Enabling Trust Through Continuous Compliance Assurance

Bonnie Morris*, Cynthia Tanner, Joseph D'Alessandro
Lane Department of Computer Science and Electrical Engineering, West Virginia University
* College of Business and Economics, West Virginia University

Abstract

As the digital world expands the building of trust and the retention of privacy in information sharing becomes paramount. The impediment to information sharing is a lack of trust between the parties, based on security and privacy concerns, as well as information asymmetry. In an effort to foster trusted information sharing we propose a trusted enclave with an embedded Continuous Compliance Assurance (CCA) mechanism as a technology solution. The CCA mechanism if not too costly in processing, would assure compliance to all regulatory policies regarding the data to be shared. A test bed which mimics the trusted enclave with the embedded CCA module was built to capture realistic performance statistics. The performance statistics gathered through the test bed indicate that real time compliance assurance is feasible, thereby enabling trusted information sharing.

Index Terms

Embedded Audit Process, Monitored Information Exchange, Regulatory Compliance, XML Based Processing, Trusted Information Sharing

1. Introduction

Continuous compliance assurance (CCA) refers to the active process of continuously testing and verifying compliance with established policies by an independent or objective third party. CCA is useful in any situation involving two or more parties who wish to engage in transactions under conditions of information asymmetry. Information asymmetry occurs when one or more parties to a contract cannot perfectly observe the actions of another party. In the context of interorganizational information sharing, CCA can be a mechanism for creating trust by reducing information asymmetry.

There are many situations in homeland security, law enforcement, and commercial supply chain operations where the sharing of private information among several organizations could be mutually beneficial. Nevertheless, organizations are often reluctant to share even when all parties agree on how the shared data are to be used. Their reluctance to share stems from concerns that the data recipients might not provide adequate security or enforce the agreed upon sharing policies. For homeland security or law enforcement applications, inappropriate use of shared information could result in subjects of investigations being tipped off or intelligence methods and sources being compromised [1] [2]. In a supply chain context, organizations are concerned about the disclosure of proprietary information about customers, vendors, or operations that could be used against the organization by competitors. Additionally, inappropriate disclosure of nonpublic personal information (NPPI) by a data recipient may result in violation of privacy laws leading to losses by the data provider from civil or criminal judgments in court.

The problem arises from information asymmetry and from the nature of information goods– information can be costlessly copied and instantaneously distributed, often leaving behind little or no evidence of the misappropriation. Once private data is disclosed, it usually cannot be made private again. Access controls are necessary but not sufficient to create trust in the data provider. Access controls limit what data may be requested and by whom, but they do not control how the data is subsequently propagated. An authorized user may copy a file to a laptop hard drive or USB drive, take it off premises and lose it or intentionally misuse the data.

A trusted enclave is necessary to protect highly sensitive or confidential information. The enclave would provide the physical and logical access controls to prevent unauthorized access or use. Additionally, all data fusion applications (e.g., applications that identify data that matches across multiple sources that should not, or data that should match that does not, or compiling data from multiple sources about an entity of interest by one or more organizations) are executed within the enclave according to user specified policies,

