SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2011; **4**:871–887
Published online 2 February 2011 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.217

RESEARCH ARTICLE

Security enhancements for UDDI

Alexander J. O'Ree and Mohammed S. Obaidat*,†

Department of Computer Science and Software Engineering, Monmouth University, West Long Branch, NJ 07764, U.S.A.

ABSTRACT

The universal description, discovery, and integration (UDDI) OASIS standard, was designed for storing, publishing, and advertising information about XML web services. Commonly used in service oriented architecture (SOA) infrastructures, the security of UDDI is often overlooked. Embedded within the specification are "optional" security measures that are commonly not implemented or enforced. In this paper we describe the UDDI security model, potential security related concerns, and mitigation strategies. Preliminary performance evaluation results are presented to show the merits of our proposed scheme. Finally, an example of the registry with additional security constraints is analyzed and evaluated for performance. Copyright © 2011 John Wiley & Sons, Ltd.

KEYWORDS

security of web services; UDDI; service oriented architecture; ebXML; REST; service discovery

***Correspondence**

Mohammed S. Obaidat, Department of Computer Science and Software Engineering, Monmouth University, West Long Branch, NJ 07764, U.S.A.

E-mails: obaidat@monmouth.edu,spyhunter99@gmail.com

†Fellow of IEEE and Fellow of SCS.

1. INTRODUCTION

Universal description, discovery, and integration (UDDI) is one of the major extensible markup language (XML) web service based standards for providing a number of different functions related to the storing, retrieving and advertising of information related to web services. The UDDI standard is a set of web service description language (WSDL) files, XML schema definition (XSD) files, and the associated documentation defining the behaviors, security mechanisms and usage of the seven web service interfaces that define the specification. Governed by organization for the advancement of structured information standards (OASIS), UDDI v3.0.2 currently enjoys wide adoption by many businesses and organizations. Based on XML SOAP, the simple object access protocol (SOAP) [1], the UDDI web service interfaces enable consumers to search for services by a number of different methods, publish new services, and validate custom extensions (technical models (tModels)) as well as facilitates the federation of registry data by replication. There are many concepts, terms, specifications and acronyms referred to in this document. The following subsections describe some of the key areas of interest related to UDDI and web services.

1.1. Web service

This work is strongly tied to web services and thus important to properly define the term, web service. There are many different definitions of both services in general and web services. One author states: "*Web services define a standardized mechanism to describe, locate, and communicate with online applications. The Web services framework is divided into three areas: communication protocols, service descriptions, and service discovery*" [2]. This definition is accurate but leaves a bit of ambiguity. For the purposes of this document, a "web service," unless noted otherwise, refers directly to XML services based on the SOAP over the hypertext transfer protocol (HTTP) or HTTP-secure (HTTPS) transport protocol [3]. Services can also run on a variety of other transport protocols and were specifically designed to be transport protocol independent. These web services are considered distributed computing and come in several different flavors and specifications, thus the importance to properly define the term. Typically, a web service represents some kind of functional portion of an application. Often, these services provide indirect interfaces to databases or perform some kind of task.

1.2. Service oriented architecture (SOA)

SOA is an application and system development paradigm that involves the use of interoperable, standards based, language independent portions of functionality that are exposed as distributable web services [4]. One of the primary goals is to reduce cost by reusing existing web services and to decrease the time to market for new products and applications. This increase in agility and decrease in development time are two of the primary driving factors for SOA adaptation. These web services create a network of distributed XML applications [5]. SOA itself represents a concept of design and development; however an SOA Infrastructure is slightly different. An SOA Infrastructure typically consists of various supporting services or capabilities in order to facilitate these web services. Example of capabilities may include security services, service registries, user directory services, and service management. One of the more information capabilities of an SOA infrastructure is the ability to register, advertise, and discovery web services *via* a service registry. This important cornerstone of SOA is an enabler for service discovery.

1.3. Service discovery

The concept of service discovery is a very important step for both services and service consumers in an SOA-based software lifecycle. Service discovery is the ability or process of finding information about a web service. This information typically includes the interface description for the service, its location, binding information, documentation, and so on. This process can happen at design time of either a service or a consumer and at run time when a consumer actively discovers the location of the target service. Without this important function, consumers of web services simply would not know of the existence of a given web service or have to rely on manual configuration of service consumers. The discovery functionality is not unique to web services. It functions similarly to the Domain Name System (DNS). Computers use DNS to translate unique identifiers (domain names) into an Internet Protocol (IP) address (location).

1.4. Service registries

The Service Registry is a core part of a SOA infrastructure. This allows for a centralized or federate location for storing, categorizing, and advertising information about web services. Another term commonly used to describe the service registry is the "yellow pages." As found in telephone books, the yellow pages act as a registry of services that business offer, as well as contact information and locations. There are several standardized interfaces for service registries and several more unique interfaces that are discussed further in this document.

1.5. Universal description discovery, and integration (UDDI)

OASIS is a not-for-profit consortium that drives the development, convergence and adoption of open standards for the global information society. The consortium has worked on countless XML based standards and many non-XML based standards since its inception in 1993 [6]. The UDDI standard is an OASIS ratified standard that is widely adopted and implemented both by commercial and open source products.

UDDI is used for storing and retrieving information about web services and services. As of the time of this writing, the current version, 3.0.2, of UDDI is in a state of "committee draft."

Figure 1 describes a few of the many cases of UDDI including a developer looking for an appropriate service to consume and a service consumer dynamically obtaining a service's location at run time. UDDI, SOAP, and WSDL are commonly referred to as the corner stone of web services and SOA architecture. UDDI has seven main interfaces or Application Programming Interfaces (APIs) listed below in Table I along with a brief description.

These APIs contain all of the necessary functions to support a UDDI Service Registry along with a number of additional options such as federation.

1.6. Service repositories

Registries are often coupled with service repositories to provide for the storage of related documentation. A repository in general, is a storage place for online resources. These resources may include everything from WSDL files to source code to documentation. Virtually any kind of file can be stored within a repository. Each file can be categorized and associated with a particular user, project, resource, or web service. Permissions and access control can be enforced, and finally the resource or document can be made external accessible *via* a simple uniform resource locator (URL) in a browser or *via* a web service API.

One such repository standard is the Electronic Business using eXtensible Markup Language (ebXML) Registry/Repository which is defined in the ebXML Registry Information Model specification. This is discussed in further detail in the next section.

1.7. Electronic business XML(ebXML) services registry and repository

EbXML is an ISO 15000 standard set defining a number of common characteristics for web services to follow in order to guarantee interoperability based on SOAP XML web services. By definition, it provides a number of different capabilities, including secure, reliable messaging, interoperability, a business process language, and a service registry and repository. EbXML started out as a joint project between the United Nations Centre for Trade Facilitation

Security Comm. Networks 2011; **4**:871–887 © 2011 John Wiley & Sons, Ltd.
DOI: 10.1002/sec

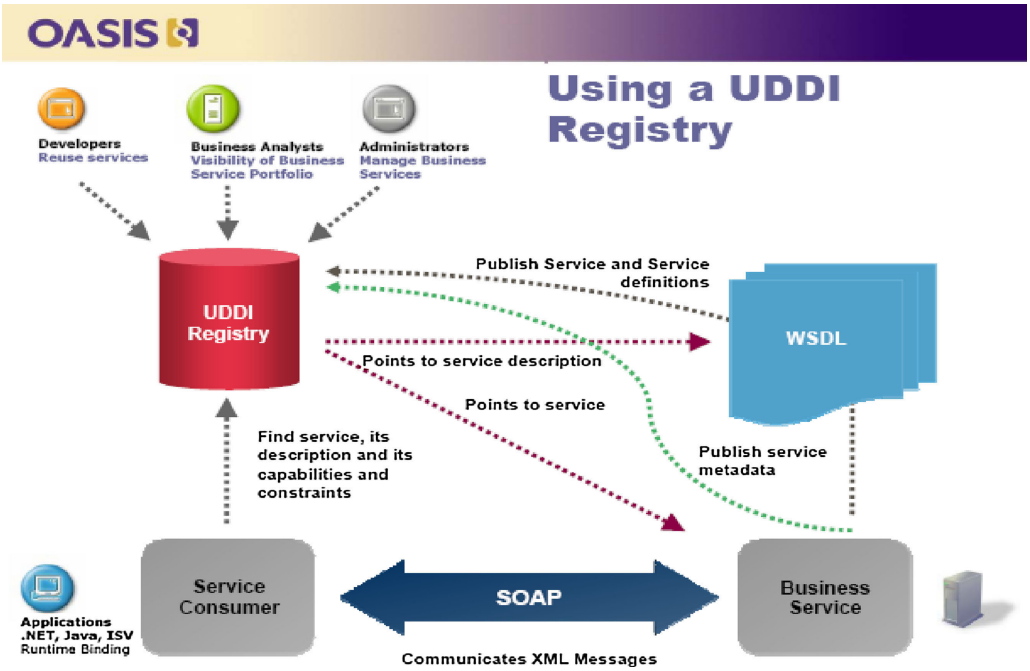

Figure 1. OASIS UDDI usage diagram [25].

