2012 IEEE Eighth World Congress on Services

SecAgreement: Advancing Security Risk Calculations in Cloud Services

Matthew L. Hale Rose Gamble
Tandy School of Computer Science
University of Tulsa
Tulsa, OK, USA
{matt-hale, gamble}@utulsa.edu

***Abstract*—By choosing to use cloud services, organizations seek to reduce costs and maximize efficiency. For mission critical systems that must satisfy security constraints, this push to the cloud introduces risks associated with cloud service providers not implementing organizationally selected security controls or policies. As internal system details are abstracted away as part of the cloud architecture, the organization must rely on contractual obligations embedded in service level agreements (SLAs) to assess service offerings. Current SLAs focus on quality of service metrics and lack the semantics needed to express security constraints that could be used to measure risk. We create a framework, called SecAgreement (SecAg), that extends the current SLA negotiation standard, WS-Agreement, to allow security metrics to be expressed on service description terms and service level objectives. The framework enables cloud service providers to include security in their SLA offerings, increasing the likelihood that their services will be used. We define and exemplify a cloud service matchmaking algorithm to assess and rank SecAg enhanced WS-Agreements by their risk, allowing organizations to quantify risk, identify any policy compliance gaps that might exist, and as a result select the cloud services that best meet their security needs.**

Keywords-cloud; audit; risk; xml; security; service level agreement; quality of security service; web services;

I. INTRODUCTION

Business and government organizations are constantly seeking to improve their bottom line and maximize their economies of scale. Cloud computing provides a number of opportunities towards advancing these goals through increased service delivery efficiency and decreased IT costs. The so-called "utility computing" [1] philosophy of cloud services is a kind of "pay as you go for what you use" structure that allows organizations to eliminate, or reduce, the capital investment costs (such as power, hardware maintenance, software updates, and bandwidth) necessary to meet peak performance constraints while still guaranteeing a high quality of service. These factors have resulted in an explosive growth of cloud services as organizations outsource data storage and computations to the cloud.

However, for mission critical systems that must satisfy confidentiality and/or integrity constraints or meet governance policies such as HIPPA[2], the NIST-800-53 [3] or the DoDI-8500 [4], this push to the cloud introduces risks of possible security violations. Since internal system details are not visible or are abstracted away as part of the cloud architecture, organizations must rely on contractual obligations embedded in service level agreements (SLAs) to discover service security offerings and assess the risk incurred by using a particular Cloud Service Provider (CSP).

Current service level agreements specifications, such as WS-Agreement [5] and WSLA [6] focus on quality of service (QoS) metrics and lack the semantics needed to express security service levels [7]. This lack of semantics inhibits service interoperability, and makes standard examination of security and risk across various CSPs difficult or impossible. In this paper, we define a framework, called SecAgreement (SecAg), that builds on the existing WS-Agreement XML schema allowing security constraints to be expressed over the service description terms (SDTs) and the service level objectives (SLOs) of the SLA. Our extensions enable CSPs to express and advertise their security offerings, increasing the likelihood an organization will use their service, and allow objective security comparisons to be made over a set of services, facilitating organizational risk assessment.

We specify a matchmaking algorithm that, given a set of requested service terms, calculates and ranks SecAg enhanced SLAs by their risk. The result allows organizations to identify where service terms match their requests and where policy or compliance gaps might exist, and ultimately select the cloud services that best meet their security needs. Our matchmaking algorithm is generally applicable to any service matchmaking scenario involving security. It allows for both organizational consumer-to-CSP and CSP-to-CSP matchmaking for choosing security compatible 3rd party services or federated cloud scenarios. The rest of the paper is organized as follows: Section II reviews relevant background, Section III specifies the SecAg extensions, Section IV discusses the matchmaking algorithm, and Section V concludes the paper.

II. BACKGROUND

When domain specific security controls (e.g. HIPPA [2], the NIST-800-53 [3] or the DoDI-8500 [4]) are applied to an information system, they form a *secure enclave* around security critical components, communication, and data. Since, data, communication, and third party components existing outside of the enclave are not designed or protected according to the polices and controls, they are generally considered insecure [8]. Herein lies the problem for organizations wishing to utilize the cloud: how can the enclave be extended into the cloud? We use WS-Agreement as a basis for extending the enclave into the cloud, Ludwig et

978-0-7695-4756-5/12 $26.00 © 2012 IEEE
DOI 10.1109/SERVICES.2012.31

al., proposed an SLA language that parallels what we see in WS-Agreement today [9]. They point to a need for a *common ontology* [9], outside the scope of this paper, that expresses the *standard terms* used in service descriptions.

A. Towards Cloud Security SLAs

Much cloud research has focused on understanding on how best to manage and monitor SLAs to detect when SLOs are violated [10-12]. Clark et al. [11] incorporate their SLA management framework into WS-Agreement specified SLAs by modifying the underlying agreement schema. They insert additional elements into SDTs (see Section III.B) to allow services to be monitored by trusted third parties.

