paragraph-level, labelling could also be considered when defining the list of data assets that are to be protected by the SAMSON file sharing PEP. Similarly, a PEDI that is intercepting email messages would need to operate not only on email messages, but also on all files that are attached to messages. A PEDI that is protecting instant messages may work on individual IM messages or at a higher order collection of messages, such as a protected chat room. The selection of the data asset granularity and the selection of data asset types that are protected by the PEP will dictate the nature of the data intercept that must be provided by the PEDI.

For SAMSON to be able to exercise policy enforcement on data assets, it must be possible to associate a security label to each individual asset. In some cases, there are third party or COTS solutions that will allow the user to apply a label at the endpoint. In other cases, the data label may have to be applied at the server itself. For example, a SAMSON protected data, accessed by users though a subscription, will have to be initially labelled at the source since the end user it merely a consumer of the data. The label itself must be able to reflect the nature of how the associated data is used and the policies that will apply to that data asset. For example, there may be the need to apply multiple caveats within the security to reflect that the asset is accessible to multiple communities. The approach taken by each PEP in the SAMSON TD architectural deployment for the security labelling of data assets is described in section 5.3: PEP Implementations.

In any event, the label applied to the data must be able to be determined either by:

- the SLS, the service that extracts, verifies and validates the security label on the data; or
- the PEDI itself.

For certain applications, such as IM chat rooms, the PEDI is better suited to determine the label on the data assets being protected since the PEDI can query the IM server directly to acquire the label on the chat room. In this case, the label is attached to chat room properties at the server, rather than being combined with the data asset itself as is the case with file and email protection.

If the information protection logic calls for a policy-based access check to be made on a file before it is delivered to a client, the PEMC will need to obtain the label on that file in order to

formulate a policy request to be sent to the Authorization Service. The PEMC extracts the security label on the file by calling the SLS.

In a similar vein, the security label must be cryptographically bound to the data that it references. In the context of the SAMSON file sharing PEP, a file contains, within its metadata structure, the security label of the file. Hence, when SAMSON encrypts a file, the security label stored within that file is encrypted along with the file. It is, therefore, not possible to modify the file or the security label since that are stored in a protected and bound form at rest.

While it is the role of the SLS to verify and retrieve the security label on a data asset, the SLS also is the logical place where validation of the security label can be performed. Validation, in the case, is an evaluation performed on the asset to ensure that the security label properties accurately reflect the information content of the asset itself. For example, a file with a community membership that included Canadian non-citizens should not include information in the national interest. It is important to note that verification, that is, ensuring that the security has not been altered, must always be done prior to releasing information, but validation may be a batch, or offline, procedure that is performed against all information assets according to a schedule.

When an information asset fails a verification test, this is a condition that must be sent to the security officer role, as it represents a potential attempt to infiltrate the security environment and obtain illicit access to protected materials. Such a scenario may require the initiation of an incident management process and, potentially, forensic analysis. A failed validation is another security concern that should be flagged to either the data owner or the security officer. This event does not denote an attack against the environment but does signal that changes to data classification rules have resulted in the target files being improperly classified. This is a situation that would require rectification before data is improperly disclosed.

5.1.2 *PEP Actions on Data*

Once the scope of the data asset has been determined, the next step is to determine, from a policy perspective, what constitutes the range of actions that can be performed against the data. Specifically, this set of actions represents the actions that require a policy-based authorization check before the operation is performed against the data. For example, in a file sharing PEP, the set of actions against files that could require a policy check include: create, open, copy, save, rename, delete, or update. When designing a PEP, a security architect must consider the following concerns:

- Granularity: If a high degree of control is needed on the operations against data, then more ‘unique’ actions are required. For example, UPDATE, CREATE, RENAME and DELETE may all be considered separate, policy enforceable actions. This would allow a policy author to create very specific policy rights across the community.

- Complexity: More actions result in very complex policies that become challenging to author and maintain.

In operational practice, a balance must be struck between these two conflicting demands: simple policy expression versus sufficiently granular access checks.

The SAMSON TD architectural deployment has expressed all data operations in terms of the policy action of "READ" and "WRITE". In this interpretation, accessing content is a READ operation whereas creating; updating or removing content is always a WRITE operation. For example, the PEP designed to protect email messages equates the sending of an email message as a policy WRITE operation and the receiving of an email message as a policy READ operation.

Table 16: PEP Operations in a Policy Context

PEP	Data Asset	READ Operations	WRITE
File Sharing	Individual Files	Open a file hosted on a SAMSON file share. Copy a file to the endpoint.	Save or copy a file to a SAMSON protected file share
	Directory Listings	List files hosted on a SAMSON protected file share	N/A
Email	Email Message	Receiving an email message.	Sending an email message.
IM	Chat Room	N/A	Joining a chat room
IM	Individual Messages	Receive an individually labeled IM message	Send an individually labeled IM message
Web	Web Service	Access web content hosted at a SAMSON protected web service	N/A

It is also significant to note that a sophisticated set of policy enforceable actions may only be of use in cases where there is an equally sophisticated policy request/response language to submit complex policy expressions and a policy engine that is able to process these sophisticated policy expressions. Policy decisions can be extended but not to include such environment attributes as: roles, dates, times and locations.

An additional consideration when planning PEP enforceable policy actions is audit. There are situations where a PEP will be called upon to perform an operation many times and the auditing of such actions may not produce useful audit records. For example, execution of a listing of files in a directory will require policy check to be performed on each file in order to allow the file to be deployed to the user or omitted from the returned list. In a directory listing of hundreds of files, with each file being evaluated in the policy, a large number of audit records would be generated. The value of the audit data being created must be considered when defining which operations on data should result in the creation of an audit record. Section 4.6:*Trusted Audit Service (TAS)* provides a detailed description of the auditing strategy for SAMSON information protection operations. Additionally, Table 15:

AuditXML Elements provides a list of data elements that are always part of a properly formatted audit record. It is important to note that individual SAMSON security services can add context-specific audit information that is relevant to that service. For example, the absence of a security label on a data object is a labelling specific error condition that can be included as part of the more general audit record format.

5.1.3 *PEP Proxy Architecture*

The most fundamental aspect of PEDI design is the manner in which information requests are intercepted so that:

- Information regarding the request can be collected, including:
    - Who is requesting the data,
    - What asset is being requested, and
    - What action is being performed on the data; and
- The decision from the PEMC can be enforced.

In the majority of cases, the ***data intercept*** strategy requires that the PEP must adhere to a proxy architecture where there are two discrete connections: from the client to the PEDI and then from the PEDI to the back end data service. This architecture assumes that SAMSON is able to intercept the data transport protocol (e.g. HTTP) and interpret the data protocol that is used by the application. SAMSON can support other architectural designs such as multi-tiered web architectures. In such architectures, the middleware acts as a broker between user requests and data services. SAMSON can be made to integrate with this middle layer, applying: policy, protection and auditing services at this central processing point. Architectures of this nature are only possible where the middleware layer supports the leveraging of external processing or security components.

The PEP design will have an impact on the SAMSON deployment architecture. As part of the defence-in-depth approach to security architecture, the concept-of-operations for a SAMSON deployment calls for cryptographic and keying operations to take place in a physically secured environment, that is, an environment separate from operational data processing. A PEP design has data protection operations taking place at the location where it has been intercepted would mean that the PEP must itself reside inside that secured environment. In such cases, the PEDI must be able to support ***intercept isolation***, specifically it must be able to be hosted in a separate zone from either the client application or the data service it is protecting.

One of the basic tenets of the SAMSON concept is the fact that “open” protocols, expressed though RFC, W3C or OASIS specifications for information exchange, both in transmission and in expression of data, can be more easily intercepted and interpreted. More importantly, however, working to an open standard means that a single SAMSON PEP can

be used to protect any number of client applications. For example, SAMSON PEPs that are written to intercept proprietary protocols like SMB and MAPI can only be used with proprietary client applications. Equivalent protocols, such as WebDAV and SMTP, are more universally used and, as a result, provide a greater range of support for applications that can leverage SAMSON information protection. SAMSON is intended as a solution to provide information protection to back end data services; leveraging open standards provides SAMSON with a degree of application or ***intercept independence*** that supports that concept.

A similar perspective can be seen from the workstation application's view on the PEP. A PEDI should be able to provide a client with a completely ***agnostic intercept***. That is, it should be possible for the endpoint applications to experience SAMSON protection in a completely transparent manner. Where a PEDI is providing a proxy-style service, it is possible that the only change at the application is the specification of new connection address information. The fundamental architecture of other services, such as instant messaging, require that the client applications specify the real connection address of the data server, but do support the configuration of proxied connections.

Generally speaking, a completely transparent proxy is possible as long as the following conditions are met.

1. The client application supports proxies connections and can specify a connection information in its own configuration settings;

2. The proxy architecture allows the entire data asset to be transmitted as a complete entity; proxies that forward portions of an asset require that the PEP buffer information as it passed through the PEP. For example, a PEP that supports file sharing must operate on the entire file in order to create an encrypted version of the file. If the proxy architecture only transmits a portion of a file at a time, the PEP will have to buffer the contents until the entire file is received.

3. The back end service does not have proprietary protocol dependencies for the client/server connection.

Complete transparency or the ability to work with applications' configuration options should remain a goal of any PEDI design.

Web service application frameworks are good candidates for hosting PEP capabilities. Where a data service can be supported through a web session, web modules can be specifically written to support SAMSON style protection for that service. These web-based services typically benefit from Windows Domain authentication and TLS-based session security. There is also an advantage of using a common API for PEP functionality; a single instance of PEP code can be used to support multiple, similar data services. The Apache Programmatic Runtime and Microsoft ISAPI modules are examples of web frameworks that can host implementations of PEP services for many data types. For example, the file sharing and web session PEPs are both implementations of Apache modules that supply SAMSON information protection of files and web services, respectively.

5.1.4 *PEP User Community*

A PEDI must be able to know which user has submitted an information request. The determination of the user's identity is necessary for:

- Retrieving security attributes for that user from the SAMSON IdM Service;
- Performing policy-based queried to the SAMSON Authorization Service; and
- Recording the user's activity in the SAMSON Trusted Audit Service.

The SAMSON concept includes the creation of an Authentication Service (a separate service from the Authorization Service), which would authenticate the user's identity over the Security SMSB. This Authentication Service would connect to a back end authentication solution and supply any SAMSON component with a trusted statement of a user's identity. This Authentication Service would be able to leverage any authentication source and extend a common authentication model to all endpoints (e.g. Windows or Linux)

The SAMSON TD architectural deployment, however, only targets Windows endpoints; as a result a modular Authentication Service has not deployed with the SAMSON TD. Instead, the Windows Active Directory has been used directly as the repository of the user's ***domain identity*** and the Windows Domain Controller performs the authentication of the user at the workstation. User **security attribute**, such as nationality, clearance and caveat membership are not stored with the user account, rather, they are stored in the security attribute repository that is accessed through the SAMSON IAS.

Given that there are two locations for user attributes: AD for the user account properties and the security attribute repository for SAMSON specific user attributes, there must be a method by which user accounts can be mapped to the equivalent security repository user entry. For example, the AD *SAMAccountName* property could map to the common name of the security attribute organizational unit within the LDAP server. The significance of this mapping is that this one property that defines a user's identity becomes universally used not only in its data access context at the PEDI, but also across all SAMSON Security Service messages, such as policy checks and audit records.

If the PEDI is using the user's domain identity as their identity within SAMSON Security Service messages, it is essential that the PEDI is able to determine the domain identity in a trusted manner. In other words, SAMSON must be able to authenticate the user's identity so that all subsequent SAMSON operations inherit that same level of trust and accountability. The PEDI must therefore be able to perform ***domain authentication*** against the domain controller or obtain and interpret Windows credentials that were acquired at the

workstation at login[5]. If the PEDI is able to obtain and use Windows domain credentials, the PEDI will have achieved a ***single sign on*** capability that further presents SAMSON as a transparent security solution.

There remains one final point in PEDI design as it pertains to the user community. If the PEDI design calls for two discrete connections: between the endpoint application and the PEDI and then from the PEDI and the back end data service, then the domain authentication will be taken from application side of the PEDI, since that is where the credentials were obtained. Depending on the nature of the PEDI design it may be necessary for the PEP to provide ***identity pass-through***, that is, the propagation of the user's identity or credentials to the back end data service. Without identity pass-through, a connection to the back end service may not be possible or there may be a loss of functionality such as discretionary access controls.

5.1.5 *PEP Data Protection*

The PEDI must be able to transform data as it traverses the PEP with the most significant transformation being ***cryptographic protection*** of data. It is the CTS, operating on files that performs these transformations. As a result, the PEDI will write incoming data to a file and request, through the PEMC, that a cryptographic transformation be applied to the data. To reiterate, when a PEP applies information protection to a file:

1. The PEDI sub-component of the PEP writes the data to a working location;

2. The PEMC sub-component of the PEP calls upon the CTS to transform the file into an encrypted form; and

3. If successful, the newly created object is read in by the PEDI and used as the data object in subsequent operations.

Using the SAMSON email PEP as an example, when the PEDI intercepts an email message being sent, the message is written to a temporary location, a request is made to protect the file, and the resulting encrypted message is read by the PEDI and sent to the message server. The PEDI must be able to support this modification of the incoming data and delivery of a modified payload. After processing the request, all temporary files are removed. It is the fact that the plaintext data, keying information and transformation activities are being performed on the machine on which the PEDI is hosted that requires that machine to have a higher degree of physical and administrative protection.

[5] Note that the acquisition of Windows domain credentials can be enhanced with PKI smartcards and/or biometrics, providing strong authentication that is inherited by SAMSON processes.

Transformation of data by the PEP forms part of the information protection strategy for an application protected by SAMSON. This strategy includes the following elements.

1. SSL/TLS protection of the session between the client and the PEDI. SAMSON is a back end protection solution and while the data asset encryption is applied and removed at the PEDI, the environment security architecture can dictate that all communications with the endpoint must be encrypted using session-based security.[6]

2. Once the PEP has encrypted the data asset, it cannot be disclosed unless it passes back through this or another similar PEP where the original key that was used to protect the asset is retrieved and applied to the cipher text.

3. The delivery of the data from the PEDI to the data service is optionally protected in SSL/TLS session, but this is an additional layer of security since the data asset is already encrypted.

4. At the data server, the data asset can be stored safely in its native format since it remains in an encrypted state.

5. System administrators can work on the data asset in its encrypted form (e.g. backup, move, copy) without ever have access to the information contained in the data artefact.

Based on this general PEDI design, SAMSON ensures that information is protected against disclosure for two reasons:

1. It is not possible to by-pass SAMSON security protections. Once SAMSON has protected a data asset, with each data asset being protected by a unique key, it is not accessible to anyone that has not requested access to the asset through the PEDI. Only the PEDI, which calls upon SAMSON through the PEMC, can initiate the actions on the data that will transform it into an object that can be disclosed.

2. When the PEDI receives a request to disclose a data asset, it calls upon SAMSON services which force policy checking and auditing to be performed before the data is release. When the PEDI delivers data to a user, it is only because a policy check has allowed the disclosure and there is an immutable record in the TAS that provides the details of the request and subsequent decision.

[6] An enhancement to SAMSON to leverage PKI to encrypt the data that is delivered to the endpoint is in early stages of design but is not part of the SAMSON TD target.

5.2 Generic Design of a Policy Enforcement Message Client

The Policy Enforcement Message Client (PEMC) is the second significant sub-component of the PEP design. It is responsible for the exchange of SAMSON security messages with other SAMSON components (SSGs) over the SMSB. It works in close coordination with the PEDI to ensure that information requests are handled in accordance with information handling rules by exchanging the necessary security service request messages with other components to leverage the full set of SAMSON information protection capabilities.

Any PEMC implementation can be described in terms of a set of common design elements that will be shared by all PEMC implementations; these elements include:

1. The message client architecture: the manner by which the PEMC communicates with the SAMSON security services over the SMSB; and

2. The information protection logic: the ordered set of security messages that the PEMC must send in order to enforce the SAMSON information protection policy in accordance with the SAMSON concept of operations.

5.2.1 *PEMC Architecture*

When contrasting the PEDI and the PEMC subcomponents of the PEP architecture it is clear that:

1. The PEDI works closely with the applications to be protected, the user community and the information assets that are to be protected. As a result, the PEDI implementation will vary considerably from instance to instance. For example, a SAMSON protected file share, email server and web-based data feed will be protected using unique PEDI implementations. Since each application will have different data constructs, different connection mechanisms, different protocols and different ways of maintaining user sessions, each PEDI is as unique as the data service it protects.

2. The PEMC, on the other hand, does not need to vary from one PEP instance to another. The role of the PEMC is much more consistent across all PEMC implementations, since the PEMC is interfacing with a known network and service environment, namely, the SAMSON Security Service on the SMSB.

Integrating any new security service, such as dirty word search, into the SAMSON information protection strategy merely requires extending the existing PEMC to make calls to the new service and interpreting that service's responses. Each PEMC, as the component of the PEP that interfaces with other SAMSON components, must support the following functions.

Connecting to the SAMSON infrastructure: The PEMC is required to connect to the SMSB in order to exchange messages with other SAMSON components. The PEMC must either be provided with or acquire the connection information, including:

- Connection information relating to the location of the core SMSB services, namely, the XMPP server;
- Any certificate information that is needed to establish TLS and, potentially, mutually authenticated network sessions;
- The PEP's XMPP identity, or JID, and credentials to establish the XMPP session; and
- The identities of other SAMSON components on the SMSB to which the PEMC will communicate, for example, the identity of the SAMSON Authorization Service.

Similarly to the Service Gateway components, as described in 3.3: *Security Services Gateways*, the XMPP functionality is best implemented by leveraging a 3rd party XMPP library, such as Gloox, Swiften or PyXMPP. Once the PEMC is connected to the SMSB, it can use the standard SAMSON message protocols, as defined in section 4.0: The SAMSON Security Services, to send and receive messages to the SSGs.

Wait for SAMSON application events: The PEDI, in its role of intercepting traffic to apply policy-based information protection, provides the PEMC with the details of a user information request. The nature of the policy check and the nature of the information that is supplied to the PEMC will depend on the application, the transaction and the information type. For example, when requesting a file from a SAMSON-protected file share, the PEDI will submit to the PEMC such details as: the user that originated the request, the file that was requested and the action that the PEDI attributes to this operation (e.g. READ). The supported SAMSON operations on data are described in detail in section 5.3: PEP Implementations.

Exercise the Information Protection Logic (IPL): With the data request in hand, the PEMC will make a series of calls to various SAMSON Security Services. The exact sequence of messages will depend not only on the information type but also the approved procedures, as stated in the concept-of-operations for the target domain. For example, the IPL determines if a particular transaction should generate an audit record and will direct the creation and submission of the audit information should a record be required. Similarly, the IPL monitors the results returned from calls to SAMSON Security Services and will handle errors or exceptions as appropriate. For example, if a file object has no security label, further processing on that file must be suspended. The IPL will direct that an audit record be sent to the TAS and an error message returned to the PEDI.

This topic is covered in more detail in the following section.

Return a Policy Decision for Enforcement by the PEDI: The IPL will almost always include an access control decision. The result of this decision, as determined by the PEMC call to the AS, must be returned to the PEDI so that the user's access request is either honoured or denied. It is the role of the PEDI to enforce this decision.

As a side note, the design of the PEMC must take into account the fact that in multi-process and multi-threaded applications, there may be multiple information requests that are in process or pending at a given time. The PEMC must ensure that IPL-based policy decisions are returned to the correct PEDI routine so that policy requests from the PEDI are always matched with the correct policy decision from the PEMC.

Wait for SAMSON security events: The XMPP infrastructure can also be used to monitor and control the behaviour of all the SAMSON components. The PEMC, just as the Service Gateways, should listen not only for application service requests and messages from other security services on the SMSB, but also for command messages from an administration process that could issue commands across the infrastructure, such as:

- Clear out any cache information;
- Re-read their configuration data;
- Shut down;
- Temporarily halt the processing of SAMSON security messages; or
- Provide a status of their current operating condition.

At this stage, it may be useful to contrast the role of the PEMC with that of the SSGs.

Table 17: PEMCs versus Security Service Gateways (SSGs)

	Policy Enforcement Message Client	Security Service Gateways
Role	To enable SAMSON information protection of the data request/response cycle	To enable the use of external security application by connecting them to the SMSB
Locations	Bridging data application intercepts to the SMSB	Bringing external security applications to the SMSB
XMPP connection	Has a unique XMPP identity on the SMSB	Has a unique identity on the SMSB
Messaging	Sends messages to service gateways to obtain a specific security capability	Responds to messages from all PEMCs to deliver a specific security capability
Auditing	Generates audit messages	Provides information that will be audited by the PEMC

5.2.2 *Information Protection Logic*

Whereas the PEDI sends a single request to the PEMC with a specific SAMSON policy-based request, the PEMC may have to call out to many SAMSON services in order to fulfill that request. The set, order and processing of SAMSON service messages that the PEMC uses to fulfill its function is referred to as Information Protection Logic (IPL). The set of messages that the PEMC can draw upon includes all the messages supported by the SAMSON Service Gateways, as documented in section 4.0: The SAMSON Security Services.

While there is no standard or specification on how IPL needs to be implemented within a PEMC, the following example will illustrate the function of the IPL. This example extends the previous example of the PEDI requesting a policy-based decision when a user attempts to access a SAMSON protected file.

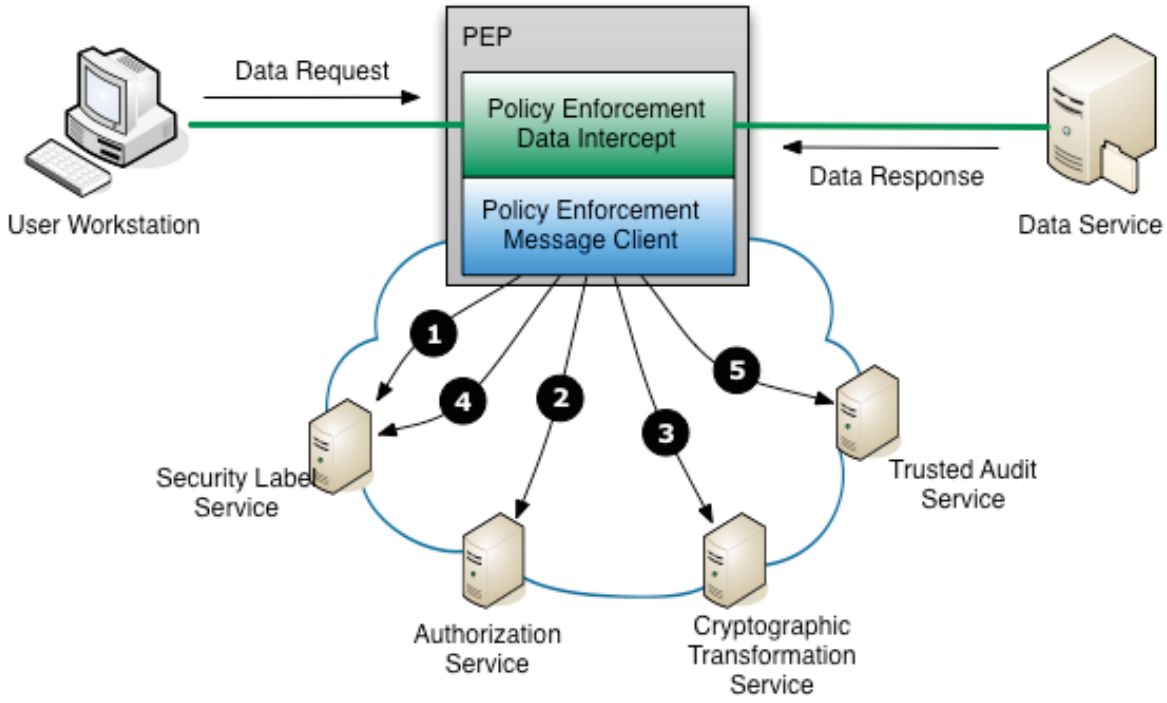


Figure 20: An Example of File Server Information Protection Logic

When the PEDI requests the policy decision from the PEMC, with the request including the user's identity, the full path to the file and the action that the application attributes to this operation, the PEDI will execute the following sequence of messages.

1. A call is made to the SLS using the GET_FILE_LABEL message type and specifying the location of the file. If successful, the SLS will respond with the security label on that file.

2. A call is made to the AS using an XACML message structure that includes the user's identity as the target, the file as the resource and the action as the XACML action. If successful the AS will return a policy decision

3. Assuming the decision is Permit, the PEMC will make a call to the CTS using the COPY_DECRYPT message type. The path to the encrypted file and the path to the desired location of the plaintext file is supplied in this message. Note that the CTS will make supplemental service calls to the KMS to obtain the key to decrypt this file. If successful, the CTS will indicate to the PEMC that the plaintext file is available for delivery to the user

4. A second call to the SLS using the GET_FILE_LABEL message type and specifying the location of the plaintext file. If successful, the two labels are compared to ensure that there has not been any tampering of either security label.

5. At this point the PEMC will return a message to the PEDI that the requested operation is permitted and that the file is available for delivery to the user. As a final step, an AuditXML message, as described in section 4.6: Trusted Audit Service, is created and sent to the TAS to create a tamper-resistant record of this transaction.

The IPL, therefore, is the logic applied by the PEMC to properly process an information request using SAMSON services. It is significant to note that IPL logic is expressed in terms of:

- The set and order of security messages that are sent to SAMSON SSGs

- The handling of responses from those services to detect the conditions under which the user's transaction should be denied (e.g. an error was received from a component or a decision to deny the request was sent from the AS)

- Logic within the IPL itself, such as the verification check to ensure that security labels on the resource are congruent and have not been altered.