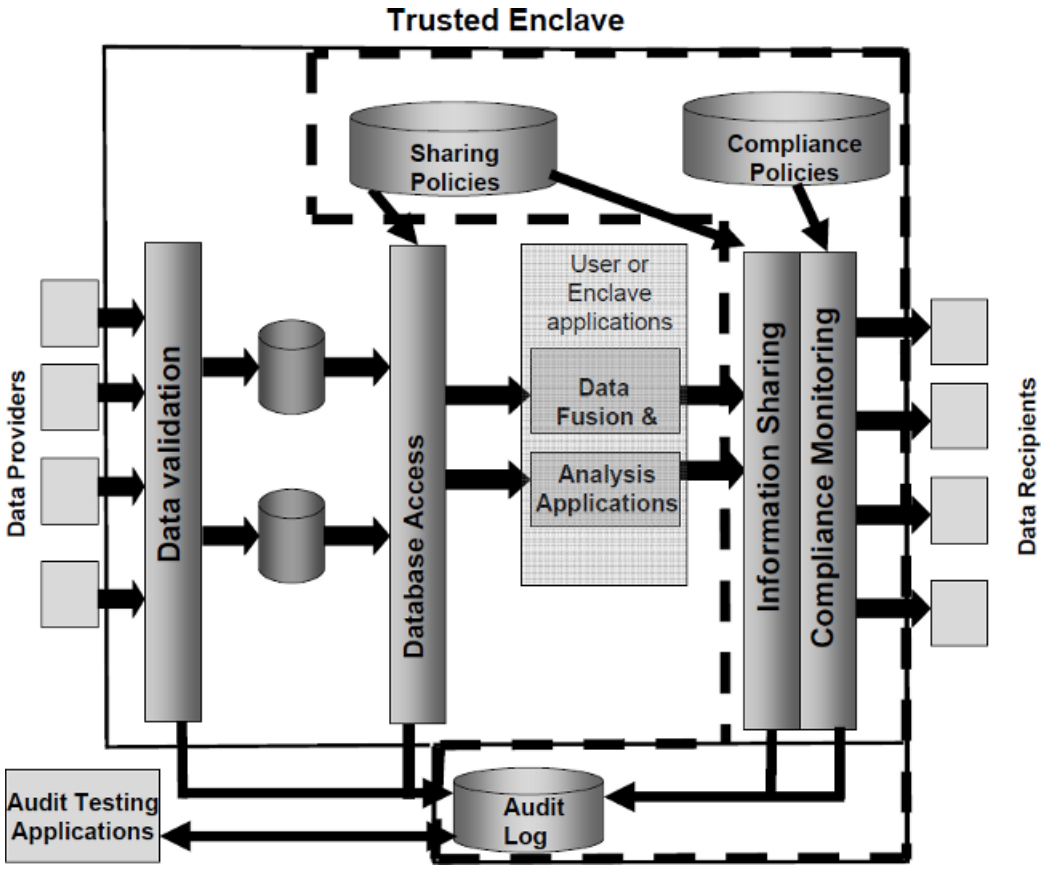

Figure 1. View of the Design of the Trusted Enclave

with the output subjected to monitoring for compliance with sharing rules. Testing for compliance assurance is based on what leaves the enclave, not on data requests by individuals or applications. Information sharing policy enforcement, data fusion and analysis, and assurance testing are done electronically according to pre-specified rules. An accessible audit log of transactions is created to facilitate on-going ex post analysis of compliance. Figure 1 depicts the design of the trusted enclave.

An example of a situation in which the trusted enclave would be useful is as follows. Only a small percentage of cargo containers are physically inspected. Information that helps authorities identify suspicious containers would benefit all shipping companies. If shipping companies share information about cargo container location, they could identify a container on the dock waiting to be loaded that is listed as "out of service" according to the container owner's records. Logistically, it is necessary to submit the data to a central location (trusted enclave) where the file matching application resides. The shipping companies' information sharing policies would identify what fusion and analysis applications are allowed to access the shared data within the trusted enclave and specify what data about containers can be shared, with whom and under what circumstances. Specifically, they could prohibit the distribution of any of the shared container data except when a suspicious container is identified by the application. Shipping companies would be willing to share such data only if they had assurance from an independent third party that the agreed upon policies or rules for using and sharing that information were being enforced because the container location data could provide useful business intelligence to their competitors and their customers competitors.

We begin our discussion by covering the topic of Continuous Compliance Assurance and the protoype we have implemented. We then transition to a discussion of a test bed we have built to assess the performance of our prototype, followed by a presentation of results collected from our test bed. We then conclude the paper.

