A Framework for Developing Secure Application in Service – Oriented Architecture

Fariba Roozbeh
Department of computer, Arak Branch, Islamic Azad University,
Arak, Iran
f_roozbeh10@yahoo.com
**Corresponding author: Fariba roozbeh*

Abstract

Service Oriented Architecture (SOA) is one of the most popular concepts to implement different systems. However it faces many challenges in terms of security. As a result, a number of standard and frame works are formed as supporters. The main purpose of this survey is to create a model for a secure Service-oriented Architecture (SOA) based on a formal model specified in the Alloy modeling language.
The proposed model is based on the basic SOA as well as CIA and include secure identities, secure interaction, secure publish and secure discover. To validate that our model is secure, we created an Alloy model for security. We create predicates that model our security definitions and the obstacles which violate these security definitions. Then we use each security definition against the obstacle that violates it to define secure elements in our model.

Keywords: *SOA, Confidentiality Integrity Availability (CIA), authentication, authorization, non-repudiation, alloy*

1. Introduction

Though not a novel concept and emerging in 1990s, service-oriented architecture (SOA) appears with new ability in performing and realizing through related equipments and protocols. This architecture includes an approach to design and implement the distributed systems in which system function is utilized as a service by users and other services. Some reasons appear in welcoming this architecture including: reducing the production costs, protecting the software due to reusability, and possibility to facile system development and upgrade.

On the other hand one must note that using this architecture necessitates ensuring the security requirements, for the unsafe technology results from inefficiency and non-operational function. Consequently, the notion of security is of particular importance in this architecture. Thus, despite the advantages of this architecture in terms of the usage, efficient security models and frameworks are included among necessary terms, not to say enough. These models also must be checked in accuracy. In this regard, there are numerous methods one of which is formal method.

This research aims at presenting a service-oriented architecture model which regards the security requirements. In this paper, we have presented a security structure, according to the features of service-oriented architecture and its basic structure, and considering the basic security principles and other security requirements for service-oriented architecture. Then, making use of Alloy analyst, we studied the mentioned structure from the security perspective.

1.1 Service-oriented Architecture

"Service-oriented architecture", as a term, represents a model in which automation logic is broken down into smaller separate units of logic which can be distributed separately[2] .

the basic structure of service-oriented architecture includes three elements: service provider, service requestor, and service registry, and three standards including web service definition language (WSDL), simple object access protocol (SOAP) and universal description discovery and integration (UDDI) makes it complete.

The illustrated structure in Fig. 1 for the general operation of service-oriented architecture, is largely accepted by references: [2]

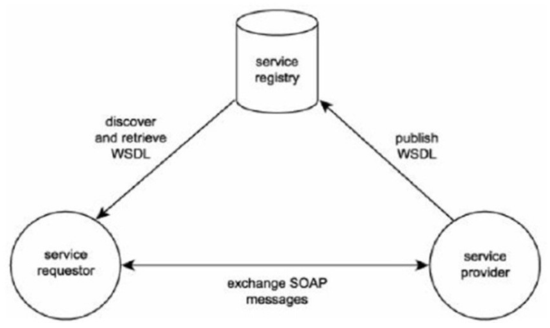

Fig. 1. Basic structure of service orientation [2]

Service provider: A service provider, usually an organization, creates and develops services. A service provider defines the service implementing, service description and business support for a particular service [5] .

Service requester: The service requester uses service and makes interactions to it. Service requester can utilize service in order to combine the operational programs through combination of available services. This element uses service registry in finding the service and is connected to service directly. Service requester may be an individual or another service [6] .

Service registry: Service registry includes a set of available services. Service registry spreads the available data in terms of its services in service registry in which the service requester can find the data about the available services[7] .

WSDL: it defines a service, and this definition conveys a couple of service aspects: service signature and data about developing and submitting the details. This data is described by XML (extensible Markup Language), a language, apart from platform, for data communication [8] .

SOAP: It presents a definition that, according to XML, can be used for exchanging data among existences in a distributed non-central context. This signal includes a header and a body [8] .

UDDI: Medium programs that publish and recognize web services and include a registry in which the service providers publish their service in order that others can recognize them. This technology arranges the services and, after presenting a description, allocates the resulting data in a central store [9] .

1.2 Security and the Service-Oriented Architecture

Security requirements for architecture and automation solutions are not novel in the world of information technology. Consequently, service-oriented operational programs need to be equipped in order to manage many traditional security requirements for protecting the data and ensuring of authorized data availability. The following includes the relation between service-oriented architecture and security principles through CIA triad and WS-security framework.

1.2.1 CIA Triad

In conceptual field, the data security is founded upon three primary principles: confidentiality, integrity and availability. These security principles make a security triad called CIA triad. Fig.2 illustrates each one of these security principles along with available technologies in its accomplishment from the service-oriented architecture.

Service-oriented architecture consists of a set of requestors, providers, services and data.

Confidentiality

By confidentiality, in data security, we mean "providing mechanisms for protecting inputs and data and private information from unauthorized existences". Unauthorized availability of the private information has Destructive consequences not only in national security programs, but also in industry and trade market [10] .

In service-oriented architecture, confidentiality takes place through a couple of mechanisms: access control and encryption. Access control guarantees that a valid

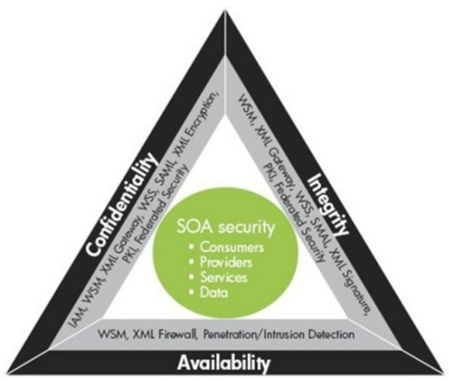
Confidentiality
IAM, WSM, XML Gateway, WSS, SAML, XML Encryption, PKI, Federated Security
Integrity
WSM, XML Gateway, WSS, SMAL, XML Signature, PKI, Federated Security
SOA security
• Consumers
• Providers
• Services
• Data
WSM, XML Firewall, Penetration/Intrusion Detection
Availability

Fig.2. Service oriented architecture and CIA[3]

existence (either a user or an operational program) has access to an entity or a service. Encryption means inserting a mathematical algorithm key to a clear context in order to create an unreadable or cipher text [3] .