Henning [13] first examined SLAs in the context of quantifiable security metrics. He identified four types of security measurements that can be quantified in SLAs, namely *performance criteria* (which measures how materials such as audit logs were delivered to the customer), *temporal criteria* (which details how objectives must be met in time, such as responsiveness to attack), *functional criteria* (which defines how any network, application or user adjustments must be handled), and *process criteria* (which quantifies task performance, such as back-ups, monitoring, or alerting). Irvine and Levin coined the term "Quality of Security Service" [14], or QoSS, as a parallel to QoS for security. Lindskog examines the need for "tunable security services" [15], defining four dimensions that characterize a tunable security service [15]: *type of protection service* (security goal, e.g. confidentiality), *protection level* (parameter-based, percentage-based or mechanism-based), *protection level specification* (policy), and *adaptiveness* (ability of a service to change protection levels at run-time).

Chaves et al. provides an overview of the existing state of security SLAs and SLA management in the cloud [16]. From their survey it is clear that security SLAs, especially in the cloud, have many open questions including: the need for better QoSS expression through improved security SLA specifications, the need for better security SLA metrics, and the need for better frameworks for both monitoring and management of security SLAs [16].

Addressing the last point, Bernsmed, et al. develop a framework for secure SLA management of federated cloud services [7]. Their framework includes a security SLA lifecycle consisting of 6 phases [7], *publishing*, *negotiation*, *commitment*, *provisioning*, *monitoring*, and *termination*. Of interest is their three piece negotiation architecture that consists of a customer with a set of security requirements, an initial CSP with a security offer, and third party CSPs that may serve to meet one or more of the terms of the security offer [7]. Because it is a high-level process, it lacks details regarding the specification structure of the SLA or the matchmaking process for selecting compatible CSPs that meet the security requirements.

B. Service Matchmaking

The need for better service interoperability has driven matchmaking research for many years. Recent efforts have focused on improved algorithms and methods for cloud federation [17], web service ranking of cloud platforms for meeting vertical compatibility needs [18], and WSDL analysis for comparing service descriptions [19]. For instance, Celesti et al. [17] propose a matchmaking algorithm that matches clouds based on their WSDL specified policies. Each policy is mapped to an XML *feature* which represents some aspect of the cloud in question. The algorithm then calculates a Euclidean distance from the home cloud (the cloud seeking federation) and the foreign cloud (the cloud being examined for compatible federation properties).

III. SecAgreement: Enhancing WS-Agreement for Expressing Security Concerns

Since cloud service implementation is largely, or entirely, transparent to the consumer, organizations must rely on SLA terms to satisfy internal policy constraints. Fig. 1 depicts a scenario where an organizational consumer contracts with a CSP to form a virtual security enclave. This CSP may then contract with N other vertically compatible 3rd party CSPs to satisfy the security needs of the organization. We hypothesize that the risk of not satisfying the security policy constraints, increases as you get further away from the service consumer. Existing SLA standards, such as WS-Agreement, do not support the analysis needed to quantify security-related risks. In the following sections we examine WS-Agreement and provide step-wise refinements that allow for the security concepts and risks to be better understood.

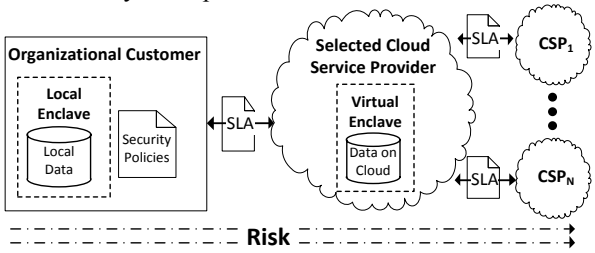

Figure 1: The Security Enclave Risk Perspective

A. Anatomy of WS-Agreement

The WS-Agreement standards specification defines a language and protocol for "advertising the capabilities of service providers and creating agreements based on creational offers" [5]. It provides a standard format, called *wsag:Template*, and a terminological basis for specifying *agreement terms*, or terms that define the service offering(s). Terms come in two specific varieties *Service Description Terms* (SDTs), which define the offered (or requested) operations that the service can perform, and *Guarantee Terms* (GTs) which define quality assurances over the described service terms. The final piece of the agreement is an optional set of *Creation constraints*, which define alternate service levels that the provider will likely accept instead of the levels listed in the guarantee terms. Fig. 2 shows the structure of a WS-Agreement template.

Our SecAg schema[1], focuses on extending the *Terms* section to accommodate the expression of security

[1] Full XML schema available in: M. Hale and R. Gamble, "SecAg XML Schema," Univ. of Tulsa, 2012. Report No.: SEAT-UTULSA-032012, www.seat.utulsa.edu/secag

constraints. We examine the two types of terms further as we develop our security extensions in the following sections.