It must be re-iterated that IPL within a PEMC is coded specifically for its role in protecting certain applications and data types. The example given above pertains to a file sharing scenario, but a PEMC that is protecting a different kind of information asset, such as chat room messages, would implement a different profile of IPL in terms of the security messages that are exchanged with SAMSON components. It suffices to say that while some IPL functions are mandatory in most cases: almost all PEMC requests will require a policy check from the AS and an audit message sent to the TAS, there will variability in how IPL is implemented within the PEMC. The IPL for each supported application type is described in detail in the following section.

Even for a given application, two environments may opt for different IPL in their PEMC: one environment may opt to opt to omit some security checks that are not as relevant according to their risk profile.

Additionally, PEMC IPL may be adjusted over time to include calls to new security services that are introduced into the SAMSON information protection space. SAMSON, as an SOA and committed to the development of *data-centric* security, has been designed to encourage the development and introduction of new security capabilities that will further enable protection and sharing of information assets. A PEP can leverage these services by updating to the IPL at the PEMC to take advantage of these new SSG capabilities.

5.3 PEP Implementations

The SAMSON TD architectural deployment includes four distinct implementations of PEP:

1. A File Sharing PEP to mediate access and protect individual files as they are stored on a SAMSON protected file server;

2. An Email PEP to mediate access and provide information protection of email messages, and associated attachments, when those message are sent and received;

3. An IM PEP to mediate access to chat rooms and protect those messages as they reside at IM server; and

4. A Web Session PEP to mediate access of web services.

Each PEP implementation is described in detail in the following sections.

5.3.1 *File Sharing PEP*

This PEP is intended to provide information protection for individual files as they are hosted on a SAMSON protected file share. As with all PEPs, the intent is for SAMSON information protection to be added to an existing network infrastructure as a security overlay; the introduction of SAMSON data-centric security practices should not necessitate modifications to either the client endpoint or the back end file server.

The File Sharing PEP requires that files have a security label. Therefore, the user must ensure that any files sent to the File Sharing PEP are labelled: either by manually applying a label or by leveraging a 3rd party file labelling solution such as the Titus labelling solution for Microsoft Office documents. The security labels that are applied to files must be compatible with the label format expected by the SLS.

For the File Sharing PEP, the granularity of the data centric security model is taken to the individual file level. When users place a file on a SAMSON protected file share via the PEP:

1. The security label on the file is used for policy checks against the AS; and

2. If permitted by policy, the file is individually encrypted with its own unique symmetric key and placed within a SAMSON container file.

As described in section 4.4.1: CTS Design Considerations / Configurations, the container file is the form taken by a SAMSON protected file while is it resident on a SAMSON protected file share. The contents of the container file can only be disclosed if the file is accessed through the File Sharing PEP. The PEP, being a SAMSON component, is able to retrieve

the original, single use symmetric key that was used to encrypt the file when it was stored and the decryption process will only take place if the requesting user has the policy right to see that file's contents. Since the PEP audits all file transactions, a permanent, trusted record of all data creation and disclosure events is maintained. In addition, files re-saved to the file server are re-encrypted with a new, single use symmetric key.

5.3.1.1 File Sharing PEP Architecture

In the SAMSON TD deployment architecture, the File Sharing PEP is implemented as an *indirect proxy*. That is, the file share location that is to be protected by the File Sharing PEP is mounted in a staging area on the PEP's host file system.

1. The PEP thus has the ability to store files to and retrieve files from the unmodified back end file server.

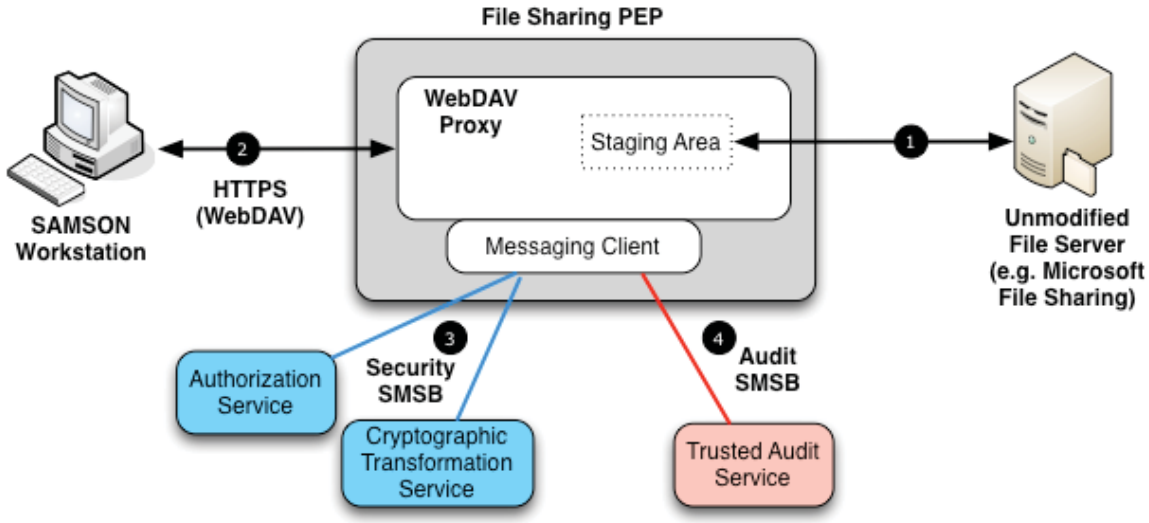


Figure 21: File PEP Architecture

2. The PEP itself provides a WebDAV interface to SAMSON users, allowing the staging location to be accessed this web interface. Since endpoint clients such as Windows 7 Explorer natively support WebDAV sessions, a user can mount the SAMSON PEP and will see the files that are present on the back end file server.

3. At the PEP, the deployed WebDAV server is enhanced with SAMSON processing logic to ensure that operations on files adhere to the protection mechanisms that are required for SAMSON's data-centric security. The File Sharing PEP is a participant on the Security SMSB so that it can leverage SSGs to access services such as policy decisions from the AS and encryption services from the CTS.

4. The File Sharing PEP is also a participant on the Audit SMSB for access to auditing services.

As detailed in section 5.2.1:PEMC Architecture, the PEP leverages SAMSON SSGs by having a messaging client sub-component in the PEP architecture that is responsible for all XMPP-based SMSB communications.

Because the actions the user performs against the WebDAV interface are forwarded (via the underlying file system mount) to the back end file server and the results can be manipulated as they transit the PEP, file sharing security is applied transparently to the user. The only requirement on the user is that access to the SAMSON protected file server must be made through the PEP itself, not directly to the back end file server. If a user accesses the back end file server directly, any files delivered to the user will remain encrypted since the PEP was not in the transaction path to decrypt the contents of the container.

5.3.1.2 File PEP Operations on Data

Connecting to the File Sharing PEP is not a SAMSON protected operation. When a user requests to mount the WebDAV service where the File Sharing PEP resides, SAMSON does not interrupt this processing and the mount action is always allowed to proceed. The first action that Windows Explorer takes after a successful mount is to request a listing of the files at the mounted location. Listing a directory is one of the three file operations that the File Sharing PEP intercepts so that information protection logic can be applied to the transaction. Each of these operations is described below.

Directory Listing

When the user's Windows Explorer requests a directory listing of the current directory (either automatically or directly by the user), the client software formulates a WebDAV listing request and sends it to the File Sharing PEP. The WebDAV proxy within the PEP allows this request to be forwarded to the backend server, but intercepts (1) the listing as it is returned. At this stage, the File Sharing PEP has a complete list of the files in the target directory stored in a WebDAV data structure. The PEP examines each of the files in the list in sequence to determine if the presence of the file can be disclosed to the user. This process is shown in the following diagram.

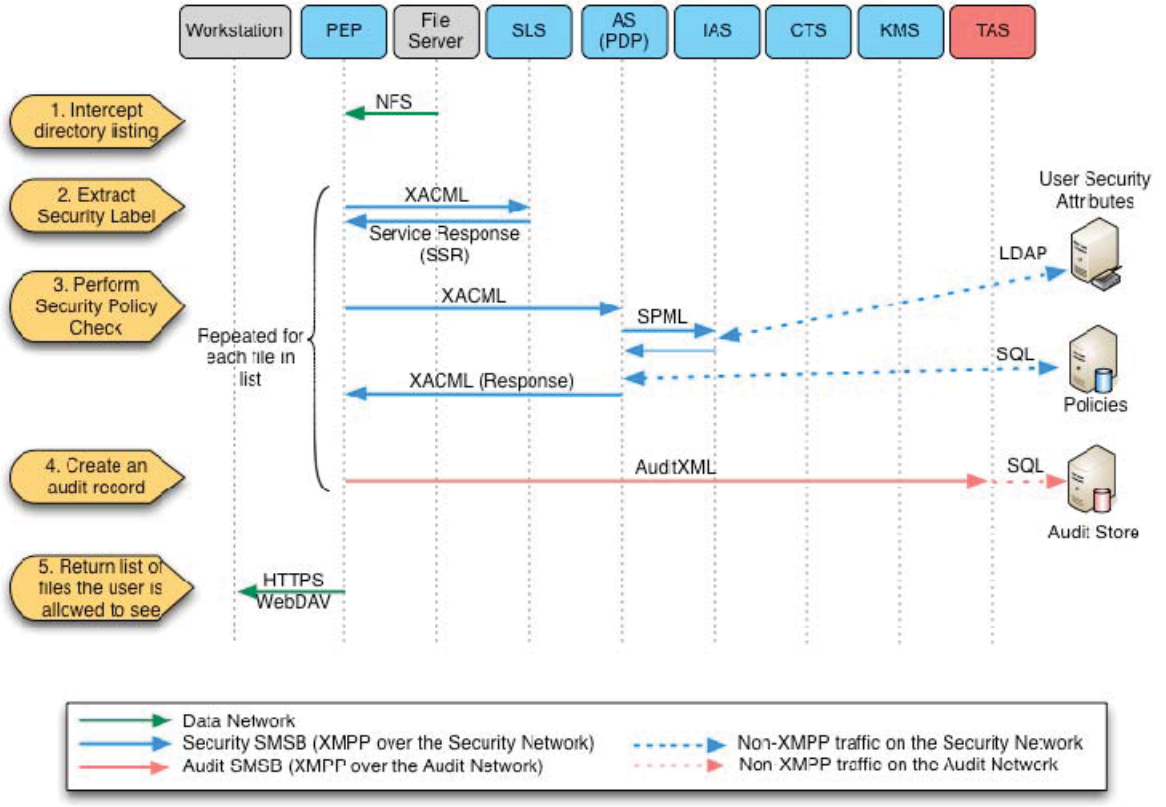


Figure 22: File PEP - Directory Listing

1. The File Sharing PEP intercepts the file listing from returned from the file server.

The PEP then repeats steps 2 through 4 for each file in that listing.

2. The PEP calls the SLS, using the message format specified in section 4.5.2: SLS Messaging and Operation, to extract the security attributes on the target file. Since the PEP and the SLS are co-located on the same system, the absolute path to the file can be specified.

3. The PEP calls the AS, using the message format specified in section 4.2.2: AS Messaging and Operation, with the user's identity, the security attributes of the file and the policy action to get a policy decision as to whether the user should be able to see the target file. For file listings, the policy action is "READ", that is, only users with the policy right to READ files can see those files in a directory listing. Also of note, in the SAMSON TD deployed architecture, the WebDAV service is configured to utilize Windows Domain credentials for authentication, hence, the user's account that is used for the policy check is the user's Windows Domain account. Depending on the policy decision, the PEP takes the following action:

a. If the AS states that the policy decision is to permit the user to see the file, the target file is allowed to stay in the WebDAV directory listing data structure.
b. If the policy decision is to deny the action, the target file is removed from the WebDAV directory listing data structure.

4. An audit record, based on the message format specified in 4.6.2: TAS Messaging and Operation, is generated by the PEP and sent to the TAS. As described in that section, the TAS will process the audit record, extend the record details and submit the record to the audit store.

5. Once all files in the directory structure have been processed, the SAMSON PEP allows the (possibly reduced) WebDAV structure to be returned to the user's Explorer session. The user will be unaware if any files have been scrubbed from the listing since the listing is filtered prior to delivery. This means that two different users will see different contents in a directory depending on each user's COI membership and the security policy that was used to authorize the action.

Storing a File

When a user stores a file to the back end file share, either by copying the file or saving an active file to that location, the File Sharing PEP will execute the process shown in the following diagram.

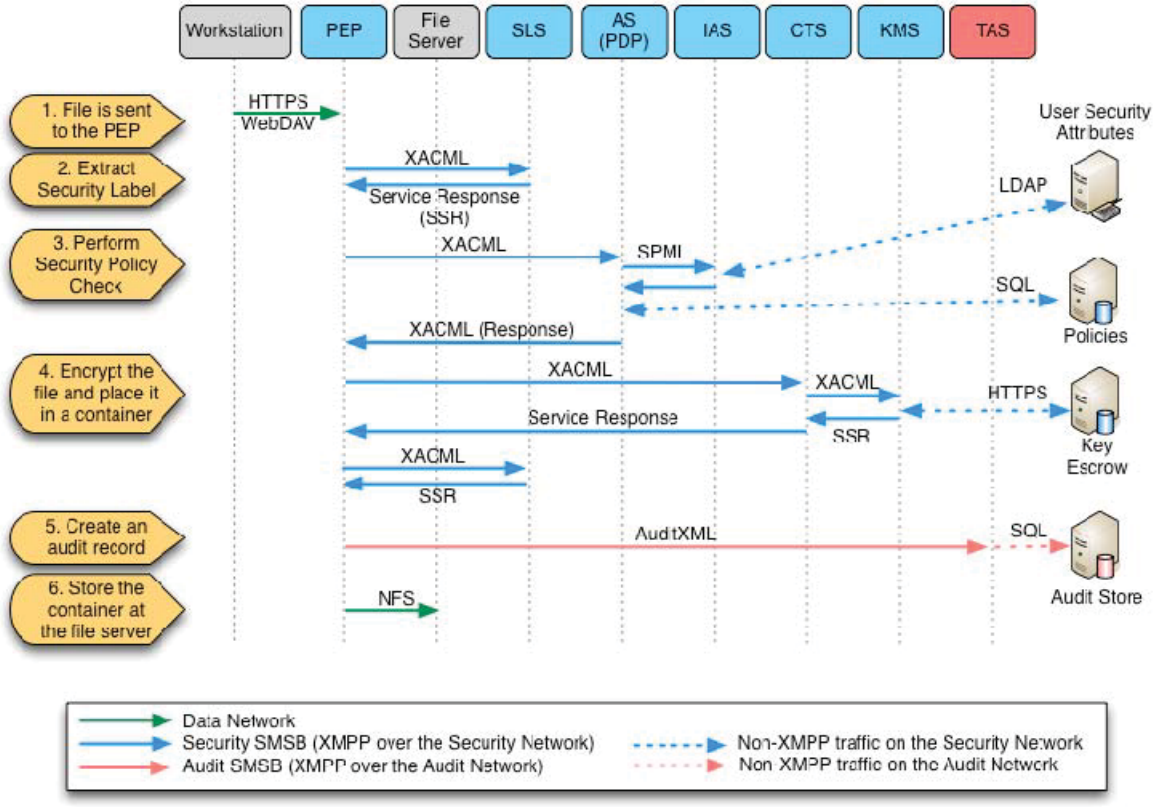


Figure 23: File PEP - Storing a File at a SAMSON Protected File Server

1. The file data is allowed to be uploaded through the WebDAV session but rather than being stored at the back end file server, the PEP places the file in a local staging area.

2. The PEP calls the SLS, using the message format specified in section 4.5.2: SLS Messaging and Operation, to acquire the security attributes on this local file. Note that this means that the file must be properly labelled at the endpoint before it is sent to the PEP. For the SAMSON TD architectural deployment, the Titus Document Labelling for Microsoft Office product was used at the endpoint to label files. The format for file labels is described in section 4.5.1: SLS Design Considerations / Configuration.

3. The PEP determines if the user has the policy right to create SAMSON protected content using the security attributes on the file. The user's identity, the security attributes from the file and the policy action of "WRITE" are sent to the AS in a policy request message using the format defined in section 4.2.2: AS Messaging and Operation.

a. The AS itself sends the IAS a user security attribute request as defined in section 4.1.2:IAS Messaging and Operations to acquire the user's COI membership list from the user security attribute repository. This information is used as part of the policy decision-making process.

4. If the policy decision is to allow the file to be stored, the PEP calls the CTS, using the message format described in 4.4.2: CTS Messaging and Operation, to encrypt the file. The CTS will leverage the KMS to obtain a new symmetric key for the operation and store this key in the escrow system. With this new key, the file is encrypted and placed in a container. The container is also populated with the other data elements as described in section 4.4.1: CTS Design Considerations / Configurations, namely, the caveat, the digest and the token.

5. The PEP will then create an AuditXML formatted audit record, based on the message format specified in 4.6.2: TAS Messaging and Operation, that specifies the file store transaction details and send the record to the TAS to create a permanent record of the transaction.

6. Once the container has been successfully created, the PEP will move the container from the working area to the staging area where it is stored at the backend file server via NFS.

The PEP will cancel the file storage transaction and send a WebDAV compliant HTTP 403 Forbidden status code back to the user's Explorer session if any of the following conditions are encountered:

- The AS states that the policy decision is to deny the action;
- Any of the SSGs return error codes to indicate that they could not process the request (e.g. the SLS return an error that there is no label on the file); or
- There is any error associated with storing the file to the back end file server.

Audit records are generated in each case with the error status code reflected in the notes sections of the AuditXML message. As described in section 4.6.1: TAS Design Considerations / Configuration, when the TAS encounters an AuditXML message that includes references an error encountered by the SSGs, it triggers a security event. When a security event is raised, the SIEM solution included as part of the SAMSON TD architectural deployment will generate a notification email message for the security officer.

The temporary files that are created as part of this action are deleted once the action is complete, regardless or whether the action completed successfully, was denied or encountered an error.

Retrieving a File

When a user requests a file from a SAMSON protected file share, the File Sharing PEP will execute the information protection logic in accordance with the following time sequence diagram.

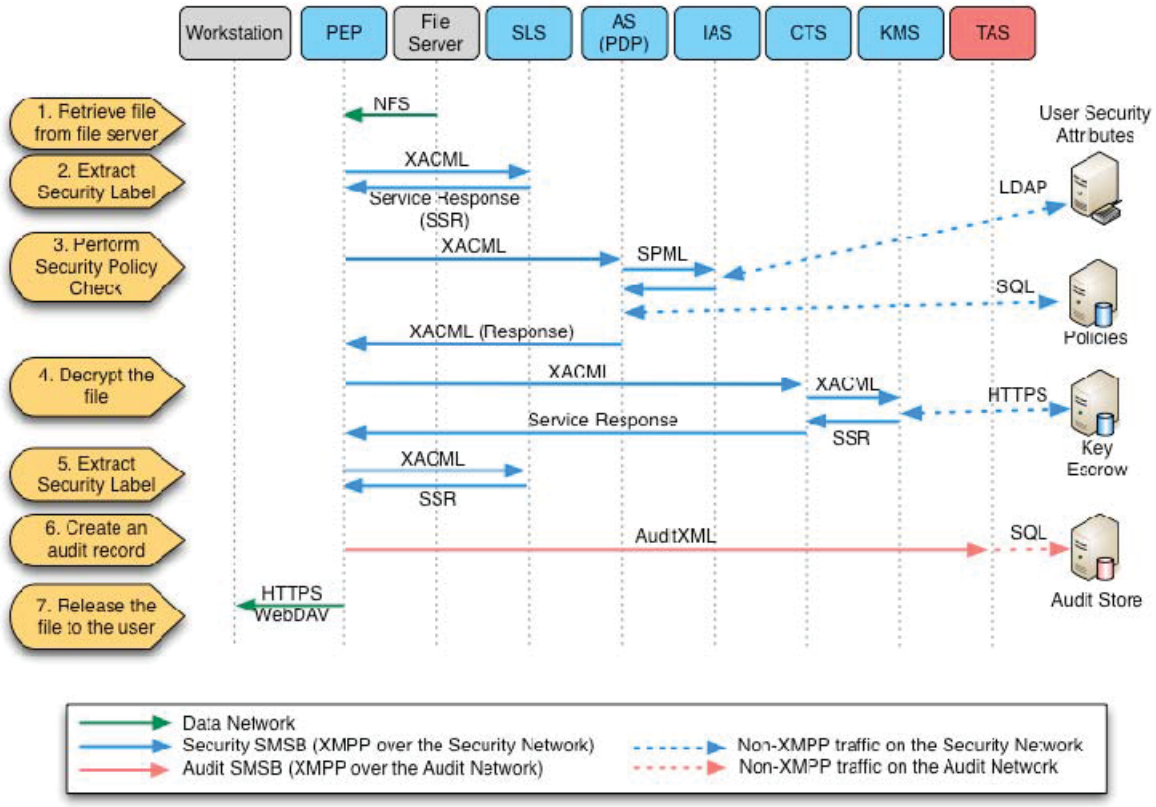


Figure 24: File PEP - Retrieving a File from a SAMSON Protected File Server

The significant data exchanges that take place in the processing of this request are described below.

1. When a user requests a file from the back end server, the file is retrieved but is placed in a local staging area on the PEP machine.

2. Through a call to the SLS, using the message format specified in section 4.5.2: SLS Messaging and Operation, the PEP acquires the security attributes on this working file. Since this is a SAMSON protected file, it will be in a SAMSON container that is readable by the SLS.

3. The PEP determines if the user has the policy right to access a protected data asset that is labelled with those security attributes. The user's identity, the security attribute from the file and the policy action of "READ" are sent to the AS in a policy request message using the format defined in section 4.2.2: AS Messaging and Operation.

    a. The AS itself sends the IAS a user security attribute request as defined in section 4.1.2: IAS Messaging and Operations to acquire the user's COI membership list from the user security attribute repository. This information is used as part of the policy decision-making process.

4. If the policy decision is to allow the file to be disclosed to the user, the PEP calls the CTS to decrypt the file, using the message format described in 4.4.2: CTS Messaging and Operation. The CTS, working on the temporary file, will perform the following actions:

    a. By comparing the digest in the file container against the container's contents, validate that the file has not been altered.

    b. Using the token in the container, call the KMS to retrieve the symmetric key that was used to encrypt the file. A key request message using the format specified in section 4.3.2: KMS Messaging and Operations.

    c. Decrypt the encrypted file stored in the container to get the original file.

5. Through a second call to the SLS, the PEP acquires the security attributes on the decrypted file. Since the security attributes in the container were taken from the security attributes from the original file when the container was created, the file's attributes and the container's copy of the attributes should match. If there is a mis-match, the container has been tampered with and the file should not be disclosed to the user. At this point additional security measures may be inserted into the SLS information protection logic such as additional security scans, virus scanning, and/or data loss protection. Alternatively, these data integrity solutions can be implemented as separate SSGs and called directly by extending the information protection logic at the PEP. If the SLS confirms that the file that was decrypted has not been altered while on the file server, the PEP places the file in the staging area where it can be delivered to the user that requested it.

6. The PEP will then create an AuditXML formatted audit record, based on the message format specified in 4.6.2: TAS Messaging and Operation, that specifies the file disclosure transaction details and send the record to the TAS to create a permanent record of the transaction.

7. If the file has been authorized for disclosure (the security policy permits the release of the information and there have been no errors in processing the request), the file is delivered to the user's workstation over WebDAV.

The PEP will cancel the file retrieval transaction and send a WebDAV compliant HTTP 403 Forbidden status code back to the user's Explorer session if any of the following conditions are encountered:

- The AS states that the policy decision is to deny the action;
- Any of the SSGs return error codes to indicate that they could not process the request (e.g. there was a mismatch between the file and the container); or
- There is any error associated with retrieving the file from the back end file server.

Audit records are generated in each case with the error status code reflected in the notes sections of the AuditXML message, as shown in Table 15: *AuditXML Elements*.

The temporary files that are created as part of this action are deleted once the action is complete, regardless or whether the action completed successfully, was denied or encountered an error.

5.3.1.3 File PEP Trust Model

The File Sharing PEP contains many levels of protections that prevent information from being disclosed to unauthorized individuals. On the front end, the Windows Explorer to WebDAV sessions are TLS protected. On the back end, the files that are exchanged between the PEP and the file server are encrypted with unique symmetric keys. Files can only be disclosed when they are accessed through the File Sharing PEP and this disclosure only occurs after a policy check permits the transaction to take place. The policy check that takes place is based on the user's identity that is taken from their Windows credentials. The actions taken by the PEP are performed on a hardened appliance. Finally, all transactions are audited so that there is a tamper-resistant record of all user activity where information has been disclosed to the SAMSON user community.

5.3.2 *Email PEP*

The Email PEP is intended to provide information protection of email messages as they are sent to, stored at and retrieved from an existing mail server in the target environment. As with all PEPs, the intent of SAMSON based email protection is for the Email PEP to be added to an existing network infrastructure as a security overlay. The Email PEP is, therefore, implemented as a proxy architecture where email traffic is directed through the proxy and SAMSON information protection logic is applied during the sending and receiving of email messages.

For the Email PEP, the granularity of the data centric model is taken to the individual email message. Specifically, both the message body and all the attachments within that message are individually evaluated against the security policy. In other words, when an email message is sent, the security policy is enforced, for both sender and receivers, not only

against the message body, but also against all files that are attached to that message. While the message is stored at the mail server awaiting delivery to the recipients, the entire message (body and attachments) is protected as a single encrypted container object.