and Electronic Business (UN/CEFACT) and OASIS [7]. One of the standards within ebXML is the registry and repository. Often referred to as the ebXML RegRep, the official name is the "ebXML Registry Information Model" and the "ebXML Registry Services and Protocols." Figure 2 depicts the overall feature set of the protocol. The ebXML RegRep contains web service definitions and an information data model for storing web service data.

EbXML-based services typically utilize a number of WS-Security standards including security assertion markup language (SAML) for message level security and HTTP secure sockets layer (SSL) for encryption. These standardized services are unique by requiring a standardized message body for all messages with the payload of the message being transferred *via* an SOAP Attachment (SWA) [8].

Table I. UDDI Interface APIs.

Inquiry–used for reading and searching for information in the registry
Publishing–used for making changes within the registry, update, add, delete
Security–used for issuing tokens for authentication purposes
Custody Transfer–used for federated UDDI clusters
Replication–used for federated UDDI clusters
Subscription–used for subscribing to updates on UDDI data elements
Value Set–used for validating custom tModels

The ebXML Registry Information Model is currently at version 3.0.1. Dated 2007, it is still being revised and expanded to support new protocols and standards, such as the Web Ontology Language (OWL) and REpresentational State Transfer (REST) [7].

1.8. Other repository standards and specifications

OASIS has a new upcoming specification, the content management interoperability services (CMIS). CMIS is targeted to provide greater interoperability and support for enterprise content management (ECM) systems [9]. Since a service repository could be implemented using a more generic ECM, the CMIS specification is a promising way to provide a standards-based approach. CMIS is still under development but has already been announced by Microsoft to have

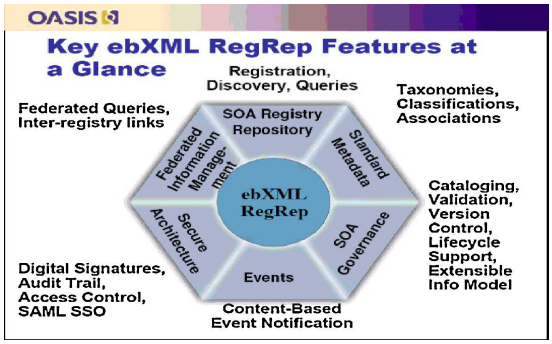

Figure 2. EBXML registry repository feature.

direct support for their online collaborative web portal and content management system, SharePoint 2010. This specification is not yet considered a standard.

1.9. Alternate service discovery approaches

There are several other approaches available for performing the same function as a service registry without using SOAP, UDDI, or ebXML. Two of them are REST and DNS.

REST is a software architecture style for creating web services with minimal overhead without using SOAP. It is based on the doctoral dissertation of Roy Fielding and is not currently considered a standard [10]. REST primarily utilizes straight XML over HTTP(S), utilizing HTTP verbs such as GET or POST to describe the actions the service performs. For example, an REST consumer would use an HTTP GET message for obtaining information in a read only fashion and use POST, DELETE, or PUT for changing the state of the server. Integrity, encryption, and authentication are all specifically handled by HTTP, SSL, and HTTP based authentication mechanisms. It is not protocol independent like SOAP. Both SOAP- and ebXML-based services are transport protocol independent and utilize XML-based standards for performing message integrity, encryption, authentication, and so forth. SOAP and ebXML only use POST HTTP messages for all transactions, read or write.

As far as service discovery is concerned, there are a few REST-based service registries in development such as WS02, which is an open source project and an attempt at creating a service registry and repository specification and implementation [11]. REST is significantly different than SOAP and is not a standard but an architectural style. It is an important upcoming technology that should not be ignored.

There is also relevant research into the area of using specialized DNS records, endpoint discovery (EPD) records, to identify web service locations [12]. In this scenario, one of the oldest and well-known infrastructures is reused in a different way. The DNS provides translation to and from domain names and IP addresses, as well as providing basic information lookups and pointers. It was created in the early 1980s by the Internet Engineering Task Force (IEFT). This research is significant; however, it does not address some of the other more interesting areas of service discovery. These areas include providing information about the classification, categorization, documentation, security requirements, and ownership of a service.

2. RELATED WORK AND LITERATURE REVIEW

UDDIe is an extended registry that supports the notion of "blue pages" to record user defined properties associated with a service and to enable discovery of services based on these. UDDIe enables a registry to be more dynamic, by allowing services to hold a time lease, a time period describing how long a service description should remain in the registry [13]. The "blue pages" can be easily translated into a folksonomy. A folksonomy, also referred to as social tagging, is a commonly used method of categorizing items on the web through the use of weighted keywords that are supplied by end users. The concept is that the more users that visit a particular web page, resource or in this case, service descriptions, the higher the ranking for the page's content for a given set of related web pages or resources. In an UDDI context, this can actually be used as a ranking system for services.

Another interesting publication involves a new XML markup language to aggregate search results from one or more UDDI registry. The primary idea of Advanced UDDI Search Engine (AUSE) is to aggregate search results from different UDDI registries based on the USML (UDDI Search Markup Language) request and its supporting intelligent search facilities [14]. Upon further research, USML is actually part of an IBM project called Business Explorer for Web Services, BE4WS which provides a standardized search interface for UDDI registries. UDDI only offers searches of the local registry node. This can be expanded however through the use of federation of multiple registries *via* UDDI's replication APIs. In this scenario, all federated registry entries are copied and synchronized from remote registries to the local instance. Due to synchronization schedules, the data available at the local UDDI node may not always be up to date. The ability to use a real time aggregated searches to remote registries is an important step for expanding the search capabilities of the UDDI search inquiry APIs.

In the commercial market place, SOA Governance is one of the major buzz phrases. This means that through the use of some tooling or products, rules can be enforced for web services before making them publically or organizationally available. These rules can govern items such as documentation, namespaces, performance requirements, interface specifications, coding practices, approval workflows, *etc*. One group of researchers have investigated using a customized UDDI Registry to perform these types of tasks before publishing the entry by adding check-in and check-out features to the registry. They also were able to mandate testing scripts for both client and service in order to verify the functionality of the web service before adding it to the registry for publication [15]. Individually, these bits of functionality are common in many pieces of software across a wide range of solution spaces such as unit testing, but applied together, they can greatly enhance the integrity, data quality, and the usefulness of a UDDI registry.

Web service management is a large problem space in SOAs which contain all but a few commercial solutions and even fewer open source solutions. Management in this context includes the monitoring and measuring of the performance of web services. This can also include tracking exceptions or service faults and enforcing service level agreements. The next logical question to answer regarding web service management is, what to do with this data and where should it be stored? One approach is to store this information in a UDDI registry.

Security Comm. Networks 2011; **4**:871–887 © 2011 John Wiley & Sons, Ltd.
DOI: 10.1002/sec

One method is to use the extensibility of UDDI tModels to publish additional metadata into the registry. Metadata such as usage statistics, performance metrics, and security policies (WS-Policies) are examples of this. In reference [16], methods were proposed to enable metadata publishing into UDDI through the use of extensions and lightweight client side mechanisms. Additional metadata types mentioned in the reference include availability, reliability, response times, and exceptions.