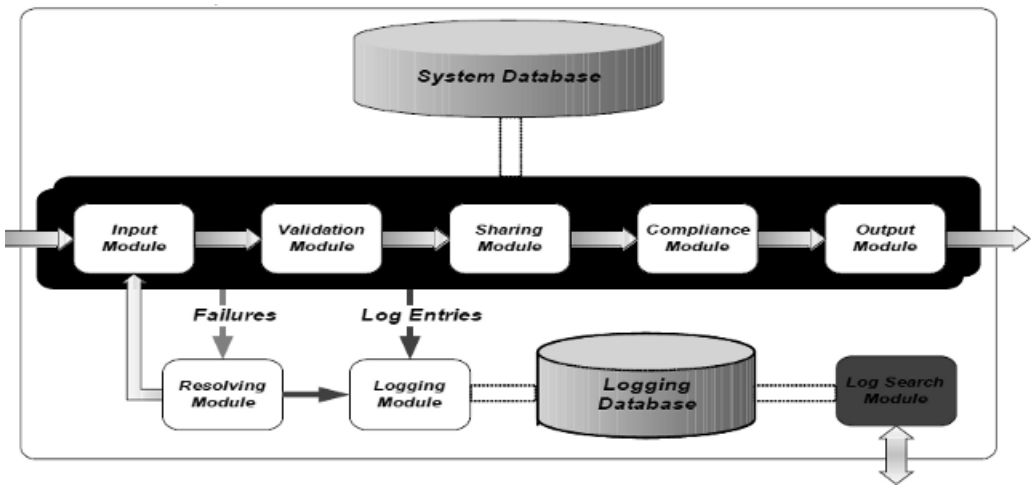

Figure 2. View of the Design of our Continuous Compliance Mechanism

2. Continuous Compliance Assurance

Extensible Markup Language (XML) is becoming the industry data exchange standard. It allows interoperability across applications and platforms [3]. Many domains have created XML data standards to enable different enterprises to exchange data easily and seemlessly. A few of these standards are: Global justice XML standard [4], XBRL for financhial applications [5], and ACORD XML [6] for life, annuity and health insurance. To enable our CCA mechanism to be incorporated in many domains we have accepted XML as the data transfer standard and have implemented the system based on XML formatted messages.

The Continuous Compliance Assurance Group at West Virginia University has built a prototype CCA system. The system accepts pre-defined XML messages and verifies that the message adheres to all applicable corporate and regulatory policies. During processing the system creates an immutable audit log of all messages, the policies applied and any resolutions applied to non-compliant messages. This embedded in-stream process can stop the delivery of a non-compliant message or simply strip the offending data from the message prior to delivery. Thus this mechanism is capable of terminating and reporting inappropriate data requests prior to delivery as opposed to audit mechanisms which are used in an ex post fashion. The current mechanisms to enforce privacy and security regulations support enforcement through access logs and supply a capability to identify inappropriate data exchanges but can not stop the exchange. As an embedded in-stream process our mechanism is uniquely capable of guarding privacy.

Our CCA module, in its prototype form, is a domain independent plug-in which processes XML formatted messages. The module is depicted in the area outlined by the dotted box in Figure 1. As a plug-in it is possible to include the module in any web enabled system which releases its data in XML format. The form of the messages must be known to the system and the system must be loaded with all policies which apply to the domain. The current prototype only includes the embedded module. The interface to define the messages, policies and resolutions for non-compliant messages is still in development.

The CCA process, illustrated in Figure 2, includes five sub-processes: Input: the identification of the message within the system, Validation: the assurance that the message conforms to the expected format, Sharing: the application of any intra- or inter-organization policies in relation to the exchange of the information, Compliance: the enforcement of all international, national and/or company regulatory policies, and Output: routing the verified message to the recipient. The system includes two additional subsystems, Resolving and Logging. Resolving forces a user defined action to be taken when a message is found to be non-compliant to regulatory policies. A message in violation of intra- or inter-company defined sharing polices is simply stripped of the offending data elements and sent forward. Logging creates an audit log of the message which allows both external independent ex post facto auditing and examination of data access and exchange.

The system database includes information on the data providers and recipients and the relationships between them, valid message formats expressed as XML schema definitions (XSD) [7], sharing policies

Table 1. Machine Specifications and Distribution of Components

Component	CPU	HD RPM	Memory	Operating System
CCA Mechanism & Enclave DB	Core2 Duo (2.53 GHz)	7200 RPM	2 GB DDR 800	Linux Server 2.6.28-11
Logging Facility	Pentium 4 HT (3.00 GHz)	10,000 RPM	1 GB DDR 400	Linux Server 2.6.28-11
Workload Generator	Pentium 4 HT (3.00 GHz)	7200 RPM	1 GB DDR 400	Linux Server 2.6.28-11

Table 2. Message type descriptions

	Number of Elements			Size (kB)		
Component	Mean	Standard Dev.	Median	Mean	Standard Dev.	Median
RAAR	2918	658	2267	144.35	115.37	112.07
Ship Manifest	920	651	694	40.26	31.72	30.43
Alert	3	0	3	0.559	0.0	0.553
Request	3	0	3	0.564	0.0	0.567

expressed as an XSLT [8], compliance policies expressed in XPath expressions [9] and resolving policies. Although not provided in the current mechanism future enhancements of the system will include an interface for the data sharing partners to specify the sharing policies pertaining to the data and a second interface for the internal auditor to specify the applicable regulations. Sharing policies are applied based on the relationship between the data provider and the data recipient. Compliance polices are applied based solely on the contents of the message. The applicable compliance policies can be determined statically based on the message type or dynamically based on the contents of the message. For example, in an application involving a global community, applicable privacy regulations can be determined based on the nationality of the individual in the message.

To determine that the CCA process, as an in-stream mechanism, will allow monitored information exchange within acceptable processing limits we created a test-bed environment geared to not only demonstrate system correctness but also to measure system viability.