In order for the security policy to be applied to a email message, the message body must also have a security label that contains the attributes that will be used in the policy check. The endpoint, therefore, must have a security labelling software solution that can apply security labels to email messages. For the SAMSON TD architectural deployment, this labelling solution is the Titus Message Classification plug-in for Microsoft Outlook. When a user composes and sends an email message, the Titus software adds a security label to the message that contains security attributes that apply to the message body.

Additionally, each file attached to the message must have its own security label. The format used by file attachments in the Email PEP solution is the same format used for the File Sharing PEP. The same SLS is used, therefore, for both the File Sharing and Email PEPs. In other words, whether users are storing Office documents on a SAMSON protected file share or sending Office documents as email attachments, the same SLS service is used to extract the security attributes from those files. As described in section 4.5.1:SLS Design Considerations / Configuration, although the same SLS implementation is used for both PEPs, a separate SLS is deployed with each PEP. Each PEP must have its own co-located SLS so that files can be evaluated locally.

The PEP leverages the CTS to:

1. Generate a new key for the email object;

2. Encrypt the message into a protected data object;

3. Place the encrypted object inside a SAMSON container, as detailed in section 4.4.1:CTS Design Considerations / Configurations.

The resulting container is in the same format as those containers created by the CTS for the File Sharing PEP.

If the sender has the policy right to create and send the email message, a copy of the encrypted message is placed in each recipient's mailbox.

This Email PEP implementation supports email traffic that uses the SMTP/POP3 mail protocols. It is expected, therefore, that the back end mail server has been enabled for SMTP/POP3 email support. SMTP/POP3 support is included as part of the Microsoft Exchange product but is not enabled by default.

5.3.2.1 Email PEP Architecture

In the SAMSON TD deployment architecture, the Email PEP is implemented using proxy style architecture. That is, the PEP:

- Supports the message protocol format used by the user's email client;
- Allows those clients to connect; and
- Forwards protocol messages on to the back end mail server.

It is while brokering the message communications between the endpoint and the back end email server that the PEP is able to insert SAMSON information protection logic.

For the SAMSON TD architectural deployment, two separate proxies are used:

1. One for the sending of email messages that uses the SMTP protocol; and
2. One for the receiving of email messages that uses the POP3 protocol.

This architecture can be seen in the following diagram.

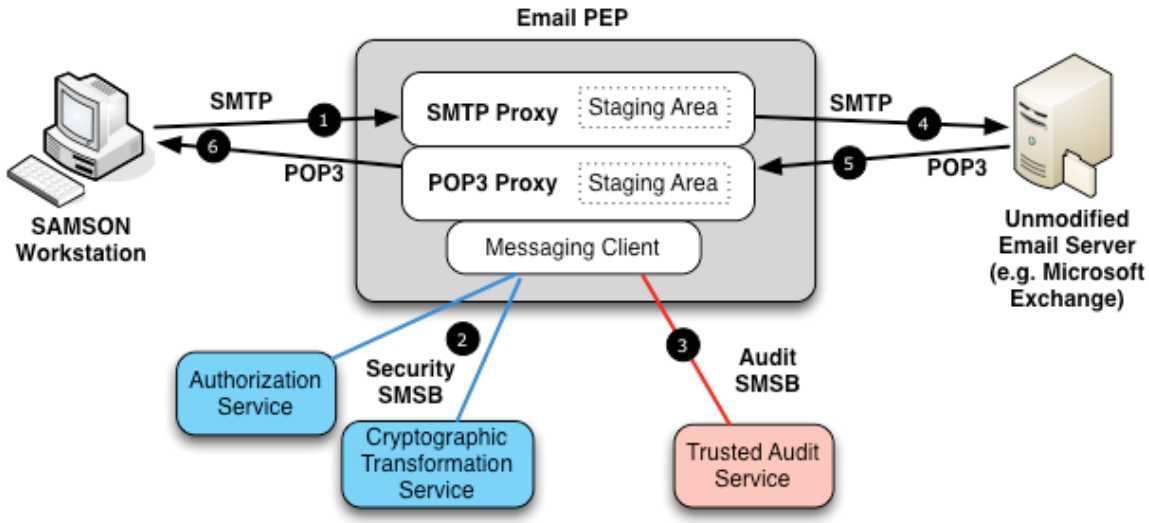


Figure 25: Email PEP Architecture

The Email PEP SMTP proxy is implemented using the ProxSMTP[7] email proxy software package. ProxSMTP is a re-purposed virus scanner that allows 3rd parties to add validation logic to its email scanning routines.

1. When the user sends an email message, the email client connects to the SMTP proxy. The proxy software receives the email message and places the contents in a staging area. The proxy software then calls the Email PEP information protection logic, that is, the software validation routine that evaluates the message to ensure it meets policy and information protection requirements.

2. As is common to the design of all PEPs, the Email PEP has a messaging sub-component that is a participant on the Security SMSB and the Audit SMSB. Through this messaging client, the PEP makes calls out to the necessary SSGs to evaluate and transform the message. For example, the Email PEP will call:

   a) The SLS to get the security attributes from any files that were attached to the message;

   b) The AS to ensure the message can be sent (and received) according to the domain's security policy; and

   c) The CTS to protect the message and format it as a SAMSON container.

3. The PEP's messaging client will also submit an event transaction record to the TAS.

4. Once it is determined that the message can be sent and the message has been properly protected, it is forwarded on to the mail server for delivery.

The POP3 proxy is also implemented using a re-purposed virus scanner, in this case, the P3Scan[8] open source POP3 mail scanner is used. The processing logic for receiving an email message is similar to the send process.

5. Users connect to the POP3 proxy and request that messages be retrieved from the back end mail server. Before the proxy returns mail message to the user they are written to a staging area where the PEP can again apply SAMSON information protection logic. The security attributes from the encrypted message's container are used to determine if the security policy allows the user to receive the email message. If permitted, the CTS decrypts the message for the user and creates an audit event of the transaction.

6. The email message is then returned to the user's email client.

[7] http://thewalter.net/stef/software/proxsmtp
[8] http://p3scan.sourceforge.net

5.3.2.2 Email PEP Operations on Messages

In order for a user's email client to work with the Email PEP, the connection settings for the client must be altered to point to the SMTP and POP3 services hosted by the Email PEP. The two PEP proxies do not perform authentication directly, rather, they forward login information to the back end service for authentication. It is only when an email message is sent (or received) that the Email PEP logic for protecting messages is invoked.

Sending a Message

When a user creates and submits an email message to the Email PEP, the PEP must be able to apply the security policy to all file attachments and ensure that the policy is applied to not only the sender of the message, but each of the recipients as well. This includes users specified as "TO", "CC" and "BCC" recipients.

When a user sends an email, the Email PEP will execute the information protection logic in accordance with the following time sequence diagram.

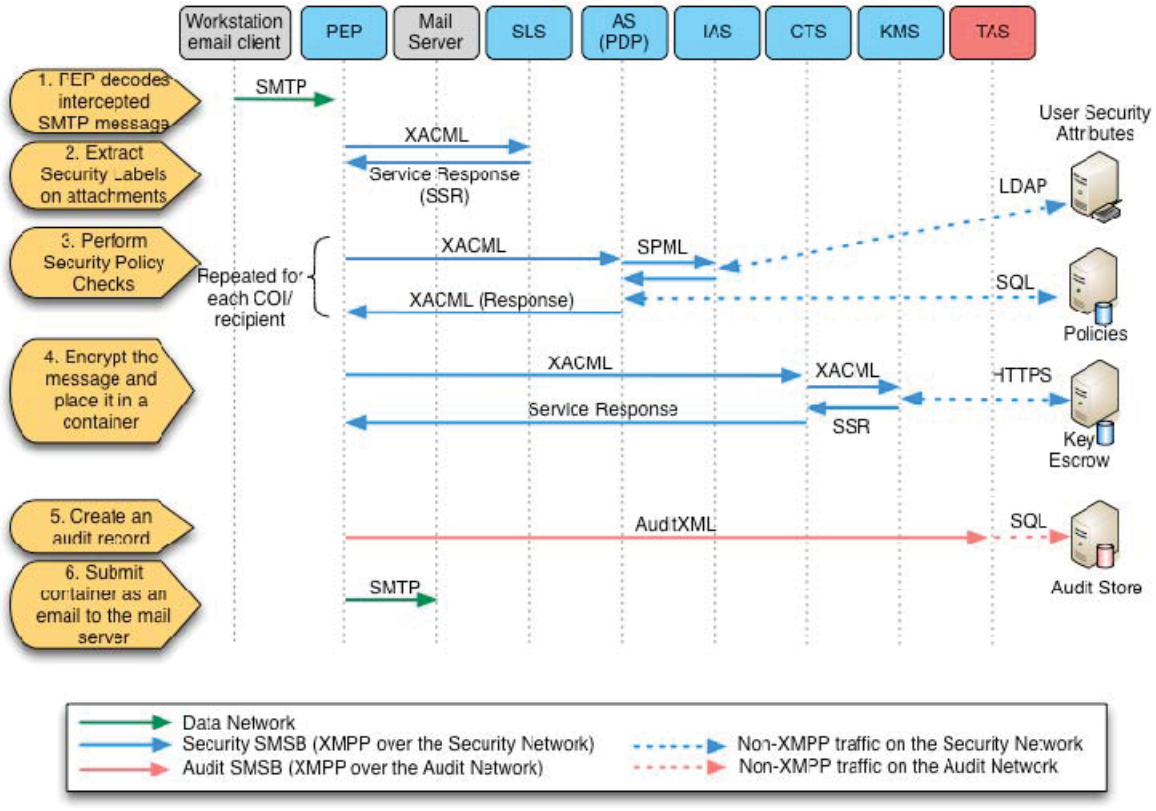


Figure 26: Email PEP - Sending a SAMSON Protected Message

1. The PEP receives the entire message from the email client and stores it in a local staging area for processing. The message is decoded if necessary (e.g. base64 decoded) and any MIME attachments are extracted from the message. At the end of this interception activity, the message body and copies of the original attached files are present in the staging area.

2. The security labels on the attached files are extracted through calls to the SLS using the message format specified in section 4.5.2: SLS Messaging and Operation. Since the Email PEP and the SLS are co-located on the same system, the absolute path to the files can be specified. Additionally, the security label on the message body is extracted from the message. The COTS endpoint labelling and visual marking software, Titus Message Classification Plug-in for Microsoft Outlook, places the security label for the message in a dedicated message header that is read from the message body. At the end of this step, the Email PEP has a unique set of security attributes that have been attached to the message and its attachments.

3. The message is evaluated against the security policy. For this step, the Email PEP sends multiple policy request messages to the AS using the message format specified in section 4.2.2: AS Messaging and Operation. The policy checks that are made include the following:

    a. A set of policy checks are made to ensure that the message **sender** is allowed to send data using the COIs on the message and the attachments. The message's FROM header is used to determine the identity the sender of the message. The action value for these policy checks is WRITE since, for the SAMSON TS architectural deployment, sending an email message is a WRITE policy operation. A separate policy check is made for each unique COI found in the security label for the attachments and the message body.

    b. A second sequence of policy checks, this time to determine if the **recipients** are allowed to receive the email message, are made using the user identities found in the "TO", "CC" and "BCC" headers within the message. Each user is individually evaluated against same COI set from the previous sender check. In this case, however, the policy action for these checks is READ. That is, the check verifies that each recipient can receive the data assets in the message.

    As with the File Sharing PEP, the AS will include the users' security attributes as part of the policy decision-making process. The user security attributes are retrieved from the IAS via a security attribute request, as defined in section 4.1.2: IAS Messaging and Operations.

4. If all policy checks result in a "permit" decision, the PEP calls the CTS to encrypt the originally intercepted base64 encoded message (which includes all MIME attachments). The PEP calls the CTS using the message format described in 4.4.2: CTS Messaging and Operation, to encrypt the file. The CTS will leverage the KMS,

using the format specified in section 4.3.2: KMS Messaging and Operations, to obtain a new symmetric key for the operation and store this key in the escrow system. With this new key, the file is encrypted and placed in a container.

5. The PEP will then create an AuditXML formatted audit record, based on the message format specified in 4.6.2: TAS Messaging and Operation, that specifies the email transaction details and send the record to the TAS to create a permanent record of the transaction.

6. The Email PEP will encode the newly created encrypted container so that it is in compliance with the mail server's message format specification. The PEP then transmits the container (as a MIME compliant email message) to the mail server with the original source and destination routing information.

In summary, the logic flow for making a policy decision about an email message is as follows. Assuming that we have the identity of the sender, the complete list of the message recipients and the set of all the COIs on the message body and attachments:

- For each COI in the set, does the sender have the policy right to WRITE data assets with these COI values?

- For each of the recipients:
  - For each COI in the set, does the recipient have the policy right to READ data assets with these COI values?

This means that for a message with **m** recipients and **n** attachments (each with a different COI) and where the message body also has a unique COI, there will be (n+1) x (m+1) separate policy checks made. It is significant to note that the Email PEP deployed for the SAMSON TD architectural deployment attempts to minimize the number of policy checks that must be made; the PEP will create a unique set of all COIs found in the message body and attachments so as not to repeat the same policy check. For example, if an email message has three attachments with the same COI in their security label, the Email PEP will only perform one policy check to cover all three attachments.

In the course of evaluating these policy checks, if there is a single "deny" decision returned from the AS, the message is not sent to any of the recipients. When an email message is prevented from being sent due to a policy violation, an informational email message is sent to the originator to indicate that the message was not delivered and exactly what policy issues prevented the message from being sent, as shown in Figure 27: *Email Policy Violation Message*.

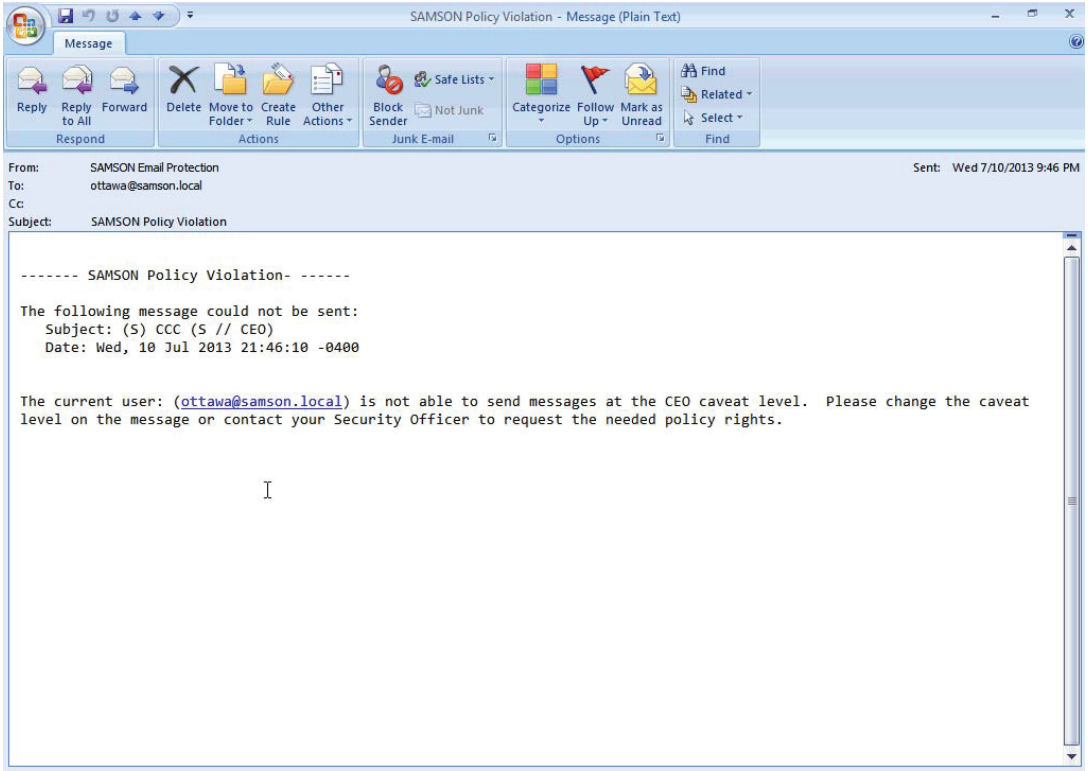


Figure 27: Email Policy Violation Message

The Email PEP does not abort a set of policy checks when a deny is encountered, rather, it continues to perform policy checks so that the informational message sent to the originator provides a complete list of why the message is not compliant with policy. The sender then has the option to resend the message by:

- Dropping recipients from the message;
- Dropping attachments from the message; or
- Changing the caveats on the message body or attachments.

The temporary staging files that are created as part of this action are deleted once the transaction is complete, regardless of whether the action completed successfully, was denied or encountered an error.

Receiving a Message

When a user connects to the mail server to retrieve email messages, the Email POP3 proxy allows the message retrieval commands to be sent to the back end mail server. When messages are returned through the proxy, however, the proxy intercepts the messages and

stores them in a local staging area. For each message, the Email PEP applies SAMSON information protection logic as shown in the following diagram.

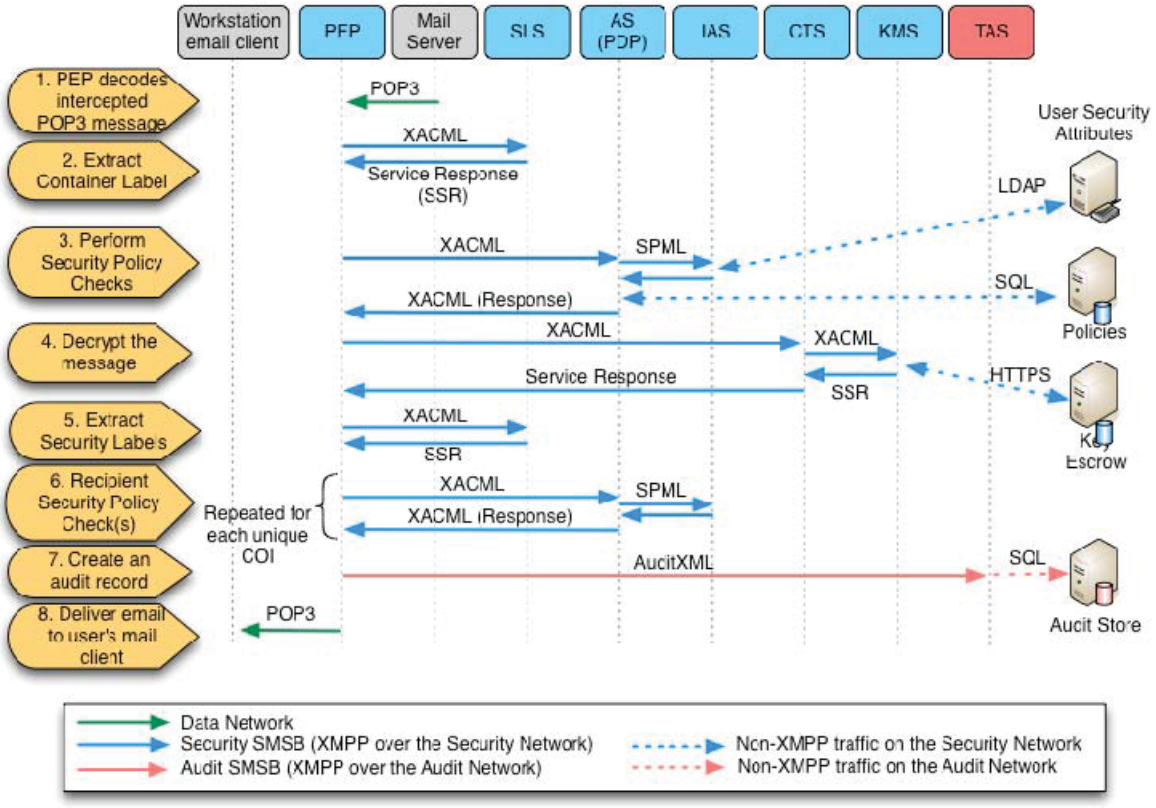


Figure 28: Email PEP - Retrieving a SAMSON Protected Message

1. When email messages are retrieved (via POP3) through the proxy, each email is first written to a local staging area on the PEP machine.

2. When the email message was sent, the Email PEP encrypted the message as a SAMSON container. Therefore, once the message is stored locally and decoded (base64 decoding) the security attributes on the email can be extracted from that container. Through a call to the SLS, using the message format specified in section 4.5.2: SLS Messaging and Operation, the PEP acquires the security attributes from this container.

3. Using the information in the message's "TO" header to identify the intended recipient, a policy check is requested from the AS using the message format specified in section 4.2.2: AS Messaging and Operation. The action value for these checks is READ since, for the SAMSON TS architectural deployment, receiving an email message is a READ policy operation. The resource for this policy check is the COI

for the message, as stated in the security label on the container. If the policy decision is to deny the recipient the message, the container is not decrypted.

4. If the policy check determines that the recipient can receive the message, the PEP calls the CTS to decrypt the file, using the message format described in 4.4.2: CTS Messaging and Operation. The CTS, working on the container, will perform the following actions:

    a. By comparing the digest in the container against the container's contents, validate that the container has not been altered.

    b. Using the token in the container, call the KMS to retrieve the symmetric key that was used to encrypt the file. A key request message using the format specified in section 4.3.2: KMS Messaging and Operations.

    c. Decrypt the encrypted file stored in the container to get the original encoded, email message.

    The resulting decrypted file is the original email message as the SMTP proxy received it. The Email PEP extracts all MIME encoded attachments from this message to obtain the files that were attached to this message.

5. The security labels on the attached files are retrieved through calls to the SLS using the message format specified in section 4.5.2: SLS Messaging and Operation. Since the Email PEP and the SLS are co-located on the same system, the absolute path to the files can be specified.

6. Using the information in the message's "TO" header to identify the intended recipient, a series of policy checks are made to the AS: one for each unique COI in the message and attachments. The action value for these checks is READ since, for the SAMSON TS architectural deployment, receiving an attachment on an email message is a READ policy operation. The resources values for these policy checks are the unique set of COIs in the security labels on each of the attached files.

7. The PEP will then create an AuditXML formatted audit record, based on the message format specified in 4.6.2: TAS Messaging and Operation, that specifies the email transaction details and send the record to the TAS to create a permanent record of the transaction.

8. The Email PEP will encode the decrypted message and signal the POP3 proxy to deliver the message to the user's email client.

It is significant to note why policy checks are needed when an email message is received. When the message was originally sent, policy checks were made to ensure that all recipients were allowed to receive the email. Why then is another set of policy checks needed when a user retrieves email? The reason is because of the delay between when a message is sent and when it is received. During that interim, the security policy may have

changed or the recipient may have been excluded from a COI. While the user was allowed to receive the email message when that message was sent, under the new, updated policy, the user may no longer be permitted to receive the message and the email should not be delivered to the user.

5.3.2.3 Email PEP Trust Model

The Email PEP contains many levels of protections that prevent information from being disclosed to unauthorized individuals. While an email message is stored at the mail server, it exists as an encrypted object inside a SAMSON container. Privileged users, such as the mail administrator, are not able to disclose messages that are stored at the server since the message must be decrypted using keys that are only available to the Email PEP. The policy and cryptographic actions taken by the Email PEP are performed on a hardened appliance. Finally, all transactions are audited so that there is a tamper-resistant record of all user activity where information has been disclosed to the SAMSON user community.

In the SAMSON TD architectural deployment, the front-end communications between the user's email client and the PEP are not encrypted. However, this solution is compatible with TLS/SSL tunnelling tools that could be used to protect the front-end session.

5.3.3 *Instant Messaging PEP*

The Instant Message (IM) PEP is intended to limit access to chat rooms and protect chat room messages that are stored at the IM server. In order to gain access to the message content within a chat room, users must go through the IM PEP so that the messages can be decrypted for the user. However, prior to being given access to a chat room, the IM PEP will validate the user's request to join a room, matching the user's security attributes and the security attributes on the chat room against the security policy. Only if a user is permitted to view the COI reflected in the chat room messages will the user be allowed to enter the room, at which time any messages that are sent or received by the user will be encrypted or decrypted, respectively. Users that attempt to access the back end IM server may be able to bypass the SAMSON access control check, but the messages that are received will not be decrypted and the information stored at the IM server will not be disclosed.

The IM PEP is implemented as a proxy architecture:

- Users connect with their IM clients to the IM PEP;
- The IM PEP establishes a connection to the back end IM server; and
- The IM PEP forwards messages between the client and the server, inserting SAMSON information protection logic as messages pass through the proxy.

In normal operations, the granularity of the IM data-centric model is taken to the chat room level. That is, each chat room is assigned security attributes and it is those attributes that are used in the access control checks. For the IM PEP, security attributes are stored as part of the chat room's room description and are stored at the IM server's database.

User's also have the option to "mark-up" specific messages within the chat room. That is, the user can apply security attributes (COIs) to individual messages. Marked-up messages are handled slightly differently that normal chat room messages. When the IM PEP receives a marked up message, the message must be individually checked against the policy. Users must have the policy right to create content using the specified attribute, or COI, for the IM PEP to allow the message to be added to the history of chat room messages. Similarly, when a user is receives IM messages, individual marked up messages must be evaluated against the security policy to ensure that they have the right to receive the message.

Marked up messages, therefore, can only be delivered to a subset of the users that have already been granted access to the chat room. For example, consider a chat room that is available to the CANUS community and within that chat room a user marks up a specific message for the CEO community. Canadian users will see all messages since they belong to both the CEO and CANUS communities but American users will only see CANUS messages. Messages that cannot be disclosed due to policy restrictions are filtered such that the chat room participants are not aware that there are additional messages that they have been prevented from receiving.

Each combination of chat room and security attribute uses a unique key for encryption. For the previous example, there will be separate keys used for the CANUS messages and the CEO messages in the chat room. The chat room's security attributes, stored as part of the description for the chat room at the IM server, including the following:

1. The default security attribute (COI) for the chat room: used in the initial access control check to gain entry to the chat room;

2. Any other security attributes (COIs) used to mark up individual chat room messages; and

3. For each unique COI in the chat room, the key token that can be used to retrieve the key that protects that COI's messages.

