NGCRC Project
End-to-End Security Policy Auditing and Enforcement in Service Oriented Architecture

Final Report: Aug 2014

Prof. Bharat Bhargava
CERIAS and Computer Science, Purdue University

Table of Contents

1. Introduction

An SOA client interacts with a service, submits its request and gets a response back. The client expects certain levels of quality of service guarantees. These can be expressed as security and privacy policies, interaction authorization policies and service performance based policies. It is impossible to guarantee whether these constraints are respected and enforced without a monitoring agent. Moreover, if a service is compromised or misbehaves, the monitoring agent needs to discover the malicious activity, provide feedback and take remedial action. There is need for novel techniques to monitor service activity, to discover and report service misbehavior and ensure security and privacy of data. We propose a novel approach that uses a service monitor to audit and detect malicious or compromised services. The research focus was to detect illegal service interactions that violate system policies. Service monitor has two main modules: Interaction authorization and Trust management. Trust based service reputation management algorithms and service interaction authorization policies were developed to stop service invocations that violate system policies by evaluating their trust based on their actions.

1.1 Problem Statement

In a service-oriented architecture (SOA) environment, services can collaborate to provide broader functionality and accomplish tasks quickly. In SOA a service can dynamically select and invoke any service from a group of services to offload a part of its functionality. This is very useful to build large systems with existing services and dynamically add services to support new features. One of the main problems with such a system is that from a client (user or service) perspective, it is very difficult to trust the service interaction lifecycle and assume that the services behave as expected and respect the client's (user/organization) policies.

Figure 1 shows an example where a user's personally identifiable information(PII) bring disseminated out of the organization boundary during an SOA transaction. It is impossible for the organization to ensure that it's organizational policies are enforced once the PII leaves the boundary. Furthermore, it is important to have means of understanding how services perform down the pipeline and dynamically adjust service interactions. For example if a dependent service or a set of services are under attack or due to various factors if their trustworthiness declines, such services should not be relied upon.

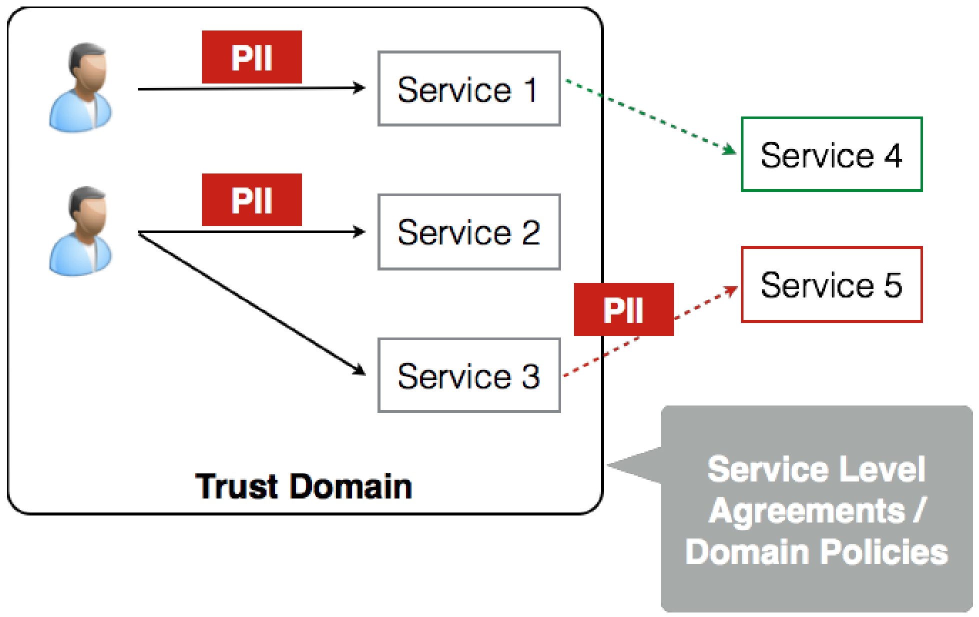


Fig. 1: SLA/Policy violations in SOA interactions

1.2 Proposed Solution

Our current solution using a service monitor audits and detects malicious or compromised services based on a monitoring agent with trust management and policy enforcement capabilities. The main features of this solution are the following:

- An agent that can monitor all services in the system and detect non-compliance of service behavior with respect to system policies and service level agreements.
- A trust management subsystem that supports pluggable trust management algorithms, as part of the monitoring agent that can track and evaluate service behavior over time based on service interactions, consumer feedback, service state and system context.
- A transparent mechanism that supports passive and active monitoring to track, authorize, block and redirect service actions.
- A policy enforcement subsystem that supports pluggable service interaction authorization policies.
- A web-based management console to experiment service interactions by enabling authorization policies and trust management algorithms to evaluate different service topologies.

1.2.1 Benefits of the proposed solution

This solution proposes a novel method of dealing with security problems in SOA. The main advantages of the proposed solution are as follows:

- Monitors all interactions among services in the enterprise.
- Provides increased awareness of security violations.

- Proactive treatment of potentially malicious service invocations.
- Dynamic trust management of services in an enterprise.
- Enables timely detection of potentially compromised services.
- Detection of bottlenecks in an enterprise SOA to improve performance.
- Easy integration of any service topology, trust management algorithms and authorization policy into a SOA system.
- Provides a platform to experiment with different service topologies and policies along with different perspectives of service trust evaluation.

2. Solution Architecture

Figure 2 below shows the architecture and components of the proposed end-to-end security framework. The details of the different components and their interactions are described in the next section.