The publishing of this additional data presents a few problems. The first is that since tModels are completely user extendible and by definition, of type xs-string[‡]. This prevents the ability for the UDDI consumer to make range value based queries. For example, if a client performs an EPD using UDDI and more than one relevant endpoint is returned, a useful method to decide which endpoint to use is to examine the metadata. Provided that performance metrics could be made available in UDDI on both endpoints, the client can then choose the fastest or the more reliable service to execute.

The second issue is that any UDDI consumer would have to know about the specific tModel ahead of time and would have customized code to match that particular tModel in order to provide any additional post processing. The usage of additional published performance data, although it could be parsed and used programmatically, it is more likely to be used in a non-programmatic fashion, such as an end user being presented with the data and then selecting the target endpoint.

This is a great capability, however programmatically performing this task is going to be specific to each implementation of UDDI and the corresponding tModels describing the performance data which are vendor dependent in some cases. Although several commercial products have the capability to publish these metrics, they are not standardized or published within similar tModel definitions. A few standards exist for this type of data, such as Web Service Distributed Management, WSDM. The third issue is that in order for any of this to work properly, this configuration would be specific to a particular implementation and instance of a registry. This defeats the point of having a standard.

With web services, the objective is to reuse web services, sometimes even for unintended purposes. This can be accomplished through using XML translations, service orchestrations, or through workflows. Workflows offer an interesting perspective on web services. One or more web services, interactions with other components, and human interaction can all take place within a workflow. There are several workflow standards such as OASIS's Business Process Execution Language (BPEL) and Wf-XML/ASAP. Wf-XML is an XML standard for executing workflows created by the Workflow Management Coalition and is an extension off of the OASIS standard of ASAP, or Asynchronous Service Access Protocol [17].

Often found is SOA infrastructures, a business process or workflow is a combination or compilation of multiple web services and supports direct human interaction, such as decision making, approval process, *etc*. Business processes, in the context of web services and workflows, often directly mimic how the actual business operates such as order processing, help desk ticketing systems, personnel on boarding, *etc*. Depending on the execution environment, these workflows can actually be exposed as web services through an SOAP interface. Because of the additional complexity involved with a composite business process service or workflow, it is challenging to properly identify it within a UDDI registry.

In reference [17], several other extensions to UDDI were proposed, including the ability to represent domain dependent or user defined relationships between model elements, such as dependencies, functional relationships, delegation, and other relationships between UDDI service entries. Web services, due to their distributed nature, can often have complex relationships and dependencies with other services. This can create a house of cards in which when one particular service fails, many other services which depend on it will also fail to operate. To make matters more difficult, troubleshooting this complex arrangement of dependencies can be very difficult unless proper documentation, management, and auditing of these dependencies are in effect. This is an unfortunately consequence of any distributed platform and thus highlights the importance of identifying these complex relationships. An extended UDDI using tModels or some other specification change would be able to store this information and perhaps with the help of commercial products, such as Amberpoint's SOA Management System (SMS), discovery the dependencies can happen automatically at runtime and then the corresponding UDDI record can be updated.

These relationships between services closely tie back to the ideas of taxonomy, ontology, and folksonomy in which categorization and classification are used to increase the availability of data by grouping or linking similar items [18]. Folksonomy, also commonly referring to as social indexing or tagging, is the process of adding tags to categorize content by end users.

In the web service development world, it is common to run into security related problems. There are so many standards for securing the message level security of an SOAP or generic XML transaction that integrating a client and service can be a challenging task. To make matters worse, SOAP runtime environments often have different interpretations of these standards, which can create additional confusion. This security mismatch has been recognized and a solution has been designed that may help alleviate this problem. A negotiation protocol was designed for enabling easier security protocol integration on the fly by enabling both the client and service to adjust as necessary without user intervention [19]. If implemented, this can help reduce integration time and cost of using SOA and secured web

[‡] In xs-string, xs stands for XML Schema, part of the official W3C definition for XML Schema definitions

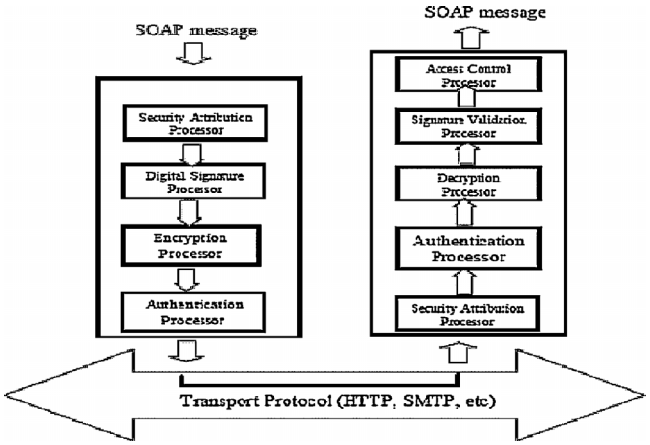

Figure 3. Architecture for secure web service container.

services. In the standards world, however, there were already a number of existing standards and specifications that can be used to publish security related data for web services. WS-MetadataExchange, WS-Policy both from W3C, and WS-Discovery from Microsoft are three of the commonly supported specifications that are typically used at development time. These protocols enable a semi automated method to configure a web service client to conform to the security protocols specified by the service's requirements. This research provides a new way to performing the same task at runtime.

In a reference [20], a message level security framework was proposed that utilizes a number of standards in order to facilitate a secure message transaction with authorization policies based on XACML, the eXtensible Access Control Markup Language (XACML). The usage of open standards to secure communication and perform access control is an excellent use case for access control to services. The example architecture is depicted in Figure 3.

In another related article, researchers created a Context-Aware Security Policy, which when implemented, used OWL and XACML based reasoning policies to control access to UDDI resources [21].

There is no question that UDDI can be and should be extended in order to expand the capabilities provided by the specification and subsequent implementations. Many of the noted features or extensions to UDDI could and should be included directly within UDDI specification. The remainder of this paper builds on a subset of these extensions.

3. UDDI ENDPOINT DISCOVERY

In terms of discovering the endpoint or access point to a web service, UDDI offers this ability through a few methods in the Inquiry API. In order to obtain an access point for a web service, an UDDI client must be executed *via* one of the following methods: find_binding, get_bindingDetail, and get_serviceDetail. This is graphically depicted in Figure 4. All of these require the client knowing a unique

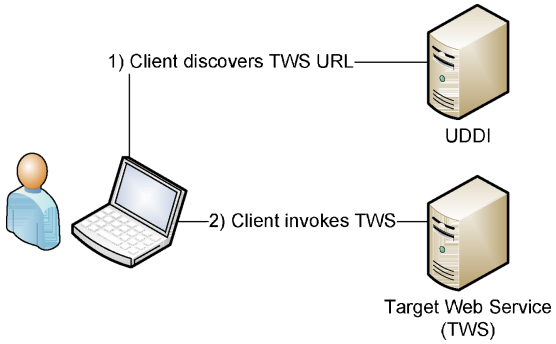

Figure 4. Service EPD procedure.

identifier ahead of time. This identifier can also be discovered *via* another Inquiry API such as find_service. All of these methods return a significant amount of data, most of which is discarded during runtime discovery.

In many real world scenarios, UDDI is extended *via* the tModel, a key, name, value triplet that can be associated with a particular registry entry. The majority of these data is returned.

Web service management is a large problem space in SOAs, which contain all but a few commercial solutions and even fewer open source solutions. Management in this context includes the monitoring and measuring of performance of web services including tracking exceptions and enforcing service level agreements. The next logical question to answer regarding web service management is, what to do with this data and where should it be stored? One approach is to store this information in an UDDI registry.

One valid use case of an UDDI registry is to use the extensibility of tModels to publish additional metadata into the registry. Metadata such as usage statistics, performance metrics, and security policies (WS-Policies) are prime examples. "UDDI enhancements for Dependability" enables metadata publishing into UDDI through the

use of extensions and lightweight client side mechanisms [16]. Additional metadata types mentioned in the research includes availability, reliability, response times, and exceptions.

The publishing of this additional data presents at least three problems. The first problem is that since tModels are completely user extendible and are yet by definition, of type string, this prevents the ability for the UDDI consumer to make range value-based queries. The second issue is that any UDDI consumer would have to know about a specific tModel and have customized code to match that particular tModel in order to provide any additional postprocessing. The usage of additional published data, although it could be parsed and used programmatically, it is more likely to be used in a nonprogrammatic fashion. For example, if a client performs an EPD using UDDI and more than one relevant endpoint is returned, a useful method to decide which endpoint to use is to examine the metadata. Provided that performance metrics are available in UDDI on both endpoints, the client can then choose the fastest or the more reliable service to execute. This is a great capability; however, programmatically performing this task is going to be specific to each implementation of UDDI and the corresponding tModels describing the performance data. Although several commercial products have the capability to publish these metrics, they are far some standardized or published within similar tModel definitions. The third issue is that in order for any of this to work properly, this configuration would be specific to a particular implementation and instance of a registry. For example, several commercial products offer the ability to publish metadata in the form of tModels to an UDDI registry. One such product is Layer7's XML Networking Gateway, which publishes URLs to WS-Policy files associated with the service. This aids in client side integration by providing security requirements and related information. Another product is Amberpoint's SMS, which has the capability to monitor the performance and throughput of web services. This data can then be published into UDDI tModels and then associated with service's record. Since tModels are defined only as strings, clients wishing to make use of this data programmatically will have to first obtain all records that need to be compared, convert the string data into an appropriate data type and then perform the comparison.

4. WEB SERVICE SECURITY AND SERVICE DISCOVERY

Since UDDI is based on web service interfaces, web server and web service security should be of concern, either when developing or deploying an UDDI node. Web services are based on XML messages and are typically sent over HTTP transport protocol, although services can also communicate over SMTP[§] and JMS[‖] These transactions are designed to go through network firewalls and often pass through intermediate devices such as XML translators or security appliances[¶]. Securing the transactions of web services therefore must be implemented *via* other mechanisms. There are a number of identified security issues with web services as well as key parameters that are often used to describe the security profile of a web service. These security concerns can be mitigated at the transport, message level, or a combination of both.

- Confidentiality–can a third party read confidential information as it is sent over the network?
- Integrity–has the inbound or outbound message been modified in any way while in transit?
- Authenticity–could someone impersonate the consumer or provider and send false information?
- Non-repudiation–can the consumer or provider deny sending information?
- Replay attacks–can this message be captured by a third party and resent in order to obtain the data?
- Authorization–does the requestor have permission to use this service, method, or piece of data?

The majority of these security concerns can be addressed through the use of standardized XML security protocols, Public Key Infrastructure (PKI) certificates and HTTP mechanisms. There are a large number of XML standards, identity management protocols, and HTTP protocol adaptations that can be used to secure a web service. Since UDDI is a set of web services, it is important to look at the specification with security in mind.

The National Institute of Standards and Technology (NIST) has recently published findings on web service security and potential security vulnerabilities. It listed a great number of potential areas of concern for web service security and UDDI [22,23]. Even though UDDI is a set of web service interfaces, it presents additional security concerns that are only partially addressed within the specification and accompanying documentation. The following sections will detail some of the findings and present example mitigation strategies. Some of these issues are common to just about any piece of software, but many are not and are clearly web service and UDDI specific.

Obaidat and Boudriga [24] emphasized the importance of protecting service registries. The key points outlined include strong user management with authentication and authorization policies, auditing and log file protection, denial of service protection, intrusion detection and finally overall network protocol security. These parameters provided an excellent starting point for the remainder of this discussion as it relates toward enhancing the security profile of UDDI.

The sections that follow describe how each of these security issues applies to the UDDI specification and how

[§] Simple Mail Transfer Protocol.

[‖] Java Messaging Service.

[¶] Example security appliances: IBM Datapower, Layer7's XML Gateway.

Table II. UDDI authentication message flow.

Actor	Destination	Credential	Returns	Recommended Protection
Client	UDDI security API	Username/password in SOAP: envelope/body	Authentication token (string)	SSL
Client	UDDI inquiry/publish API	Authentication token in SOAP: envelope/body	Requested data	None
UDDI	Client	None	Replies with requested data	None

the proposed modifications are applied on UDDI to help mitigate the risks currently found within the specification.

5. UDDI SECURITY MODEL

The current UDDI specification leaves it up to the implementer of the registry to decide on which method or methods to use for securing access to the registry. It does not mandate any kind of authentication or authorization on most interfaces. Several key areas of UDDI's security model are discussed in the following sections, including authentication, authorization, and message transaction protection.

5.1. Authentication

UDDI's security model for authentication is *via* the optional Security API. If a registry node supports authentication and the Security API, then in order to access protected resources, a client must first execute a request to the Security API's getAuthToken, passing it the username and "cred" as it is referred to in the specification. The Security API then returns an "authToken" which is then included with subsequent callouts to the other UDDI APIs. An important point is that the "authToken" is included in subsequent callouts directly within the SOAP message body of the other APIs. The drawback to this is that it makes authentication very not modular or flexible. Each API and method now needs to have authentication and authorization code incorporated into it. Ironically, having the UDDI's security mechanisms written directly into the WSDL API, especially as an optional API, is against many of the standards that OASIS has produced, such as SAML, WS-UsernameToken, *etc*. All of these authentication mechanisms are SOAP message security headers and purposely cannot be placed within the body of the SOAP message. Additionally, they all offer additional protection such as XML signature or encryption on sensitive data, such as a password. If the Security API is configured for a non-encrypted channel such as HTTP, both the username and password will be sent in clear text.

Table II depicts the message flow of an UDDI style message transaction with authentication.

In reality, service and data access control mechanisms are not difficult to implement or enforce. Many application frameworks include this capability free of charge. There are several standards that help define ways to do this, such as SAML, SAML Protocol, and XACML.

The "cred" element for the Security API is supposed to be some sort of credentials that only the user knows. The specification states that it really could be anything so long as it is documented. According to the XSD for UDDI, it is of type string. In practice, most UDDI implementations, specifically jUDDI, Centrasite, Systinet, and OpenUDDI, use username/password combination for authentication. The UDDI specification states the security mechanisms provided can be extended through the use of the Security API which is itself, an optional API. The specification explicitly states that an implementer may extend this functionality. "A node MAY provide an alternative mechanism for obtaining authInfo elements" [25]. There is no further clarification on this from the specification.

One interpretation would be to include the security token as the "cred" element. This increases the difficulty level significantly from an implementation perspective. A UDDI implementer would have to reinvent the security architecture, such as the SAML in order to put the identity markup up in the body of the message. This goes directly against the SAML standard of placing the SAML assertion markup within the Security XML node of the SOAP message header [26]. Another interpretation of this extension would be to have a custom version of the Security API, which requires an SAML or WS-UsernameToken for authentication, which in turn is used to generate the "authToken". For clarification, the "authToken" is also of type string from the UDDI XSD.

In contrast, ebXML Registry and Repository standard is very clear on what is required and recommended. "A registry MUST enforce all Access Control Policies including restriction on the READ action when processing a request to the HTTP binding of a service interface" [7]. This particular specification is very security conscious and continues further. "The Registry MUST be able to authenticate the identity of the User associated with client requests in order to perform authorization and access control and maintain an Audit Trail of registry access" [7].

Additionally, the ebXML Registry provides direction for specifying how users are authenticated and authorized. This is accomplished *via* the OASIS standard, SAML. Several usage scenarios are defined in Section 11.4 of the ebXML RegRep specification. The advantage to using SAML comes down to how the SOAP interfaces are implemented and

how they serialized on the wire in the SOAP message header. SAML is a widely accepted and implemented standard.

It is obvious that security enforcement within the UDDI specification is open ended so that implementers are left with designing their own approach. This means that creating a matching UDDI consumer can be challenging considering the potential for a variety of authentication mechanisms, which must be specified within the functional code of both the client and service because of the embedded security elements. By standardizing on the functional interface of the service and detaching the authentication mechanisms from the interface, the security requirements can be added or enforced after the fact with security handlers.

An example use case would be the usage of WS-UsernameToken or WS-Trust SAML tokens[#], which both provide a secure authentication method within the security header of the message. Both are well-known standards and can interoperate over a large array of platforms and applications.

5.2. Authorization

Authorization is another area of concern which asks the question, "does this authenticated user have permission to perform a given action on a specific item?" The UDDI specification leaves it up to the implementer to decide on what, if anything should have access controls on it.