Because the security label for the chat room, and associated key token, are stored at the IM server, there is no need to place encrypted messages inside a SAMSON container.

The SAMSON TD architectural deployment uses an unmodified Transverse IM client (version 1.5.3) and an unmodified OpenFire 3.7.1 IM server. The connection between the IM client and the IM PEP uses a TLS protected XMPP protocol and the IM PEP redirects traffic through to the back end IM server over a TLS protected XMPP link. The back end IM server is configured to authenticate users with their Windows domain credentials. The IM PEP, being in the middle of the IM client to IM server communication is able to utilize the authenticated user identity for security policy checks.

5.3.3.1 IM PEP Architecture

In the SAMSON TD deployment, the IM PEP is implemented in the form of an application proxy. That is, the PEP:

- Supports the message protocol format used by the user's IM client;
- Allows those clients to connect; and
- Forwards protocol messages on to the back end IM server.

It is while brokering the message communications between the endpoint and the back end IM server that the PEP is able to insert SAMSON information protection logic.

The core of the IM PEP is a modified Spectrum2[9] IM proxy/gateway. Spectrum is a generic proxy that brokers communication between many messaging protocols. For the SAMSON TD, Spectrum2 is being used in its trivial configuration, specifically, connecting an XMPP front end to an XMPP backend. However, this configuration is what allows the IM PEP to insert SAMSON information protection into the message handling process. The IM PEP architecture can be seen in the following diagram.

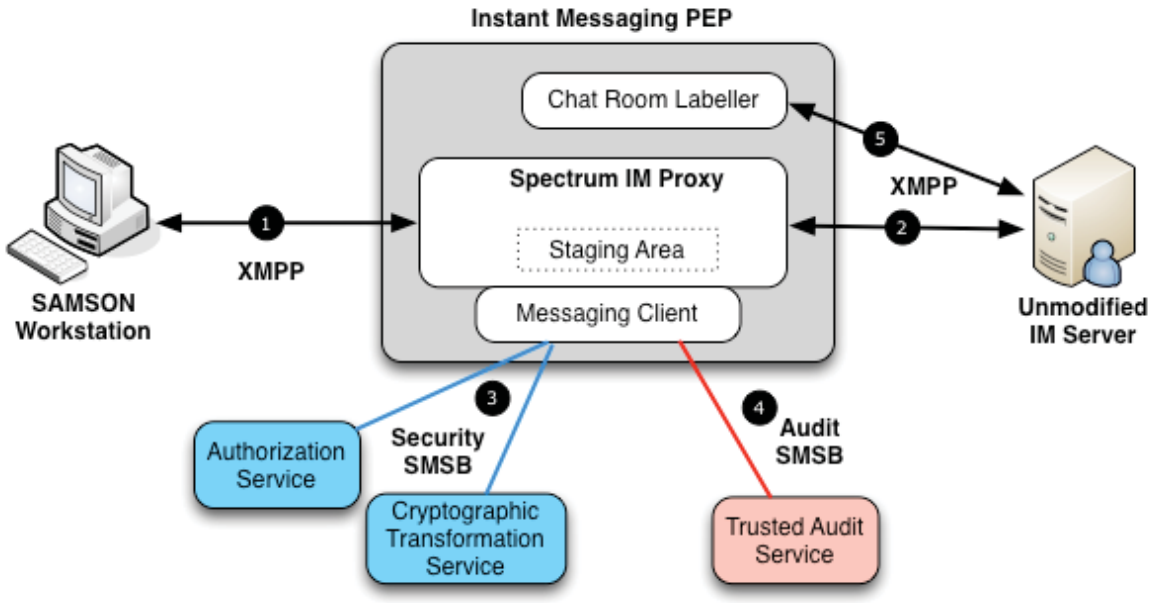


Figure 29: IM PEP Architecture

[9] http://spectrum.im

1. Users connect to the IM PEP using their IM client application

2. The PEP forwards the connection on to the back end IM server. XMPP session establishment messages are passed to the IM server so that the user's session is authenticated with the user's Windows domain login information. When a message is sent from or is to be sent to the user's IM client, the IM PEP first applies SAMSON information protection logic, including performing authorization checks prior to granting access to a protected chat room, encrypting and decrypting individual IM messages and creating an audit record of each transaction.

3. To leverage SAMSON security services, the IM PEP is a participant on the Security SMSB to access policy decisions from the Authorization Service and encryption services from the Cryptographic Transformation Service.

4. The IM PEP is also a participant on the Audit SMSB for access to auditing services.

As detailed in section 5.2.1:PEMC Architecture, the PEP leverages SAMSON SSGs by having a messaging client sub-component in the PEP architecture that is responsible for all XMPP-based SMSB communications.

The IM PEP protects and forwards messages transparently to the user. The only requirement on the user is that access to the SAMSON protected IM server must be made through the IM PEP itself, not directly to the back end IM server. If a user accesses the back end IM server directly, any messages delivered to the user will remain encrypted since the PEP was not in the transaction path to decrypt the message.

Out-Of-Band Messaging

For certain transactions, the IM PEP must get or set information at the IM Server directly. This situation will occur in two situations:

1. On start-up when the IM PEP retrieves from the IM server the list of chat rooms, the list of COI's in use within the chat room and the key token (used to acquire the actual key) for each chat room/COI combination.

2. When a new COI is used in an existing chat room, the IM PEP will acquire a new key for these messages and a key token that can be used to reacquire the key for subsequent operations. In order for this key token to be accessible in the future (e.g. after the IM PEP is restarted) this key token must be written back out to the IM server. When the IM PEP is restarted (as described in the previous situation) this new chat room/COI key token will again be obtained from the IM Server.

It is important to note that these OOB IM communications are between the IM PEP and the IM server alone. The user does not take part in these sidebar communications nor is there any manual action required on the part of the user. These data exchange only take place when the IM PEP itself requires information or service from the IM server in order to fulfil its

role as an IM information protection proxy. The IM PEP takes care of the establishment and closing of these sidebar sessions automatically.

OOB messaging is also used to create SAMSON protected chat room, although this is done through a utility rather than by the IM PEP itself. Chat rooms must be initialized before they can be made available to SAMSON users. The initialization process is performed using a jointly deployed SAMSON utility at the IM PEP: the chat room labeller (item 5 in Figure 29: IM PEP Architecture). When there is the need for a new chat room for a specific community, the Security Officer uses this utility to:

1. Create the new chat room;
2. Assigning the new chat room to a COI (the default community); and
3. Acquiring a new cryptographic key that will be used to protect messages in the chat room.

The utility places the default COI and key token within the description field for the chat room as it is defined at the IM server.

5.3.3.2 IM PEP Operation on Data

The IM PEP has three main categories of operations that involve SAMSON information protection:

1. Listing and joining chat rooms;
2. Sending or receiving a normal chat room message; and
3. Sending or receiving a marked up message within an existing chat room.

Each operation is described in detail below. For each operation, the user's identity is taken from the IM session properties; the IM server authenticates users when the session is established. This authentication is based on the user's Windows domain login[10].

Additionally, since the COI and key token information assigned to a chat room do not change over time, there is no need to continuously re-query the IM server for these chat room security attributes. When the IM PEP is started, the PEP queries the IM server (over an OOB message exchange described above) for a list of available chat rooms and each chat room's security attributes. To obtain this information, the IM PEP uses it own separate connection to the XMPP server to request chat room security information. This information is retrieved using standard XMPP protocol IQ messages[11] to retrieve the chat room's

[10] For the Transverse client, the supported authentication is username/password based on the user's Windows domain account. Other IM client can leverage Windows domain credentials for SSO operation.
[11] XEP-0045: Multi User Chat, an XMPP protocol extension

description. This XMPP session is transient; once the chat room information has been acquired, the PEP closes the OOB connection with the IM server.

The security attributes for each chat room, therefore, is represented in a local cache on the PEP and can be referenced when formulating SMSB message requests to leverage the SSG services.

Each IM operation where the PEP applies information protection logic is described in detail below.

Listing Available Chat Rooms

A user can request a list of available chat rooms. In keeping with the SAMSON information protection philosophy, users should only see those chat rooms for which they have a policy right to access. The list of available chat rooms should therefore be filtered based on the result of a policy decision by the AS. In this way, filtering a list of chat rooms is similar to filtering a directory listing as described in the File Sharing PEP.

After a user connects to the IM server via the IM PEP, the IM client can request a list of the available chat rooms.

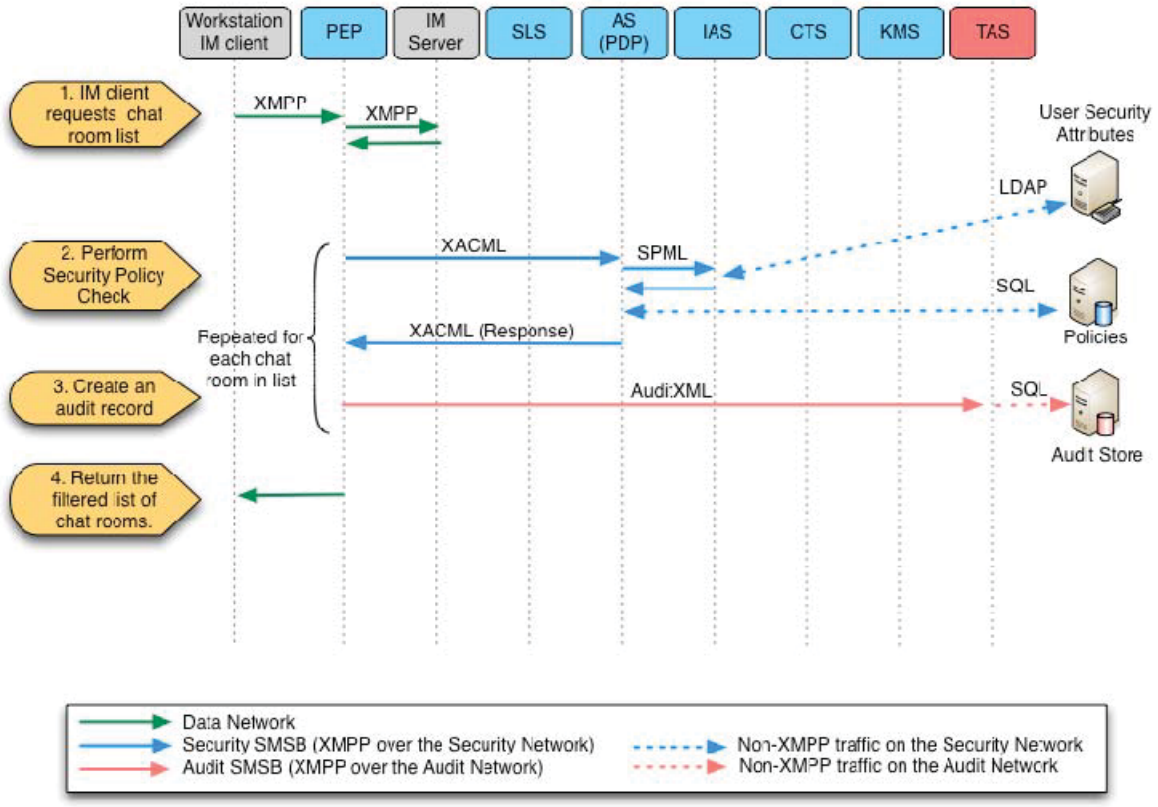


Figure 30: IM PEP - Chat Room Listing

1. The IM client sends an XMPP DISCO (discovery) message that requests the list of available MUC rooms (multi-user chat). The IM PEP forwards this request (unmodified) on to the back end IM server. The IM server returns the list of chat rooms to the PEP where the PEP will filter the list according to the security policy.

2. For each chat room in the list, the PEP calls the AS to determine if the user has the policy right to see that room. The PEP formulates and sends a policy request message to the AS, using the message format specified in section 4.2.2: AS Messaging and Operation. In this message, the user's identity is taken from the IM session and the resource is the default COI for the chat room as specified in the local data structure (cache) for chat room security attributes. The action is "READ" since in the SAMSON TD architectural deployment, viewing a chat room is deemed to be a READ operation. If the policy decision is to permit the user to see the chat room, the chat room is allowed to remain in the chat room list that is returned to the user's IM client, otherwise, the chat room is removed from the list.

3. For each policy decision, the IM PEP will create an AuditXML formatted audit record, based on the message format specified in 4.6.2: TAS Messaging and Operation, that specifies the IM chat room list operation details and send the record to the TAS to create a permanent record of the transaction.

4. The filtered list of available chat rooms that the user has a policy right to see is returned to the user's IM client.

Joining a Chat Room

Once a user has connected to the XMPP domain, via the IM PEP, they can request to join a SAMSON protected chat room.

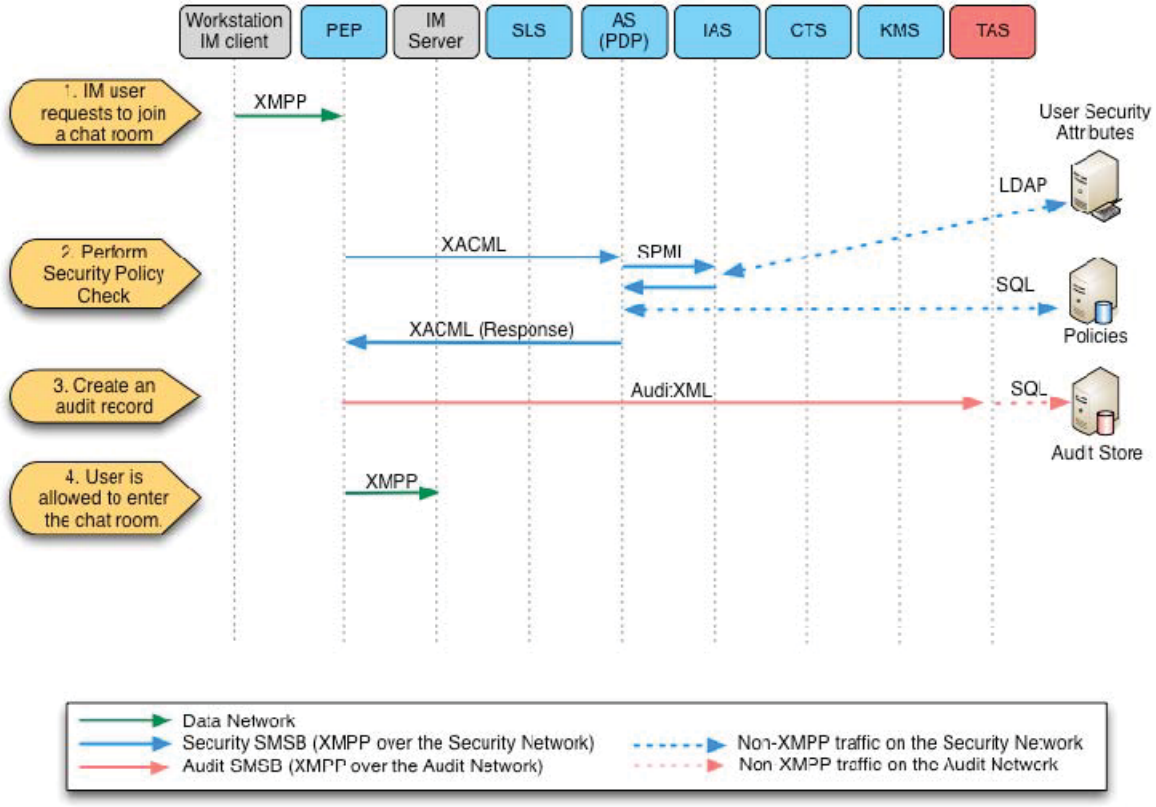


Figure 31: IM PEP - Joining a SAMSON Protected Chat Room

1. The user sends a request to join a chat room and the IM PEP intercepts this join request.

2. The PEP calls the AS to determine if the user has the policy right to join that room. The PEP formulates and sends a policy request message to the AS, using the message format specified in section 4.2.2: AS Messaging and Operation. In this message, the user's identity is taken from the IM session and the resource is the default COI for the chat room as specified in the local data structure for chat room security attributes. The action is "WRITE"; in the SAMSON TD architectural deployment entering a chat room is a WRITE operation.

3. For each policy decision, the IM PEP will create an AuditXML formatted audit record, based on the message format specified in 4.6.2: TAS Messaging and Operation, that specifies the IM chat room join operation details and send the record to the TAS to create a permanent record of the transaction.

4. If the policy decision was "permit", the user is allowed to enter the chat room, otherwise, the messaging session with the chat room is not established and the user cannot send messages to or receive messages from the chat room.

Sending an IM message to a Chat Room

In order for a user to be able to send a message to a chat room, that user must already have joined the chat room. Joining a chat room through the IM PEP implies that the user was granted access to the chat room after a security policy check was made to ensure that the user has the policy right to access the chat room's content. Therefore, it is not necessary to perform a policy check on individual messages since the user's participation in the chat room already implies that they have the policy right to send messages through the chat room.

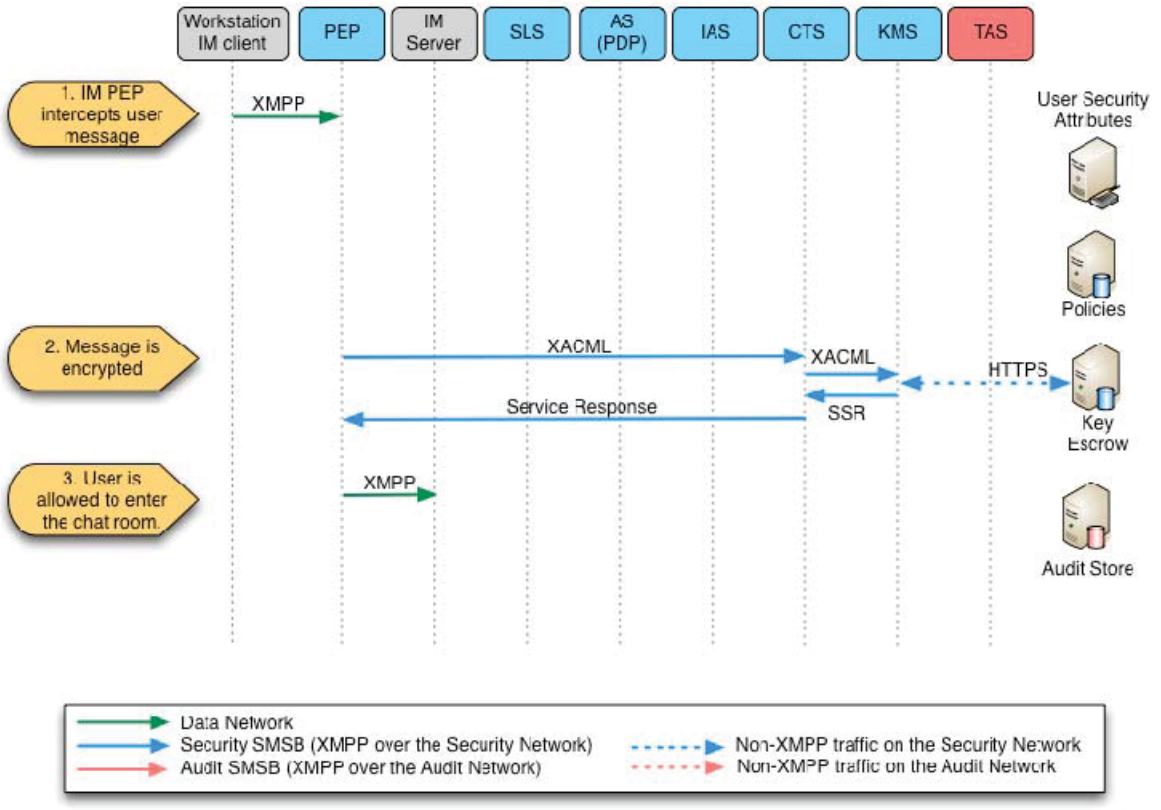


Figure 32: IM PEP - Sending a Message within a SAMSON Protected Chat Room

1. User sends a message to the IM server; the IM PEP intercepts the message. The IM PEP writes the message content out to a working file in the local staging area on the PEP machine.

2. The PEP calls the CTS using the message format described in 4.4.2: CTS Messaging and Operation, to encrypt the message in the working file. When the IM PEP was started, the PEP retrieved the security attributes for the chat room,

including the key token that references the key that is to be used to encrypt this chat room's messages. The CTS will leverage the KMS, using the format specified in section 4.3.2: KMS Messaging and Operations, to retrieve the key for this chat room. The CTS encrypts the working file to create an encrypted version of the working file.

3. The PEP reads in the encrypted working file and base64 encodes the data (the encryption process will create characters that cannot be processed by the IM server unless the resulting object is encoded). The PEP replaces the plaintext text in the message with the encrypted and encoded text.

4. The PEP transmits this newly protected message to the IM server.

It is significant to note that there is no audit record generated for this operation. Since there is no policy decision being made, there is little information to record in an audit record (other than the fact that a message was sent).

Receiving a Message from a Chat Room

Messages are delivered to all participants in the chat room in two cases:

1. When a user enters a chat room, a set of recent messages are sent to the user (this number of messages that are sent to a user is configurable at the IM server); and

2. When a user sends a new message to the chat room, this message is then sent to all participants in the chat room.

Messages are stored in encrypted form at the IM server. Therefore, when the message traverses the IM PEP on the way to the user's IM client the IM PEP will decrypt the message for the user. Since each user has a separate connection to the IM server, separate copies of the message are sent to each user and each message must be individually decrypted.

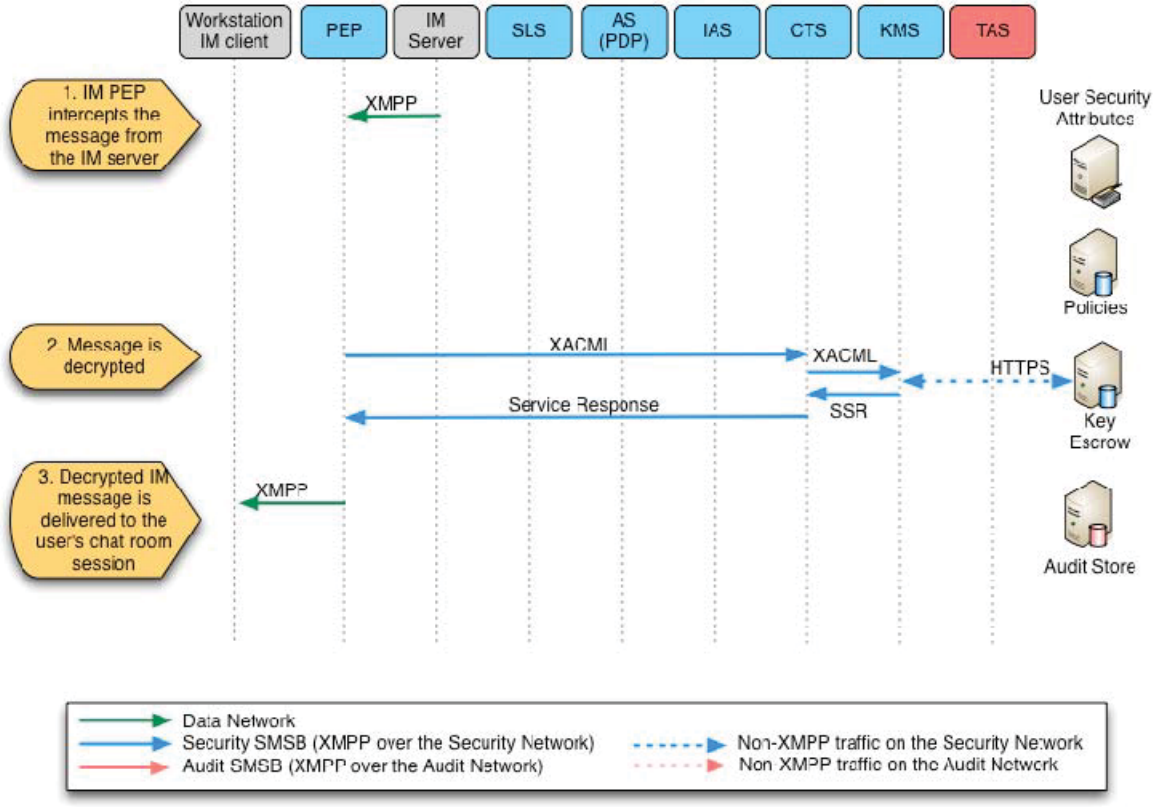


Figure 33: IM PEP - Receiving a Message within a SAMSON Protected Chat Room

1. The IM server sends a message to the user's IM client; the IM PEP intercepts the message. The IM PEP writes the message content out to a working file in the local staging area on the PEP machine. The PEP will decode the message to restore the object as it was encrypted by the PEP when it was sent.

2. The PEP calls the CTS using the message format described in 4.4.2: CTS Messaging and Operation, to decrypt the message in the working file. When the IM PEP was started, the PEP retrieved the security attributes for the chat room, including the key token that references the key that is to be used to encrypt this chat room's messages. The CTS will leverage the KMS, using the format specified in section 4.3.2: KMS Messaging and Operations, to retrieve the key for this chat room. The CTS decrypts the working file to create a decrypted version of the working file.

3. The PEP reads in the decrypted working file and replaces the plaintext text in the message with the decrypted text. The PEP transmits this newly released message to the user's IM client.

As with the message send operation, there is no audit record generated for this operation.