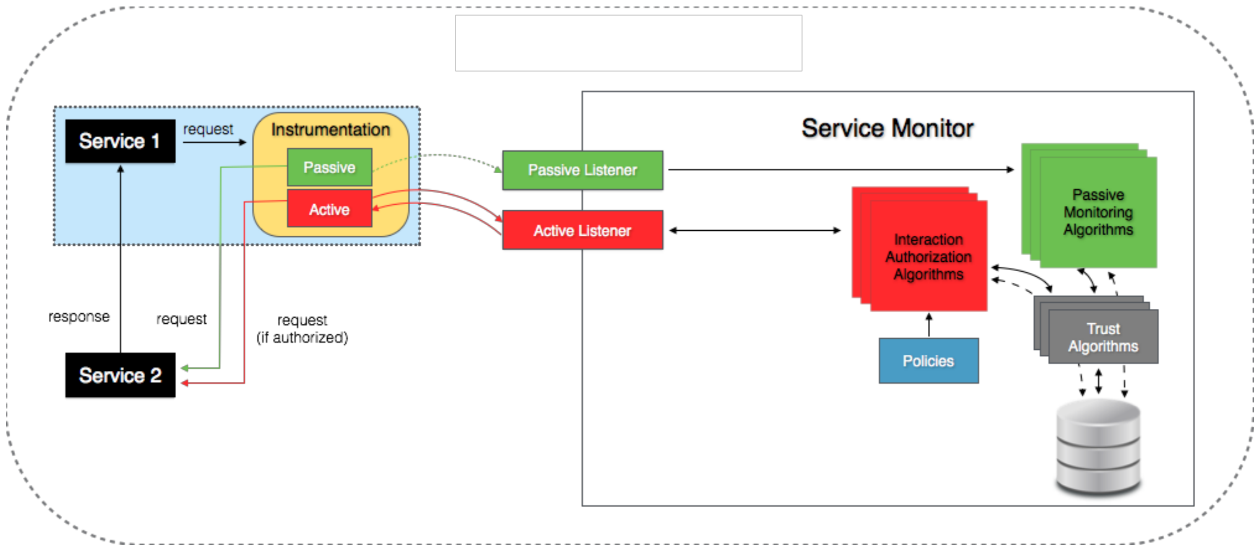


Fig. 2: Solution Architecture

2.1 Service Monitor

The service monitor is the core module installed as part of the SOA system. The service monitor was developed as a set of services and a management console with the following features:

2.1.1 Monitoring Modes

The service monitor can either record and analyze all service interactions or it can intervene to manage interactions as configured. There are two modes of monitoring: passive and active. Details of each monitoring mode are described below.

Passive Monitoring: The passive monitoring mode as depicted in Figure 3 involves the recording of all service interactions by the service monitor in the form of (caller, callee) pairs. In this mode, all service invocations are intercepted, but not blocked. The effect of the interception is only on the analysis of service interactions and the update of trust levels of services based on the interactions. The recording and trust update operations following the interception are performed in an asynchronous manner transparent to the service invocation.

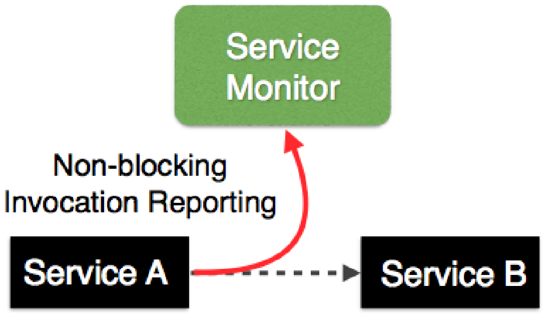


Fig. 3: Passive Monitoring

Active Monitoring: The active monitoring mode as depicted in Figure 4 involves interception of all service invocations to dynamically check their compliance with system policies. The service monitor in this mode uses its interaction authorization component to decide the appropriate action for the service invocation (e.g. allow, block, redirect). The service invocation is blocked until an authorization is received based on the evaluation of associated policies, and cancelled if the invocation does not comply with the policies.

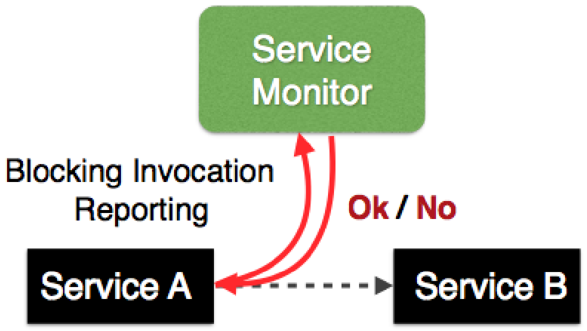


Fig. 4: Active Monitoring

2.1.2 Trust Management

Service trust is a measure of service behavior over time. Enterprise policies and Service Level Agreements (SLAs) use trust values to argue about quality of service. Service Monitor uses the trust management module to evaluate service behavior and maintain dynamic trust values for monitored services. The trust management module integrated with the service monitor provides the ability to conveniently integrate and evaluate different service trust management algorithms with the set of available scenarios. This module uses a database of essential information about services including service name, url, port, parameters and trust level. The trust level of a service at any point in time indicates how that service compares to other services in terms of compliance with policies, security requirements, SLAs etc. This information can be used in service composition policies to restrict invocations of services with trust levels below a certain threshold.

The trust management module is responsible for updating the trust value of services based on various parameters like service state, service behavior and system context. In our current implementation, trust levels of services are updated based on their interactions with other services. Specifically, the trust level of a service *s1* is updated upon its invocation of another service *s2*. The interface implemented by the different trust management algorithms in the framework has the form:

```
alg(interaction_id)
```

where the `interaction_id` is the key that can be used to look up interaction details. Algorithm implementations are added to the framework as self-contained modules, which expose this function. The current framework implements a set of trust algorithms and additional algorithms from the rich literature in trust management can easily be integrated into the system by adding their implementation files to a specific directory holding all trust algorithm implementations. The algorithm used by the trust management module at any point in time can be selected dynamically from among the algorithms available in the framework.

2.1.3 Interaction Authorization

The service monitor has the ability to decide the fate of a pending service interaction when a service is under active monitoring. The decision is handed over to an "interaction authorization" algorithm. The interaction authorization algorithm, if enabled, in the current framework makes authorization decisions based on compliance with the global policies in the system.

The interface implemented by the interaction authorization algorithm has the form:

```
authorize(from, to, id, callback)
```

Implementations are added to the framework as self-contained modules which expose the above function. The `from` and `to` parameters provide caller and callee metadata. `cb` is the callback provided to return the authorization decision. This authorization decision is communicated by the service monitor to the service instrumentation.

Policies: Certain service level agreements, organizational rules/regulations and quality of service requirements may be expressed in terms of interaction authorization policies. The interaction authorization algorithms may enforce one or more such policies. The policy setup process is described as follows:

- **Policy definition:** Service invocation policies are defined at the global (enterprise) level and registered with the Service Monitor.
- **Policy monitoring:** Active listeners monitor service invocations within a service invocation chain.
- **Policy decision-making:** Service Monitor decides which action must be taken based on the evaluation of enabled policies.
- **Policy enforcement:** Interaction Authorization algorithms are applied to allow/disallow service interactions based on system policies in the active monitoring case.