There is no clear guidance on enforcing access control or obtaining user credentials aside from the generic authInfo elements. Typically, when a web service is written, the API is provided from a WSDL, file which explicitly defines the message and data contracts for each operation. This turns into code that implements the interface specifications as defined with the WSDL and any other accompanying documentation and requirements. Once the functional portions are completed, additional security handlers[**] are added to the deployment. These security handlers provide a framework for signing and verifying XML digital signatures and encryption, enforcing access control, and a slew of other functions that are not functionally specific to the service. The handler approach normally does not modify the content of the body of the XML message and only adds elements to the security header section or provides additional cryptography services, such as XML Encryption or XML Digital Signatures. Since it is separate from the functional body of the service interface, it enables the security framework to be modular and easy to migrate from one security model to the next. The security requirements can even be advertised as metadata in the service's WSDL and/or WS-Policy attachment.

For comparison purposes, the ebXML Registry specification is much more specific about authorization. The registry "... MUST perform authorization and subsequently enforce access control rules based upon the authorization decision. Authorization and access control is an operation conducted by the registry that decides WHO can do WHAT ACTION on WHICH RESOURCE" [8]. The ability to apply and enforce a fine grained, role based, or attribute-based access control mechanism to UDDI elements is critical to maintaining information security. Without it, potentially sensitive information about web services can be divulged to unauthorized users. This includes locations, owners, security mechanisms, *etc*. Furthermore, there are several business scenarios in which a business entity would only wish to share specific registry information with authorized business partners [27]. In these cases, information security is a fine grained access control mechanism referred to as role-based access control (RBAC), in which group membership is used to filter data based on XACML policies [28]. Access control can also be based off of a simple list, attributes, group membership, and so on.

In a real world example, in the commercial UDDI implementation, HP SOA Systinet, automatically registers a large number of internal services. In the version used for evaluation, there was no authentication or authorization required for the web user interface or for the Inquiry API, potentially exposing these internal services to attack. This practice should be avoided as potentially unsecured endpoints could be advertised and therefore exploited [23].

5.3. Message and transport level security

The current UDDI standard recommends HTTP transport for Inquiry and Value set APIs. The remainder of the APIs are recommended for SSL but not required. Table III is an excerpt from the UDDI specification.

6. UDDI THREATS

As architected and implemented with the recommended security model, UDDI v3 still suffers from a number of security vulnerabilities. The following describes some of these problems and their potential impacts.

6.1. Registry endpoint replacement

UDDI is often considered the corner stone for SOA infrastructures. One of the most common use cases is runtime service discovery. This process generally occurs when an application or service requires something from another downstream service and the location of this target web service (TWS) is unknown. The application then attempts to discover the TWS's location by another web service callout to an UDDI server. In this case, UDDI is being used as a veritable phone book of web service information. This also

[#] WS-UsernameToken is an OASIS specification for transmitting usernames and encrypted passwords.

[**] The term "security handler" is often interchanged with enforcer, behavior, or filter.

Table III. UDDI v3 Recommend Security Model [25].

API set	tModel	Recommended transport	Recommended security mechanisms integrity/confidentiality	Authentication
Inquiry	uddi-org:inquiry_v3	HTTP		
Publication	uddi-org:publication_v3	HTTP	SSL V3	
Security	uddi-org:security_v3	HTTP	SSL V3	
Custody Transfer	uddi-org:custody_v3	HTTP	SSL V3	
Replication	uddi-org:replication_v3	HTTP	SSL V3	Mutual authentication
Subscription	uddi-org:subscription_v3 uddi-org:subscriptionListener_v3	HTTP	SSL V3	
Value set	uddi-org:valueSetValidation_v3 uddi-org:valueSetCaching_v3	HTTP		

As one can tell from this table, there is no explicit requirement or recommendation for message level protection other than the use of SSL on some of the interfaces.

presents some unique security concerns that are either not addressed or only partially addressed within the specification.

If a web service consumer participates in service discovery, in which the URL to the service is "discovered" at run time by an additional web service call to a service registry, there is the potential for registry endpoint replacement to occur. This happens when a client requests a web service endpoint from a registry and either a "man in the middle" attack occurs or the registry is falsified (entirety or just a specific data item). This attack returns a false URL endpoint to the service. The client, trusting that the registry has returned a valid endpoint, then sends a request to a malicious web service that records the inbound message. Potentially sensitive information can be disclosed this way, including the security profile of the actual service. The current UDDI standard does not mandate or recommend the use of message level XML digital signing on the message transaction level. XML signing of specific registry elements is optional; however this leads to an implementation related issue. The validation of this signature is outside the scope of most web service development and runtime frameworks; thus it is up to the UDDI client to implement, validate, and optionally enforce this style of signature. One easy solution would be to sign all XML messages to and from the registry and enforce HTTP with SSL by using a PKI certificate issued by a trusted authority, both of which would successfully mitigate this vulnerability.

Registry endpoint replacement can also happen if security is compromised within the registry itself and it advertises false information (an unauthorized user changes entry data). Again, the signing of individual UDDI elements can be used to mitigate this issue to some degree. This would not protect against a "man in the middle" attack coupled with the removal of the digital signature element. Additionally, the digital signature could be replaced by resigning the falsified element with another certificate. The signature could also be removed from the registry by an unauthorized user. Providing that the client does not require the signed element, this change would not be noticed. Further discussion on unauthorized changes to UDDI data is beyond the scope of the specification and this document; however, an implementer of an UDDI registry should understand the importance of safe guarding information from attack.

6.2. Information disclosure

The information stored within an UDDI server can often vary. The specification offers an extensibility point called a tModel, or tModel as it is referred to in the specification. These tModels have been used for everything from OWL-S[††] semantic tagging [29] to storing service performance metadata [16]. All of this information can be considered sensitive information per the particular instance of a service registry.

For each service and business entry, a service registry can store data related to the owner, technical, and administrative contacts. There are growing concerns for information and personal privacy and the risks from social engineering. This simple contact information could potentially be sensitive. Considering that the UDDI specification does not require encryption for the most commonly used Inquiry API, it is obvious that this information can be susceptible to interception.

There is one gleaming problem with the UDDI v3 recommendations for security. The specification recommends SSL for the Security API when a client receives an authentication token. This token can then be sent over a non-secured channel. UDDI does not recommend nor require SSL for the Inquiry API. A malicious user can therefore obtain this token over unencrypted HTTP transmission and then impersonate the user. By not *requiring* authentication and authorization to an UDDI server or at the very least, supporting access control on specific UDDI elements and adequately protecting the authentication channel, sensitive information can be changed or disclosed to unauthorized persons.

[††] Web ontology language for services.

6.3. Man in middle attacks

Although the "man in the middle" attack has been around for quite some time, its effects on an UDDI node can be detrimental. According to the UDDI specification, all Inquiry API calls, which includes those used for service EPD, are recommended to be transmitted in clear text over HTTP. There is no recommendation for XML digital signatures (message level), XML encryption, or transport layer encryption. The specification does at least partially address this by including an optional XML digital signature on a specific UDDI data element, such as a business or service. This however can easily be overcome by simply removing the digital signature element in transit. Unless the requestor of this data knew *a priori* about the signature, the removal of the signature would most likely go unnoticed. The most common usage of XML digital signatures is the signing of SOAP message headers, such as Timestamps or WS-Addressing headers, and the signing of the entire body element of the SOAP message envelope. In general, SOAP development frameworks readily support the signing of the entire body of the message, not individual items. It is outside of the norm. Since this is an item level signature and not a message level signature, an application developer would either have to rely on a software development kit provided by the SOAP framework and/or the UDDI developer, or simply ignore the signature elements. This places all Inquiry API message transactions in high risk for several specific security vulnerabilities. UDDI is used as a corner stone for finding web services and thus the importance of securing these message transactions is critical. The simplest thing to do is to require message level signatures, but still continue to support item level signatures. This way message integrity and data integrity can be verified.