Integrity

Another basic principle of security in service-oriented architecture is integrity, which has some definitions with the same meaning:

- Integrity is a purpose during whose accomplishment no data or input can change, or if they are clearly authorized to change[11] .
- Integrity guarantees that the content of the signal, from moving from source to delivering to the recipient in the destination, has not changed [2] .

In general, it is perceived from integrity that a protected signal is regarded as a unified unit and a

process cannot make any change in it partly or completely.
Integrity is discussed from two viewpoints: data integrity and the integrity of origin [11] .
The data integrity guarantees that the data is not under risk, and consequently, it is reliable in a period. The integrity of origin, on the other hand, guarantees that the information about recipient is valid. Both of these are implemented through equal encryption which is the very digital signature [11] .

Availability
As stated formerly, data availability is one of the security principles. When authorized users cannot reach the sources, there is no need for principles such as confidentiality and integrity. Thus, availability is of the same importance as confidentiality and integrity.
Availability ensures us that the users easily reach to the authorized data [10]. In addition to availability as an important aspect of reliability, it also guarantees an existing source. In terms of security, availability means undeniability. Perhaps, one makes use of a source, reaches the data or call a service under particular Conditions; such usages must be undeniable[11] .

1.2.2 WS-security
The WS-security is regarded as the main component of service-oriented solutions. Security operations can be located on the data exchange in layered form to protect the data content of the recipient [2]. WS-Security framework and its descriptions providing the primary QOS (Quality of service) requirements, enables the organizations to:

- Use service-oriented solutions for the process of the private and particular inputs.
- Limit the services availability if necessary.

As it is illustrated in Fig.3 the WS-Security framework uses WS-Policy framework.

1.3 Alloy
Alloy is referred both to a language and a tool; it was created by Daniel Jackson and the Software Development group in MIT University. This language conveys a modeling based on first-order logic used for defining limitations and complex behaviors.[12] The idea of Alloy is to provide a simple and partly automatic approach for software developers to write and test the official features of the software design. The Alloy system includes these three elements [12]:

- Alloy Logic, a combination of relative algebra and predicate logic, determines the combination method for the relationship between various primary inputs in Alloy and the value of the statement result.
- Alloy language used for expressing the specifications according to Alloy logic, defines the key terms and Alloy structural descriptions.
- Alloy analysis that creates model samples to confirm the consistently of a description or to violate the assertions.

Signature
A signature represents set of atoms. Atoms are the primary existing entities or the very basic elements having three following specifications:
Indivisible, Immutable, uninterpreted
1| **sig** Person {}
Creates a new group called 'Person'; if put key term 'Abstract' before 'Person', it means that the 'Person' group has no element except those belonging to its subdivisions. In order to create limitation in the field of atoms in a group, Alloy uses key words such as 'lone', 'one', 'some', 'no'.

Alloy Operators
Alloy has some operators of which the most important we introduce here:
1| a + b, a - b, a & b *// union, difference and intersection of a and b*
2| ~e *// e transposed*
3| ^e *// transitive closure of e*
4 |*e *// reflexive-transitive closure of e*
5|a.b *// (relational) join of a and b*
6| a -> b *//product of a and b*
7| a **in** b *// true if a is a subset of b*

Expressions
Alloy logic supports three styles of expression writing

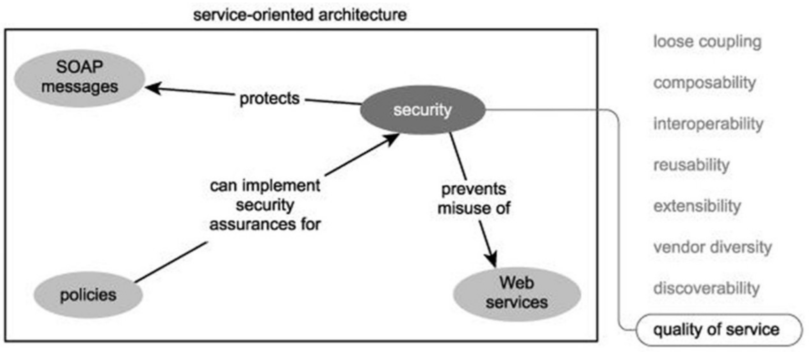

Fig.3 .Security in service-oriented architecture **[2]** including "Predicate Calculus Style", "Navigation Expression Style", and "Rational Calculus Style". These styles can be combined if necessary.

Facts, functions and predicates

There is no high level expression in Alloy. Each expression is contained in a block, enclosed in braces. A fact block holds expressions which form a constraint on the model, as they must always hold (example 1, lines 16-19). A predicate defines a reusable limit (example 1, lines 33-35). A function defines a reusable expression (example 1, lines 28-31). Predicates and functions can have either no or numerous arguments. Functions can return a result.

A function or predicate which can be applied to a single value can always be applied to multiple values as well.For instance, Grandpas function (example 1, lines 28-31) can be applied on one or more 'Person' and return their results.

Modules

Alloy specifications are stored in text files with the file extension .als. Each file is called a module and it can enter the elements of a module using the open statement. If *foo.asl* includes

```
sig ThisIsFoo {}
```

Then bar.asl can use the following signature:

```
open foo
sig ThisIsBar extends ThisIsFoo {}
```

If names become ambiguous, e.g., because **bar.als** contains a signature with the same name as one in **foo.als**, then the module name can be prepended to the signature name.

The module can also be renamed dynamically by using the as keyword:

```
open foo as f          // foo is now known as f
sig ThisIsFoo {}   // signature with an ambiguous name
sig ThisIsBar extends f/ThisIsFoo {}
// extends the imported signature
```

Imported modules are per default expected in the same directory.

Analysis using the Alloy Model Finder

There are two kinds of tests that the analyzer can run. For a simulation, the analyzer tries to find a model which respects all the constraints given, called an "example". In the checking mode, the analyzer tries to find a "counterexample" which violates an assertion. For both tests, an appropriate command, **run** and **check** respectively, must be contained in the specification. It is followed by the name of a predicate or assertion which is to be tested and an upper limit for the number of atoms for each signature. By default, this limit applies to all signatures, but exceptions can be defined: The scope for a specific signature may be set to a higher or lower value, or the number of atoms can be set to a fixed value by using the exactly keyword (example 1, lines 38 & 41). If no limit is defined for an atom, as default, Alloy will apply the test for number 'three'.