Following are some possible interaction authorization policies that can be enforced by the service monitor.

- [Privacy] Service may not communicate with “advertising” services which carry a trust level below a certain threshold. This acts as a remote “ad-block” for the user where he/she wishes not to receive advertisements from services who are not trusted.
- [Privacy] Service may not leak the “address” information of the user to “advertising” services. This policy would prevent leakage of identity information of the user.
- [Confidentiality] Service may not interact with any service over plain text.
- [Confidentiality] Service may not transmit user’s credit card information to any external service below a certain trust level.

Implementation of such policies can be based on a standard policy language. For instance, we demonstrate the use of XACML to express these interaction authorization policies in our implementation.

3. Implementation

We have implemented an end-to-end security framework with the service monitoring approach for REST-based services, using the node.js framework (http://nodejs.org/) for network applications. In the developed framework, end-to-end security is provided by tracking all service invocations (including inter-service communication), and either recording (passive monitoring) or blocking (active monitoring) illegal service invocations (i.e. those not complying with system-level policies). The capabilities of the current framework can be listed as follows:

- Invocations in an end-to-end chain of services for a request can be tracked and recorded.
- Multiple global (system-level) policies can be specified for enforcement on service invocations (policy language: XACML).
- Service invocations can be actively monitored to block those not complying with global

policies.

- Different trust algorithms can be integrated into the framework to evaluate the trustworthiness of services in the system.
- Trust values of services can be updated based on their interactions with other services and feedback from services about interactions.

At the heart of the framework is the service monitor module, which includes submodules for managing trust, tracking service invocations and making service invocation authorization decisions. The service monitor and its components are described in detail below.

3.1 Service Instrumentation

REST services monitored by this framework are implemented as node.js applications. A node.js application uses the "request" module when invoking a REST service. This framework provides two instances of instrumentation of the "request" module:

- **Non-blocking Instrumentation:** This instrumentation intercepts a REST call to a remote service and forwards interaction metadata to the service monitor application. It is important to note that the call to the service monitor does not block the regular operation of the external service invocation.
- **Blocking Instrumentation:** This instrumentation intercepts a REST call to a remote service, forwards interaction metadata to the service monitor and waits for the service monitor to authorize the interaction.

In both instrumentations, metadata forwarded to the service monitor includes invoker identity, target service identity, target operation, transport information (HTTP/HTTPS), and start and end times of the operation. Each instrumentation sends two messages to the service monitor - pre-invocation and post-invocation.

The non-blocking instrumentation uses a UDP to send messages to the service monitor in a fire-and-forget manner. This provides a very fast method to send out metadata about an interaction. The blocking instrumentation uses an HTTP connection to send a request with interaction metadata and waits for a response from the service monitor.

3.1.1 Service Feedback

The non-blocking and blocking instrumentations include an optional hook for the caller applications to provide feedback about a service interaction. This is invoked after the completion of a service request and before sending the final set of metadata to the monitor. The caller application can implement the following function to to provide feedback of any form by calling the results callback function.

```
global.eval_interaction = function(target, start, end, results){
     //Invoke results() with any feedback data as JSON object.
```

```
        results({...});
};
```

3.2 Service Monitor

The service monitor is implemented as a centralized Node.js service which continuously watches all interactions and takes appropriate actions. It has the following modules:

3.2.1 Interaction Authorization Algorithms

Interaction authorization module is designed to plug-in independent interaction authorization algorithms conveniently. Furthermore the service monitor allows multiple interaction authorization algorithms to be enabled simultaneously. These algorithms can be based on organizational SLAs, rules and QoS requirements. We provide the ability to express these constraints as policies.
These interaction authorization algorithms are invoked when monitored services uses the blocking request instrumentation. The request is blocked while waiting for a response from the service monitor and the instrumented request take appropriate actions based on instructions received. For example, the interaction authorization algorithm may decide to block or redirect the service request based on the context.

Policy Implementation Details:

- **Policy Language:** XACML 3.0 specifications are followed to define policies in XML.
- **Policy Engine:** WSO2 Balana implementation of XACML 3.0 is used in the framework as the policy engine for making authorization decisions.

Creating and Testing a Policy: A new policy is defined using XACML and undergoes unit testing using the *policy development and testing module* provided in the framework. An access control request in XML format is used to test all possible scenarios of the policy that generates the expected “Deny” and “Permit” outcomes.

Deploying a Policy: Once tested, a new policy is integrated into the service monitor. The XML request format is converted to a JSON template. This template is populated with appropriate values by the interaction authorization algorithm when invoking the access controller.

Details of three example interaction authorization policies are described below. Any policy that can be specified using XACML can easily be integrated into the framework.

Policy 1: Remote Untrusted Ad-Block

Deny	
Resource	any
Subject	any
Action	READ
Env: http://endtoendsoa.cs.purdue.edu/policy/trust_level	< 5
Env: http://endtoendsoa.cs.purdue.edu/policy/operation	substr("advertisement")

This policy blocks invocations of services that include advertisements if those services have a trust level lower than 5. The operation value ('advertisement') is inferred from the url used to access the service (callee).

Policy 2: No Plain Text Transport

Deny	
Resource	any
Subject	any
Action	READ
Env: http://endtoendsoa.cs.purdue.edu/policy/transport	plain

This policy blocks invocations to services that do not use the "https" protocol. http://endtoendsoa.cs.purdue.edu/policy/transport attribute value will be set to "plain" if data is transmitted over plain text. We infer the value of the transport protocol from the url used to access the resource.

Policy 3: Block Credit Card Transmission

Deny	
Resource	any
Subject	any
Action	WRITE
Env: http://endtoendsoa.cs.purdue.edu/policy/trust_level	< 10
Env: http://endtoendsoa.cs.purdue.edu/policy/data	Includes(Credit Card #)

This policy blocks the posting of a user's credit card number to a payment service with a trust level less than 10.