Sending a Marked Up Message

In the SAMSON TD architectural deployment, the IM PEP provides for the protection messages that are individually labelled with their own COI information. These "marked up" messages are specified by placing a COI (enclosed in brackets) in the message's initial characters. For example, the message "(CEO) This is for Canada only" is a marked up message for the CEO community. Marked up messages are handled differently that non marked up messages, specifically:

1. Each marked up message is checked against the security policy to ensure the sender has the policy right to create a message for this community;

2. Each marked up message is encrypted with a key that is unique for that chat room/COI combination; and

3. Since a marked up message requires a policy check, the creation of these messages is an auditable event.

Recall that when the IM PEP is started, it retrieves chat room security attributes for each chat room, including the key tokens for the keys that were used to protect the chat room's messages (both marked up and non-marked up). If, during the course of a chat room session, a user requests to mark up a message with a new COI, a new key must be generated to protect this new chat room/COI combination. Also, the chat room's security attributes must be pushed back to the IM server.

The handling of a chat room marked up message is shown in the following diagram.

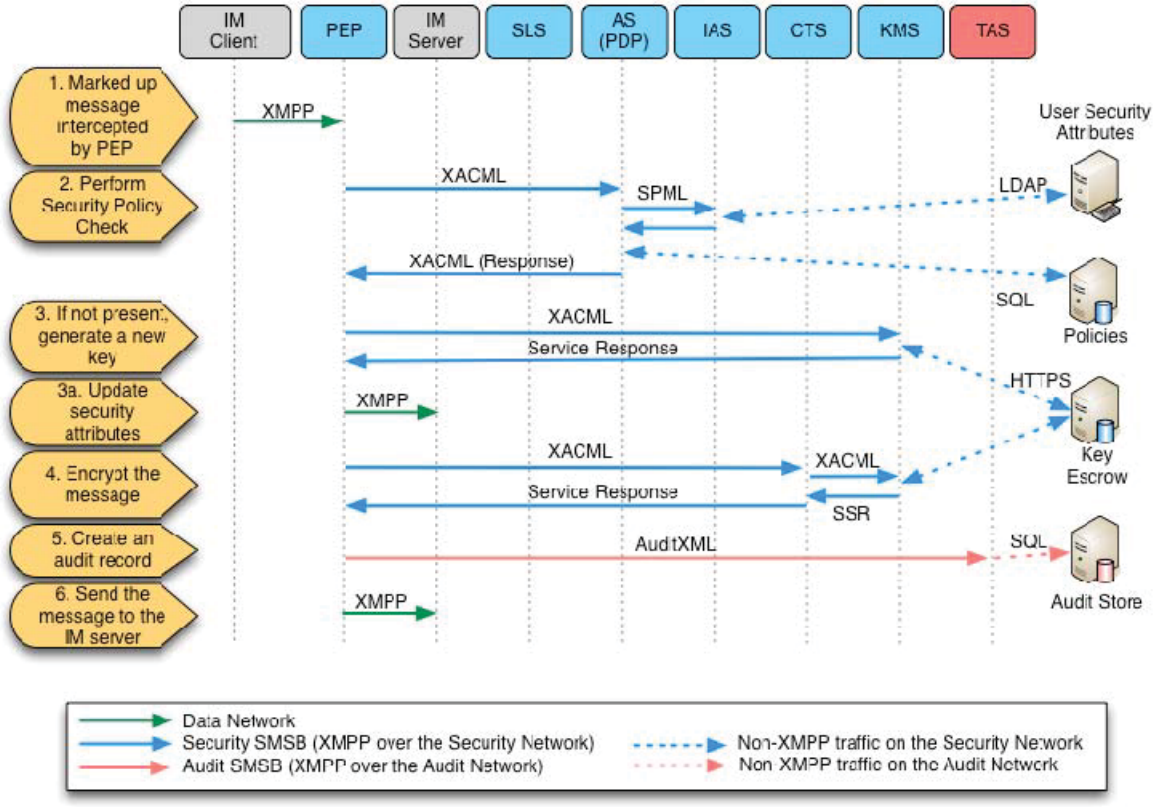


Figure 34: IM PEP - Sending a Marked Up Message

1. The user sends a marked-up message to the IM Server and the IM PEP intercepts the message. The PEP detects that this is a marked up message by parsing the message and determining the COI that should be applied to this message. The IM PEP writes the message content out (including the marked up text) to a working file in the local staging area on the PEP machine.

2. The PEP calls the AS to determine if the user has the policy right to send a message that has been marked up with that COI. The PEP formulates and sends a policy request message to the AS, using the message format specified in section 4.2.2: AS Messaging and Operation. In this message, the user's identity is taken from the IM session, the resource is the marked up COI in the message and the action is "WRITE". In the SAMSON TD architectural deployment, marking up a message is a WRITE operation.

3. The IM PEP checks the local cache to see if there is a key token for this chat room/COI combination. If no such key exists, this is the first time this COI will have been used in this chat room and a new key must be generated to protect these messages. The PEP requests a new key from the KMS, using the format specified in section 4.3.2: KMS Messaging and Operations to generate a new key. In this

exchange, the KMS will create a new key, store the key in the escrow system and return the key token to the IM PEP. At the end of this exchange, there is now a key available to protect the message.

   a. This key token, however, must be stored at the IM Server so that there is a permanent record of the chat room / COI key token. Recall that when the IM PEP starts, it reads (in an OOB message exchange) the security attributes for all SAMSON protected chat rooms. If a new key has been generated by the PEP, the associated key token must be sent to the IM Server so that the mapping of chat room / COI to key token can be retrieved the next time the IM PEP is started[12]. The updated security attributes for the chat room is sent to the IM Server in an OOB message exchange as described in the previous section.

4. At this stage, a key will be available in the key escrow system (referenced through the key toke identified) that can be used to protect the marked up message. The PEP calls the CTS using the message format described in 4.4.2: CTS Messaging and Operation, to encrypt the message in the working file. The operation requested from the CTS is to encrypt the working file using the key token supplied in the request (FILE_ENCRYPT_TOKEN). The CTS, in turn, will leverage the KMS, using the format specified in section 4.3.2: KMS Messaging and Operations, to retrieve the key associated with this key token. The CTS then encrypts the working file.

5. The IM PEP will create an AuditXML formatted audit record, based on the message format specified in 4.6.2: TAS Messaging and Operation, that specifies the marked up message operation details and send the record to the TAS to create a permanent record of the transaction.

6. The PEP reads in the decrypted working file and replaces the plaintext text in the message with the following elements concatenated together:

   a. the COI of the marked up message (in brackets); and

   b. a base64 encoded version of the contents of the working file where the encrypted message was created by the CTS

   The PEP transmits this newly protected message to the user's IM client.

Note in this last step, the encrypted message is sent to the IM server with the COI data prepended (unencrypted) to the message. This mean that the marked up message the IM server received takes the form:

[12] Note that security attributes of chat rooms are not stored at the PEP itself. Every effort has been made to keep the PEPs stateless. As appliances that do not store security information, PEPs can be load balanced, hardened and maintained in a more controlled manner.

```
(COI data)h8f26hfg3rggw5hw9fwhf2fh74g2f
```

Note that the COI information, the security attribute for the message, is not encrypted. The COI data is prepended to the message so that when the message is delivered to participant in the chat room, the IM PEP recognizes this as a marked up message and provides the IM PEP with the COI to which this message belongs. The manner by which the IM PEP transmits encrypted marked up messages to chat room participants is discussed in the next section. However, since the entire message was encrypted, the marked up text will exists in two places:

1. Inside the encrypted message text; and

2. Prepended to the encrypted message.

Having the COI data both inside and outside the encrypted message is necessary to support the process when users receive the marked up message.

Receiving a Marked up Message

When a chat room message is sent from the IM server to a chat room participant, the IM PEP examines the message to determine if it is a marked up message. As described in the previous section, a message that has COI information prepended to it is deemed to be a marked up message. If the message is not marked up, the IM PEP will treat the message as a normal encrypted message and execute the procedure described in the section above *Receiving a Message From a Chat Room*. If the message is a marked up message, however, the process described in the following diagram is followed:

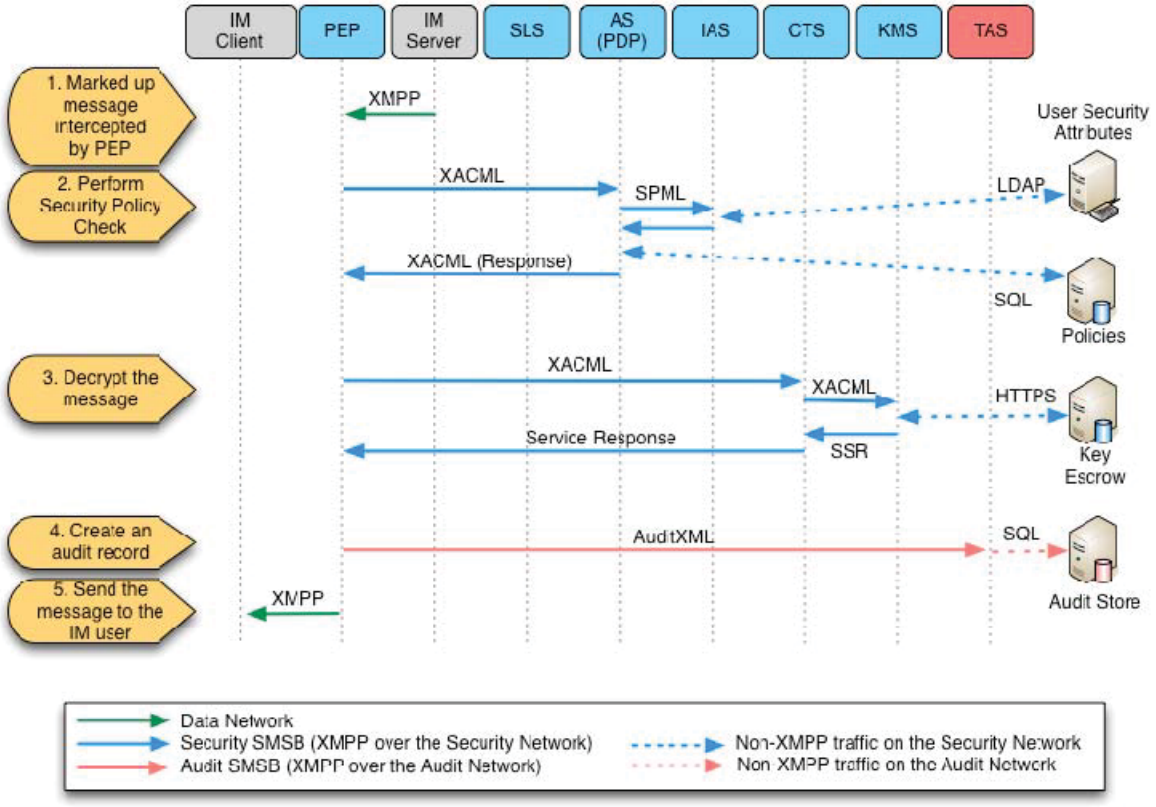


Figure 35: IM PEP - Receiving a Marked Up Message

1. The IM PEP intercepts the chat room message as it is sent from the IM Server to the chat room participant. The IM PEP examines the message to see if it is marked up and, if so, extracts the COI from the message. The PEP writes out the encrypted portion of the message, that is, everything after the prepended COI information and base 64 decodes the message to get the originally encrypted content from when the message was sent.

2. The PEP calls the AS to determine if the user has the policy right to receive a message that has been marked up with that COI. The PEP formulates and sends a policy request message to the AS, using the message format specified in section 4.2.2: AS Messaging and Operation. In this message, the user's identity is taken from the IM session, the resource is the marked up COI in the message and the action is "READ". In the SAMSON TD architectural deployment, reading a marked up message is a READ operation.

3. Using the local cache of chat room security attributes, the key token for this chat room/COI combination is used in a call to the CTS to decrypt the protected message. The PEP calls the CTS using the message format described in 4.4.2: CTS

Messaging and Operation, to decrypt the message in the working file. This call to the CTS includes the key token for the message so that the CTS can request the appropriate key from the KMS with which to apply the cryptographic transformation.

4. The IM PEP will create an AuditXML formatted audit record, based on the message format specified in 4.6.2: TAS Messaging and Operation, that specifies the marked up message operation details and send the record to the TAS to create a permanent record of the transaction.

5. The PEP reads in the decrypted working file and replaces the plaintext text in the message with the decrypted text. The PEP transmits this newly released message to the user's IM client.

Note that since marked up messages are encrypted in their entirety (the marked up COI and the message content), the IM user will see the prepended COI information and recognize this as a marked up message that was sent to a restricted community. Many IM clients can be tailored to detect when strings of characters have been used, such as "(CEO)". To enhance the user experience at the endpoint, the SAMSON TD architectural deployment configured the Transverse chat software to use of flag icons to identify when CEO and CANUS marked up messages were received. The endpoint chat software should be configured to clearly indicate caveat level and classification in the visual markings in the chat message / room.

5.3.3.3 IM PEP Trust Model

The IM PEP contains many levels of protections that prevent information from being disclosed to unauthorized individuals. Both the front-end (IM client to IM PEP) and back end (IM PEP to IM Server) communications are TLS encrypted so that messages cannot be disclosed in transit.

While an IM chat room message is stored at the IM server, it exists as an encrypted object that is uniquely keyed to the chat room / COI combination. Privileged users, such as the IM administrator, are not able to disclose messages that are stored at the IM server since the message must be decrypted using keys that are only available to the IM PEP. The policy and cryptographic actions taken by the IM PEP are performed on a hardened appliance. Finally, policy-based transactions are audited so that there is a tamper-resistant record of all user activity where information has been disclosed to the SAMSON user community. Policy-based transactions are transactions that require a security policy check prior to execution and, for IM information protection, consists of the following actions:

- Joining a chat room
- Marking up a new message; and
- Receiving a marked up message.

5.3.4 *Web Session PEP*

The Web Session PEP (Web PEP) is one variety of PEP that operates on web data. This PEP limits access to only those users that have a policy right to use the web services protected by the PEP. For the SAMSON TD architectural deployment, this PEP does not currently provide cryptographic protection; rather, only access to the back end web service is subject to policy access restrictions. As with all PEPs, the intent is for SAMSON information protection to be added to an existing network infrastructure as a security overlay; the introduction of SAMSON data-centric security practices should not necessitate modifications to either the client endpoint or the back end web server.

5.3.4.1 Web PEP Architecture

Deployment of the Web PEP requires that the PEP be configured as a proxy for the back end web server. That is, when users connect to the Web PEP the information requests are forwarded on the back end web service. The proxying behaviour of the Web PEP is achieved using the open source *mod_proxy* module that is part of the Apache 2 Web Server. With the proxy module in place and properly configured, the Web PEP intercepts information request that are destined for the back end web server.

The PEP logic itself is implemented as a separate Apache module within the same Apache deployment that is supplying the proxy behaviour. The PEP module is referenced before the proxy module in the call stack so that users must first pass the SAMSON security policy check before their information request is proxied through to the target web service.

The Web PEP is fully compatible with the concept of *virtual hosts*: the idea that a single web server can service multiple FQDNs at the same time. A user can request information from three separate FQDNs not realizing that it is the same web server handling the processing for all three requests. This architecture is shown in the following diagram.

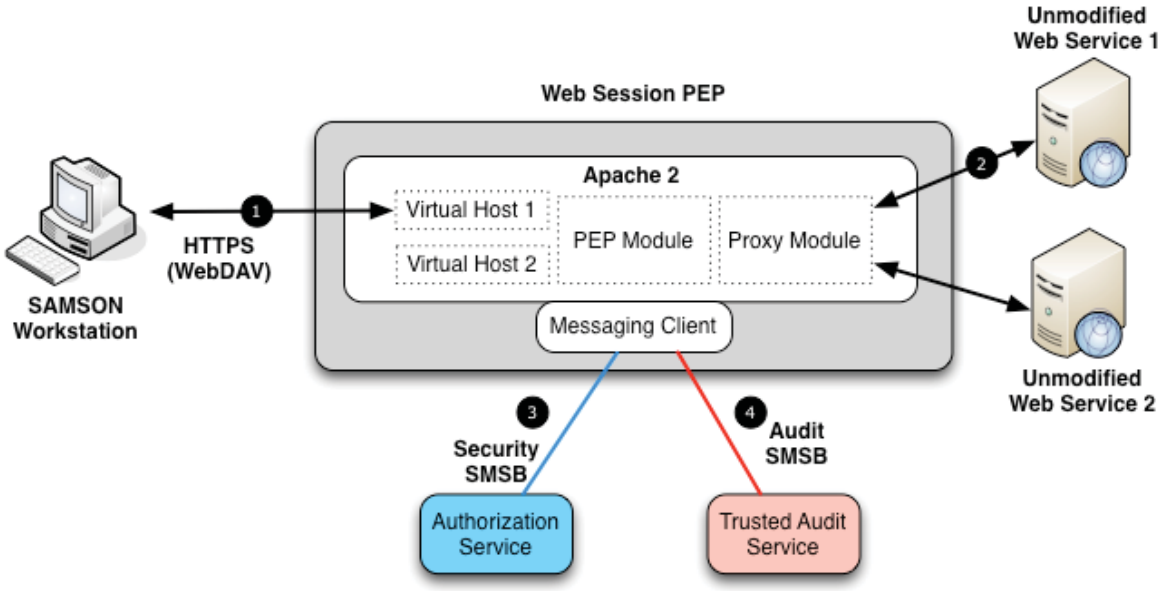


Figure 36: Web Session PEP Architecture

1. In this scenario, a user connects, via HTTPS, to the FQDN of the first Apache virtual host on the DATA network.

2. If the PEP information processing logic determines that the user has the policy right to access the back end web server, the information request is forwarded on the to associated back end web service.

3. Since the architecture of the Web PEP follows the general PEP design, the Web PEP has a messaging client that is a participant on the Security SMSB. Through this messaging infrastructure, the Web PEP leverages the AS to obtain the security policy decision as to whether the user should be allowed access to the back end web service.

4. The Web PEP is also a participant on the Audit SMSB so that an audit record of the user's access to the protected web service can be stored at the TAS.

As with all other PEPs, the policy decision to grant a user access to protected data requires the user's identity and the security label on the protected resource. For the SAMSON architectural deployment, the user's identity is determined though an Apache-based login capability that prompts the user to supply their Windows domain account and associated password. When the Apache server receives this information, the server performs an LDAP authentication against the Windows Active Directory to verify the user's credentials. Once

authenticated, the Apache server will know the user's identity for the duration of that web session.

The security label for each web service is set at the Web PEP. The SSL certificate that is used to protect the session between the user's workstation and the PEP includes, as one of the certificate attributes, the COI for the back end web service. For the SAMSON TD deployment, the SSL certificates that were used at the PEP included Netscape certificate extensions and the web service's COI was allocated to the NS_COMMENT field within that certificate extension. A separate certificate is used for each virtual host.

When the Web PEP Apache2 server is started, it reads and caches the COI information from the hosting certificated for each virtual host so that when a user connects to a virtual host, the PEP has the appropriate COI to apply for the policy check to see if the user has the right to access the back end service.

5.3.4.2 Web PEP Messaging and Operation

When a user requests access to a web service via the Web PEP, the PEP will first ensure that the user's session is authenticated. As previously described, if the session is not authenticated, the Apache server will prompt the user for a username/password and then authenticate the user through the Windows Active Directory. Once authenticated, the Web PEP will know the identity of the user and the COI that applies to the web service being accessed.

The information processing logic to handle the request is shown in the following diagram.

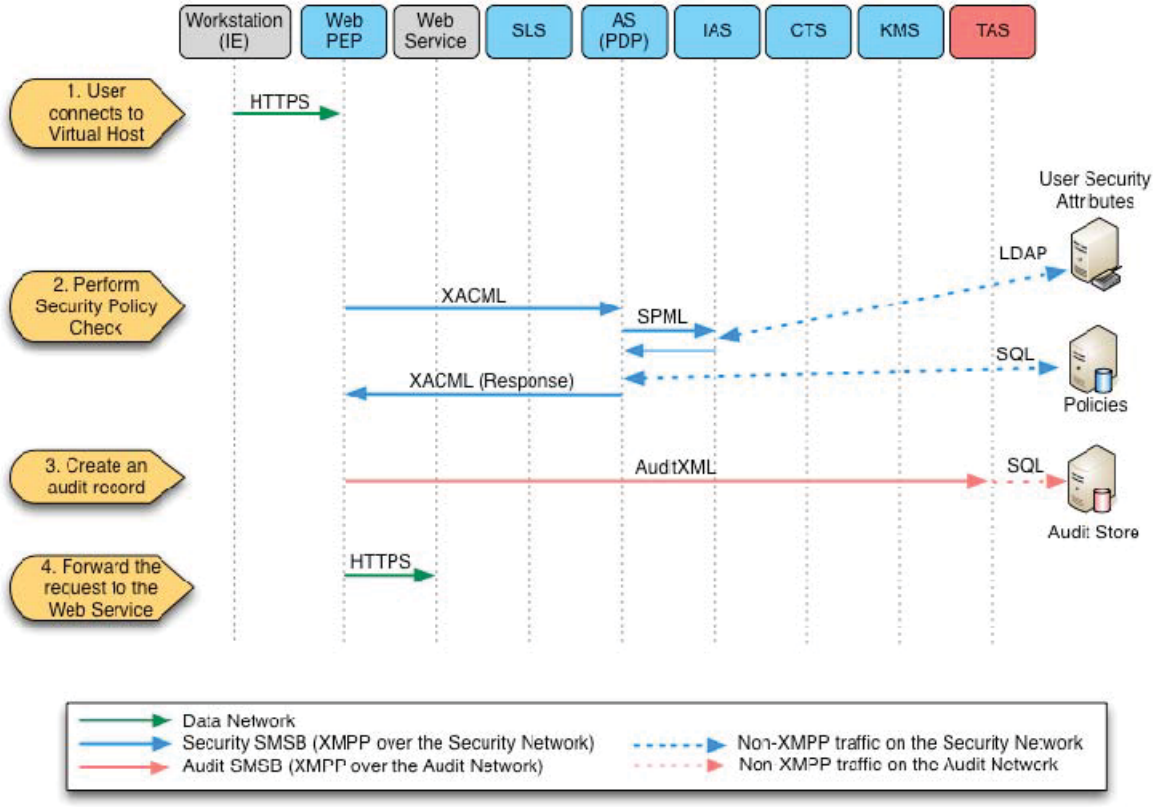


Figure 37: Web PEP - Accessing a SAMSON Protected Web Service

1. The user's information request is sent to the Web PEP over TLS/SSL. The PEP has the COI for the requested virtual host from the SSL certificate for the virtual host. The PEP also has the user's identity for this web session.

2. The PEP calls the AS to determine if the user has the policy right to access this Web Service, given the COI to which it belongs. The PEP formulates and sends a policy request message to the AS, using the message format specified in section 4.2.2:AS Messaging and Operation. In this message, the user's identity is taken from the web session, the resource is the COI from the certificate and the action is "READ". In the SAMSON TD architectural deployment, accessing a web service is a READ operation.

3. The Web PEP will create an AuditXML formatted audit record, based on the message format specified in 4.6.2:TAS Messaging and Operation, that specifies the web service access operation details and send the record to the TAS to create a permanent record of the transaction.

4. If the policy decision was to allow the user access to the web service, the PEP allows the message to be forwarded to the destination by the Apache proxy module.

If the policy decision denies the user access to the back end service, an HTTP error code 403 (Not Authorized) is returned to the user's web session.

5.3.4.3 Web PEP Trust Model

The front-end connection between the user's workstation and the Web PEP and back end connection between the Web PEP and the web service are protected using TLS/SSL. While not configured for the SAMSON TD architectural deployment. It is recommended that the back end connection between the Web PEP and the web service be either:

- Hosted on its own network so that the only way to access the back end service is through the Web PEP; and/or
- Limited with host-based firewall at the web service so that only the Web PEP is the only machine allowed establishing a connection with the web service.

6.0 Self-Protecting SAMSON Services

Whereas section 4.0: The SAMSON Security Services describes the SSGs and section 5.0: The SAMSON Data Intercept Strategy describes how the PEPs leverage the SSGs to provide data centric security to applications, this section presents the SAMSON self-protection mechanisms that are present in the SAMSON TD architectural deployment. These self-protection mechanisms include the use of SAMSON PEPs to protect SAMSON administrative interfaces. This section also presents SAMSON's use of operational security tools for monitoring system integrity.

There are three administrative interfaces that are protected by SAMSON:

1. The administrative interface to edit the security policy;
2. The administrative interface to edit user security attributes; and
3. The audit review interface.

A description of the architecture of each interface and the manner by which the interfaces are protected by SAMSON is provided in the following sections.

6.1 Policy Administration Interface (PAI)

The Policy Administration Interface (PAI) allows the Security Officer to create, modify and delete the security policies that are used by the AS. In the SAMSON TD architectural deployment, the PAI is a service that:

1. Provides a web-based interface to the security officer on the front end; and
2. Includes database connectivity that can be used to retrieve and set policies at the security policy repository.

Since the PAI is a web interface, access to the service can be controlled through the use of a SAMSON Web PEP, as documented in section 5.3.4: Web Session PEP.

The architecture of the PAI, therefore, can be viewed in the following diagram.