Figure 2: The structure of a WS-Agreement Template

B. Service Description terms

SDTs define the particular operations exposed as part of the service offering. The WS-Agreement specification of SDTs consist of two WSDL attributes, *wsag:Name*, and *wsag:ServiceName* and one WSDL field of type *xs:any*. The Name field is a mandatory attribute that uniquely identifies the SDT and allows it to be referenced anywhere in the template using a structural reference such as XPATH [5]. The ServiceName is also mandatory and identifies a particular service across multiple SDTs. This allows agreements to define a "packaged service" [5] where each SDT identifies a specific component of the service identified by the ServiceName field. Finally the xs:any field allows for an arbitrary XML specification of the SDT. Fig. 3 shows the structure of these elements as specified in the WS-Agreement standard and an example (that we develop further in the next section and in Fig. 4).

```
<wsag:ServiceDescriptionTerm                1
   wsag:Name="xs:string"                    2
   wsag:ServiceName="xs:string">            3
   <xs:any> … </xs:any>                     4
</wsag:ServiceDescriptionTerm>              5
                                            6
<wsag:ServiceDescriptionTerm                7
   wsag:Name="ReadComponent"                8
   wsag:ServiceName="StorageService01">     9
   <s><func>read files<func>               10
   <func>limit access using                11
   <ac>//@adminonlypolicy</ac></func>      12
   <func>audit events <ae>"read"</ae>      13
   <ae>"readfail"</ae>                     14
   </func></s>                             15
</wsag:ServiceDescriptionTerm>              16
```

Figure 3: Structure and example of WS-Agreement SDT

The issue we take with this specification structure is that it leaves security out of the standard. Thus, all security constraints are pushed to the domain specific <xs:any> term (line 4) used to define the service offering. The allowance of a hodgepodge of terminology prevents the necessary standard expression of security constraints. In addition, the lack of standardized expression inhibits service interoperability, and makes examining security and risk across SDTs from various Cloud Service Providers (CSPs) much more difficult.

C. SecAg Enhanced Service Description Terms

To address the need for security properties and assessment using WS-Agreement, we advocate for an enhanced SDT structure that incorporates a key XML attribute, *secag:ComponentType,* and two additional XML fields, *secag:AuditableEvents* and *secag:AccessControlList*. Each of these elements are defined as (≜) follows:

- secag:ComponentType ≜ optional field of type *xs:string* that identifies a standardized component term with respect to security functionality (such as "auditor," "auditable" or "access control")
- secag:AuditableEvents ≜ optional field that contains a set of secag:AuditableEvent fields, each of which have a standard name (such as "read") and an arbitrary <xs:any> field identifying the domain specific implementation of the auditable event that the service component generates as part of the SDT
- secag:AccessControlList ≜ optional field that contains a set of secag:AccessControl fields, each of which consists of a standard name and an <xs:any> field that references the piece of the WSDL defining applicable access control policy elements

To illustrate the need for these extensions, we return to the example in Fig. 3 where the service offering includes an SDT describing a file read component. Assume "read" and "readfail" are auditable events recorded by the auditor resource (lines 13-14) and access to the component is governed by the access control policy called "adminonlypolicy" (line 12). Without the SecAg SDT extensions, all of this functionality is mixed together and would likely be stated differently for other services.

The SecAg extensions allow us to do several things to mitigate this. First, using the secag:ComponentType field, we can abstractly state the type of security constraints on the component (in this case it is "auditable"). Second, the secag:AuditableEvents and secag:AccessControlList elements separate the component functionality from the security functionality. Finally, just the presence of the two terms abstractly marks the component as producing auditable events and being governed by a set of access controls. Fig. 4 shows the security enhanced structure of an SDT using the extensions (lines 1-12) and the transformed example (line 14-28). Note that, throughout the paper, we adhere to traditional XML specification notation, where '+' means "one or more," '?' means "at most one or none," '*' means "zero or more," and '|' means "or".

Thus, by abstracting security concerns away from the domain specific level and up to the standards level we

empower CSPs to represent operation-based security constraints in an interoperable way that can be compared and ranked according to risk, such as by our algorithm presented in Section IV.

```
<secag:ServiceDescriptionTerm                1
   wsag:Name="xs:string"                     2
   wsag:ServiceName="xs:string"              3
   secag:ComponentType="xs:string">          4
   <xs:any> … </xs:any>                      5
   <secag:AuditableEvents>                   6
      <secag:AuditableEvent/> +              7
   </secag:AuditableEvents>?                 8
   <secag:AccessControlList>                 9
     <secag:AccessControl/> +               10
   </secag:AccessControlList>?              11
</secag:ServiceDescriptionTerm>             12
                                            13
<secag:ServiceDescriptionTerm               14
   wsag:Name=" ReadComponent"               15
   wsag:ServiceName="StorageService01"      16
   secag:ComponentType="auditable">         17
   <s><func>read files</func></s>           18
   <secag:AuditableEvents>                  19
    <secag:AuditableEvent name="read"/>     20
    <secag:AuditableEvent name="readfail"/> 21
   </secag:AuditableEvents>                 22
   <secag:AccessControlList>                23
     <secag:AccessControl name="adminOnly">24
       <ac>//@adminonlypolicy</ac>          25
     </secag:AccessControl>                 26
   </secag:AccessControlList>               27
</secag:ServiceDescriptionTerm>             28
```

Figure 4: Structure and example of SecAgreement SDT

D. Service Properties

Before we can define guarantee terms we must first look at another element of the wsag:Terms section called *wsag:ServiceProperties*. Service properties are used to define the "measurable and exposed properties" [5] of a service, such as response time or throughput, and they form the basis for defining SLOs. A service property consists of two attributes we have previously examined, wsag:Name and wsag:ServiceName, and one new element called *wsag:VariableSet*. A VariableSet is simply a container for a set of *wsag:Variable* elements. Each variable consists of two attributes, a wsag:Name and a *wsag:Metric* and one field *wsag:Location*. A metric is a reference to the datatype of a variable and is specified using a URI (such as xs:string). The Location element is a structural reference to some field in the service terms that may be embedded in the domain specific service descriptions. Essentially the role of a variable is to provide high level handles on low level details for later use in the Guarantee Terms section. Fig. 5 shows the structure of the wsag:ServiceProperties element (lines 1-10) and an example (lines 12-25) that parallels the earlier example from Fig. 3.

In the example, there are two variables, a set called "AuditLog" and a float called "LogCapacity," that together define the "AuditProperties" that apply to the storage service from our previous examples. The variables point to a location called "//SDT/AuditComp" [not shown] that houses the audit system for the service. By using wsag:Variable and wsag:ServiceProperty, the internal data housed at the locations can be exposed, as part of the Agreement, for later use in defining Guarantee Terms and SLOs.

```
<wsag:ServiceProperties                      1
   wsag:Name="xs:string"                     2
   wsag:ServiceName="xs:string">             3
   <wsag:VariableSet>                        4
      <wsag:Variable wsag:Name="xs:string"   5
        wsag:Metric="xs:URI">                6
<wsag:Location>xs:anyType</wsag:Location>    7
      </wsag:Variable> +                     8
   </wsag:VariableSet>                       9
</wsag:ServiceProperties>                   10
                                            11
<wsag:ServiceProperties                     12
   wsag:Name="AuditProperties"              13
   wsag:ServiceName="StorageService01">     14
   <wsag:VariableSet>                       15
     <wsag:Variable wsag:Name="AuditLog"    16
        wsag:Metric="secag:Set">            17
       <wsag:Location>//SDT/AuditComp/Log   18
        </wsag:Location></wsag:Variable>    19
     <wsag:Variable wsag:Name="LogCapacity" 20
        wsag:Metric="xs:float">             21
       <wsag:Location>//SDT/AuditComp/Cap   22
        </wsag:Location></wsag:Variable>    23
   </wsag:VariableSet>                      24
</wsag:ServiceProperties>                   25
```

Figure 5: Structure & Example of wsag:ServiceProperties

E. SecAg Security Service Properties

Our version of the service property, an element called *secag:ServiceSecurityProperties*, contains everything the basic form has (Fig. 5 lines 1-10) with one additional attribute. This attribute, called *secag:SecurityDomain*, is attached to the main element (Fig. 5 lines 1-3) and is of type *xs:string*. This very slight change allows CSPs to group service security properties according to a set of common security terms defined in a domain specific ontology framework such as OWL-S as suggested in [19].

F. Guarantee Terms

As you might guess, one the most important features of service level agreements are the service levels. In WS-Agreement, service levels are characterized by a set of Guarantee Terms which define quality assurances (in the form of SLOs) over the SDTs using the defined service properties. Up to this point we have developed the SecAg enhanced versions of WS-Agreement SDTs and service properties. Next, we examine guarantee terms.

A *wsag:GuaranteeTerm* contains two attributes, a wsag:Name (described previously), and a new attribute called *wsag:Obligated* which is one of three values (provider, consumer, or third-party). It consists of four main elements: *wsag:ServiceScope*, *wsag:QualifyingCondition*,

wsag:ServiceLevelObjective, and *wsag:BusinessValueList*. Each of these, and their subfields, are defined as follows:

- wsag:ServiceScope ≙ element that describes to which agreement elements the guarantee term applies (could be one or more services, one or more SDTs or one or more sub fields of an SDT)
- wsag:QualifyingCondition ≙ optional element that may be used to express a precondition that must hold for SLO evaluation
- wsag:ServiceLevelObjective ≙ element containing an assertion (expressed using a domain specific logic) over some subset of service properties and/or constant factors that expresses the condition that must be met to satisfy the guarantee term
- wsag:BusinessValueList ≙ element that contains a list of business values (in the form of penalties and rewards) that are associated with satisfying or failing to satisfy a service level objective.

```
<wsag:GuaranteeTerm Name="xs:string"                    1
  Obligated="wsag:ServiceRoleType">                     2
   <wsag:ServiceScope                                   3
      ServiceName="xs:string">xs:any                    4
   </wsag:ServiceScope>*                                5
   <wsag:QualifyingCondition>xs:any                     6
   </wsag:QualifyingCondition>?                         7
   <wsag:ServiceLevelObjective>                         8
      <wsag:KPITarget/> |                               9
      <wsag:CustomServiceLevel/>                       10
   </wsag:ServiceLevelObjective>                       11
   <wsag:BusinessValueList>                            12
       <!-- Omitted for clarity-->                     13
   </wsag:BusinessValueList>                           14
</wsag:GuaranteeTerm>                                  15
                                                       16
<wsag:GuaranteeTerm                                    17
  Name="AuditFailureSystem"                            18
  Obligated="ServiceProvider">                         19
   <wsag:ServiceScope                                  20
      ServiceName="StorageService01">                  21
      //SDT/AuditComp</wsag:ServiceScope>              22
   <wsag:QualifyingCondition>                          23
       ServiceState = "Ready"                          24
   </wsag:QualifyingCondition>                         25
   <wsag:ServiceLevelObjective>                        26
      <wsag:CustomServiceLevel>                        27
<t><name>AuditFailureSystemExists</name>               28
<ex>HasAuditFailure EQ True</ex></t>                   29
<t><name>98%AuditFullShutdown</name>                   30
<ex>count(//Var/@Name=AuditLog)GTEQ                    31
0.98*(//Var/@Name=LogCapacity) IMPLIES                 32
ServiceState = "Shutdown"</ex></t>                     33
      </wsag:CustomServiceLevel>                       34
   </wsag:ServiceLevelObjective>                       35
   <wsag:BusinessValueList/>                           36
</wsag:GuaranteeTerm>                                  37
```

Figure 6: Structure and example of wsag:GuaranteeTerm

Fig. 6 (lines 1-15) shows the structure of these elements in WS-Agreement. Note that the SLO term is composed of either a *wsag:KPITarget* or *wsag:CustomServiceLevel* element, where KPITarget contains a name and an <xs:any> field that identifies a target element as a *key performance indicator* (e.g. response time) and CustomServiceLevel simply contains an <xs:any> field. Finally, lines (17-37) show an example guarantee, called AuditFailureSystem, which insures an audit failure system exists and that the service will be shutdown if the audit log is 98% full. We discuss the problems with this example and compare it to our enhanced guarantee term in the next section.

G. SecAg Enhanced Guarantee Terms

There are several issues with the standard term that inhibit security interoperability and SLO comparison. First, the restriction that a Guarantee term be limited to a single SLO means grouping similar SLOs requires nesting them in a single SLO (and thus obfuscating the core meaning) or spreading them across multiple, differently named, guarantee terms (and thus obfuscating the grouping). For instance, suppose there are two SLOs that both pertain to an overarching audit failure system. Using the existing WS-Agreement structure would require creating two separate guarantee terms that break define each SLO individually, or group them together under a single mega SLO as in Fig. 6 lines 27-34, where each <t> marks a domain specific declaration structure that identifies each SLO.

To mitigate this problem, our first change is to allow multiple SLOs in a single guarantee term. For this to work and still be compatible with the other fields in the WS-Agreement guarantee terms, all SLOs must have the same service scope, qualifying conditions, and business value list. In this sense, the only change introduced is a separation of concerns – allowing each SLO to express one quality assurance as part of the overall term.

Secondly, as we have done with both the SDTs and service property enhancements, we include a new optional attribute *secag:SecurityType* that allows for ontologically defined standard security terms to identify a guarantee term. The security type indicates that the term matches a standardized guarantee, allowing for at-a-glance comparison of the high level guarantee terms.

Next, and perhaps most importantly, we define an augmented SLO element, *secag:ServiceLevelObjective*, that includes four attributes and a standardized expression format, embedded in a *secag:Predicate* element, that facilitates comparison. The four attributes are a wsag:Name, *secag:StandardSLO*, *secag:ServiceLevel*, and *secag:Importance*. The new terms are defined as follows:

- secag:StandardSLO ≙ attribute of type xs:string that identifies a standardized term associated with an SLO ontology
- secag:ServiceLevel ≙ attribute of type xs:float that quantifiably represented the offered service level (takes on the value of 1 or 0 for true or false levels)
- secag:Importance ≙ attribute of type xs:float used by the requestor to identify how important an SLO is, with respect to satisfying its in-house security

controls. An importance factor constitutes a *risk weight*, which is the amount of risk incurred by the requesting organization if the SLO is not met (we discuss risk and risk weights in the next section).

The standardized expression language includes both numerical and logical comparators. Each expression is specified as a secag:Predicate element that embeds an arbitrary number of sub expressions. Other XML expression formats such as WSML [20] exist, but are unnecessarily complicated for expressing security SLOs. Thus we provide the bare minimum number of operators for expressing constraints in a standard, comparable way. Table I shows the available operators in SecAg for expressing security constraints in SLOs and qualifying conditions. Each operation is specified in prefix notation in the XML.

TABLE I. SECAGREEMENT XML EXPRESSION OPERATORS

SecAgreement Element	Operator
secag:And	∧
secag:Or	∨
secag:Not	¬
secag:GT	>
secag:GTEQ	≥
secag:LT	<
secag:LTEQ	≤
secag:EQ	=
secag:NEQ	≠
secag:ElementOf	∈
secag:Implies	⇒
secag:Add	+
secag:Subtract	–
secag:Multiply	*
secag:Divide	÷
secag:InRange	True if a ∈ (n, m) for three numbers a, n, and m
secag:COUNT	returns number of items in a set

Finally, Fig. 7 shows how the enhanced guarantee terms look while Fig. 8 presents the refined example from Fig. 6. As you can see, the two SLOs previously grouped into a single SLO using the domain specific syntax are now separated into two distinct SLOs under the same guarantee term and specified using standard elements. Furthermore, the service levels are abstracted to the top level, facilitating comparison, and importance factors are attached to identify the risk weights of the SLOs.

```
<secag:GuaranteeTerm Name="xs:string"                    1
  Obligated="wsag:ServiceRoleType"                       2
  SecurityType="xs:string">                              3
   <wsag:ServiceScope                                    4
      ServiceName="xs:string">xs:any                     5
   </wsag:ServiceScope>*                                 6
   <wsag:QualifyingCondition>xs:any                      7
   </wsag:QualifyingCondition>?                          8
   <secag:ServiceLevelObjective> +                       9
      <secag:Predicate>…</secag:Predicate>               10
   </secag:ServiceLevelObjective>                        11
   <wsag:BusinessValueList/>                             12
</secag:GuaranteeTerm>                                   13
```

Figure 7: Structure and Example of secag:GauranteeTerm

```
<secag:GuaranteeTerm                                     14
  Name="AuditFailureSystem"                              15
  Obligated="ServiceProvider"                            16
  SecurityType="AFS">                                    17
   <wsag:ServiceScope                                    18
      ServiceName="StorageService01">                    19
      //SDT/AuditComp</wsag:ServiceScope>                20
   <secag:QualifyingCondition>                           21
     <secag:Predicate>                                   22
       <secag:EQ>                                        23
<secag:String>ServiceState</secag:String>                24
<secag:String>"Ready"</secag:String>                     25
       </secag:EQ>                                       26
     </secag:Predicate>                                  27
   </secag:QualifyingCondition>                          28
   <secag:ServiceLevelObjective                          29
     Name="AuditFailureSystemExists"                     30
     StandardSLO="AFSStructuralExistence"                31
     Level="1.0" Importance="1.0">                       32
     <secag:Predicate>                                   33
       <secag:EQ>                                        34
<secag:Bool>//Var/@Name=HasAuditFailure                  35
        </secag:bool>                                    36
<secag:Bool>True</secag:Bool>                            37
       </secag:EQ>                                       38
     </secag:Predicate>                                  39
   </secag:ServiceLevelObjective>                        40
   <secag:ServiceLevelObjective>                         41
      Name="98%AuditFullShutdown"                        42
      StandardSLO="AuditFailureResponse"                 43
      Level="0.98" Importance="0.7">                     44
      <!-- Omitted due to space -->                      45
   </secag:ServiceLevelObjective>                        46
   <wsag:BusinessValueList/>                             47
</secag:GuaranteeTerm>                                   48
```

Figure 8: Structure and Example of secag:GauranteeTerm

IV. RISK CENTRIC MATCHMAKING

Now that we have developed the SecAgreement extensions, we propose a two part risk analysis and risk-based matchmaking algorithm that utilizes the SecAg structure. Before we delve into the algorithm for calculating risk, we briefly outline how one might arrive at the "standard terms" we have mentioned throughout and how an organization might begin to assess risk weights.

A. Mapping Policy to Secag Terms to Understand Risk

Policy and information assurance are well studied security concepts that are typically addressed by an organization's compliance with a set of security controls. Each control governs some aspect of an organization's information systems in ways that satisfy the policy. Policy and security control documents such as the NIST-800-sp53 [3] group controls into *families*, and *categories* and assign them unique identifiers such as AU-12 – Audit record generation or AC-5 – Separation of Duty. Each of these are then broken down into more detailed control statements. In previous work [21] we examined these control categories, related them to other similar documents such as the DoDI 8500-2 [4], and arrived at meta-level security policy

elements. Work such as this can specify a set of common security terms, as we have shown in our examples, which can be used by CSPs to decrease the impact of semantics on interoperability and comparison processes. The standard terms can also guide organizational assignment of risk weights by mapping organizational defined or used security controls to SLA terms. In that sense, a risk weight on an arbitrary service term would be proportional to the emphasis the organization places on the policy elements to which the term is mapped.

B. A Risk-based Approach to Service Matchmaking

We begin our matchmaking algorithm by specifying the SLO matching process. Let $\boldsymbol{r}$ be a feature vector that contains N requested service level features, $rf_1 \ldots rf_N$, each of which can be a quantifiable value or a Boolean [where True=1 and False=0]. These features are each given a weight, w_i, which reflects the level of risk incurred if the requested feature is not present in the service level offering (f_i) (e.g. $rf_i = 1,\ f_i = 0$) or not fully compliant (e.g. $rf_i = 1,\ f_i = 0.9$). For instance, logging auditable events might be more important than having a real-time alerting mechanism and would thus have a higher weight. After the organization assesses each service level offering, the N weights are placed in the column vector $\boldsymbol{w}$. Finally, assume we have M potential CSPs vying for the service contract. Let $\boldsymbol{F}$ be an M x N matrix where the ith row is a row vector $\boldsymbol{s}_i$ corresponding to the ith CSP and f_{ij} is the jth service level feature offered by that CSP. Eq. (1) shows $\boldsymbol{r}$, $\boldsymbol{w}$, and $\boldsymbol{F}$.

$$\boldsymbol{r} = \begin{pmatrix} rf_1 \\ rf_2 \\ \vdots \\ rf_N \end{pmatrix} \boldsymbol{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix} \boldsymbol{F} = \begin{pmatrix} \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \\ \vdots \\ \boldsymbol{s}_M \end{pmatrix} = \begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1N} \\ f_{21} & f_{22} & & f_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ f_{M1} & f_{M2} & \cdots & f_{MN} \end{pmatrix} \quad (1)$$

To calculate how closely CSP service level offerings meet the requested service level features, given the feature importance, we calculate the weighted Euclidean distances between each pair (rf_j, f_{ij}), with risk weight w_j, and sum them together to produce a total distance d_i for the ith CSP. Eq. (2), iteratively calculates a column vector $\boldsymbol{d}$, which holds the M CSP offer distances.

$$d_i = \sum_{j=1}^{N} w_j \sqrt{\left(f_{ij} - rf_{j,}\right)^2} \quad 1 \le i \le M \qquad (2)$$

If we were to stop here, the service selection that most minimizes risk would simply be the minimum value in $\boldsymbol{d}$. However, SLOs are only one piece of the risk calculation. An organization also cares about the service meeting all of its business needs, or having the most exact match with respect to the security enhanced SDTs. Many researchers have previously examined this kind of problem and have developed comparators for examining service offerings.

For our algorithm we decided to adapt work done by Klusch et. al [19], who propose a hybrid matchmaking algorithm using SAWSDL. Their algorithm consists of two parts. First they define a partial ordering of service matches ranging from *Exact* (when the requested and offered term are identical), *Subsumes* (when the offered term subsumes the requested term), *SubsumedBy* (when the requested term subsumes the offered term), and *Fail* (requested term does not relate to any offered term in any of the previously stated ways). They also define a 5th match type, called *Plug-in,* but since it is a special type of subsumes we group them together for simplicity. The second part, a text similarity analyzer, assesses service terms to determine their semantic relevance to each other (using various measures such as keyword comparison, string edit distance, substring containment and Wordnet similarity [19]), and arrives at a minimum value representing the best possible match between requested and provided terms.

We adapt their comparator in two ways. First, we incorporate risk weights, labeled rw_j, into each of the requested service terms as we did with the requested service levels. These risk weights represent how important a given service term, which is embedded in SecAgreement as a ServiceDescriptionTerm, is with respect to satisfying the business, security, and policy goals of the organization. Second, we use their partial ordering to establish a fuzzy comparator. We could simply select 0, 0.37, 0.66, and 1 respectively, but this would indicate that there are equal distances between the different satisfactions of the request. In reality, *Exact* and *Subsumes* both mean that the requested functionality will be present in the offer (more exactly for exact) whereas *SubsumedBy* and *Fail* mean the request is not fully met (in certain ways). Thus, we assign *Exact* the value 0, *Subsumes* the value 0.2, *SubsumedBy* the value 0.6, and *Fail* the value 1 to better represent the degree of match.

To use this comparator, we define a function called *match*(t, $\boldsymbol{o_i}$), where t is some requested term describing a service operation and $\boldsymbol{o_i}$ is a vector containing the offered operations for the ith CSP. The *match* function will apply the hybrid matchmaking algorithm to the secag SDTs defined in Section III.C and return one of the 4 comparator values. To complete our extension we developed Eq. (3), which iteratively sums the product of the match function, applied to the jth requested operation (that is, t = ro_j) and the ith offered operation vector $\boldsymbol{o_i}$, and the corresponding risk weight rw_j over all j, to arrive at a value od_i (for offer distance) which describes how closely the ith CSP matches the requested operations. Together the od_i terms form a column vector $\boldsymbol{od}$ of length M.

$$od_i = \sum_{j=1}^{N} rw_j * Match(ro_j, \boldsymbol{o_i}) \quad 1 \le i \le M \quad (3)$$

Just like with the service level offerings distance vector $\boldsymbol{d}$, taking the minimum of $\boldsymbol{od}$ would give us the optimal CSP for that assessment (that is it would be the CSP that offers operations that most closely match the requested ones).

Taking a step back, the goal that we started with was to understand how much risk an organization would incur if it were to use a particular CSP. We decided the calculation required two parts: understanding the risk incurred by a CSP's non-fulfillment of security SLOs, and understanding the risk incurred by a CSP not providing a requested operation. We now have values that characterize the risk associated with each of these computed parts, so to arrive at a single risk value for the ith CSP we make one final

calculation, a simple weighted sum. Eq. (4) defines the final risk calculation as weighted vector addition, where c_1 and c_2 are organizationally defined constants that define the relative importance of risk associated with the service level offerings and the service operation offerings respectively. We recommend the values of 0.8 and 1 respectively for c_1 and c_2 based on the fact that most organizations value the presence of operational features, SDTs, over SLOs.