3.3 Trust Management

The trust management subsystem is responsible for managing the trust values of services based on different trust management algorithms. The algorithm implementations can be easily plugged into the framework as self-contained modules. The current framework implements the trust algorithms described below, and additional algorithms from the rich literature in trust management can easily be integrated into the system by adding their implementation files to a specific directory holding all trust algorithm implementations. Multiple trust management algorithms can be dynamically enabled at any point via the service monitor management console.

3.3.1 Trust Algorithms

1. **Simple Average:** This is a basic algorithm for evaluating trust. This algorithm sets the trust value of the caller to the average of the trust values of the caller and the callee.

```
from.trust_level = (from.trust_level + to.trust_level)/2
```

2. **Weighted Moving Average:** In this algorithm, the trust value of the caller is updated using a weighted average of the trust levels of the caller and the callee. The trust value of the caller is updated using the following formula, where *w* is a real number in the range [0,1].

```
from.trust_level = w * from.trust_level + (1-w) * to.trust_level
```

While a weight in the range [0, 0.5) attaches greater importance to the original trust level of the caller, a weight in the range (0.5, 1] has greater emphasis on the trust value of the callee. For the case of *weight = 0.5*, this is equivalent to the *simple average* algorithm.

3. **Client Feedback**: This algorithms complements the *weighted moving average trust* algorithm. While the *weighted moving average trust* algorithm evaluates trust for the caller, the *client feedback* algorithm evaluated the trust for the callee. We adapted the self-organizing trust model [CB13] for peer-to-peer systems and extended it for service trust evaluation in SOA. This algorithm is based on the feedback of consumers about their interactions with services. The feedback includes two main components: **'satisfaction'** and **'weight'**. This is formally defined as follows:

Satisfaction (s_{ij}): It is defined as *service i's* degree of satisfaction about an interaction with *service j* as perceived by *service i.*

Weight (w_{ij}): It is defined as the significance of *service i's* interaction with *service j* as perceived by *service i.*

This algorithm calculates trust using a moving window of *n* most recent service feedbacks. A **fading factor (f_{ij}^{k})** is used to decide the contribution of a particular feedback in the computation of the trust value. The fading factor is calculated as follows:

$$f_{ij}^{k} = k / sh_{ij}$$

k	**:** position of (most recent) interaction in the window
sh_{ij}	**:** size of feedback window

Competence belief (cb): It is defined as the average behavior of a service based on consumer feedbacks and is calculated as follows:

$$cb_{ij} = \frac{1}{\beta_{cb}} \sum_{k=1}^{sh_{ij}} \left(s_{ij}^{k} \cdot w_{ij}^{k} \cdot f_{ij}^{k}\right)$$

Integrity belief (ib): It is defined as the level of confidence in predicting a service's behaviour based on variation in consumer feedback. It is calculated as follows:

$$ib_{ij} = \sqrt{\frac{1}{sh_{ij}} \sum_{k=1}^{sh_{ij}} \left(s_{ij}^{k} \cdot w_{ij}^{k} \cdot f_{ij}^{k} - cb_{ij}\right)^{2}}$$

Trust level: Trust value of a service is evaluated using the average of competence beliefs and integrity beliefs based on the feedback of all consumers of that service. It is calculated as follows:

$$t_i = avg.\ cb - (avg.\ ib / 2)$$

3.4 Management Console

The management console is a UI for Service Monitor. It presents different functional views, which are described as follows:

- **Scenario List:** This view shows different SOA scenarios categorized as Active and Passive scenarios. Users can select a scenario and go to the scenario page.

- **Scenario Page:** This view shows details of the selected scenario, for e.g. the list of services in the scenario with the option of starting/stopping them, trust values for these services with the option of updating trust values for experimentation, link to scenario statistics page and a try button to play the scenario.
- **Trust Management and Interaction Authorization Algorithms List:** This view shows different trust management algorithms and interaction authorization algorithms. Users can enable/disable these algorithms via this page. The enabled algorithms are highlighted to show their status.
- **Interaction List:** This view shows the logs of service interactions captured by the Service Monitor.
- **Service List:** This view lists all the services in the system along with their endpoint details, trust values, options to enable/disable them and links to the service statistics page.
- **Service/Scenario Statistics Pages:** These pages provide detailed statistics about service usage and scenario trust level distributions over time for all services in a scenario for each trust management algorithm.

3.4.1 Service Topology Configurations

All scenarios were developed using a set of node.js/express services. These services use the aforementioned blocking and non-blocking instrumented request modules. These services are grouped into scenarios and are independently tested to ensure regular operation of the scenario before being integrated into the service monitor management console.

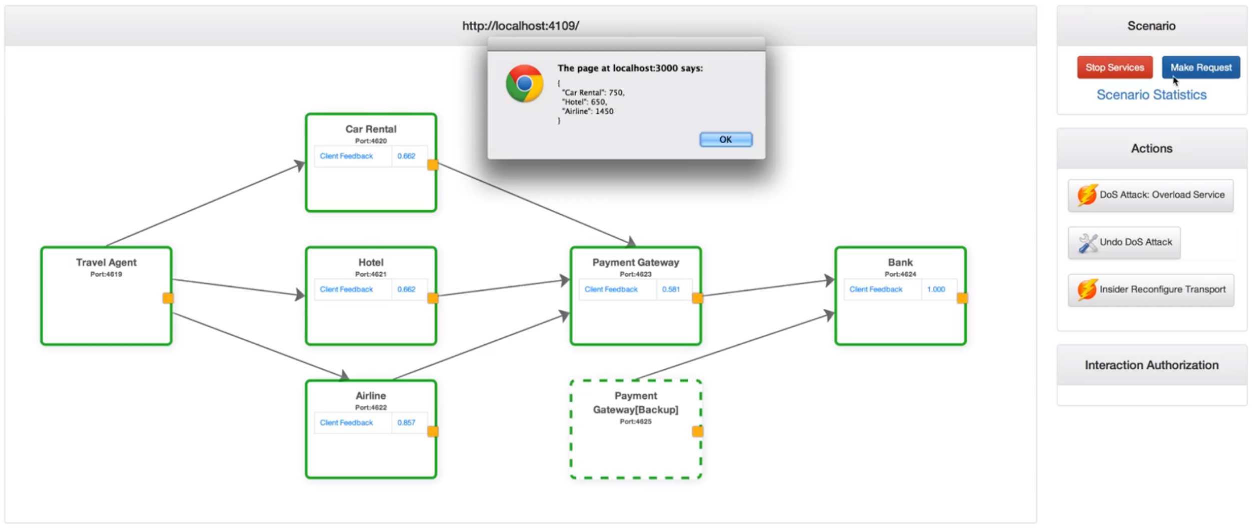
A descriptor file is used to define a service topology that is to be integrated into the management console. An example of such a descriptor is shown in Figure 5. Figure 6 shows the rendered topology in the management console based on Figure 5 descriptor. Table 1 provides descriptions for each field in the service topology descriptor file.

```
{
    "id" : 6,
    "access_url" : "http://localhost:4109/",
     "description" : "Composition of 6 services which demonstrates two types of
failures.",
    "name": "Demo Scenario",
    "services" : [9, 10, 11, 12, 13, 14, 15],
    "connections" :[[9,10] ,[9, 11] ,[9, 12] ,[10, 13] ,[11, 13],
                   [12, 13], [13, 14], [15, 14]],
    "pos" : [{"id":9, "x" : -300, "y" : 200},
              {"id": 10 , "x" : 100, "y" : -75},
              {"id": 11 , "x" : 100, "y" : 160},
              {"id": 12 , "x" : 150, "y" : 400},
              {"id": 13 , "x" : 500, "y" : 180},
              {"id": 14 , "x" : 900, "y" : 200},
              {"id": 15 , "x" : 500, "y" : 400}],
    "actions" : [{"name" : "DoS Attack: Overload Service", "type" : "attack",
                  "invoke_url" : "http://localhost:5113/dos_attack"},
                 {"name" : "Undo DoS Attack", "type" : "fix",
                  "invoke_url" : "http://localhost:5113/fix_dos_attack"},
                {"name" : "Insider Reconfigure Transport", "type" : "attack",
                  "invoke_url" : "http://localhost:4111/attack_transport"}],
    "status" : [ {"service" : 15, "status" : "svc_backup"}]
}
```

Fig. 5: Example service topology descriptor

Table 1: Descriptions of each field of the scenario configuration

Field	**Description**
`id`	A numerical identifier of the topology.
`access_url`	The URL of the initial service of the service topology. This is the service which relies on the other services of the topology.
`description`	A description of the scenario.
`name`	Name of the scenario.
`services`	This is a list of service id values. These id values are extracted from the service monitor database.
`connections`	This is a description of the directed edges of the service topology. These represents requests made from one service to another. If service A calls service B, the entry [A, B] will be added to the list of connections.
`pos`	This optional field provides positioning details for rendering the service topology on service monitor for each service. When this information is not present, the service monitor will automatically position the service nodes in the order they appear in the 'services' field.
`actions`	This field provides means to show shortcuts to various actions that can be programmed into the scenario for experimentation purposes. For example a service may include provisions to simulate a denial-of-service attack using a special endpoint. The action field allows the developer to provide the URL and a label for it to be displayed in the management console.
`status`	This field is used to highlight special services, for e.g. backup services in the UI.



The page at localhost:3000 says:

```
{
  "Car Rental": 750,
  "Hotel": 650,
  "Airline": 1450
}
```

Fig. 6: Service topology rendered in the management console

3.5 Installation Instructions

Installation instructions to install the monitoring framework on Ubuntu Linux is provided here. These instructions serve as guidelines for installation on any other operating system as well.

Project Page	https://www.cs.purdue.edu/homes/rranchal/ngcsoa_webpage/
Source Code	https://code.google.com/p/end-to-end-soa/
License	MIT License Copyright 2013

Obtaining the Source: The source code can be downloaded using git. If git does not exist it can be installed by running the command:

```
  sudo apt-get install git
```

To download the source run the following command:

```
  git clone https://code.google.com/p/end-to-end-soa/
```

Required Software Packages: The following software packages must be installed prior to running the main install procedure.

Table 2: Required Software Packages and Installation Instructions

Software Package	**Installation Instructions**

nodejs	`sudo apt-get install nodejs`
mysql	`sudo apt-get install mysql`

maven (> version 2.0)	`sudo apt-get install maven`
	If upgrading from an older version of maven ensure that maven2 is no longer installed by running:
	`sudo apt-get remove maven2`

Java7	Verify the installed version of java by running:
	`java -version`
	If the output indicates tha version of at least "1.7.0" then java7 is already installed.
	To install on Ubuntu run:
	`sudo apt-get install default-jre`

MySQL Database Setup: A MySQL database named soa_trust is used by the framework for storing vital information. This database must be set up prior to running the main install procedure. A file containing an SQL script for database setup is provided in the "node" directory of the project source.

- To execute this file to create the database run:

  ```
  mysql -u ROOT_USER -p < db.sql
  ```

  Where `ROOT_USER` is the name of the root mysql user that was created as part of MySQL installation.
- Set the MySQL username and password in `node/monitor/db/index.js` to a user that has read and write access to the soa_trust database.
- Optional: Use any init.x script in the experiment directory to assign default trust values.

Install & Run
Step 1: Build the policy using maven:
From the "policy" directory of the project source run:

```
mvn clean install
```

Step 2: Install the framework:
From the "node" directory of the project source run:

```
./install
```

Running the Framework:

To start service monitor:

```
./start
```

To stop everything:

```
./stop
```

Port Assignments: The monitor will use port 3000 and all services in scenarios are assigned ports starting with 4101. Please make sure there are no conflicts.

4. Demo Overview

The scenario developed for demonstration of the framework features involves a travel reservation service relying on three other types of services (airline, advertising and payment) to complete a service request.

4.1 Sample Scenarios & Demo

A set of screen capture videos were produced to demonstrate the service monitor and its capabilities. Table 3 provides the details of the videos along with URLs to access them on the Web.

Table 3: Demonstration video details

Video	Description	Link
Part 1	Introduces a typical SOA topology of a travel agent system which depends on a set of remote services. This introduces the features of the management console to start a topology, invoke the main service, view interaction details, enable trust modules, and enable interaction authorization modules. An interaction authorization algorithm that is based on client feedback trust values is used to demonstrate redirection of service requests.	`http://youtu.be/eJTT075rWQM`
Part 2	Demonstrates an insider attack on the hotel service. The simulated insider attack changes the communication protocol of one of the service to an insecure protocol. This further	`http://youtu.be/cbwfB0u9gfc`

	shows how an interaction authorization module blocks insecure interactions.	
Part 3	Demonstrates the use of a XACML based interaction authorization module to evaluate contents of a request and take actions. In this case the policy detects the presence of credit card details in request data.	`http://youtu.be/cEzy6frCX34`

5. Experiments

We evaluated the system for performance and security. These are discussed as follows:

5.1 Performance

We conducted a series of experiments to measure the overhead of the passive and active monitoring on the total round-trip time of the service interactions and compared it with the baseline setting without any monitoring. Figures 7, 8 and 9 show the interaction round trip request-response steps for the baseline, passive and active scenarios respectively. The experiments are conducted in a LAN setting for a chain of two REST services (client calls service 1 and service 1 calls service 2). The data was collected over 5 runs, with 50 concurrent requests in each run and averaged. Table 4 shows the results obtained for average round trip time per request in each run. Figure 10 shows the summarized results of the experiments. The outcome of the experiments show that the overhead of the proposed security measures is negligible in passive monitoring. However, the experiments show 53% of overhead in active monitoring scenario. This overhead is due to blocking the service interaction for authorization based on 2 enabled XACML policies. Further experiments show that the overhead of increasing the number of policies does not have a significant effect on the performance.

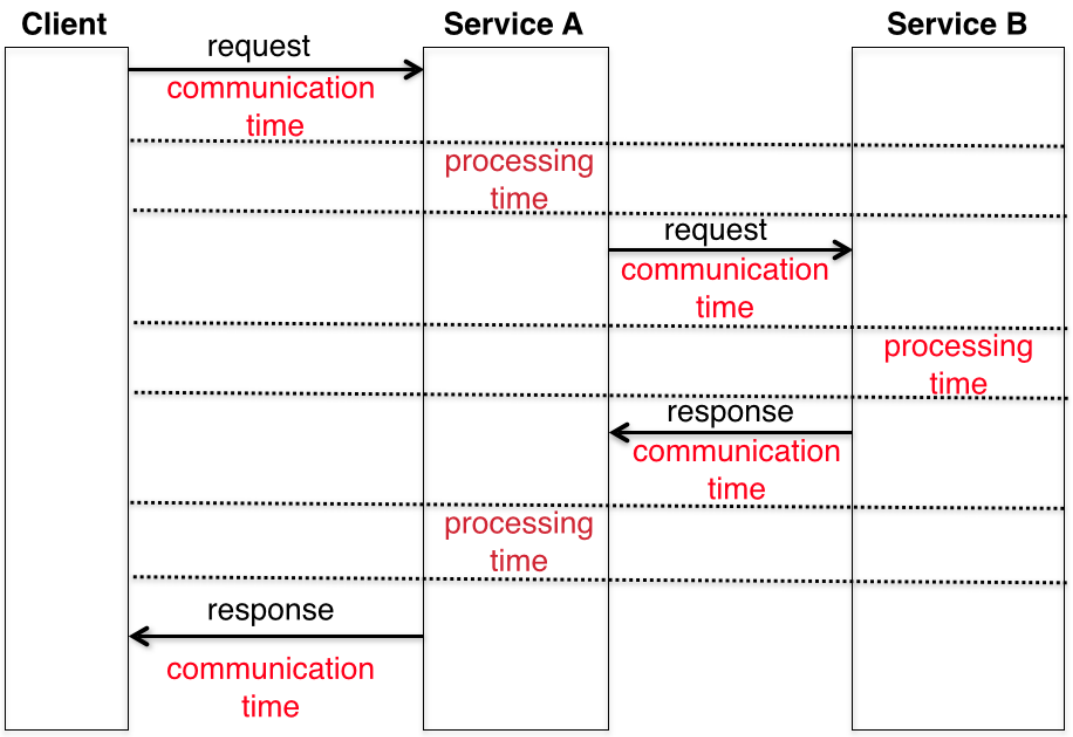


Fig. 7: Baseline Scenario Steps (without monitoring)