6.4. Common web service security issues

Since UDDI is web service based, all known web service vulnerabilities should also be concerned. Clearly, there are web service specific security concerns. Under the US Department of Commerce, the NISTs have published documentation that helps identify and describe a large portion of these concerns [23]. Although several of items listed are common to any piece of hardware or software, the majority of them are web service specific. After consolidating the several similar finds, over 50 specific issues were published. Ranging from message replay attacks to user principle impersonation, all of these issues should be considered for an implantation of UDDI.

7. PROPOSED SECURITY ENHANCEMENTS

After considering the potential security risks related to the UDDI specification, the following defines a number of recommended changes in order to enhance the security structure. Some of these changes can be implemented with minimal or low impact to existing UDDI implementations and deployments. Other changes require revision of the UDDI WSDL and XSD files, as well as the associated documentation. Moving forward, these recommendations should be integrated within the next version of the UDDI specification.

7.1. Transport and message level security

Web service traffic is commonly passed through intermediate web servers or devices for a number of different reasons, even with HTTPS SSL. For this reason, it is equally important to ensure message level and transport level security. Commonly, security is defined as a process, not as a something that is solved and forgotten about. This is for two reasons:

(1) There are always new security vulnerabilities being discovered. There are numerous government and private organizations that actively promote the awareness and resolutions to security vulnerabilities.
(2) The security of an electric system must always be thought of in a layered approach. Often, systems are designed with a single security enforcement point, such as a network firewall. While this approach provides a hard outer shell, the components behind the firewall are soft. For this reason, everything that is on a network should be properly secured, whether or not it is directly accessible from an external network [30].

To mitigate any potential security and risks issues, the following changes are recommended to the UDDI standard.

Message level:

- Mandate the support and requirement of XML digital signatures on all message transactions
- Optional support for XML encryption
- Requirement for an authentication token
- Requirement for signed WSU[‡‡]; Timestamps. By specifying message creation and expiration timestamps, it will also help prevent message replay attacks.
- Requirement for signed unique WS-Addressing message identifiers.

Transport level:

- Requirement for encryption either *via* HTTPS SSL/TLS communication or by XML encryption on all APIs and applicable GUIs.

Policy level:

- Requirement for authentication and authorization for all resource access requests
- Requirement for protected audit log.

[‡‡] Web service utility.

These changes will undoubtedly add overhead to UDDI transactions due to expensive encryption and signing algorithms. The expense of message level security is discussed in section VIII. This additional overhead outweighs the threat of information security compromise.

As a general recommendation, message transactions should be encrypted. UDDI is based on SOAP, which as defined, is transport protocol independent. Most commonly, SOAP traffic is transmitted over HTTP-based protocols. The usage of SSL/TLS[§§] satisfies this protection requirement, but XML Digital Encryption could also be used.

Additionally, in order to protect again message replays, unique message identifiers should be used. This is defined within the WS-Addressing specification.

The usage of SSL/TLS does not always guarantee message integrity though. Often, SSL reverse proxies are used in protection environments for load balancing purposes. These can also be used for malicious purposes. By requiring XML digital signatures on both requests and responses, message integrity can be validated and confirmed by both parties, especially if an X509 PKI[‖‖] Infrastructure is in place and used for the signatures.

7.2. Access control

UDDI explicitly states that authentication and authorization are optional. When implemented, the user authenticates to the Security API, and then uses an issued token (referred to as authToken in the spec) within the body of all subsequent request messages. Many of the commonly used XML-based standards for user authentication and authorization, such as SAML, XML Digital Signatures, WS-Username, *etc*., are all within the message header. This helps facilitate the separation of application code and security code. Adoption for the previously mentioned standards is high both with the Java and Microsoft based technologies. Therefore, in order to enhance the UDDI security profile for authentication, the recommendation is to remove Security API and all references to the "authToken." Authentication should be supported *via* an implementation or instance specific profile within the SOAP message header. For example, if WS-Trust or SAML is to be supported, a Secure Token Service (STS) should also be included within the implementation and documented as such [26]. As an added bonus, an STS can be used to support multiple authentication schemes, all of which translate user identity to a common SAML format.

All of the previous renditions of UDDI included support for anonymous access and it should also be optionally supported with a secured version of UDDI. The difference is in authorization. The specification leaves authorization to be optional and even states that describing item level policies is difficult [25]. A secured UDDI instance should have a strong authorization policy that utilizes group, role, attribute, or list-based access control. Anonymous access should be optional as per site and instance requirements.

The proposed recommendation is to explicitly define the requirement and usage of a strict access control model that is capable of applying and enforcing group, user, attribute, or role-based access that will be applied to data objects for all API interfaces, as well as any other user interface provided by the registry[¶¶] [31]. The suggested approach is defined in the following passages.

For web service interactions, authorization should be optional. If the request that does not include a supported identity token, the request shall then have the same rights as an anonymous user which may or may not be allowable at the given UDDI node. Such a decision should be instance specific. Requests with a qualifying token shall be verified for validity, authenticated, and checked against all applicable policies and access control mechanisms for the requested UDDI data (service, business, binding template, *etc*.).

In a related work, XACML-based policies were used to control access to an UDDI registry [27]. This can also be done using a rules based engine that leverages the SAML Protocol (SAML-P) specification in which an entity can execute an Access Decision Query for a particular resource. This can be directly applied to UDDI in which the resource name is the UDDI key for the requested element. With this model, an implementation could be very modular.

If an application GUI is provided, all of the same rules should also be used for authentication and access control. This can also aid in supporting federated users *via* a single sign on mechanism, such as WS-Federation or WS-Trust. All GUI components must utilize the web service APIs to provide a single point of entry and policy enforcement. Although the use of usernames and passwords is the most common, it should be discouraged with the recommended approach of using PKI certificates. The US Department of Defence explicitly states that PKI certificates are to be used over usernames and passwords for authentication.

"The use of userids and passwords may lead to compromise of the userid and password, thus providing access to unauthorized individuals. . . Per the DoDI 8520.2 all private web servers are required to request a subscriber certificate.." [32].

In order to maintain compatibility with older clients, anonymous access should optionally still be supported. This access would provide access to only information specified by the owner of the registry instance and the underlying data.

7.3. Authentication

The problem with exposing data on a network is that it can become vulnerable to attack and exploitation. The need for proper authentication and authorization for controlling

[§§] Secure socket layer/transport layer security.

[‖‖] Public key infrastructure.

[¶¶] Many UDDI registry implementations have other web services and user interfaces outside of the specification. These must also be protected.

[‖‖] Apache CXF 2.2.4.

access to resources, such as a web service, is clear. There have been a number of other related publications that specifically address authentication and authorization techniques for general web services.

Authentication for web services can be performed many different ways. REF _Ref246057345 \h MERGEFORMAT Table IV shows a variety of authentication mechanisms that are commonly found with web services. All of these mechanisms are outside of the SOAP message body.

Moving forward, one of the most versatile and flexible authentication models to use is the SAML, coupled with the usage of a WS-Trust based STS. This implies the replacement of the UDDI Security API with an STS-based web service. The STS can then act as the authentication point using a variety of authentication mechanisms in a secure fashion (SSL recommended). The STS then returns an appropriate formatted SAML Assertion matching the user identity of the UDDI client. The client then uses the provided assertion, which is digitally signed by the STS, as the authentication token to the remaining UDDI APIs. The remaining UDDI services are then configured to trust the STS as a token issuer (and only the STS). Through these mechanisms, a secured transaction can be implemented with flexible authentication mechanisms. As an added bonus, the WS-Trust specification is well supported by both Java[¶¶] and Microsoft[***] based platforms. This is the recommended approach for a secured UDDI implementation. Implementers of course, can always choose to use alternate approaches, authentication mechanisms, and access control techniques. It is recommended to utilize WS-Security based mechanisms within the SOAP message security header for the transmittance of user identity and security related information, not *via* the SOAP message body artifact, "authInfo." SSL or XML Encryption should also be used in order to ensure the privacy of message transactions.

8. PERFORMANCE IMPACTS

Over time, there have been many papers that discuss the performance impacts of web service security utilizing message level and transport level security. During experimental research, these elements were combined with a real world commercial UDDI server provided by BEA Systinet v6.5. Since this particular product only offers the traditional username/password support *via* the "authToken," a proxy service was created using the Windows Communication Foundation (WCF) in C#.

This particular proxy service essentially acted as a security enforcement point for the UDDI instance. The SAML[†††] Token Profile 1.1, a portion of the WS-Trust specification,

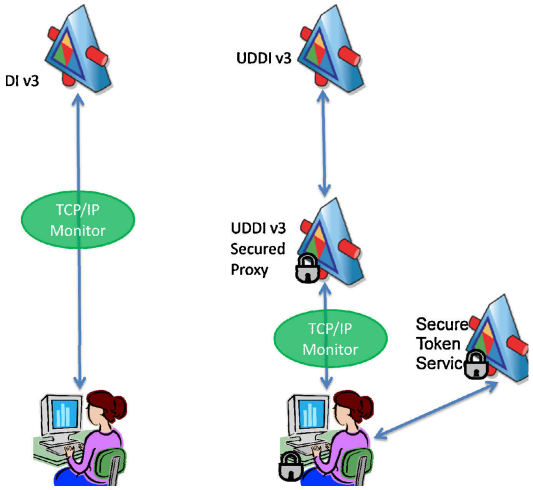

Figure 5. Message level security overhead in bytes.

was utilized. This added the requirement for clients first accessing an STS for authentication *via* X509 Certificate in order to obtain a qualifying SAML assertion. This assertion was then included with the request message along with an XML Digital Signature and an encrypted public key certificate from the client. The proxy service then validated the XML signatures and ensured that the SAML assertion was used by the trusted STS, all of which was performed utilizing built in features of the WCF framework. In a production environment, an additional component could be added to control access to specific resources within UDDI, such as business elements, data fields, *etc*. The proxy service then forwarded the original request to the UDDI server. On the return trip, responses from UDDI were signed with an X509 certificate and returned to the client and verified.

During this experiment, two key items were recorded *via* the use of a TCP/IP[‡‡‡] Monitor: message size and latency. Each of the UDDI Inquiry API methods listed in Figure 6 was executed 100 times for consistency. As depicted in Figure 5, the TCP/IP Monitor was used to intercept and record XML SOAP message transactions as recorded on the wire.

8.1. Message size

It was recorded that on average, the additional security information creates a message size increase by approximately 27 KB. This represents the actual messages to and from the client to the UDDI server, including the Secure Token Request (STR) messages to and from the STS.

[¶¶] WCF .NET Framework 3.0 or higher.

[***] Security assertion markup language.

[†††] Security assertion markup language.

[‡‡‡] Transmission control protocol/internet protocol.

Table IV. Example web service authentication methods.

Layer	Authentication	Authorization
Transport	HTTP basic, HTTP Digest, HTTP NTLM, HTTP client certificate, Kerberos	Access control lists, PKI certificates (only allow trusted certificates) Role/group base control lists
SOAP message	WS-username, SAML*, PKI certificate, XML signatures (proof of ownership)	Access control lists, PKI certificates, Attributes, roles, or group based access control, XACML**, SAML-protocol

*Security assertion markup langauge.

**XML access control markup language-OASIS standard.

8.2. Latency of security processing

During our latency test, several unexpected observations were found. When instantiating the client object that performed the service request and then subsequently executed the web service API method repeatedly, it was noted that the WCF framework keeps the connection to the web server open. This eventually led to the testing of four different profiles, Secure Persistent, Secure Terminal, Unsecure Persistent, and Unsecure Terminal. The persistent cases were when the service proxy object within the client's code was reused for each iteration of execution. The terminal cases were when this same object was released from memory and recreated.

In the Secure Persistent profile, an initial spike in execution time (approximately 78 ms) was noted. The response time then averaged a respectable 22 ms. The Secure Terminal profile averaged 68 ms for every execution. The additional delay for the Secure Terminal profile was later confirmed to be related to the caching of the SAML token provided by the STS. This was a significant difference.

The actual data is presented in Figure 7, which clearly shows the initial spike from the STS transaction. Each time the connection was terminated and reopened, the client requested a new token from the STS, thus the additional 50 ms delay. The unsecure profile averaged 4.8 ms for persistent connections and 5.25 ms for terminal connections. This was most likely due to the opening and closing of sockets. In the figure, there is obvious spiking for all transactions. The source of this spiking was never accurately determined, but it was most likely caused by disk I/O.

The most relevant calculation was the comparison of Secure Persistent to Unsecure Persistent. This scenario represents a real world use case of UDDI. In many cases, multiple calls to UDDI may be necessary in order to obtain the required information. On average, service call outs to the secured UDDI server took 3.7 times longer than to the unsecured UDDI server at 22 ms. This was calculated using the following equation.

$$\frac{\text{Average(Secure Persistent)} - \text{Average(Unsecure Persistent)}}{\text{Average(Unsecure Persistent)}}$$

The reason for subtracting the unsecure from the secure average was because for this evaluation, a proxy service was used to simulate a real environment, thus the time it took the UDDI server to respond is not important, only the security overhead was of interest. Overall, the average latency for the Secure Persistent profile was 22 ms. This finding is clearly visible in Figure 8 which depicts the average latency for the four profiles.

Although there was a clear difference in performance between the secure and unsecure profiles for UDDI, the time delay was minimal. The average of 22 ms over 4 ms was hardly something that an end user would notice. In this case, the ends justified the means for securing UDDI with message level security. The performance impacts outweigh the risk of leaving an important aspect of SOA open to vulnerabilities.

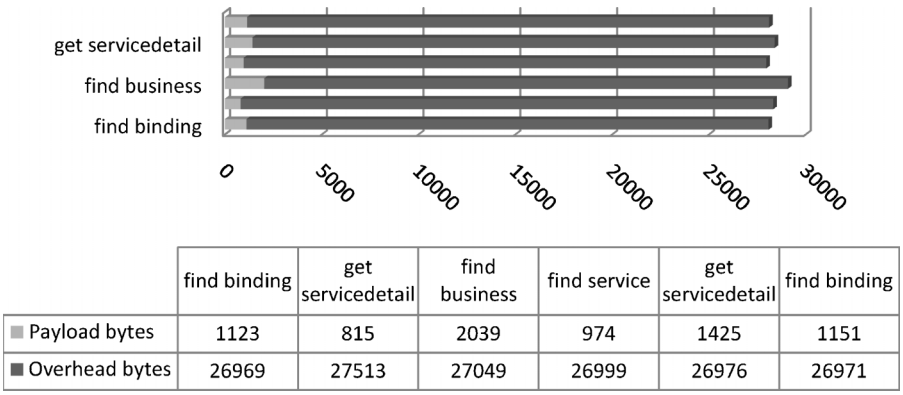

Figure 6. Evaluation architecture.

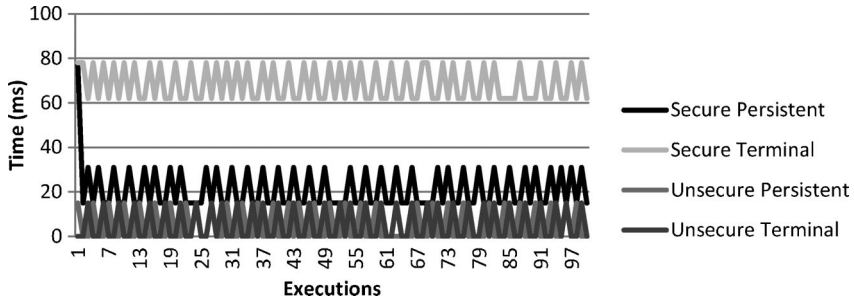

Figure 7. Message security latency comparison, find_business.

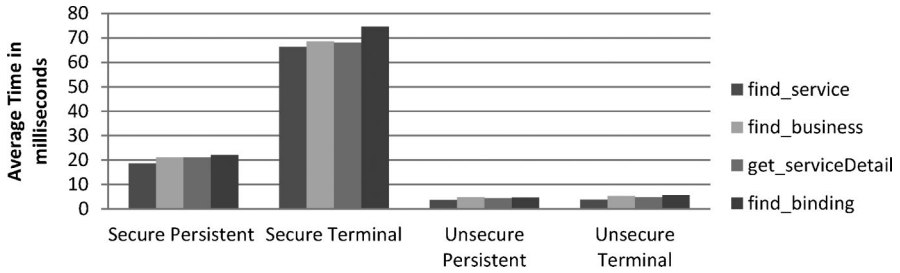

Figure 8. Message level security latency comparison in bytes.

9. ADDITIONAL ENHANCEMENTS

UDDI is a set of web service definitions (WSDL), data schemas (XSD), and documentation describing the behaviors and usage of an implementation of the specification. From the documentation, the usage of various data types such as "findQualifiers" is quite specific. During an Inquiry callout, the request message can have none or more findQualifiers defined which can only have one of the values defined in the documentation. The schema, however, defines a findQualifier as a generic string. This forces developers to hard code a set of constant strings inside application code and thus prone to typographic error. A more efficient approach would be to have an enumerated type defined matching the set of acceptable values. This same issue can be seen with other data types within UDDI, such accessPoint's useType. Simple inconsistencies and poorly defined schemas greatly increase the amount of effort required for client integration.

In practice, performing what should be the simplest of tasks, EPD, can actually be quite difficult in order to properly handle all cases. For example, for each service registered within an UDDI registry, the entry can have multiple binding templates. For each binding template can either have an accessPoint or a hostingRedirector (deprecated). Each accessPoint has a value, which is typically a URL and an optional attribute of useType. The useType can either be an endPoint, bindingTemplate, hostingRedirector, wsdlDeployment, or something else. In a real world instance of UDDI utilizing the commercial product, HP Systinet, the value of the accessPoint, although typically is an URL, can also be a Java classpath. The attribute useType is also commonly specified as the namespace of the SOAP version used for the given service which is not one of the recommended types. As one can imagine, trying to parse of all of these possibilities and correctly processing them can be challenging.

Clearly, the correct path forward is to refine the XSD of UDDI in order to clearly define what should or should not be used for a specific data element using enumerated data types. This is a relatively simple change which will greatly ease integration and usage issues. Additionally, it should actually increase the amount of generated code *via* some WSDL to code utility such as Java Axis's wsdl2java or Microsoft's svcutil.exe, and decrease the amount of developer written code which thus reduces development time and cost.

10. CONCLUSIONS

UDDI offers some great capabilities, but when examining the specification's recommended security model, it is clear that the specification should either be revised or extended to include new techniques to prevent information assurance and security related problems. Moving in a forward direction, the UDDI specification should be changed in order to meet new needs and capabilities. A stronger, yet more agile security design is necessary in order to promote wider adaption, implementation, and usage. The preceding referred to a number of standardized specifications and security adaptations that if implemented correctly, would successfully mitigate many of the inherent security concerns presented by the UDDI specification. The summation of these changes

should either be proposed as an extension to UDDI or incorporated into the specification itself. It is up to the UDDI Technical Committee to decide upon these recommendations in order to move the specification forward.

REFERENCES

1. W3C. (2007, April) World Wide Web Consortium. [Online]. Available at: www.w3.org/TR/soap
2. Curbera F, Duftler M, Khalaf R, Nagy W, Mukhi N, Weerawarana S. Unraveling the Web Service Web: An Introduction to SOAP, WSDL and UDDI. *IEEE Internet Computing* 2002; **6**(2): 86–93.
3. World Wide Web Consortium. (2007, April), Simple Object Access Protocol. [Online]. Available at: www.w3.org/TR/soap
4. Epstein J, Matsumoto S, McGraw G. Software Security and SOA. *IEEE Security and Privacy* 2006; **4**(1): 80–83.
5. Robinson SH, Knauerhase RC, Milenkovic M, *et al*. Toward Internet Distributed Computing. *IEEE – Computer* 2003; **36**(5): 38–46.
6. OASIS. OASIS – Organization for the Advancement of Structured Information Standards. [Online]. Available at: www.oasis-open.org/who
7. OASIS. (2007). OASIS Webinars. [Online]. Available at: www.oasis-open.org/events/webinars/2007-06-04-ebXML-Registry-and-Repository.wmv
8. OASIS. (Feburary 2007) OASIS. [Online]. Available at: www.oasis-open.org/committees/download.php/23648/regrep-3.0.1-cd3.zip
9. OASIS. (2009, September) Content Management Interoperability Services (CMIS). [Online]. Available at: www.oasis-open.org/committees/cmis
10. Fielding RT. 2000; Architectural Styles and the Design of Network-based Software Architectures. *Doctoral Dissertation, Department of Computer Science*, [Online]. Available at: www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
11. WSO2. WSO2. [Online]. Available at: wso2.org
12. Alor-Hernandez G, Posada-Gomez R, Alberto A. Aguilar-Lasserre AA, Abud-Figueroa MA. Web services discovery and invocation by using DNS-EPD. In *IEEE Electronics, Robotics and Automotive Mechanics Conference (CERMA 2007)*, Vol. DOI 10.1109/CERMA.2007.24, September 2007; 695–700.
13. ShaikhAli A, Rana OF, Al-Ali R, Walker DW. "UDDIe: An Extended Registry forWeb Services. In *2003 IEEE Symposium on Applications and the Internet Workshops (SAINT'03 Workshops)*, 2003; 85.
14. Zhang L-J, Li H, Chang H, Tian C. XML-based advanced UDDI search mechanism for B2B integration. In *Fourth IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS'02)*, 2002; 9.
15. Paul R, Cao Z, Yu L, Saimi A, Xiao B, Tsai WT. Verification of web services using an enhanced UDDI server. In *The Eighth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, 2003; 1–8.
16. Gorbenko A, Romanovsky A, Kharchenko V. How to enhance UDDI with dependability capabilities. In *Annual IEEE International Computer Software and Applications Conference*, Vol. 33, 2008; 1023–1028.
17. Spies M, Schoning H, Swenson K. Publishing interoperable services and processes in UDDI. In *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, Vol. October, 2007; 503.
18. Mikroyannidis A. Towards a Semantic Social Web. *Computer*, Vol. November 2007; 113–115.
19. Yee G, Korba L. Negotiated security policies for E-services and web services. In *IEEE International Conference on Web Services (ICWS'05)*, 2005; 605–612,
20. Peng Q. Secure communication and access control for web services container. *Fifth International Conference on Grid and Cooperative Computing (GCC'06)*, October 2006; 412–415.
21. Trabelsi S, Gomez L, Roudier Y. Context-aware security policy for the service discovery. In *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, 2007; 477–482.
22. Singhal A. Web services security: challenges and techniques. In *Eighth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'07)*, 2007; 282.
23. US Dept of Commerce – National Institute of Standards and Technology, "Guide to Secure Web Services," U.S. Government, Gaithersburg, MD, Special Publication 800-95, 2007. [Online]. Available at: csrc.nist.gov/publications/nistpubs/800-95/SP800-95.pdf
24. Obaidat MS, Boudriga NA. Security of e-Systems and Computer Networks. Cambridge University Press: Cambridge, UK, 2007.
25. OASIS. (2004, Oct.) UDDI Version 3.0.2. [Online]. Available at: uddi.org/pubs/uddi_v3.htm
26. OASIS. (2005, March) OASIS – Organization for the Advancement of Structured Information Standards. [Online]. Available at: www.oasis-open.org/specs/#samlv2.0
27. Dai J, Steele R. UDDI access control. In *Third International Conference on Information Technology and Applications (ICITA'05)*, Vol. 2, 2005; 778–783.
28. OASIS. (2004) XACML Profile for Role Based Access Control. [Online]. Available at: docs.oasis-open.org/xacml/cd-xacmlrbac

29. Montrose B, Kim A, Khashnobish A, Luo MKJ. Adding OWL-S Support to the Existing UDDI Infrastructure. In *IEEE International Conference on Web Services*, 2006.
30. Howard K, Erickson P. A case of mistaken identity? News accounts of hacker, consumer, and organizational responsibility for compromised digital records. *Journal of Computer-Mediated Communication* 2007; **12**(4). Available at: jcmc.indiana.edu/vol12/issue4/erickson.html
31. Jiancheng N, Zhishu L, Zhonghe G, Jirong S. Threats analysis and prevention for grid and web service security. In *IEEE Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007)*, 2007; 526–531.
32. US Government Defense Information Systems Agency. (24 April 2009) DISA Field Security Operations Web Tomcat Site Checklist. [Online]. Available at: iase.disa.mil/stigs/checklist