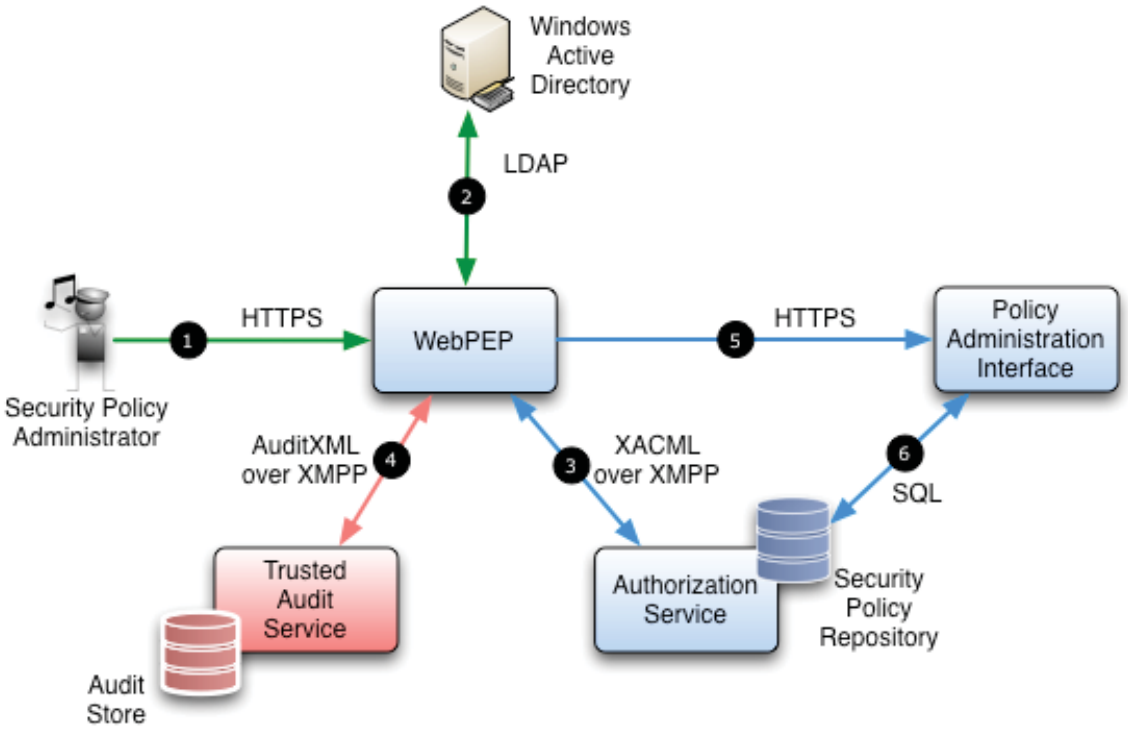


Figure 38: Self-Protected Policy Administration Interface

The Security Policy Administrator connects to the Web PEP that has been configured to gate access to the PAI. Access to the Web PEP is made over the Data network.

1) This connection is accordance with the Web PEP design, that is, the connection is over a TLS protected link and the server certificate used to protect this web host contains an attribute that defines the COI for the PAI. As previously described, the Web PEP in the SAMSON TD architectural deployment uses the NS_COMMENT attribute in the Netscape certificate extension in the server certificate to house COI data.

2) Additionally, as previously stated, the Web PEP requires that a user authenticate to the Windows domain to establish their identity. The Web PEP accepts the user's credentials and verifies them with Windows Active Directory using LDAP authentication.

3) At this stage, the Web PEP has the user's identity and the COI for the resource being requested. As described in section 5.3.4: Web Session PEP, the Web PEP has the information necessary to submit an authorization request to the AS. If the user has the policy right to access this resource, the information request is proxied to the PAI. If the access request is denied, an HTTP 403 Forbidden message is returned to the user. In either case, an audit record is written to the TAS.

4) An audit record of the attempt to access the Policy Administration Interface is stored through the Trusted Audit Service.

5) Access to the PAI is limited in two ways:

   a) The proxied request is sent over the SECURITY network to which users have no direct access; and

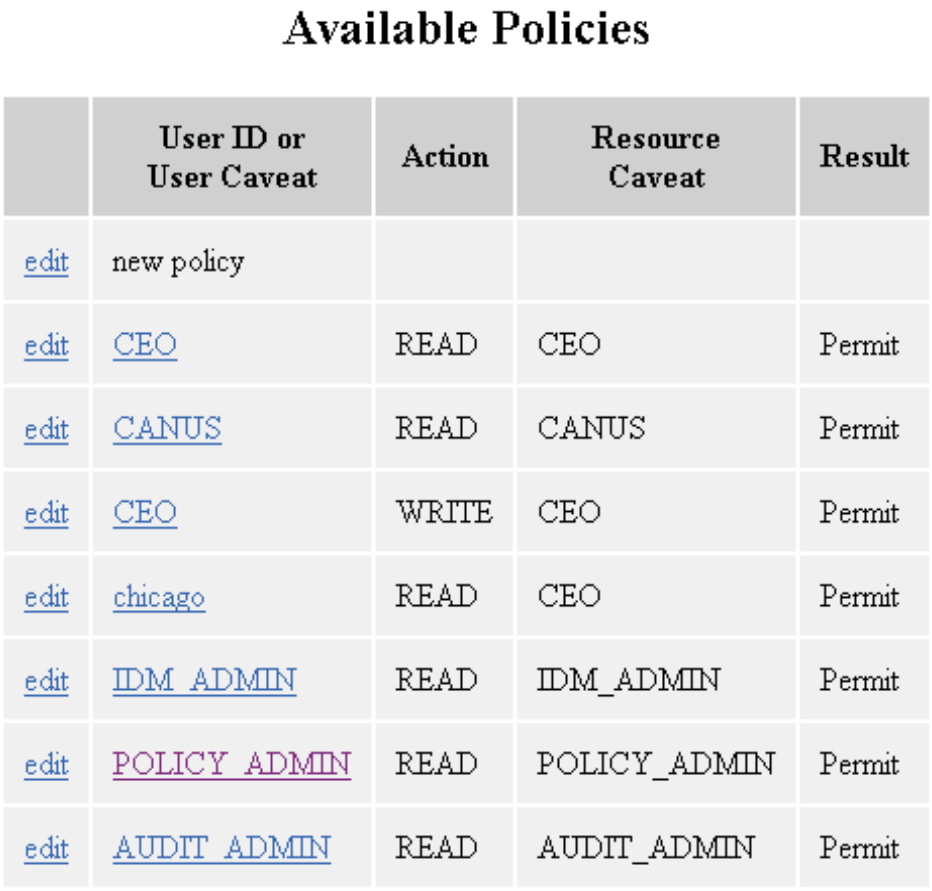
   b) The PAI is configured (host-based firewall) to only allow access from the Web PEP.

   As such, only authorized users will get access to the PAI. Through this interface, current policies are displayed, new policies can be created and existing policies can be deleted. The PAI interface is shown in Figure 39: PAI Web-based Interface.

6) In the SAMSON TD architectural deployment the Security Policy Repository is a MySQL database. The PAI data exchanges with the Security Policy repository are made using SQL over the SECURITY network. The PAI holds, within its configuration, the necessary database account access to write policies.

Once the Security Policy Repository has been updated, any policies are immediately enforced. Since for every policy request the AS retrieves all applicable policies, the AS is always referencing the latest set of policies.

To grant access to the PAI, therefore, there must be a rule that grants the SO (user, group or role) READ access to the PAI's COI. It is significant to note that the PAI is used to alter the very security policies that are used to gate access to the PAI. As a result, it is possible to lock the Security Officer out of the PAI entirely by deleting the policy rule that allows access to the PAI. Re-instatement of such a policy rule must be manually entered at the AS command line. An example of such a rule is shown below.



Available Policies

	User ID or User Caveat	Action	Resource Caveat	Result
edit	new policy			
edit	CEO	READ	CEO	Permit
edit	CANUS	READ	CANUS	Permit
edit	CEO	WRITE	CEO	Permit
edit	chicago	READ	CEO	Permit
edit	IDM_ADMIN	READ	IDM_ADMIN	Permit
edit	POLICY_ADMIN	READ	POLICY_ADMIN	Permit
edit	AUDIT_ADMIN	READ	AUDIT_ADMIN	Permit

Figure 39: PAI Web-based Interface

In this sample policy list, users that are members of the POLICY_ADMIN community have the policy right to access (READ) resources that are assigned to the POLICY_ADMIN community. A Security Officer would need to be part of the POLICY_ADMIN COI to be granted access to the PAI.

It is assumed that the target environment generates the certificates that are used to label the Web PEP virtual hosts. Since the HTTPS interface is on the DATA network, that is, the target environment's operational network, the certificates that protect those services should link into the target environment's CA.

6.2 Identity Attribute Administration Interface (IAAI)

The Identity Attribute Administration Interface (IAAI) allows the Identity Administrator to set the security attributes for SAMSON users. For the SAMSON TD architectural deployment, the management of user security attributes (the attributes used by SAMSON for policy decision) is governed by the following workflow:

- The Windows domain (Active Directory) is the authoritative source for the user's account and credentials (not managed through SAMSON)

- The Security Attribute Repository, on the SECURITY network, is the authoritative source for users' security attributes, including nationality, clearance level and membership in communities of interest.

- The IA must manually enter the entries for user's security attributes; there is currently no mechanism to automatically load a list of users from Active Directory into the repository.

- To ensure the integrity of the architecture, the IAS and the security attribute repository are not directly accessible to the user community. However, a user's security attributes must be available at the endpoint in order to support labelling solution (labelling solution must be able to display those attributes that are available to the user). As a result, attributes from the authoritative source (the security attribute repository) are pushed to the non-authoritative source (Active Directory) where they can be accessed from the DATA network. The attributes that are read from Active Directory do not require strong integrity since they are only used for labelling; when policy decisions are performed on data assets, the security attributes from the authoritative source (high integrity) are used.

In the SAMSON TD architectural deployment, the IAAI is a service that:

1. Provides a web-based interface to the identity administrator on the front end; and

2. Includes LDAP connectivity that can be used to retrieve and set user security attributes at the security policy repository.

For this deployment, the IAAI uses phpLDAPadmin[13]: an open source, web-based LDAP client that provides basic administration for LDAP servers.

Similarly to the PAI, the IAAI is a web interface and access to the service can be controlled through the use of a SAMSON Web PEP. The architecture of the IAAI, therefore, can be viewed in the following diagram.

[13] http://phpldapadmin.sourceforge.net/wiki/index.php/Main_Page

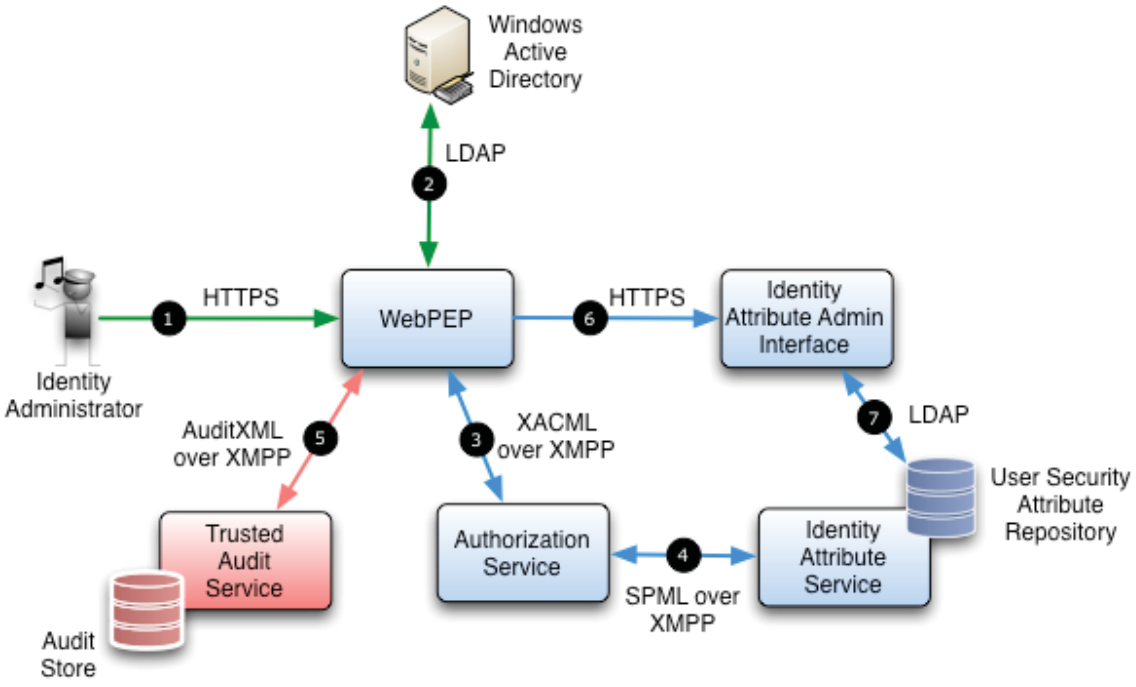


Figure 40: Self-Protected Identity Attribute Administration Interface

The IA connects to the Web PEP that has been configured to gate access to the IAAI. Access to the Web PEP is made over the Data network.

1) This connection is accordance with the Web PEP design, that is, the connection is over a TLS protected link and the server certificate used to protect this web host contains an attribute that defines the COI for the IAAI. As previously described, the Web PEP in the SAMSON TD architectural deployment uses the NS_COMMENT attribute in the Netscape certificate extension in the server certificate to house COI data.

2) Additionally, as previously stated, the Web PEP requires that a user authenticate to the Windows domain to establish their identity. The Web PEP accepts the user's credentials and verifies them with Windows Active Directory using LDAP authentication.

3) At this stage, the Web PEP has the user's identity and the COI for the resource being requested. As described in section 5.3.4: Web Session PEP, the Web PEP has the information necessary to submit an authorization request to the AS. If the user has the policy right to access this resource, the information request is proxied to the IAAI. If the access request is denied, an HTTP 403 Forbidden message is returned to the user. In either case, an audit record is written to the TAS.

4) As described in section 4.2.2: AS Messaging and Operation, the AS retrieves user's security attributes from the IAS and uses these attributes in the evaluation of policy decisions.

5) An audit record of the attempt to access the Identity Attribute Administration Interface is stored through the Trusted Audit Service.

6) Access to the IAAI is restricted in two ways:

   a) The proxied request is sent over the SECURITY network to which users have no direct access; and

   b) The IAAI is configured (host-based firewall) to only allow access from the Web PEP.

   As such, only authorized users will get access to the IAAI. Through this interface, current policies are displayed, new policies can be created and existing policies can be deleted. The IAAI interface is shown in *Figure 40: Self-Protected Identity Attribute Administration Interface.*

7) In the SAMSON TD architectural deployment the Security Attribute Repository is an LDAP directory. The IAAI data exchanges with the Security Attribute Repository are made using LDAP over the SECURITY network. The IAAI holds, within its configuration, the necessary LDAP account access to write and modify directory entries.

Once the security attribute repository has been updated, users' new security attributes are reflected in any subsequent policy checks. That is, when the AS queries the IAS for users' security attributes, the latest security attributes are retrieved from the security attribute repository.

To grant access to the IAAI, therefore, there must be a rule that grants the IA (user, group or role) READ access to the IAAI's COI. It is significant to note that the IAAI is used to alter the very user security attributes that are used to gate access to the IAAI. As a result, it is possible to lock the Identity Administrator out of the IAAI entirely by deleting the user security attributes that allows access to the IAAI.

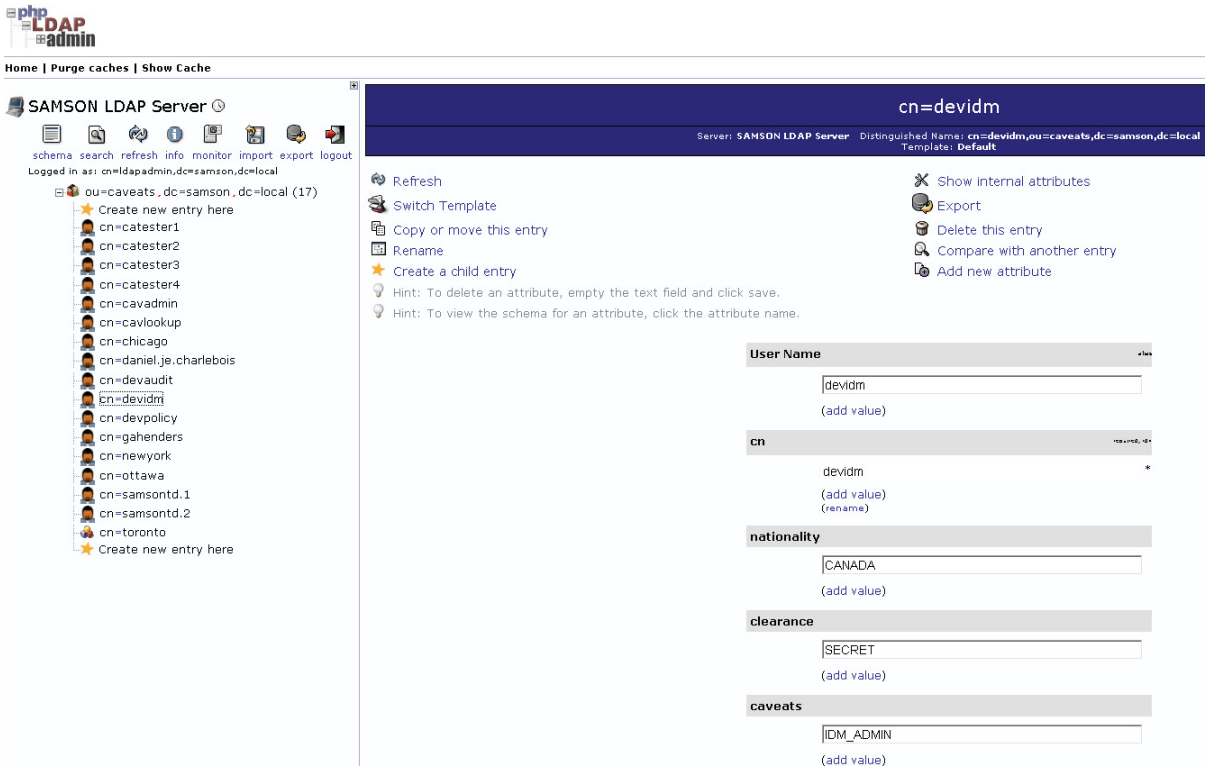


Figure 41: IAAI Web-based Interface

In this sample security attribute list, users that are members of the IDM_ADMIN community have the policy right to access (READ) resources that are assigned to the IDM_ADMIN community. An Identity Administrator would need to be part of the IDM_ADMIN COI to be granted access to the IAAI.

6.2.1 *Identity Attribute Synchronization*

The previous section described the use of authoritative and non-authoritative sources of user security attributes. The authoritative source, used in policy decisions, is the Security Attribute Repository. The non-authoritative source, used to provide end points with security attribute information for purposes such a labelling, is Active Directory. It was stated that attributes are pushed from the authoritative source to the non-authoritative source. This process is described in this section.

The pushing, or one-way synchronizing, of security attributes is done by a SAMSON support process: the Identity Attribute Synchronization Utility. This utility is typically deployed on the IAS system itself although it is not dependant on being co-located with any other SAMSON component. Once started, the process will periodically perform the synchronization activities.

The synchronization activity is shown in the following diagram.

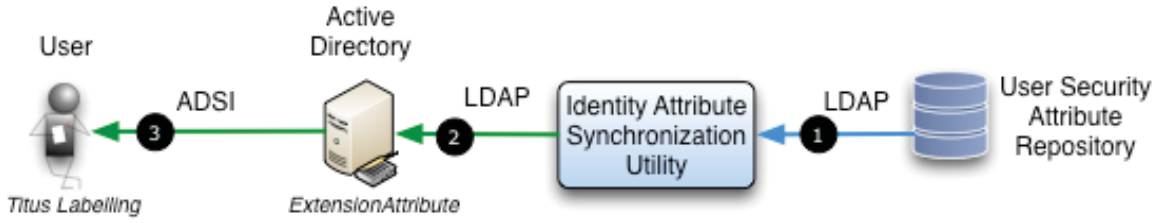


Figure 42: Synchronizing Security Attributes

1) The utility connects to the Security Attribute Repository on the SECURITY network and retrieves the security attributes for each user.

2) The utility connects to Active Directory on the DATA network and updates each user's AD entry with their security attributes.

The mapping between the LDAP and AD schema equates each user's LDAP common name with their Active Directory *SAMAccountName*.

For the SAMSON TD deployment architecture, only the COI information is synchronized. This COI data is stored in an *extensionAttribute*: a set of free-to-use locations within the AD user schema.

When launches, the utility requires the operator to enter appropriate account credentials for both the LDAP and AD connections. Once successfully started, the utility will continue to synchronize the security attribute repository with Active Directory.

With security attributes synchronized and available on the DATA network, endpoint software products such as the Titus suite of labelling solutions can retrieve attribute data by querying the chosen *extensionAttribute*. This information can be accessed through a variety of AD supported communication protocols including LDAP or, as is the case with Titus, the AD Scripting Interface (ADSI).

6.3 Audit Review Interface (ARI)

The Audit Review Interface (ARI) allows the Audit Reviewer to examine audit records that have been placed in the Trusted Audit Store by the Trusted Audit Service. In the SAMSON TD architectural deployment, the ARI is a service that:

1. Provides a web-based interface to the Audit Reviewer on the front end; and

2. Includes database connectivity that can be used to retrieve audit records from the TAS.

Since the ARI is a web interface, access to the service can be controlled through the use of a SAMSON Web PEP, as documented in section 5.3.4: Web Session PEP.

The architecture of the ARI, therefore, can be viewed in the following diagram.

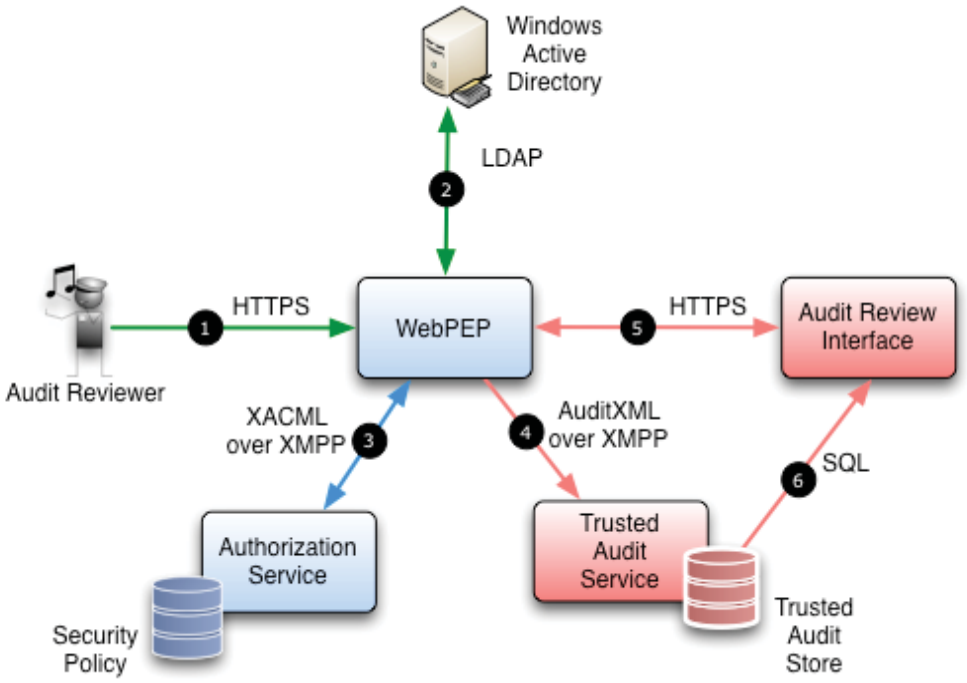


Figure 43: ARI Web-Based Interface

The Audit Reviewer connects to the Web PEP that has been configured to gate access to the ARI. Access to the Web PEP is made over the Data network.

1) This connection is accordance with the Web PEP design, that is, the connection is over a TLS protected link and the server certificate used to protect this web host contains an attribute that defines the COI for the ARI. As previously described, the Web PEP in the SAMSON TD architectural deployment uses the NS_COMMENT attribute in the Netscape certificate extension in the server certificate to house COI data.

2) As previously stated, the Web PEP requires that a user authenticate to the Windows domain to establish their identity. The Web PEP accepts the Audit Reviewer's credentials and verifies them with Windows Active Directory using LDAP authentication.

3) At this stage, the Web PEP has the user's identity and the COI for the resource being requested and an policy check is made to the AS to determine if the user has the policy

right to access the ARI. If the user has the policy right to access this resource, the information request is proxied to the ARI. If the access request is denied, an HTTP 403 Forbidden message is returned to the user. In both cases, the Web PEP sends an audit record of the transaction to the TAS.

4) An audit record of the attempt to access the Audit Review Interface is stored through the Trusted Audit Service. As described in section 4.6.2: TAS Messaging and Operation, the TAS will process the audit record, amend the record with chain-of-custody information and submit the records to the Trusted Audit Store.