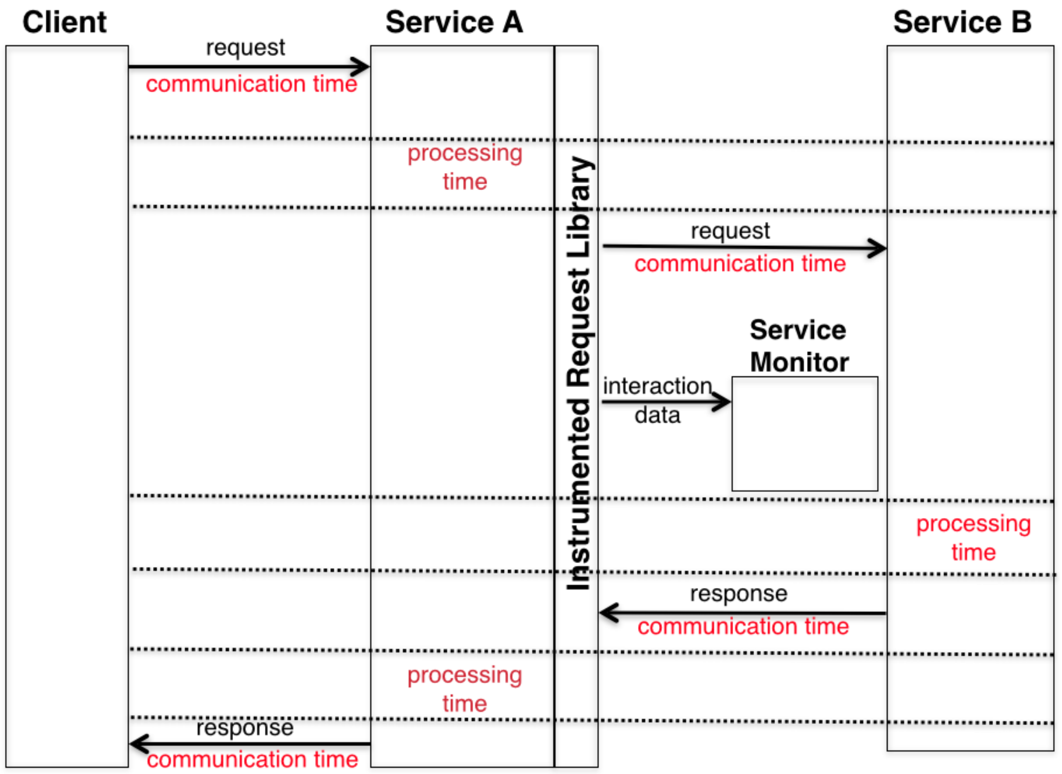


Fig. 8: Passive Monitoring Scenario Steps

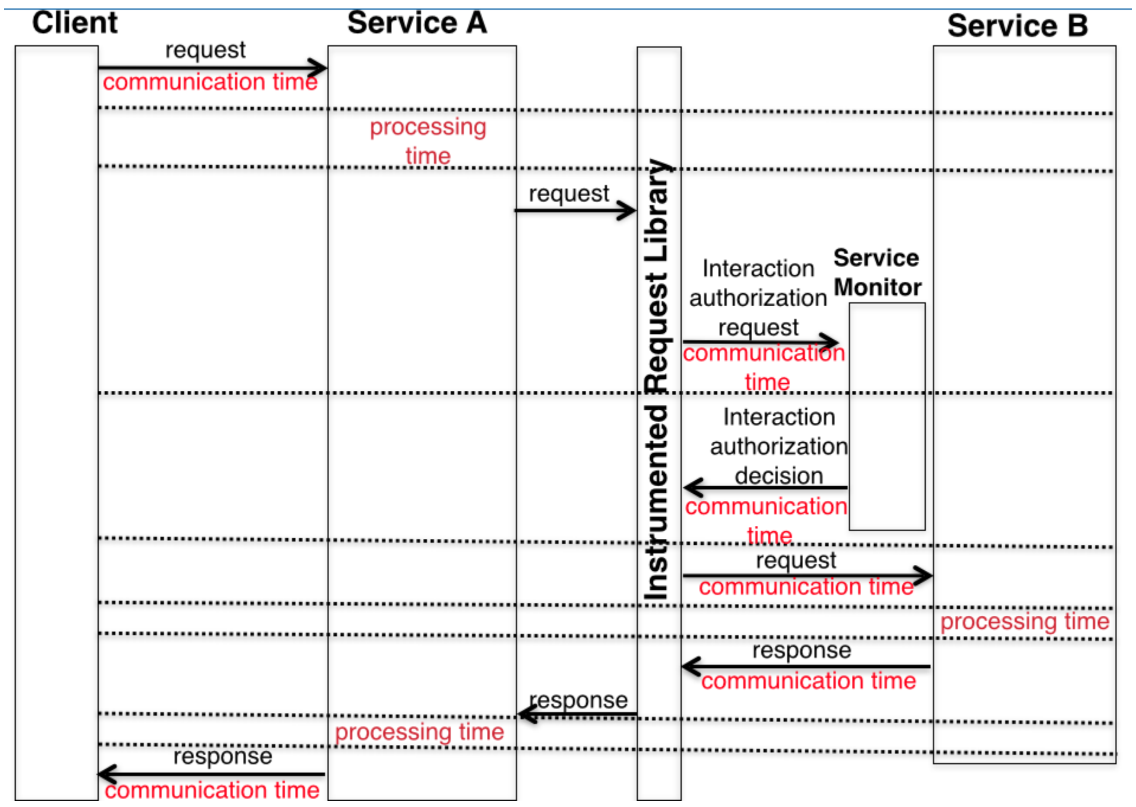


Fig. 9: Active Monitoring Scenario Steps

Table 4: Average round trip time per request (ms) per run

	Baseline	Passive	Active
Run 1	10.11	11.48	24.35
Run 2	10.09	10.77	15.62
Run 3	10.20	10.13	13.65
Run 4	10.83	9.34	13.56
Run 5	9.52	10.92	13.57

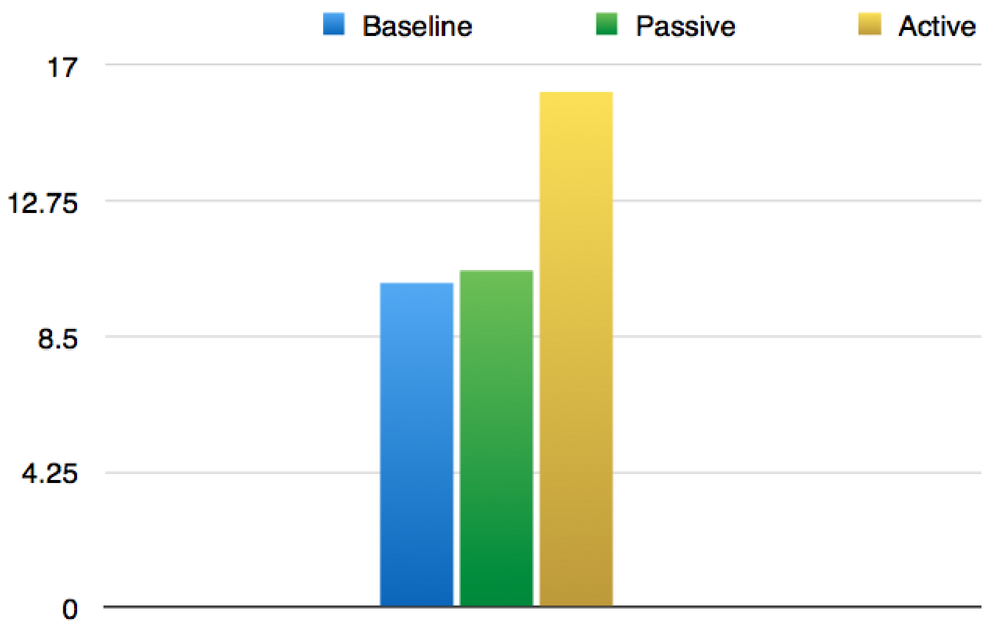


Fig. 10: Average round trip time per request (ms)

5.2 Attack Simulations

We use a scenario based on a travel agent service (Figure 6) to simulate a set of attacks and evaluate system security under monitoring. The scenario setup consists of three services that are consumed by the travel agent service (car rental, hotel reservation and airline). These three services rely on a payment gateway service. The service monitor had a set of trust management algorithms and a set of interaction authorization algorithms enabled, which were used to identify and take actions in the context of these attacks. The system was monitored using Active Monitoring mode, which used the blocking instrumentation, for all the simulations.

5.2 Denial-of-Service Attack

We simulated a DoS attack on this payment gateway service. The DoS attack was simulated by introducing a delay in the request processing at this service. This is triggered by a remote command that can be invoked by sending a HTTP GET request to the payment gateway service. This delay trigger was designed in a manner that the delay can be increased by issuing multiple requests to it. We also developed means to undo this DoS attack by introducing another operation to remove the delay.

Initially, the client feedback trust algorithm was deployed in this scenario. The three services that depend on the payment gateway service generated weak feedback values due to the delay in processing times at the payment gateway. This caused the client feedback trust value of payment gateway to deteriorate. Simultaneously we enabled an interaction authorization algorithm that takes the client feedback trust values of services into account. A threshold trust value was set

(Eg. 0.5) as the trigger to redirect requests to a backup service as the current service is perceived to be under attack. In this situation the payment gateway service has a backup service ready and the information of this backup service is made available via service configuration.
We show that the service monitor successfully instructs the client instrumentation at caller services to redirect requests to the backup service when the trust value of the payment gateway service drops below a predefined threshold. Figure 11 shows the service configuration after redirection of service requests.

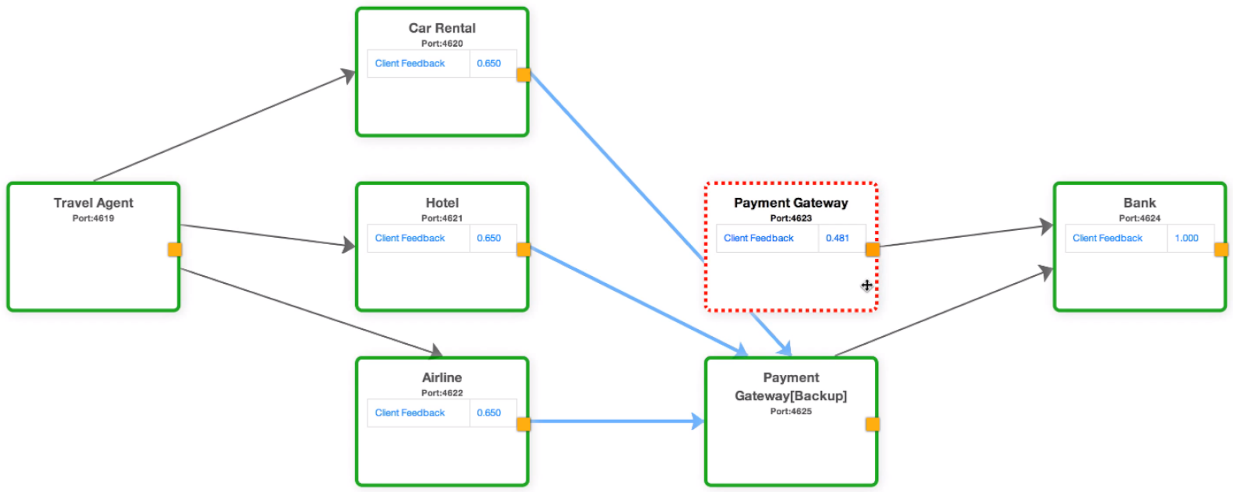


Fig. 11: Redirection of service requests to the backup payment gateway service during DoS attack on the primary payment gateway service.

5.3 Insider Attack

Hotel service was used to simulate an insider attack. The payment gateway service was developed with two endpoints hosted on HTTP and HTTPS. The three services use the HTTPS endpoint. In this attack simulation an insider with access to the hotel service changes the transport from HTTPS to HTTP. The service monitor has an interaction authorization algorithm enabled that enforces the communication to use secure protocols (Eg. HTTPS). As soon as the hotel service (which is under attack) attempts to invoke the payment gateway over HTTP, the service monitor blocks that request and interrupts service operation. Such an interaction authorization algorithm can further take action to notify an administrator. This shows the novelty of using interaction authorization algorithms in capturing an organization’s policies and service level agreements.

5.4 Data Leakage

The travel agent sends a credit card number to all three services - car rental, hotel reservation and airline as part of the request. We developed an interaction authorization algorithm that controls the dissemination of sensitive information such as credit card information. This algorithm controls information flow based on the trust levels of target services. We simulated a scenario where the trust level (client feedback) of the hotel service decreased below a threshold set in the interaction authorization algorithm. This triggered the service monitor to block the interaction and prevent the dissemination of credit card information to untrusted services.

6. Conclusion

Typical SOA consumers issue requests to a front-end service and have no visibility in interactions that happen beyond the front-end service. In addition, the details of the services involved in an end-to-end service invocation chain are usually not shared with the consumers. It is difficult to trust all of the services in a service composition and assume they are behaving as expected.

We designed and developed a monitoring platform for REST services. This platform included instrumentation components to be used with services and a service monitoring service. The monitoring service was designed to be able to plug in trust management and interaction authorization functionality. The trust management algorithms are able to take advantage of feedback provided by caller instrumentation. The interaction authorization algorithms can be based on standardized access control policy and we used XACML policy language in our implementation. We developed a set of performance experiments to evaluate the overhead introduced due to blocking instrumentation and interaction authorization. A set of simulations were carried out to demonstrate the effectiveness of the proposed design. These simulations are made available in the form of a demonstration.

References

- [AB12] M. Azarmi, B. Bhargava, P. Angin, R. Ranchal, N. Ahmed, A. Sinclair, M. Linderman, and L. ben Othmane, “An End-to-End Security Auditing Approach for Service Oriented Architecture,” 31st IEEE Symposium on Reliable Distributed System (SRDS), 2012.
- [BC06] S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn, “vTPM: virtualizing the trusted platform module,” USENIX-SS’06, Berkeley, CA, USA, 2006.
- [BK08] A. Benameur, F. Kadir, and S. Fenet, “XML Rewriting Attacks: Existing Solutions and their Limitations,” IADIS Applied Computing, IADIS Press, Apr. 2008.
- [CB13] A. Can and B. Bhargava, "SORT: A self- organizing trust model for peer-to-peer systems,” IEEE Trans. Dependable Secure Comput., 10:14- 27, 2013.
- [VE06] J. Viega and J. Epstein, “Why applying standards to Web services is not enough,” IEEE Security & Privacy, 4(4):25-31, 2006.

- [WB14] WSO2 Inc., “WSO2 Balana XACML Implementation,” github.com/wso2/balana