3. Test Bed

We have built a test bed which consists of: a workload generator, our CCA mechanism, enclave database (small scale model), analysis tools, and our logging facility. Each of these components is implemented in the Java programming language. Communication between the components is facilitated through the use of the Java Messaging Service (JMS) [10] and a pair of ActiveMQ [11] message brokers. The decoupling of components in this manner allows us to experiment with different configurations as the components will always communicate in the same way (through the JMS interface). Table 1 describes our current configuration. Our machines are connected through the college's local area network, thus communication between components is subject to some network delays. It should be noted that this configuration is dictated by current shortages on resources and that it would be entirely possible to further separate any of the components (specifically move the enclave database to a different machine).

3.1. Workload Generator

Our workload generator provides our CCA mechanism with a continuous stream of messages to process. Our goal is to represent some specific use case of our system though the modeling of scenario specific messages. We currently have implemented four basic message types: Risk Assesment and Analysis Report (RAAR), Ship Manifest, Alert, and Request. XSDs representing each of these data models can be found here [12]. The RAAR is a message containing reports on suspicious information related to activities. Table 2 displays statistics related to the size and structure of each of the message types, gathered from the generation of 20,000 messages. We feel these messages are a representative subset of a larger super set of messages related to the example discussed in the introduction (cargo shipping). The workload generator randomly generates one of the four previously described messages and places that message on a queue, where it waits for the CCA Mechanism to accept it for processing. It does this at a rate of approximately 3 messages per second.

Another aspect of our workload generator is the mimicking of the trusted enclave environment described above, specifically the relationships between sharing partners. To achieve this we created a model enclave database containing prospective users of the system. We randomly create pairs of users and then

Table 3. Compliant Throughput Results per Message Size

Message Size (kB)	49.99	99.99	149.99	199.99	249.99	299.99	349.99	399.99	449.99	499.99	inf.
Messages per Second	47.62	20.00	15.15	12.50	10.10	8.70	7.63	6.70	6.17	5.75	4.12
Kilobytes per Second	161.13	1488.76	1871.81	2145.11	2212.46	2390.25	2463.06	2488.11	2614.91	2717.38	2668.03
Throughput Time (sec)	.021	.05	.066	.08	.099	.115	.131	.15	.162	.174	.243
Percent of Messages	75%	6%	5%	3.2%	1.5%	1.1%	1%	1%	1%	1%	3.7%

assign each of these pairs a message type at random (message types described in Table 2). Based on the assigned message type, we generate the necessary policies: Validation and Sharing, and link these policies to the user pair. Generating messages in this manner ensures that each message will test the full capabilities of the system: verify user relationship, enforce partner sharing agreement, and verify compliance to applicable policies.

3.2. CCA Mechanism

As discussed above, the CCA process consists of seven core modules: Input, Validation, Sharing, Compliance, Output, Logging, and Resolving. Input accepts messages from the Workload Generator (through a queue) and verifies the user's credentials and also performs a look-up of all applicable validation, sharing, and compliance policies. Validation next verifies the message's form by validating the message against a XSD. There exists a one-to-one relationship between message and validation policy. If the message is found to be of a valid form, it is next passed to the Sharing module where the message's content is verified against a single sharing policy, realized as an XSLT. The XSLT strips data elements that are prohibited from distribution by the provider's policies. Finally, compliance verifies that the provider's policy is not in violation of any corporate or regulatory policy. What we refer to as compliance policies are realized as regular XPath 1.0 expressions. These XPath expressions search the document for offending data. If none of the expressions return a result, the message is deemed compliant, otherwise it is non-compliant. One or many compliance policies exist per message (the set of applicable policies are retrieved during input).

If at any point during the process, a message fails, the message is immediately forwarded to a special module: Resolving. Failures include: no defined user relationship (detected at Input), invalid message form (detected at Validation), or non-compliance to a corporate or regulatory policy (detected at Compliance). Currently, Resolving only acts as a forwarding mechanism for failed messages.

3.3. Logging Facility

An important aspect of the CCA Mechanism is the Logging module which maintains an up-to-date log of the activity occurring throughout the processing of a message. This is achieved by sending a copy of the message to the Logging module before and after each module operates on it. Characteristics of these messages are then selected and stored into an audit log. Some of the characteristics include: providing and receiving users of the message, message usage, message content, as well as the time of the log entry. Capturing and storing this information allows for both external independent ex post facto auditing and examination of data access and exchange.

3.4. Gathering Statistics

Our test bed is designed to capture throughput performance statistics for our CCA mechanism. Through our use of the JMS interface and ActiveMQ, we are able to attach properties to each of the messages. These properties are only visible through the JMS interface and in no way affect the form of the XML document being processed. Leveraging the use of these properties, we are able to record timestamps of significant events in the processing cycle. For instance, we record the times a message enters and exits a module. This allows us to assess individual module performance. Using our result capture strategy, we are able to assess our system on varying levels of grainularity.

Messages that have passed through the CCA mechanism are collected onto a final queue. Our test bed includes a tool for automatically retrieving and assessing the messages residing on this queue. The assessment includes the calculation of average throughput time, average messages per second, average bytes per second, and average time spent in each of our modules, among other statistics. In the next section, we report our throughput numbers.

4. Test Bed Results

We sent 20,000 messages through our system and collected the results, using the configuration displayed

in Table 1. Due to the way we generate our messages, a certain percentage of the messages will fail (about 25-33%). Of our 20,000 messages, 14,486 of them were found to be compliant, and therefore successful. The most computationally intensive case occurs when a message is found compliant (it visits each module). As a result, we direct our performance analysis to this case. Table 3 displays the results collected from this test, i.e. these message throughputs only pertain to the 14,486 successful messages. The messages are broken down into tiers by size (in kilobytes); the upper limits on each of those tiers are displayed in the first row. Each reported throughput value is an average for that specific tier. We also report the time per message (for each tier) and the percentage of messages belonging to that tier. We can see that as the message grows larger, the number of messages our system can process per second steadily decreases, while the number of kilobytes we are able to process per second increases to a maximum of approximately 2717 kilobytes/second. From the statistics we gathered, our average throughput of compliant messages is: 23.81 messages per second and 1513.34 kilobytes per second.

5. Conclusion

In this paper we have presented a trusted enclave environment with an embedded CCA process to facilitate information sharing in a secure setting. The CCA mechanism provides real-time auditing of messages for compliance to both participant sharing policies and regulatory policies. One of the major impediments to real-time or embedded in-stream auditing is the cost of increased processing time [13]. In a report to the president on Revolutionizing Health Care Through Information Technology, the committee found that other commercial auditing systems consume excessive storage overhead and degrade system performance, leading administrators to either turn off the auditing feature or regularly purge the audit logs [14]. In order for our CCA module to be successful in enabling trusted information exchange compliant with all governmental, international and corporate regulations, we must yield performance results which will overcome the need to turn off the auditing feature. While we continue to explore methods to increase our processing throughput, we believe that these results demonstrate that the mechanism is a viable process to include in information sharing applications to ensure both information security and privacy.

6. Acknowledgements

This work was funded in part by Lockheed Martin Corp., VIACK Corp., and the University of Oklahoma through the ICFS program.

References

[1] U. S. G. A. Office, "Information sharing: The federal government needs to establish policies and processes for sharing terrorism-related and sensitive but unclassified information," Tech. Rep., 2006. [Online]. Available: http://www.gao.gov/new.items/d06385.pdf

[2] Lee and Whang, "Information sharing in a supply chain," *Internation Journal of Technology Management*, vol. 20, no. 3, pp. 373–387, 1999.

[3] L. Seligman and A. Rosenthal, "Xml's impact on databases and data sharing," *Computer*, vol. 34, no. 6, pp. 59–67, 2001.

[4] "Global justice xml." [Online]. Available: http://www.it.ojp.gov/jxdm/

[5] "Xbrl." [Online]. Available: http://www.xbrl.org/Home/

[6] "Acord xml standard." [Online]. Available: http://www.acord.org/Standards/propertyxml.aspx

[7] "W3c specification: Xml schema language." [Online]. Available: http://www.w3.org/XML/Schema

[8] "W3c specification: Xml transformations." [Online]. Available: http://www.w3.org/TR/xslt

[9] "W3c specification: Xpath." [Online]. Available: http://www.w3.org/TR/xpath

[10] "Java message service (jms)." [Online]. Available: http://java.sun.com/products/jms/

[11] "Activemq message broker." [Online]. Available: http://activemq.apache.org/

[12] "Cca message data models." [Online]. Available: http://joe-dalessandro.net/cca/xsd

[13] Debreceny, Gray, Ng, Lee, and Yau, "Embedded audit modules in enterprise resource planning systems: Implementation and functionality," *Journal of Information Systems*, vol. 19, no. 2, pp. 7–28, 2005.

[14] G. Johnson, "Compliance with data protection laws using hippocratic database active enforcement and auditing," *IBM Systems Journal*, vol. 46, no. 2, 2007.