5) Access to the ARI is restricted in two ways:

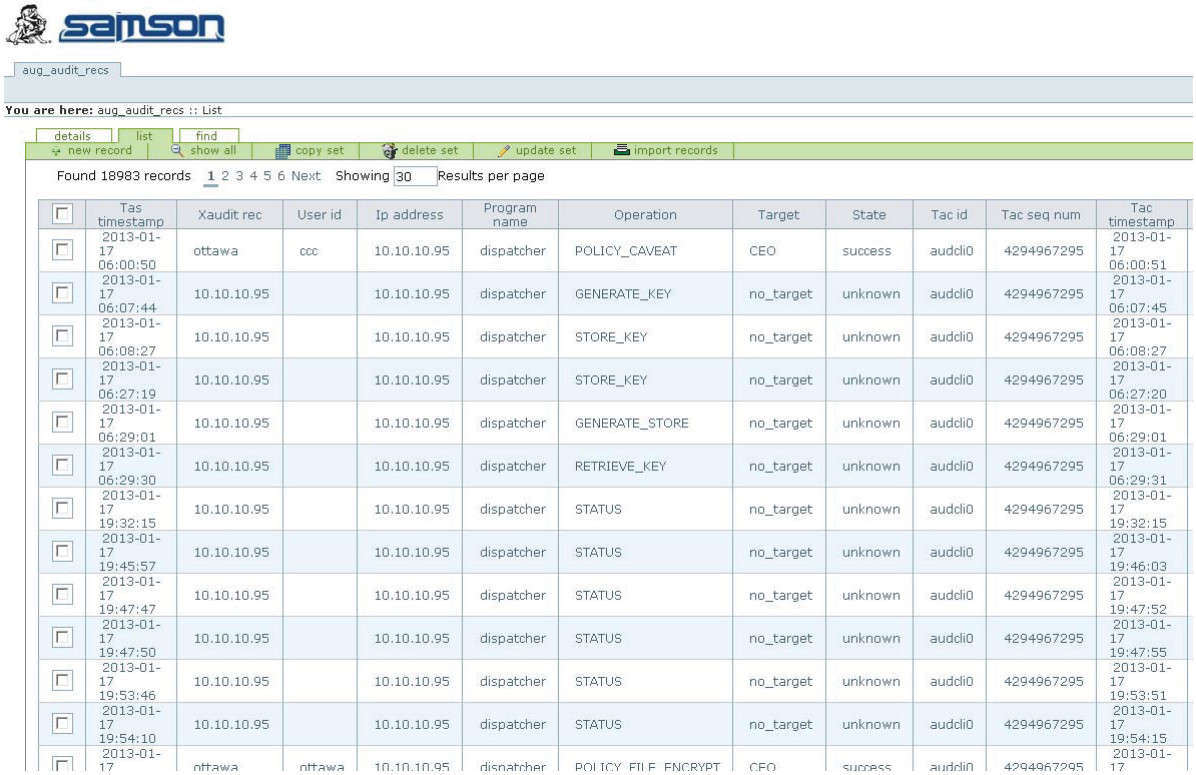
   a) The proxied request is sent over the AUDIT network to which users have no direct access; and
   b) The ARI is configured (host-based firewall) to only allow access from the Web PEP.

As such, only authorized users will get access to the ARI. Through this interface, the Audit Reviewer can view and search for audit records as part of a forensic or monitoring process. The ARI interface is shown in Figure 43: ARI Web-Based Interface.

6) In the SAMSON TD architectural deployment the Trusted Audit Store is a MySQL database. The ARI data exchanges with this repository are made using SQL over the AUDIT network. The ARI holds, within its configuration, the necessary database account access to read audit records.

For the SAMSON TD architectural deployment, a 3-tier web service was created using the Xataface[14], a web application framework that can generate forms and processing logic for MySQL hosted data stores. Under this framework, the deployed ARI presents the following interface.

[14] http://xataface.com



Tas timestamp	Xaudit rec	User id	Ip address	Program name	Operation	Target	State	Tac id	Tac seq num	Tac timestamp
2013-01-17 06:00:50	ottawa	ccc	10.10.10.95	dispatcher	POLICY_CAVEAT	CEO	success	audcli0	4294967295	2013-01-17 06:00:51
2013-01-17 06:07:44	10.10.10.95		10.10.10.95	dispatcher	GENERATE_KEY	no_target	unknown	audcli0	4294967295	2013-01-17 06:07:45
2013-01-17 06:08:27	10.10.10.95		10.10.10.95	dispatcher	STORE_KEY	no_target	unknown	audcli0	4294967295	2013-01-17 06:08:27
2013-01-17 06:27:19	10.10.10.95		10.10.10.95	dispatcher	STORE_KEY	no_target	unknown	audcli0	4294967295	2013-01-17 06:27:20
2013-01-17 06:29:01	10.10.10.95		10.10.10.95	dispatcher	GENERATE_STORE	no_target	unknown	audcli0	4294967295	2013-01-17 06:29:01
2013-01-17 06:29:30	10.10.10.95		10.10.10.95	dispatcher	RETRIEVE_KEY	no_target	unknown	audcli0	4294967295	2013-01-17 06:29:31
2013-01-17 19:32:15	10.10.10.95		10.10.10.95	dispatcher	STATUS	no_target	unknown	audcli0	4294967295	2013-01-17 19:32:15
2013-01-17 19:45:57	10.10.10.95		10.10.10.95	dispatcher	STATUS	no_target	unknown	audcli0	4294967295	2013-01-17 19:46:03
2013-01-17 19:47:47	10.10.10.95		10.10.10.95	dispatcher	STATUS	no_target	unknown	audcli0	4294967295	2013-01-17 19:47:52
2013-01-17 19:47:50	10.10.10.95		10.10.10.95	dispatcher	STATUS	no_target	unknown	audcli0	4294967295	2013-01-17 19:47:55
2013-01-17 19:53:46	10.10.10.95		10.10.10.95	dispatcher	STATUS	no_target	unknown	audcli0	4294967295	2013-01-17 19:53:51
2013-01-17 19:54:10	10.10.10.95		10.10.10.95	dispatcher	STATUS	no_target	unknown	audcli0	4294967295	2013-01-17 19:54:15
2013-01-17	ottawa	ottawa	10.10.10.95	dispatcher	POLICY_FILE_ENCRYPT	CEO	success	audcli0	4294967295	2013-01-17

Figure 44: ARI Web-based Interface

To grant access to the ARI, therefore, there must be a policy rule that grants the Audit Reviewer (user, group or role) READ access to the ARI's COI. It is significant to note that since audit records are viewed immediately after they have been posted to the Trusted Audit Store, one of the most recent audit records the Audit Reviewer will see is the audit record for the transaction that granted the Audit Reviewer access to the ARI itself. This is another clear demonstration of SAMSON services protecting SAMSON interfaces.

The AuditXML schema is presented in Annex A.6:*AuditXML Schema*.

6.4 Audit Integrity Checker (AIC)

Section 4.6.2: TAS Messaging and Operation describes the auditing strategy for the creation of chained records that are resistant to tampering. The *chain-of-custody* protection of the audit records provides integrity for the system as a whole. If all SAMSON transactions are audited and audit records cannot be illicitly altered, then the audit trail becomes the authoritative source of SAMSON activities and can be used in incident management and forensic analysis.

SAMSON protection of audit records is not reliant on the access control over the audit store (although hardening and access management are part of the deployment strategy for all SAMSON components including audit), rather it is the generation of record and block level digests that ensure that any tampering of audit records can be detected. When audit records are accepted at the TAS, they are amended to include calculated digest values that become part of the transactional audit record, as shown in Figure 18: Audit Record Digests.

Figure 16: TAS Deployed Architecture demonstrates the role of audit integrity tools, tools that can re-evaluate audit records to ensure that the audit chain has not been modified after being posted to the audit store. The SAMSON TD architectural deployment includes one such tool: the Audit Integrity Checker (AIC). The AIC, co-located with the TAS, is a utility that performs a complete review of the entire audit chain from the first record of the first audit block to the most recently added audit record. The AIC is executed from the TAS system command line when the Security Officer requires a verification of the integrity of the audit records and, therefore, the system as a whole. In an operational setting, the AIC can be scheduled to run autonomously through system maintenance facility such as *cron* where the Security Officer can set the frequency of the integrity check.

The AIC repeats the digest calculation that was used to generate the original digests on the audit records. As previously described, these digests are calculated as follows.

The *Audit Record Digest* is a SHA1 hash digest of the concatenation of the following data elements that were taken from the original audit record and are now expressed in the audit store database:

- The unique identifier for the TAS (the tasID);
- The block number and block sequence number;
- The quality of service identifier (the qosID is not used in the architectural deployment);
- The TAS timestamp;
- The original auditXML audit record;
- The user's identity
- The ipAddress of the system that generated the record;
- The name of the program that generated the record;
- The operation on the data;
- The resource requested in the transaction (target);
- The policy decision;
- The identity of the client that generated the audit record;
- The sequence number of the audit record generated at the client; and
- The timestamp of the audit record generated at the client.

The SHA1 function calculates an SHA1 160-bit checksum for the string, as described in RFC 3174 (Secure Hash Algorithm). The value is returned as a string of 40 hex digits. This calculated value is compared to the originally calculated digest that was stored with the record in the trusted audit store.

The *Audit Chain Digest* is a SHA1 hash digest of the concatenation of the current records' record digest and the previous record's chain digest. This calculated value is compared to the originally calculated chain digest that was stored with the record in the trusted audit store.

The *Block Digest* is a SHA1 hash digest of all the record digests in the block and is compared with the block digest value stored with the audit block.

When verifying the integrity of a series of audit records, the AIC performed the following operations:

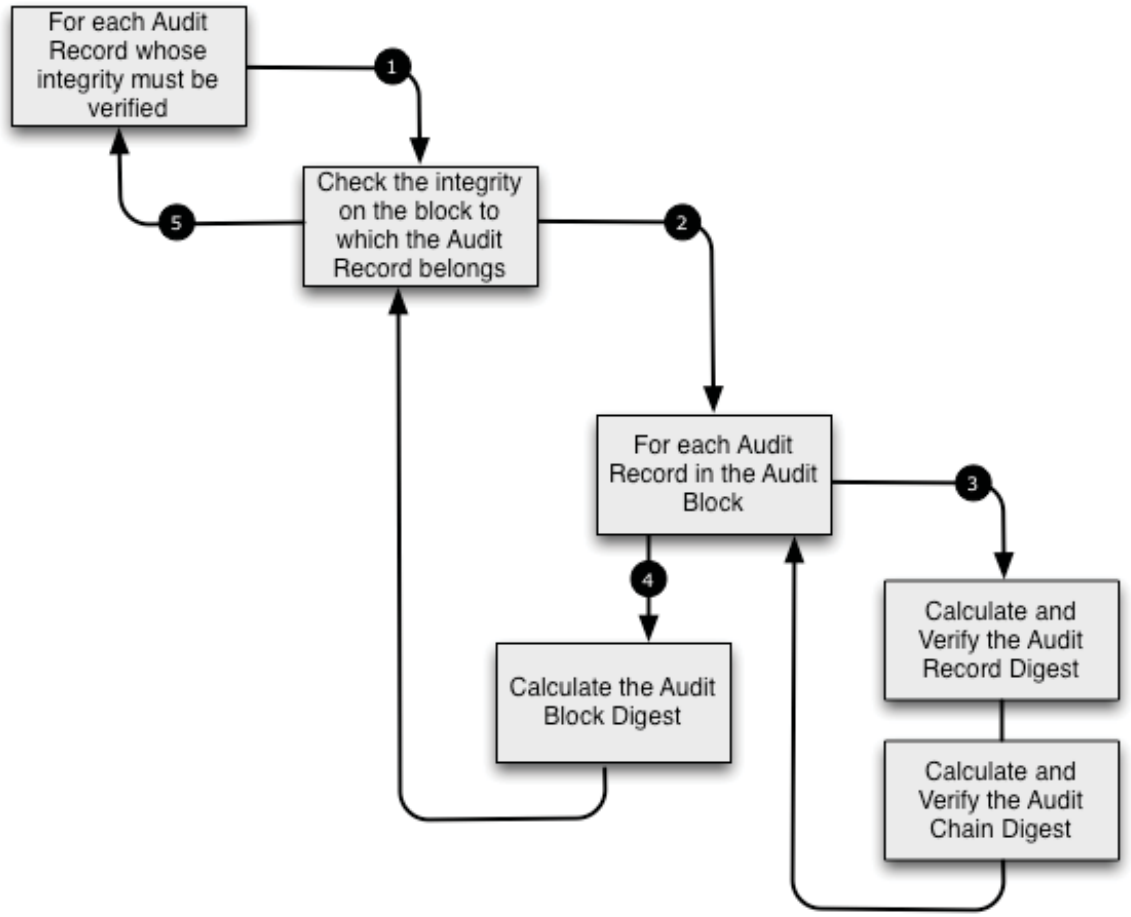


Figure 45: AIC Verification Process

1. Each record is examined individually, although the verification of a record requires the verification of the audit block to which the record belongs.

2. To verify a block of audit records, each record in that block must be individually checked and the block chain verified. As a result, each record in the block is verified to ensure that the block itself maintains its integrity.

3. For each record in the block, the audit record digest and the audit chain digest is calculated. If the integrity of any records in the block cannot be asserted (i.e. the record has been modified) then the block level verification process is deemed to have failed and no records in that block can be trusted. If the digests for all record in the block are correct, the block is deemed to have maintained its integrity and all records in that block have not been altered.

4. The Audit Block Digest is then calculated and compared against the digest of the block itself. The calculated record, chain and block level digests are subsequently used for all other audit records in the current block.

Once the record, chain and block level digests have been calculated for one record, these digests can be used to validate any record in that block. If the span of records goes beyond the current block and into the next block, there digests for the next block will have to be calculated.

Once a block is deemed to have been tampered with:

- All processing on that block stops;
- All audit records in that block are deemed to be untrusted; and
- A security event is raised through the SAMSON Security Event Management process and a notification alert is sent to the Security Officer 6.5 *SAMSON Security Event Management.*

6.5 SAMSON Security Event Management

As described in section 4.6.2: TAS Messaging and Operation, the TAS can bridge to a SIEM solution so that security violations and SAMSON error conditions can be collected, managed, and sent to the designated security and/or administrative person to respond, as a notification message. The SAMSON TD architectural deployment includes two components that enable this capability:

1. The TAS creates and forwards syslog messages to a *syslogd* server based on specific conditions and;

2. A COTS SIEM solution has been deployed in the target environment to read TAS generated syslog messages, process them to generate security alerts and forward notification messages via an email server.

The COTS SIEM solution for the SAMSON TD architectural deployment is AlienVault version 4.1. The SIEM deployment is configured to read in syslog messages stored by the co-located *syslogd* server. (CentOS 6.3 *ksyslogd* daemon). The AlienVault SIEM solution processes each syslog event to determine if it should be raised as a security event. Security events are sent to the Security Officer as email messages via the Microsoft Exchange server.

The processing sequence for raising security events and notifications is shown in the following diagram.

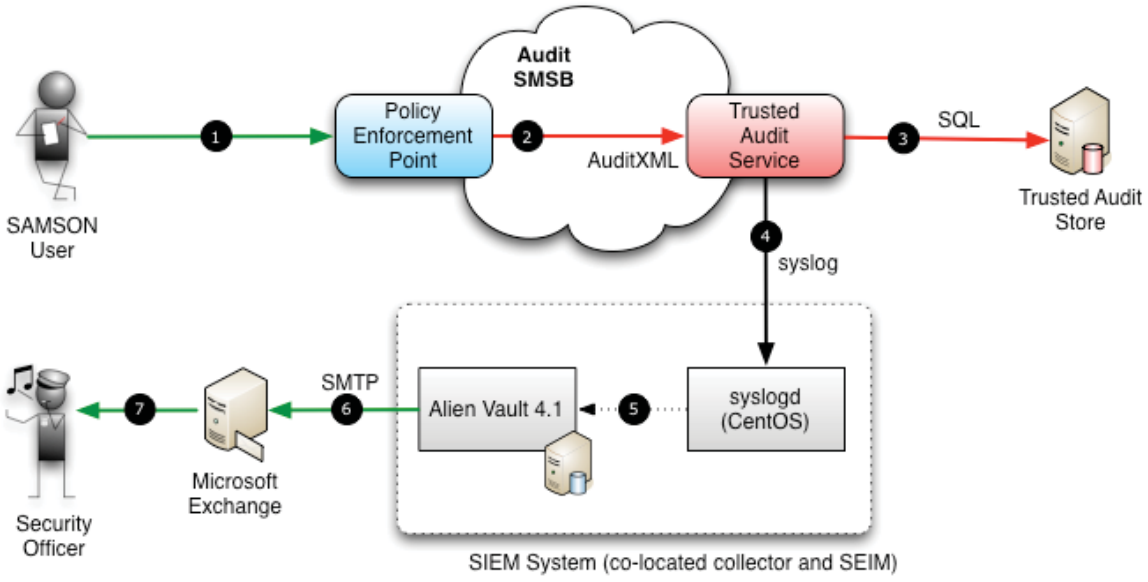


Figure 46: SAMSON Security Event Handling and Notifications

1. SAMSON User performs a SAMSON transaction that is processed by a PEP.

2. During the course of processing the transaction, the PEP generates an audit records that is sent to the TAS 4.6.2: *TAS Messaging and Operation.*

3. The TAS will extend the audit record to include integrity digests and store the record at the audit store.

4. If the audit record is for an auditable event (i.e. represents a security incident or includes a processing error code), the TAS generates a syslog record. The syslog record is described using the following format (10 elements, comma separated):

    a. The time that the event record was generated (the TAC timestamp);
    b. The IP address of the TAS that processed the audit record;
    c. The identity of the user that created the audit record;

d. The resource that was requested as part of the transaction;
e. The requested policy action on the resource (e.g. READ/WRITE);
f. The IP address of the PEP that created the audit record;
g. The program name (PEP) that generated the audit record;
h. The requested SAMSON operation on the resource (e.g. file policy check);
i. The error code associated with the transaction; and
j. The text for the error condition associated with the transaction.

A sample syslog message, generated for a request for a file that does not exist, would be as follows:

```
Mar 14 22:40:52 10.10.10.87
catester1,/usr/local/apache2/htdocs/data/CEO
document.docx,WRITE,10.10.10.95,filesystem,POLICY_FILE,40111,N
o such file or directory : /usr/local/apache2/htdocs/data/CEO
document.docx
```

In this case, user catest1 has requested a file “CEO document.docx” and the SAMSON PEP was not able to process the request, as the file does not exist.

The configuration of the SAMSON TAS includes the specification of the target syslogd server and the logging facility level to use for SAMSON log events. The TAS uses this information to submit the syslog message to the target syslogd server.

5. For the SAMSON TD deployment architecture, the syslogd server is configured to write SAMSON syslog messages to a separate file. AlienVault uses this syslog file as a event source, reads records from this file as they are appended and processes the events reflected in each record.

6. AlienVault allows specific actions to take place depending on the kind of event messages that are read from the event sources. All SMASON events are handled the same way:

    a. Raise an alert (a local ticket managed through the interface); and

    b. Generate an email message and send it to a target user, typically the Security Officer, via SMTP though the domain exchange server.

7. The Security Officer is then able to retrieve the Security Event notifications and respond to the event appropriately.

When viewed though the AlienVault SIEM web interface, security alerts are presented as follows. Figure 47: AlienVault Security Incidents shows the general list of recent security events and Figure 48: AlienVault Security Event Details shows specific information for an individual security event.

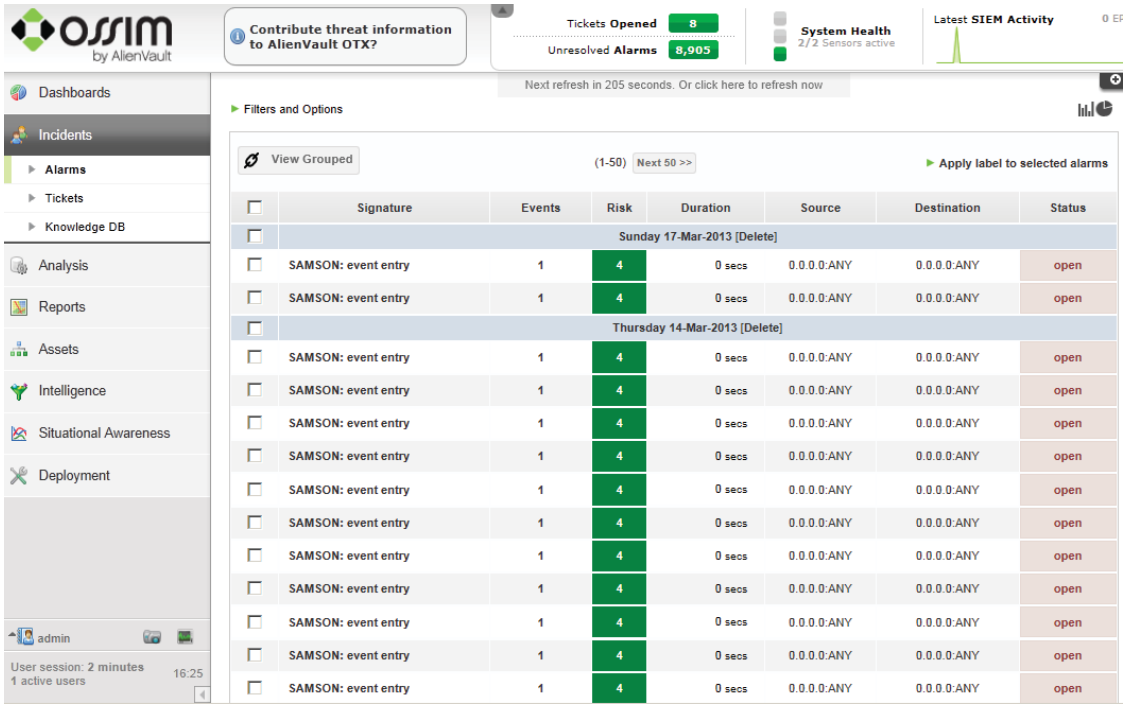


Figure 47: AlienVault Security Incidents

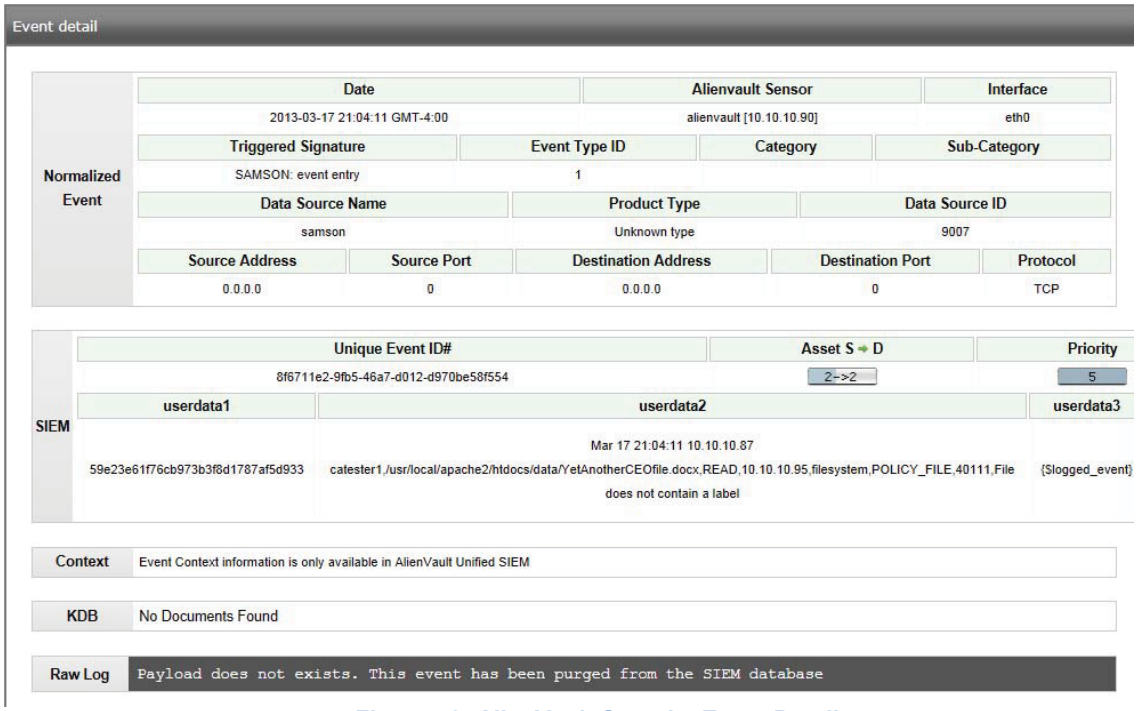


Figure 48: AlienVault Security Event Details

When AlienVault raises a security event, a notification email using a specific message structure is sent to the Security Officer.

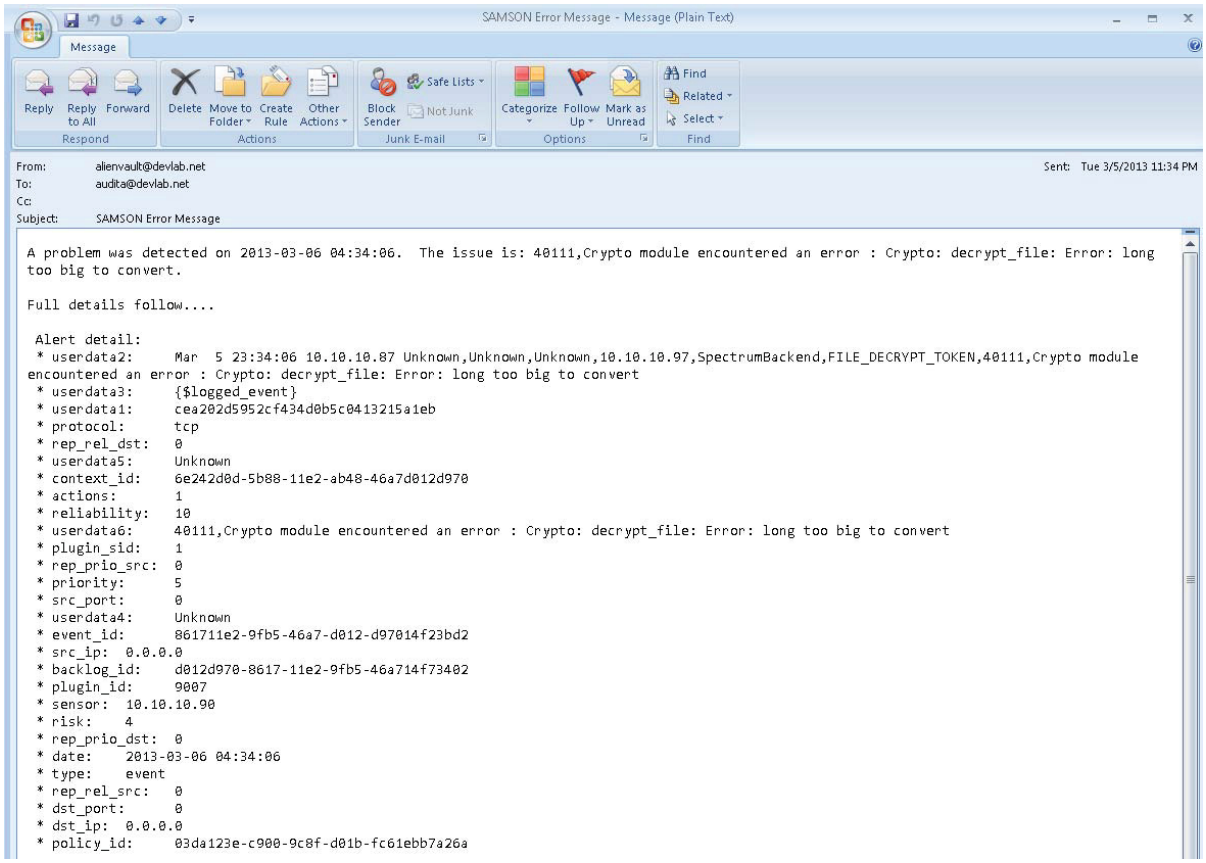


Figure 49: A Security Event Notification Email

Annex A: Message Formats

This annex provides details for the messages that are received from and responded to by the six core SAMSON services. For each message type, the message format is presented and the role of individual attributes within the message is explained.