$$c_1 * \boldsymbol{d} + c_2 * \boldsymbol{od} \quad (4)$$

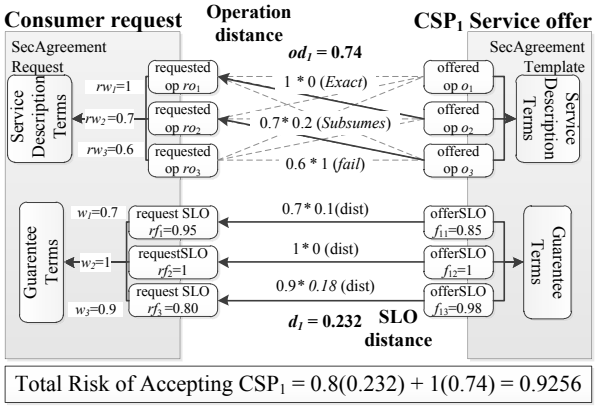

Figure 9: Example single CSP matchmaking

Fig. 9 presents a diagrammatic representation of our risk calculation algorithm applied to this example scenario. On the left the consumer provides a SecAg enhanced WS-agreement which contains three requested service operations: a read/write operation, ro_1, an auditor component ro_2, and a login component ro_3. They also request three particular service level objectives: 95% logging of auditable events rf_1, the presence of an audit failure mechanism rf_2, and 80% audit log full alerting rf_3. The service provider responds with a service offer that contains 3 operations: a service administration component o_1, a read/write component o_2, and a general auditor o_3 and 3 SLOs: 85% logging of auditable events f_{11}, the presence of an audit failure mechanism f_{12}, and 98% audit log full alerting f_{13}. Upon applying our algorithm, we find that the offered operation term o_2 exactly matches ro_1 and can thus provide the read/write operations, o_3 subsumes ro_2 and thus can provide an auditing capability, but in a more general way than requested. Finally, after looking through all offered operations, *match* returns fail for ro_3. Each of these are weighted according to the organizationally defined risk weights (left of Fig. 9) and summed according to Eq. (3) to arrive at $od_1 = 0.74$, or the risk incurred as a result of using the offered service operations. Now Eq (2) is applied to the offered and requested service levels and we arrive at a total SLO risk factor $d_1 = 0.232$ these are combined according to Eq. (4) to arrive at the final risk factor calculation for CSP_1 of 0.9256. In a real matchmaking scenario this would be done for the M available service providers and the consumer would select the CSP with the lowest risk factor.

V. Conclusion

SecAgreement extends WS-Agreement to allow security metrics to be expressed on the terms of SLAs. In this paper we examined each extension and exemplified its use. After establishing the specification elements needed to incorporate risk into SLAs, we developed a matchmaking algorithm that allows for services to be ranked by the amount of risk they would incur for a customer organization. Future work will be focused on better understanding the re-negotiation process involved when a cloud service provider decides to use another provider introducing additional risk that was not included in the original match process.

Acknowledgment

This material is based on research sponsored by the Air Force Office of Scientific Research, under agreement number FA-9550-09-1-0409. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views expressed in the paper are those of the authors and do not reflect the official policy or position of the Department of Defense or U.S. Government.

References

[1] A. Minkiewicz, "Cloud Nine, are we there yet?," *Journal of Software Technology,* vol. 14, no. 4, pp. 4-8, 2011.
[2] HHS, "Summary of the HIPAA Privacy Rule," 2003.
[3] NIST, "Special Publication 800-53," 2009.
[4] DoD. "Instruction DODI 8500.2, IA Implementation," 2003.
[5] A. Andrieux *et al.* "Web Services Agreement Specification (WS-Agreement)," http://www.ogf.org/documents/GFD.107.pdf.
[6] IBM. "WSLA "; http://www.research.ibm.com/wsla/WSLA093.xsd.
[7] K. Bernsmed *et al.*, "Security SLAs for Federated Cloud Services," in Sixth Int'l. Conference on Availability, Reliabilty and Security, 2011.
[8] M. Hepner *et al.,* "Forming a Security Certification Enclave for Service-Oriented Architectures," in Services Computing Wrks., 2006.
[9] H. Ludwig, *et al.*, "A Service Level Agreement Language for Dynamic Electronic Services," in 4th Int'l Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, 2002.
[10] P. Patel *et al.*, "Service Level Agreement in Cloud Computing," in Cloud Workshops at OOPSLA, 2009.
[11] K. P. Clark *et al.*, "Secure Monitoring of Service Level Agreements,," in Int'l Conf. on Availability, Reliability, and Security, 2010.
[12] C. Chen *et al.,* "AOP Based Trustable SLA Compliance Monitoring for Web Services," in 7th Int'l Conf. on Quality Software, 2007.
[13] R. Henning, "Security Service Level Agreements: Quantifiable Security for the Enterprise?," New Security Paradigms Wrks., 1999.
[14] C. Irvine, and T. Levin, "Quality of Security Service," in Proceedings of the New Security Paradigms Workshop, 2000.
[15] S. Lindskog, "Modeling and Tuning Security from a Quality of Service Perspective," Chalmers Univ. Of Tech. Sweden, 2005.
[16] S. Chaves *et al.*, "SLA Perspective in Security Management for Cloud Computing," in Int'l Conf. on Networking and Services, 2010.
[17] A. Celesti *et al.*, "How to Enhance Cloud Architectures to Enable Cross-Federation," in 3rd Int'l Conf. on Cloud Computing, 2010.
[18] R. Hu *et al.*, "WSRank: A Method for Web Service Ranking in Cloud Environment," in Int'l Conf. on Dependable, Autonomic and Secure Computing, 2011.
[19] M. Klusch *et al.*, "Hybrid Adaptive Web Service Selection with SAWSDL-MX and WSDL-Analyzer" The Semantic Web: Research and Applications," L. Aroyo *et al.*, eds., pp. 550-564: Springer, 2009.
[20] Working group, "WSML" http://www.wsmo.org/TR/d16/d16.1/v1.0/
[21] M. Hale and R.Gamble, " A Semantic Model for Software Security Compliance Certification" Report No. SEAT-UTULSA-122011, 2011 (under review).