To find an example or counterexample, the Alloy Analyzer translates the specification into the input to a SAT solver[13] . Once the solver has found a satisfying assignment, the Alloy software tool visualizes the result. The "Evaluator" window allows the live evaluation of expressions in the context of the solution.

Example Specification and Analysis

The following Alloy specification (example 1, grandpa.asl) formalizes a simple example. It is a module header which determines its name. The complete name of module equals its path and is stored in system file. Our example Module is stored in a file called 'language/grandpa.asl'. The 'person' group solely includes a couple of elements: 'man' and 'woman'. Each man can have either no or one woman having 'wife' relationship to it. Each woman can have either no or one man having 'husband' relationship to it.

A fact is a limit to be satisfied all the time. For instance, the fact beginning in line 16 of above example says that no one can ever be his forebear, and if one is a husband of another, the other is his wife, and vice versa.

'No self father' assertion, in this example, says that no one can ever be his father, and this is always valid, and no counter example is found for it. 'grandpas' function primarily determines that the parents include mother, father or their spouses, and that the grandfather is a man who is the father of the parents. The following example illustrates Alloy specifications.

```
1|  module language/grandpa
2|
3|  abstract sig Person {
4|     father: lone Man
5|           mother: lone Woman
6|  }
7|
8|  sig Man extends Person {
9|           wife: lone Woman
10|          }
11|
12|  sig Woman extends Person {
13|          husband: lone Man
14|          }
15|
16|  fact {
17|       no p: Person | p in p.^(mother+father)
18|          wife = ~husband
19|          }
20|
21|  assert NoSelfFather {
22|          no m: Man | m = m.father
23|          }
24|
25|  // This should not find any counterexample.
26|  check NoSelfFather
27|
fun grandpas [p: Person] : set Person {
        let parent = mother + father +
    father.wife + mother.husband |
        p.parent.parent & Man
        }

pred ownGrandpa [p: Person] {
        p in p.grandpas
        }

// This generates an instance similar to Fig1-3
run ownGrandpa for 4 Person

// This generates an instance similar to Fig 2-3
run Grandpas
```

Example. 1. some specification of Alloy

Instructing the Alloy Analyzer to run all tests on this specification produces the following output in the command log of the software:

```
Executing "Check NoSelfFather" :No counterexample
found. NoSelfFather may be valid.
Executing "Run ownGrandpa for 4 Person" :
ownGrandpa is consistent.
Executing "Run grandpas": grandpas is consistent
```

The primary output of Alloy is a directed graph, which can be seen in the “instance window” of the analyzer software. The atoms, or objects, are shown as nodes, while the relations are the edges. The nodes’ shapes and colors are chosen arbitrarily and convey no special meaning. If there exists more than one atom for a signature, numbers are assigned to the atoms. Note that Alloy does not necessarily produce small or minimal examples. There may, and almost always will be atoms which are not needed to fulfill the constraints.

The first command, Check 'No Self Father' is an assertion; as you see Alloy did not find any counter example; thus the assertion is valid.

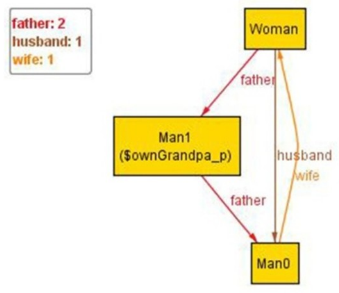

Fig. 6.The result of executing Run 'owngrandpa for 4

The second command is Run 'Own Grandpa for 4 Person' that performs 'own grandpa' for four Person. In Fig. 6 one of the created results for this command, the result of Run 'Own Grandpa for 4 Person', is illustrated.

The last command is Run 'grandpas' that performs the 'grandpas' function. Fig .4 presents one of the results created for this command.

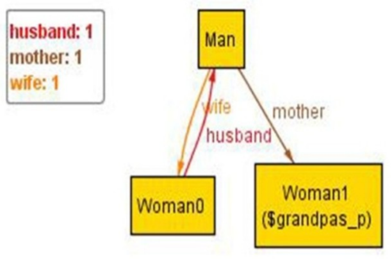

Fig .4. The result of executing Run 'grandpas'

2. Available Models of Service - Oriented Security

We discuss briefly some available models for service-oriented architecture security that provides its requirements.

IBM: Reference Model for service-Oriented Architecture Security
To accomplish the goals and security requirements in service-oriented architecture, IBM has also presented a logical architecture illustrated in Fig. 5 .This architecture can be defined in three abstract levels: Business security services, IT security services, and security policy management. Also, there is a security enabler for presenting security functions to IT security services[4] .

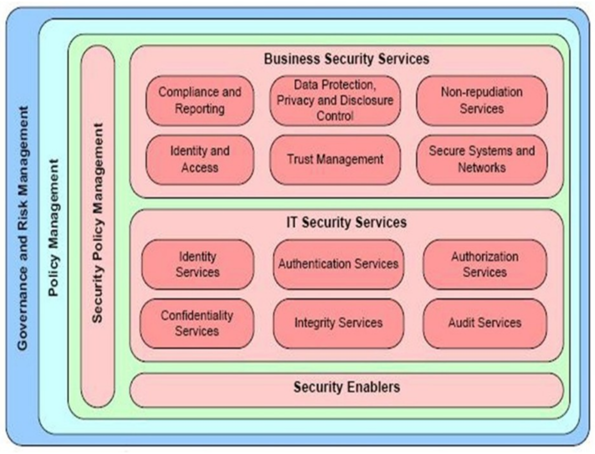

Fig. 5. IBM Model for service-Oriented Architecture Security [4]

NSTISSI: A General Model for Data System Security
CNSS (Committee on National Security Systems) represents a model for data systems that at the same time functions as a tool for system evaluation and development. The model is unique in that it stands independent of technology[1] .

As it is illustrated in Fig .7, three dimensions are supposed for this model that addresses all security requirements of a data system [1] .

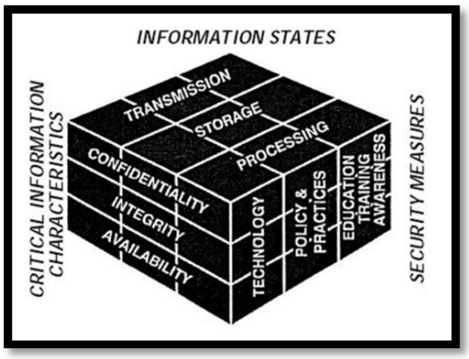

Fig .7.General model for data systems security[1]

3. The Proposed Security Model

Our recommended security model is proposed according to the basic structure of service-oriented architecture and basic security requirements. We also concerned security principles such as authorization, authentication, and non-repudiation. In Fig.8 , the recommended model is illustrated.

Secure identities: we concern a secure identity for all three available elements in service-oriented architecture. Reaching the sources can differ from one identity to another. As an instance, discovering a registered service in the service store must be limited to a particular recipient. A requestor makes use of identity to reach his intended service. Both service provider and service requestor may use their identities for encryption and registration of exchanging messages. In order that a service is available for an identity, primarily, one has to investigate whether the presented identity is valid, and then, one has to find whether the definite identity is authorized to reach the intended service or not. As a result, the two features of authorization and authentication must be considered in the field of the identities in secure service-oriented architecture.

Secure interaction: An interaction between the service-provider and service requestor, to prevent the threats, must be secure. In this regard, security means authentication, authorization, confidentiality, integrity, and non-repudiation [14]. In the interaction between a service provider and a service requestor, both parties must be informed of each other's identities and availability licenses. Thus, in a secure interaction both parties must be studied in terms of authentication and authorization. On the other hand, according to the security triad, the confidentiality, integrity and availability must be preserved as well. In security issues, availability means non-repudiation [11]. That includes two parts: sending non-repudiation and receiving non-repudiation.

Secure publish and discovery: Solely the authorized service provider whose identity is studied can register in service store. The service requestor also must be authenticated before reaching the service store to see whether he has the availability license or not. During the service publish and discovery, the integrity and confidentiality of the store must be protected. If, during this connection, above requirements are observed, one can ensure that the connection is secure.

Error! Reference source not found. includes observed security principles for the recommended model elements.

Table 1. Elements of security principles in secure service – oriented architecture

Security Principles	Secure identity	Secure interaction	Secure publish and discovery
confidentiality		√	√
integrity		√	√
availability			
authentication	√	√	√
authorization	√	√	√
non-repudiation		√	

4. Modeling a Secure Service-Oriented Architecture

Using Alloy language, we created the features of intended models which are to be studied as follows. In modeling, some definitions of [15] are used.

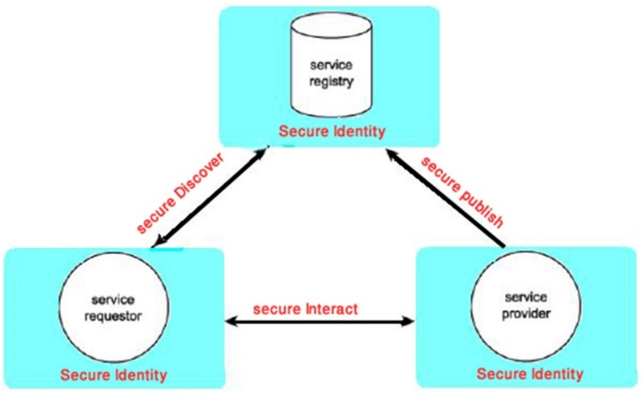

Fig.8 . proposed model for the security of service-oriented architecture

Authentication: if an authentic protected message is received by an identity, the identity will recognize the message and knows that the source was authorized to write that message. Generally, for authentication, the authority to write theory is used for recognizing the sender.

Pred authentication () {
all t: time | **all** m:Protected_Msg | **all** r:Identity |**one** record: Sent | **one** c: CanWrite |(m->t **in** r.knows) => (c.writer = m.lastWriter) **&&**
(record.sender =c.writer) **&&**
(c.msg = m) **&&** (record.msg =m) **&&** (c->t **in** r.knows) **&&** (record->t **in** r.knows)
}

Performing above statement, Fig.10 is formed.

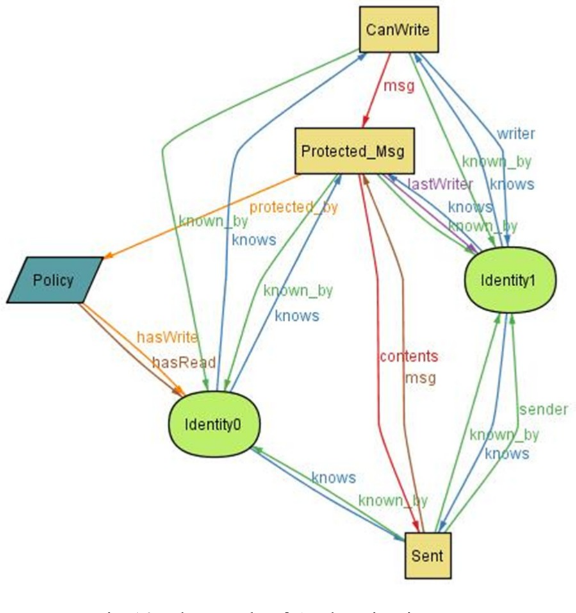

Fig.10 . the result of Authentication

Authorization: it means, if an identity tends to read or write a message, it must be included in the content of authorized identities.

pred authorization(){
all I: Identity |**all** c: CanWrite |
I **in** c.writer =>
I **in** c.msg.protected_by.hasWrite
all I: Identity | **all** R: CanRead|
I **in** R.Reader =>
I **in** R.msg.protected_by.hasRead
}

if an identity is not informed of the message content and is not authorized to reach it, it cannot reach it at any other time.

The result of above definition is illustrated in Fig.9 and Fig.11. According to Fig.9 'identity 1' is an identity able to read 'canread', for it is authorized by policy to read. In Fig.11, 'Identity 1' is an identity that is able to write in protected messages 'CanWrite', for it is authorized to write by policy.

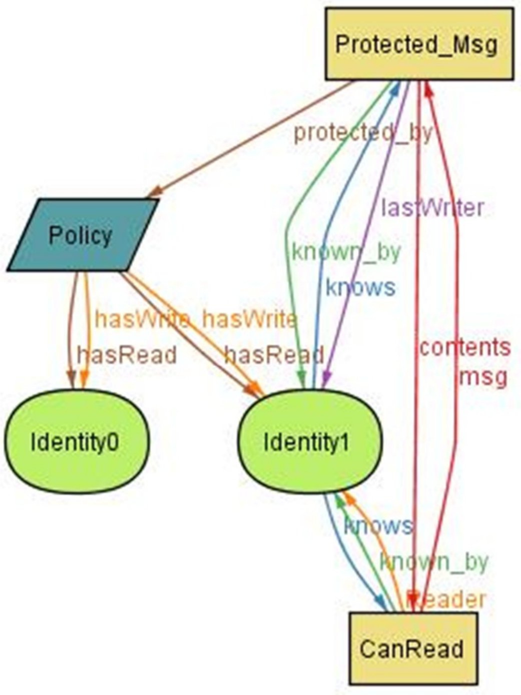

Fig.9.the result of Authorization (can read)

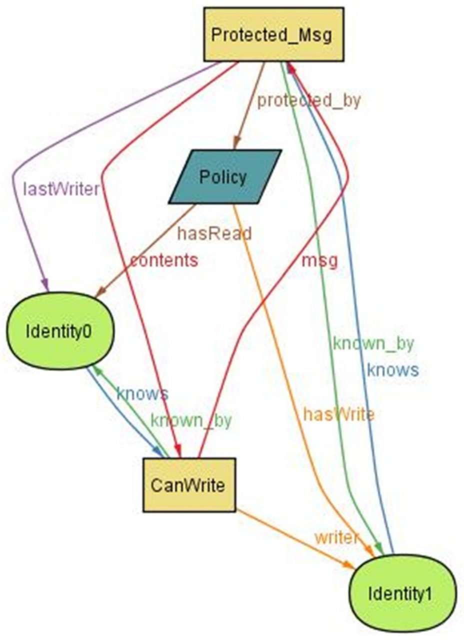

Fig.11. the result of Authorization (can write)

Confidentiality: in general, confidentiality means that the determined readers can reach the message content. That is, er time.

```
pred Confidentiality(){
        all t: time - t0/last[] | all a: Identity | all
        m:Protected_Msg |
        let t' = t0/next[t] |
        ((m->t in a.knows) &&(m.contents->
        t not in a.knows) &&
        (a not in m.protected_by.hasRead)) =>
        (m.contents->t' not in a.knows)
    }
```

The result of performing above definition is illustrated in Fig.12.

According to Fig.12, the components of 'Protected_Msg' are protected via 'Policy0', and this policy cannot reach 'Identity0'. Consequently, although in 'time0', 'Identity0' is infirmed of 'Protected_Msg', it cannot reach the contents in 'time1'

Integrity: it is understood from integrity that a protected message is regarded as a unity, and an unauthorized identity cannot make a change in a whole or a part of the message. Operationally, integrity means that an attempt in changing a message without writing authority destroys the protected message. It is studied in this definition that there is at least one identity for a protected message that is the source of message and is authorized to write.

```
pred Integrity(){
        all m :Msg | some p: Identity |
        m in Protected_Msg =>
        (p in m.protected_by.hasWrite &&
        m.lastWriter = p )
}
```

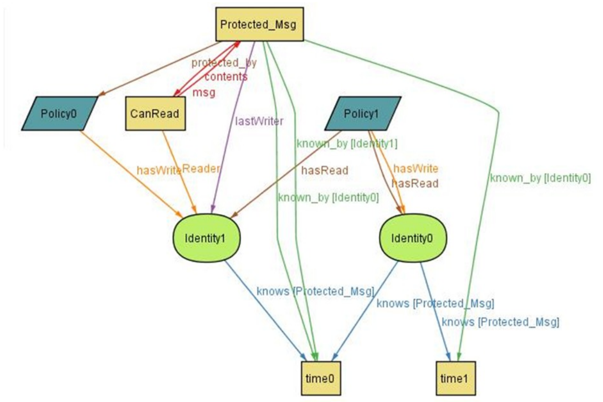

Fig.12. the result of confidentiality

The result of operating the integrity definition is illustrated in Fig.13. The identity that writes on 'protected_Msg' is the source of message and the 'last writer' which was authorized 'hasWrite' to write on it.

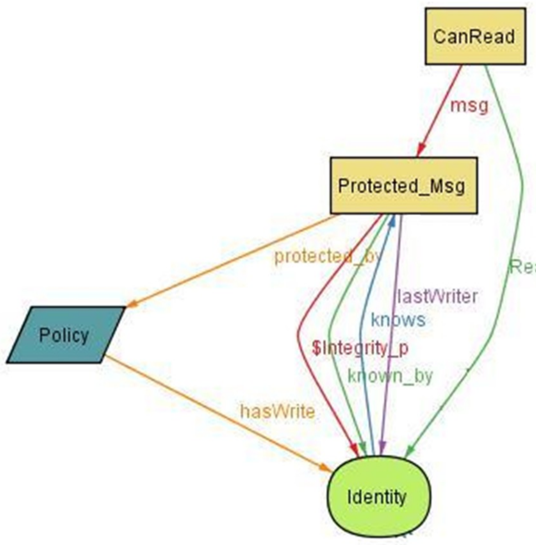

Fig.13.the result of integrity

Non-repudiation send: it refers to the disability of the message source to send the received message. There is at least one identity that knows another identity has sent the message and that is the source of the message.

```
pred NonRepudiationSenderSide(){
        all t: time | all m: Msg |
        all p,q: Identity |all record: Sent |
        (record.sender = q) &&
        (record.msg = m) &&
        (record->t in p.knows) =>
        ( m.lastWriter = q )
        }
```

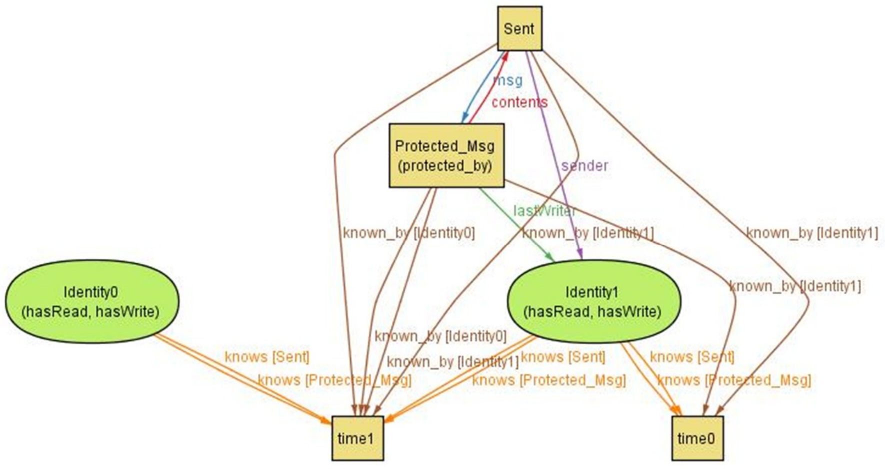

Fig.14.the result of integrity (send)

The result of operating the definition of non-repudiation send is illustrated in Fig.14 in which 'identity0' knows what [sent], and knows that 'identity1' is the 'lastWriter' and it has sent the message (sender).

Non-repudiation receive: it refers to disability of the recipient to receive the sent message. There is at least one identity that knows another identity has received the message and knows it.

```
pred NonRepudiationReceiverSide(){
        all t:time | all m: Msg |
        all p,q: Identity| all record: Recvd |
        (record.recvr= q) &&(record.msg = m) &&
        (record->t in p.knows) =>
        (m->t in q.knows)
          }
```

The result of non-repudiation receive is shown in Fig.16. 'identity0' has received the 'Msg' and 'identity1' knows that 'identity0' has received this message (Knows[received]). 'Identity0' knows 'Msg'.

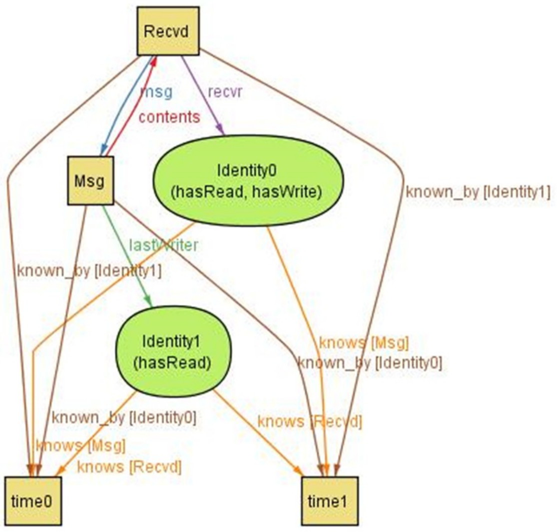

Fig.16. the result of integrity (receive)

5. Evaluating the Proposed Model

To evaluate the proposed model, first we define the threats of every security requirement; then, according to proposed model, we claim that the principles for the proposed model are observed. To do this, we operate the definitions of security principles along with their threats, and obviously it is concluded that it does not accord such systems.

Modeling the Security Threats

Fraud: this threat invalidates the authentication requirement. A special identity sends a protected message whose source is another identity message. Fig.۱۵ shows the created example in Alloy. In this figure, 'Identity0' is the sender of a protected message that 'Identity1' is its resource. In other words, an identity introduces itself instead of the other, and this invalidates the authentication requirement.

```
pred Fraud{
    all t: time | some m: Protected_Msg |
    some r: Sent |   some p: Identity  |
    (r.sender != m.lastWriter) &&
   (r.msg = m) && (m->t in p.knows)
}
```

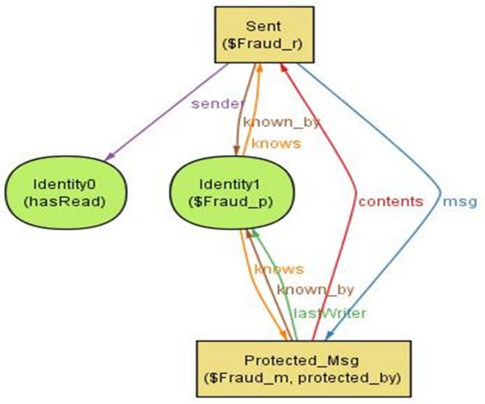

Fig.۱۵ . Fraud

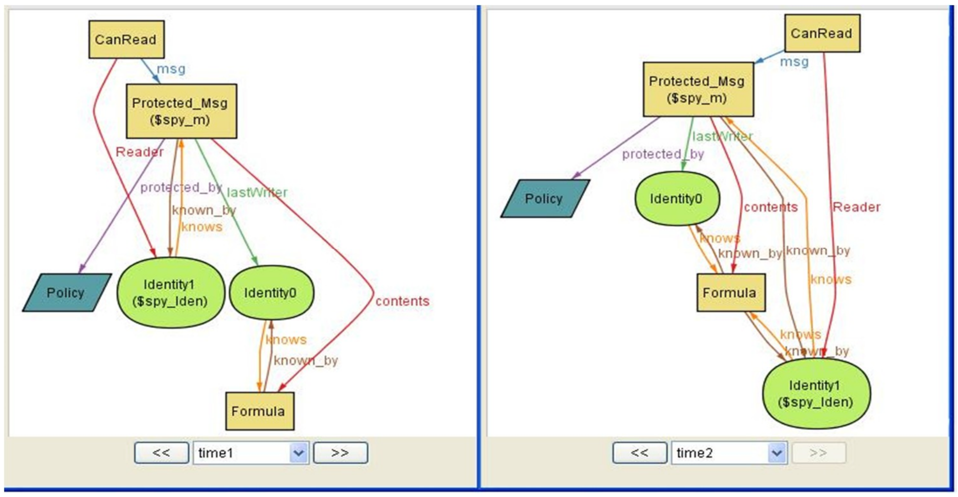

Fig.17. spy

Unauthorized availability: this threat rejects the authorization security requirement. In this case, the identity reads a message without having authority for reading it. It reads the message content, or with no authority, writes on a protected message. Fig.18 shows the created example in Alloy in which 'Identity0', having no authority of 'hasWrite' for writing, has made changes on the protected message. This threat rejects the authorization requirement.

pred Unauthorized{
Some s: CanWrite |s.writer **not in** s.msg.protected_by.hasWrite
Some d: CanRead |d.Reader **not in** d.msg.protected_by.hasRead
}

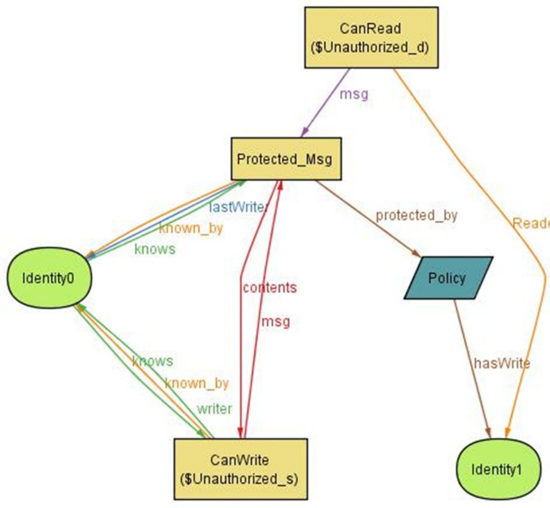

Fig.18.unauthorized availability

Spy: this threat invalidates the security requirement of confidentiality. In this case, some identities are able to get informed of message content without an authority to read.

Pred spy(){
some Iden:Identity | **some** m:Protected_Msg | **some** t: (time – t0/last[]) – t0/prev[t0/last[]] | **let** t' = t0/next[t] | **let** t'' = t0/next[t'] | (Iden **not in** m.protected_by.hasRead) **&&**
(m->t in Iden.knows) **&&** (m.contents-> t **not in** Iden.knows) **&&**
(m.contents-> t'' **in** Iden.knows) }

Fig.17 illustrates the created example in Alloy. According to Fig.17, at 'time0' the 'itdentity1' is informed of 'protected_Msg'; however, it does not know its contents, and is not allowed to reach it. But at 'time1', 'Identity1' can reach the contents of the message. This point invalidates the definition of confidentiality.

Distortion: this threat invalidates the security requirement of integrity. In this case, there are some protected messages whose resource, that is, the identity written on them last time, has not been authorized to write.

pred distortion(){
some m: Protected_Msg | (m.lastWriter **not in** m.protected_by.hasWrite)
}

Fig.20 illustrates the example created in Alloy. It indicates that 'Identity0' is the source of the protected message. While the message is protected through 'policy0' and according to this policy 'Identity1' is authorized to write on the protected message. Accordingly, the definition of integrity is invalidated and the intended message is distorted by 'Identity0'.

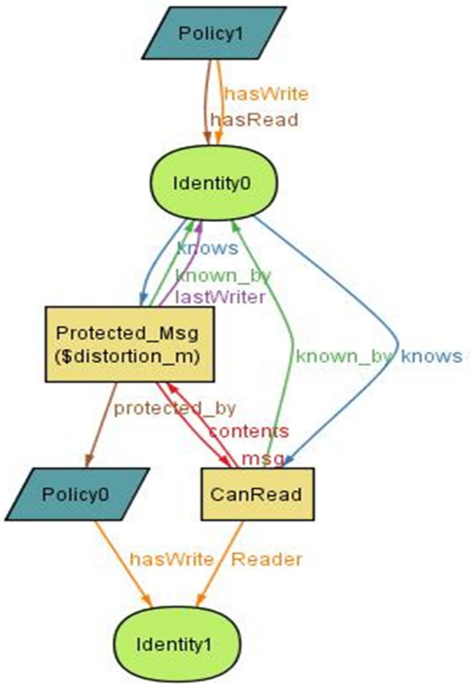

Fig.20. Distortion

Sending denial: this threat invalidates the security requirement of non-repudiation. In this case, an identity sends a particular message, while according to other identities; it is the source of message of another identity.

```
pred DeniableSending(){
    all t:time | one p,q: Identity | one m: Msg | one r:
    Sent |r.sender = q &&
     r.msg = m && r->t in p.knows &&
     q != m.lastWriter
          }
```

Fig.19 illustrates the created example in Alloy. According to this figure, 'Identity0' sends the protected message in a way that others suppose 'Identity1' as the source of the message; that is, 'Identity0' has sent a denied sending. This definition invalidates the undeniable sending.

Receive denial: this threat invalidates the security requirement of undeniable receive. In this case, the identities know that an identity has received a message; but the recipient identity is not aware of the message content; that is, the receiver identity denies receiving the message.

```
pred DeniableReception(){
    all t: time | one p,q: Identity| one m: Msg | one
    record: Recvd |record.recvr = q && record.msg =
    m && record->t in p.knows && m->t not in
    q.knows
          }
```

Fig.21 shows the created example in Alloy. 'Identity1' knows that 'Identity0" has received the message, but 'Identity0' is not aware of the message; that is, there is no relation of 'knows' in terms of 'Identity0' to the message. In general, receiving the message by 'Identity0' is denied.

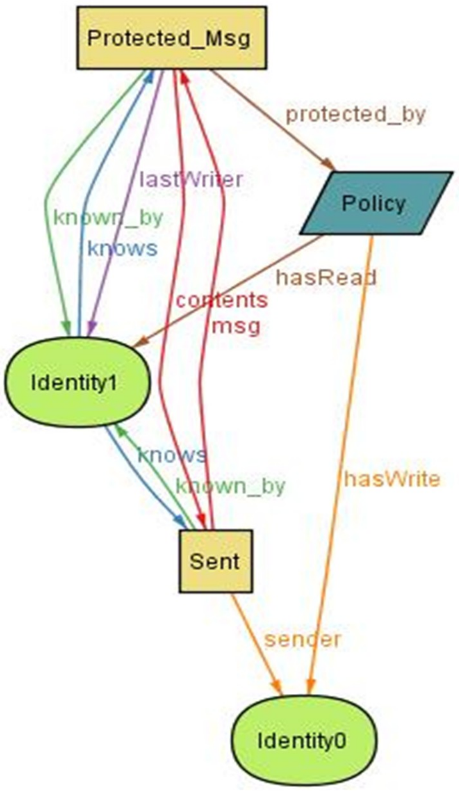

Fig.19 . sending denial

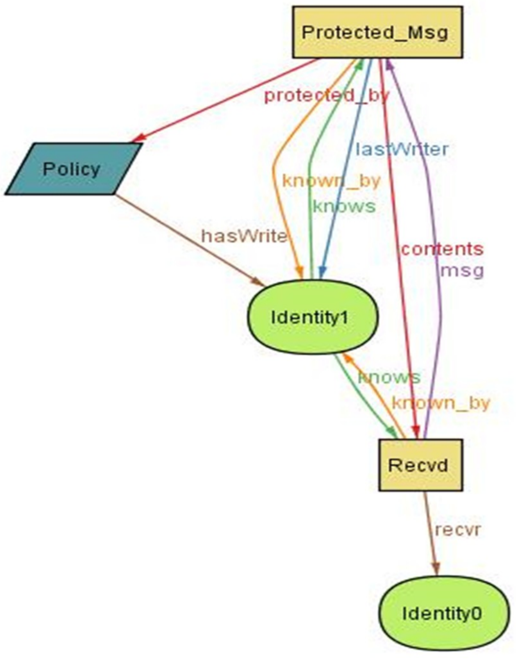

Fig.21. Receive denial

Studying the presented Security Model

Up to this point, the definition of each security principle, along with their threats was presented. Also, the created examples in Alloy were illustrated. Now, according to the proposed model, we assert that the mentioned principles of the intended model are observed. To accomplish this aim, we practice the security principles definitions with their threats, and then show that this is not a compatible system.

As stated above, to have a secure identity needs the practice of authorization and authentication on identities. We assert that an identity is secure; to confirm it, we must show the discordance between the definition of authentication and the threat of unauthorized availability. For our assertion, we make use of the following command:

```
assert secureIdentity {
        !(authentication[]and Fraud[]) and
        !(authorization[]and Unauthorized[] )
        }
```

Then, we study the above assert through this command:

```
check secureIdentity
```

and the result of Alloy analysis will be:

```
Executing "Check secureIdentity":No counterexample found. Assertion may be valid"
```

It concludes the accuracy of our asserion, and that the identity is secure through given specification.

For a secure interaction between service provider and service requestor, we had to observe authentication, authorization, integrity, confidentiality, and non-repudiation. According to the mentioned definitions for these principles and their threats, we use the following code for a secure interaction:

```
assert secureinteract {
   !(authentication[]and Fraud[]) and
   !(authorization[] and Unauthorized[]) and
    !(Confidentiality[] and spy[]) and
    !(Integrity[] and distortion[]) and
   !(NonRepudiationReceiverSide[] and
    DeniableReception[]) and
   !(NonRepudiationSenderSide[] and
   DeniableSending[])
}
```

Then, we study the above assert through this command:

```
check secureinteract
```

and the result of Alloy analysis will be:

```
Executing " check secureinteract":No counterexample found. Assertion may be valid
```

This conclusion proves our assertion as correct and the identity is secure through given specification.

To have a secure publish between service provider and the service registry, and also to have a secure discovery between the service requestor and the service registry, we had to practice the authentication, authorization, confidentiality, and integrity. According to the definitions of the principles and their threats, to have a secure publish and discovery, we make use of the following commands:

```
assert securepublish {
        !(authentication[]and Fraud[])and
        !(authorization[] and Unauthorized[]) and
        !(Confidentiality[] and spy[]) and
        !(Integrity[] and distortion[])
        }

assert securediscover {
        !(authentication[]and Fraud[])and
        !(authorization[] and Unauthorized[])
        !(Confidentiality[] and spy[]) and
        !(Integrity[] and distortion[])
        }
```

Then, we study the above assertion through these commands:

```
check securepublish
check securediscover
```

The following is the conclusion of Alloy analysis, and our assertion is correct and the identity is secure through given specification.

```
Executing " check securepublish":No counterexample found. Assertion may be valid
Executing" check securediscover":No counterexample found. Assertion may be valid
```

6. Conclusion

In this article, first we studied the features and the basic structure of service-oriented architecture; then we investigated the subject and the importance of the security and described a couple of models presented for this architecture. In accordance with using Alloy as a tool for evaluating the particular model, in this article, we described the specification of this tool and language through an example. Using the mentioned notions in terms of service-oriented architecture and security, and based on basic structure and security principles of this model and some other security requirements, a security model was presented for service-oriented architecture. A secure identity was regarded for the three existing elements in this architecture; the authorization and authentication must be applied on this secure identity. In suggested model, a necessity of the secure relation between service provider and service requestor was asserted; and we believe this relation is secure when principles such as authentication, authorization, confidentiality, integrity and non-repudiation are concerned. In order that the service provider can publish his service data in service registry, authentication, authorization, confidentiality, integrity must be preserved in this relationship. Also, for reaching the service requestor to the particular date through service registry, the authentication, authorization, confidentiality, integrity principles must be concerned in their relation.

After presenting a security model including the determined specification, the model must be validated. We did this through Alloy: first, we defined the security principles in Alloy and operated them to ensure the definitions are compatible. Then, we studied the security threats against these principles, and made them models to ensure that they are prototype-able in a non-secure model. Finally, to verify the security of the model, we operated each security definition against the threat violate it, and we got sure that these security definitions do not accord their threats in our model. As a result, our proposed model is secure.

References

[1] McConnell, J. (1994). National Training Standard for Information Systems Security (INFOSEC) Professionals, DTIC Document.

[2] Erl, T. (2005). Service-oriented architecture (SOA): concepts, technology, and design, Prentice Hall Englewood Cliffs.

[3] Tipnis, A., Lomelli, I.(2009).Security: A Major Imperative for a Service-Oriented Architecture – HP SOA Security Model and Security Assessment, HP Viewpoint Paper.

[4] Buecker, A., P. Ashley, et al. (2008). Understanding SOA Security Design and Implementation, IBM Redbooks.

[5] Papazoglou, M. P. (2003). Service-oriented computing: Concepts, characteristics and directions. Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on, IEEE.

[6] Rahaman, M. A., A. Schaad, et al. (2006). Towards secure SOAP message exchange in a SOA. Proceedings of the 3rd ACM workshop on Secure web services, ACM

[7] Papazoglou, M. P. and W.-J. Van Den Heuvel (2007). "Service oriented architectures: approaches, technologies and research issues." The VLDB journal 16(3): 389-415

[8] Haas, H. and A. Brown (2004). "Web services glossary." W3C Working Group Note (11 February 2004).

[9] Newcomer, E. (2002). Understanding Web Services: XML, Wsdl, Soap, and UDDI, Addison-Wesley Professional.

[10] Danielyan, J. C. E. (2005). Sun Certified Security Administrator for Solaris, Dreamtech Press.

[11] Hafner, M. and R. Breu (2009). Security engineering for service-oriented architectures, Springer.

[12] Jackson, D.(2006). Software Abstractions Logic, Language, and Analysis., MIT press.

[13] Jackson, D. (2002). "Alloy: a lightweight object modelling notation." ACM Transactions on Software Engineering and Methodology (TOSEM) 11(2): 256-290.

[14] Nezhad, H. R. M., H. Skogsrud, et al. (2005). "Securing Service-Based Interactions: Issues and Directions." IEEE Distributed Systems Online.

[15] Grisham, P. S., C. L. Chen, et al. (2006). Validation of a Security Model with the Alloy Analyzer, October.