Annex A.1 Identity Attribute Service Messages

The formats described in the following message request/response cycle represent the SSG interface API presented by the IAS.

IAS Request Message

The message request format is as follows:

```
<spml:searchRequest
 xmlns:spml='urn:oasis:names:tc:SPML:1:0'
 xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core'
 requestID='REQUEST_IDENTIFIER'>

       <dsml:filter>
              <dsml:equalityMatch name='LOOKUP_VALUE'>
                     <dsml:value>ACCOUNT</dsml:value>
              </dsml:equalityMatch>
       </dsml:filter>
       <spml:attributes>
              <dsml:attribute name='ATTRIBUTE'/>
       </spml:attributes>
</spml:searchRequest>
```

The individual message attributes that can be set by the calling process are presented below.

Table 18: IAS Request Message Content

Information Element	Description	Value
Request Identifier	A Unique identifier for this request which allows multiple requests to be processed simultaneously.	Alphanumeric value
Lookup Value	This is the name that uniquely identities the user for which security attributes are requested.	The value provided for this field is the user's unique identity as determined through their domain credentials.
Account	This is the name that uniquely identities the user for which security attributes are requested.	The value provided for this field is the user's unique identity as determined through their domain credentials.

Information Element	Description	Value
Attribute	This is the list of attributes that are requested from the security attribute repository for the specified user	Multiple attributes can be specified from the following list: *nationality*, *clearance*, *caveats.* Caveats will return a comma separated list of the communities of which the user is a member.

IAS Response Message

The message response format is as follows:

```
<spml:searchResponse
 xmlns:spml='urn:oasis:names:tc:SPML:1:0'
 xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core'
 result='urn:oasis:names:tc:SPML:1:0#success'>
       <spml:searchResultEntry>
              <spml:identifier
               type='urn:oasis:names:tc:SPML:1:0#GenericString'>
                     <spml:id>USERNAME</spml:id>
              </spml:identifier>
              <spml:attributes>
                     <dsml:attr name='ATTRIBUTE'>
                           <dsml:value>VALUE</dsml:value>
                     </dsml:attr>
              </spml:attributes>
       </spml:searchResultEntry>
</spml:searchResponse>
```

The IAS will send the above response message after populating the requested data into the following attributes.

Table 19: IAS Response Message Content

Information Element	Description	Value
UserName	This is the name that uniquely identities the user for which security attributes are requested.	The value provided for this field is the user's unique identity as determined through their domain credentials.
Attribute	This is the list of attributes that are requested from the security attribute repository for the specified user	Multiple attributes can be specified from the following list: *nationality*, *clearance*, *caveats.* Caveats will return a comma separated list of the communities of which the user is a member.
Value	See table below	See table below

The SAMSON TD architectural deployment supports the request for the following user attributes. These attributes are present in the user attribute repository and represent the data elements (per user) that are requested through the IAS.

Table 20: Security Attribute Value

Requested Attribute	Response Element	Value	Example
nationality	nationality	Text representing the user's nationality	CANADA
clearance	clearance	Text representing the user's clearance	SECRET
caveats	caveats	A comma separated list containing all the communities to which the user belongs	CEO,CANUS

Annex A.2 Authorization Service Messages

The formats described in the following message request/response cycle represent the SSG interface API presented by the AS.

AS Request Message

```
<xacml-context:Request
 xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:cd:04"
 xmlns:xacml-context="urn:oasis:names:tc:xacml:2.0:context:schema:os"
 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
<xacml-context:Subject>
      <xacml-context:Attribute
       AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
       DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
         <xacml-context:AttributeValue>ACCOUNT</xacml-context:AttributeValue>
      </xacml-context:Attribute>
   </xacml-context:Subject>
      <xacml-context:Resource>
         <xacml-context:Attribute
          AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
          DataType="http://www.w3.org/2001/XMLSchema#anyURI">
            <xacml-context:AttributeValue>RESOURCE</xacml-context:AttributeValue>
         </xacml-context:Attribute>
      </xacml-context:Resource>
      <xacml-context:Action>
         <xacml-context:Attribute
          AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
          DataType="http://www.w3.org/2001/XMLSchema#string">
            <xacml-context:AttributeValue>ACTION</xacml-context:AttributeValue>
         </xacml-context:Attribute>
      </xacml-context:Action>
   <xacml-context:Environment/>
</xacml-context:Request>
```

The individual message attributes that can be set by the calling process are presented below.

Information Element	Description	Value
ACCOUNT	The user account for which the policy check is being made; for application PEPs this is usually the user account under which the security operation is taking place.	The value provided for this field is the user's unique identity as determined through their domain credentials.
RESOURCE	The data artifact that is being evaluated for an access control check; this may be a high-level construct such as a community of interest or a low-level artifact such as a filename.	The value provided for this field is the security attribute on the information asset being requested
ACTION	The operation that is being requested against the RESOURCE.	The value provided by the PEP that corresponds to the action against the data. For example, the File PEP will equate a directory listing as a policy READ action.

AS Response Message

The AS response to an XACML formatted policy request is an XACML formatted response message.

```
<xacml-context:Response
 xmlns:xacml-context="urn:oasis:names:tc:xacml:2.0:context:schema:os">
   <xacml-context:Result>
      <xacml-context:Decision>DECISION</xacml-context:Decision>
   </xacml-context:Result>
</xacml-context:Response>
```

The AS will send the above response message after populating the requested data into the following attribute.

Information Element	Description	Value
DECISION	This is the policy decision returned by the PDP accessed through the AS	One of Permit, Deny or Error

Annex A.3 Key Management Service Messages

The formats described in the following message request/response cycle represent the SSG interface API presented by the KMS.

KMS Request Message

```
<ssr:Request
 xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:cd:04"
 xmlns:ssr="urn:oasis:names:tc:xacml:2.0:context:schema:os"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:Samson.sqml.0.1 https://samsontd.ca/schema/sqml.01.xsd">
   <ssr:Subject>
      <ssr:Attribute
       AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
       DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
         <ssr:AttributeValue>SUBJECT FIELD</ssr:AttributeValue>
      </ssr:Attribute>
   </ssr:Subject>
   <ssr:Resource>
      <ssr:Attribute
       AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
       DataType="http://www.w3.org/2001/XMLSchema#anyURI">
         <ssr:AttributeValue>RESOURCE FIELD</ssr:AttributeValue>
      </ssr:Attribute>
   </ssr:Resource>
   <ssr:Action>
      <ssr:Attribute
       AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
       DataType="http://www.w3.org/2001/XMLSchema#string">
         <ssr:AttributeValue>ACTION FIELD</ssr:AttributeValue>
      </ssr:Attribute>
   </ssr:Action>
   <ssr:Environment>
      <ssr:Attribute
       AttributeId="urn:oasis:names:tc:xacml:1.0:environment:environment-id"
       DataType="http://www.w3.org/2001/XMLSchema#string">
         <ssr:AttributeValue>ENVIRONMENT FIELD</ssr:AttributeValue>
      </ssr:Attribute>
   </ssr:Environment>
</ssr:Request>
```

The individual message attributes that can be set by the calling process are presented below.

Table 21: KMS Request Message Content by Message Type

Message Type	Subject	Resource	Action	Environment
Generate a key	N/A	N/A	GENERATE_KEY	N/A
Generate and Store a key	N/A	N/A	GENERATE_STORE	N/A
Store a supplied key	N/A	The key to be stored	STORE_KEY	N/A
Retrieve an existing key	N/A	The key's token	RETRIEVE_KEY	N/A

KMS Response Message

The message response format is as follows:

```
<ssr:SAMSONSvcResponse xmlns:ssr="SAMSONSvcResponse">
   <ssr:List name="kesOP">
      <ssr:Value key="RESPONSE_TYPE">RESPONSE</ssr:Value>
   </ssr:List>
```

```
   <ssr:Status code=ERROR_CODE>ERROR_TEXT</ssr:Status>
</ssr:SAMSONSvcResponse>
```

The KMS will send the above response message after populating the requested data into the following attributes. Note that the KMS uses XACML context messages for key action requests and the SAMSON Service Response (SSR) format for key action responses. The ssr:List name attribute for KMS service responses is “kesOP”.

Table 22: KMS Response Message Content by Message Type

Response Field	Requested Action	Response Type	Response Data
RESPONSE_TYPE	GENERATE_KEY	“key”	The returned key from the escrow system.
	GENERATE_STORE	“keytoken”	The retuned key and token from the escrow.
	STORE_KEY	“token”	The returned token from the escrow.
	RETRIEVE_KEY	“key”	The returned key from the escrow.
ERROR_CODE	ALL	Integer	“0” if success, error code if a processing error has occurred.
ERROR_TEXT	ALL	Text	Error Description

Annex A.4 Cryptographic Transformation Service Messages

The formats described in the following message request/response cycle represent the SSG interface API presented by the CTS.

CTS Request Message

The message request format is as follows:

```
<ssr:Request
 xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:cd:04"
 xmlns:ssr="urn:oasis:names:tc:xacml:2.0:context:schema:os"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:Samson.sqml.0.1 https://samsontd.ca/schema/sqml.01.xsd">
   <ssr:Subject />
   <ssr:Resource>
      <ssr:Attribute
       AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
       DataType="http://www.w3.org/2001/XMLSchema#anyURI">
         <ssr:AttributeValue>RESOURCE FIELD 1</ssr:AttributeValue>
         <ssr:AttributeValue>RESOURCE FIELD 2</ssr:AttributeValue>
      </ssr:Attribute>
   </ssr:Resource>
   <ssr:Action>
      <ssr:Attribute
```

```
        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
        DataType="http://www.w3.org/2001/XMLSchema#string">
          <ssr:AttributeValue>ACTION FIELD</ssr:AttributeValue>
       </ssr:Attribute>
    </ssr:Action>
    <ssr:Environment>
       <ssr:Attribute
        AttributeId="urn:oasis:names:tc:xacml:1.0:environment:environment-id"
        DataType="http://www.w3.org/2001/XMLSchema#string">
          <ssr:AttributeValue>ENVIRONMENT FIELD</ssr:AttributeValue>
       </ssr:Attribute>
    </ssr:Environment>
</ssr:Request>
```

The individual message attributes that can be set by the calling process are presented below.

Table 23: CTS Request Message Content by Message Type

Message Type	Resource Field 1	Resource Field 2	Action Field	Environment Field
Encrypt to get a container	Plaintext file	Container file	COPY_ENCRYPT	The label on the data.
Decrypt a container	Container file	Plaintext file	COPY_DECRYPT	The label on the data.
Encrypt a file	Plaintext file	Encrypted file	FILE_ENCRYPT_TOKEN	The token for the key to use.
Decrypt a file	Encrypted file	Plaintext file	FILE_DECRYPT_TOKEN	The token for the key to use.

CTS Response Message

The message response format is as follows:

```
<ssr:SAMSONSvcResponse xmlns:ssr="SAMSONSvcResponse">
   <ssr:List name="cryptoOP">
      <ssr:Value key="RESPONSE_TYPE">RESPONSE</ssr:Value>
   </ssr:List>
   <ssr:Status code=ERROR_CODE>ERROR_TEXT</ssr:Status>
</ssr:SAMSONSvcResponse>
```

The CTS will send the above response message after populating the requested data into the following attributes. Note that the CTS uses XACML context messages for cryptographic action requests and the SAMSON Service Response (SSR) format for cryptographic action responses. The ssr:List name attribute for KMS service responses is “cryptoOP”.

```
<ssr:SAMSONSvcResponse xmlns:ssr="SAMSONSvcResponse">
   <ssr:List name="cryptoOP">
      <ssr:Value key="RESPONSE_TYPE">RESPONSE</ssr:Value>
   </ssr:List>
```

```
   <ssr:Status code=ERROR_CODE>ERROR_TEXT</ssr:Status>
</ssr:SAMSONSvcResponse>
```

Table 24: CTS Response Message Content by Message Type

Response Field	Requested Action	Response Type	Response Data
List	COPY_ENCRYPT	target	Container File
	COPY_DECRYPT	target	Plaintext File
	FILE_ENCRYPT_TOKEN	target	Ciphertext File
	FILE_DECRYPT_TOKEN	target	Plaintext File
Status	All	ERROR_CODE	"0" is operation is successful and error code if an error is encountered during the processing of the request
	All	ERROR_TEXT	Textual description of the error

Annex A.5 Security Label Service Messages

The formats described in the following message request/response cycle represent the SSG interface API presented by the SLS.

SLS Request Message

The message request format is as follows:

```
<ssr:Request
 xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:cd:04"
 xmlns:ssr="urn:oasis:names:tc:xacml:2.0:context:schema:os"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:Samson.sqml.0.1 https://samsontd.ca/schema/sqml.01.xsd">
   <ssr:Subject />
   <ssr:Resource>
      <ssr:Attribute
       AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
       DataType="http://www.w3.org/2001/XMLSchema#anyURI">
         <ssr:AttributeValue>RESOURCE FIELD</ssr:AttributeValue>
      </ssr:Attribute>
   </ssr:Resource>
   <ssr:Action>
      <ssr:Attribute
```

```
        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
        DataType="http://www.w3.org/2001/XMLSchema#string">
          <ssr:AttributeValue>ACTION FIELD</ssr:AttributeValue>
       </ssr:Attribute>
    </ssr:Action>
    <ssr:Environment />
</ssr:Request>
```

The individual message attributes that can be set by the calling process are presented below.

Table 25: SLS Request Message Content by Message Type

Message Type	Subject Field	Resource Field	Action Field	Environment Field
Extract the security label from the file	N/A	The target file being queried	FILE_GET_LABEL	N/A

SLS Response Message

The message response format is as follows:

```
<ssr:SAMSONSvcResponse xmlns:ssr="SAMSONSvcResponse">
   <ssr:List name="assignedlabel">
      <ssr:Value key="caveat">SECURITY_LABEL</ssr:Value>
   </ssr:List>
   <ssr:Status code=ERROR_CODE>ERROR_TEXT</ssr:Status>
</ssr:SAMSONSvcResponse>
```

The SLS will send the above response message after populating the requested data into the following attributes. Note that the SLS uses XACML context messages for labelling action requests and the SAMSON Service Response (SSR) format for labelling action responses. The ssr:List name attribute for KMS service responses is “assignedlabel”.

Table 26: SLS Response Message Content by Message Type

Response Field	Requested Action	Response Type	Response Data
Ssr:List	GET_FILE_LABEL	caveat	Comma separated list of caveats on a file.
Ssr:Status	All	ERROR_CODE	“0” is operation is successful and error code if an error is encountered during the processing of the request
	All	ERROR_TEXT	Textual description of the error

Annex A.6 AuditXML Schema

Audit messages are represented using the following schema.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:ta="jabber:iq:samson:trustedaudit"
 targetNamespace="jabber:iq:samson:trustedaudit"
 version="0.1"
 elementFormDefault="qualified"
 attributeFormDefault="unqualified">
   <xs:element name="auditRecord" type="ta:auditRecordType" />
      <xs:complexType name="auditRecordType">
         <xs:sequence>
            <xs:element name="principal" type="ta:principalType" />
            <xs:element name="action" type="ta:actionType" />
            <xs:element name="tacOrigin" >
               <xs:complexType>
                  <xs:attribute name="tacId"
                   type="xs:NMTOKEN" use="required" />
                  <xs:attribute name="tacSeqNum"
                   type="xs:integer" use="required" />
               </xs:complexType>
            </xs:element>
            <xs:element name="notes"
             type="xs:string" minOccurs="0" maxOccurs="1"/>
            <xs:element name="tacTimestamp" type="ta:timestampType" />
         </xs:sequence>
         <xs:attribute name="xauditVersion"
          type="xs:string" use="required" fixed="0.1" />
      </xs:complexType>
      <xs:complexType name="principalType">
         <xs:choice>
            <xs:sequence>
               <xs:element name="userId" type="xs:string" />
               <xs:element name="ipAddress" type="ta:ipAddressType"
                minOccurs="0" maxOccurs="1" />
               <xs:element name="programName" type="xs:string"
                minOccurs="0" maxOccurs="1" />
            </xs:sequence>
            <xs:sequence>
               <xs:element name="ipAddress" type="ta:ipAddressType" />
               <xs:element name="programName" type="xs:string"
                minOccurs="0" maxOccurs="1" />
            </xs:sequence>
            <xs:sequence>
               <xs:element name="programName" type="xs:string" />
            </xs:sequence>
         </xs:choice>
      </xs:complexType>
      <xs:complexType name="actionType">
         <xs:sequence>
            <xs:element name="operation" type="xs:string"/>
            <xs:element name="target" type="xs:string"/>
         </xs:sequence>
      <xs:attribute name="state" type="ta:stateType" use="required" />
   </xs:complexType>
   <xs:simpleType name="ipAddressType">
      <xs:restriction base="xs:string">
         <xs:pattern value="(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-
4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-
4][0-9]|[01]?[0-9][0-9]?)" />
      </xs:restriction>
```

```
   </xs:simpleType>
   <xs:simpleType name="stateType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="success" />
         <xs:enumeration value="failure" />
         <xs:enumeration value="denied" />
         <xs:enumeration value="unknown" />
      </xs:restriction>
   </xs:simpleType>
   <xs:complexType name="timestampType">
      <xs:attribute name="timestamp" type="ta:epochType" use="required" />
   </xs:complexType>
   <xs:simpleType name="epochType">
      <xs:restriction base="xs:string">
         <xs:pattern value="(0|[1-9][0-9]*)" />
      </xs:restriction>
   </xs:simpleType>
</xs:schema>
```

Annex B: Configuration Options

Annex B.1 Identity Attribute Service Configuration

The SAMSON deployment includes a configuration file that specifies the connection information that is used to query the identity attribute repository. Those configuration details are described in the following table.

Table 27: IAS Configuration Elements

Configuration Attribute	Description	Format
LDAP Address	This is the network address where the LDAP-based security attribute repository can be queried.	An IP address
LDAP port	This is the port on which the LDAP server listens for connections	An Integer (0 to 65535)
SSL	States whether TLS/SSL sessions should be used	Yes/No
binddn	An account that can bind to the LDAP server to perform queries against the directory	FQDN
bindpw	The password for the *binddn* account	Text
users_ou	The organizational unit where SAMSON user's security attributes are stored	Distinguished Name
caveats	The name of the attribute in the OU scope that stores COI memberships for the user	Text
clearance	The name of the attribute in the OU scope that stores the user's clearance.	Text
nationality	The name of the attribute in the OU scope that stores the user's nationality.	Text
userObjectClass	The class of the objects stored at the OU scope. This is necessary to state for querying the LDAP server.	Text

Annex B.2 Authorization Service Configuration

The SAMSON deployment includes a configuration file that specifies the connection information that is used to query the security policy repository. Those configuration details are described in the following table.

Table 28: AS Configuration Elements

Configuration Attribute	Description	Format
PAP Address	The network location of the backend policy storage facility	An IP address
PAP Port	The port on which the storage facility is listening	An Integer (0 to 65535)
PAP DB Name	The name of the database where the policies are stored	Text
PAP Table	The name of the table where the policies are stored	Text
PAP Account Name	The database account that can retrieve security policies	Text
PAP Account Password	The password for the PAP Account	Text

When the AS starts, this information is retrieved from the configuration and used to communicate with the security policy storage facility.

Annex B.3 Key Management Service Configuration

The SAMSON deployment includes a configuration file that specifies the connection information that is used to query the StrongAuth key escrow system. Those configuration details are described in the following table.

Table 29: KMS Configuration Elements

Configuration Attribute	Description	Format
Handler	The type of KMS used in this deployment	"SA" for StrongAuth "SAMSON" for a database key escrow system
SA_address	The full address of the StrongAuth Key Escrow's SOAP based interface	URL
SA_decrypt_user	The name of the SA user with decrypt privileges	Text
SA_encrypt_user	The name of the SA user with encrypt privileges	Text
SA_password	The password for both users	Text
Kgs_source	The crypto library to use for key generation	"RSA" for RSA crypto module (FIPS compliant) "SAMSON" for local crypto libraries.

When the KMS starts, this information is retrieved from the configuration and used to communicate with the key escrow system.

Annex B.4 Cryptographic Transformation Service Configuration

The SAMSON deployment includes a configuration file that specifies the libraries to use for cryptographic operations. Those configuration details are described in the following table.

Table 30: CTS Configuration Elements

Configuration Attribute	Description	Format
Crypto_module	The crypto library to use.	"RSA" for RSA crypto module (FIPS compliant) "SAMSON" for local crypto libraries.
Rsa_library_path	The location of the crypto libraries on the CTS system	Absolute path to the RSA runtime crypto libraries. (RSA only)

When the CTS starts, this information is retrieved from the configuration and used to leverage the specified crypto library.

Annex B.5 Trusted Audit Service Configuration

The SAMSON deployment includes a configuration file that specifies the connection information that is used to access the back end audit store. Those configuration details are described in the following table.

Table 31: KMS Configuration Elements

Attribute Section	Sub Attribute	Description	Format
TAS Database	**type**	The type of database used as the Audit Store	"MySQL"
	host	The location of the Audit Store	IP address
	port	The port on which the Audit Store is listening	Port number
	schema	The specification of the auditXML schema	Absolute path to the schema definition on the TAS.
TAS Actuator	**tasId**	The identifier for this instance of the TAS	Alphanumeric
	dbUser	The account used to access the audit store	Alphanumeric
	dbPassword	The password for the dbUser account	Alphanumeric

When the TAS starts, this information is retrieved from the configuration and used to communicate with the Audit Store.

Annex C: Acronyms and Abbreviations

ABAC	Attribute-based Access Control
ADSI	Active Directory Scripting Interface
API	Application Programming Interface
AS	Authorization Service
C&C	Command and Control
CAGE	Coalition Attack Guidance Experiment
CANUS	(rel to) Canada-United States
CEO	Canadian Eyes Only
CF	Canadian Forces
CFD	Chief of Force Development
COI	Community of Interest
COTS	Commercial Off The Shelf
CTS	Cryptographic Transformation Service
CWID	Coalition Warrior Interoperability Demonstration
DB	Database
DISCO	Discovery
DND	Department of National Defence
DRDC	Defence Research & Development Canada
DSML	Directory Services Markup Language
FIPS	Federal Information Processing Standard
FQDN	Fully Qualified Domain Name
HA	High Availability
HTTP	Hypertext Transfer Protocol
IAAI	Identity Attribute Administration Interface
IAS	Identity Attribute Service
IdM	Identity Management
IM	Instant Messaging
IPL	Information Protection Logic
IQ	Information Query

JID	Jabber Identity
KMS	Key Management Service
LDAP	Lightweight Directory Access Protocol
MAPI	Messaging Application Programming Interface
MIME	Multipurpose Internet Mail Extensions
OASIS	Organization for the Advancement of Structured Information Standards
OOB	Out of Band
PAI	Policy Administration Interface
PDP	Policy Decision Point
PEDI	Policy Enforcement Data Intercept
PEMC	Policy Enforcement Message Client
PEP	Policy Enforcement Point
PKI	Public Key Infrastructure
POP3	Post Office Protocol (3)
R&D	Research & Development
RFC	Request for Comment
SAMSON	Secure Access Management for Secure Operational Networks
SAMPOC	Secure Access Management Proof Of Concept
SD	Supporting Deliverable
SIEM	Security Information and Event Management
SLS	Secure Labeling Service
SMB	Server Message Block
SMSB	Secure Messaging Service Bus
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SMTP	Simple Mail Transfer Protocol
SPML	Service Provisioning Markup Language
SQL	Structured Query Language
SSG	Samson Service Gateway
SSL/TLS	Secure Sockets Layer / Transport Layer Security
TAC	Trusted Audit Client
TAS	Trusted Audit Service

TD	Technology Demonstrator
TDP	Technology Demonstrator Program
TLS	Transport Layer Security
UDDI	Universal Description, Discovery and Integration
W3C	World Wide Web Consortium
WebDAV	Web Distributed Authoring and Versioning
WSDL	Web Services Description Language
WS-Encryption	Web Services Encryption
WS-Security	Web Services Security
XACML	eXtensible Access Control Markup Language
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol