TransactionMinder v5.5

Securing Web Services and Business-to-Business Integration Environments

Netegrity White Paper

PUBLISHED: JULY 12, 2002 (REV. 3.O -MC)

Executive Summary

Web Services Overview

In simple terms, a Web service is a standards-based interface to functionality exposed for use over the Internet. Exposed functionality can be a whole business application or an application component designed to do a very specific job. Web services can be exposed by companies through their Web site portal or through partnership agreements with other companies.

Web services can be requested and consumed by users directly, or by business applications through a business-to-business integration server. In the latter case, a Web service can seamlessly be integrated with the requesting application's process to execute a transaction.

In a simple Web service scenario, a user makes a request for a price or the status of an invoice, the receiving Web site processes the request and returns a response to the user. More complex Web services involve multi-step transactions. For example, the user (or a server on behalf of the user) can place a purchase order for a product with an online store. The online store processes the purchase order and returns a response to the user (or server) including the status of the purchase order.

Web services are not a new concept, of course, but what is new is how those services are exposed by providers, and how they are accessed and consumed by requesters. Web services rely on a standard way of communication, based on the eXtensible Markup Language (XML). In this way, Web services consumers and Web services providers can understand each other without resorting to proprietary protocols or specific integration applications.

Typically, in a Web service environment, the Web service provider publishes a description of the functionality it exposes. The Web service description is stored in a registry. The Web service consumer looks up the services offered by providers in that registry (this is known as the discovery step).

Web services are characterized by three factors:

- what they do (the functionality they expose),
- where they are (the Web site which exposes that functionality),
- how they can be accessed (the set of published interfaces necessary to use the exposed functionality).

Web services information is defined in XML documents that are exchanged between consumers and providers, using an industry-standard messaging framework. Because they are based on standards, Web services are independent of operating system infrastructures and programming languages, thus enabling loosely-coupled environments and interoperable solutions across multiple, heterogeneous enterprises.

Web Services Challenges

For all their benefits, Web services are not without challenges. One of them is security.

Today, Web services platforms are limited to basic transport-level security. The XML messages that are exchanged in Web services requests and responses must include security information that goes beyond the transport layer.

Transport-level security enables point-to-point sessions. However, in Web services environments, many intermediaries can be involved in a transaction. For example, XML documents may be routed through various servers and they can be processed by multiple backend business applications necessary to complete a transaction. Each intermediary involved in a transaction can be a Web service in its own right. Intermediaries require security information from incoming requests and may need to provide additional security information to the next intermediary involved in the transaction process. Transport-level security by itself falls short of such requirements.

What is needed in addition to transport-level security is a way to provide security information for the XML messages exchanged in Web services and other business-to-business transactions. In other words, in addition to securing the communication of messages (which transport-level security does well), we need to be able to selectively secure the content of the documents used in that communication, thus providing support for fine-grained access-control and dynamic authorization decisions.

Some companies address access-control and authorization issues by implementing security in business applications used to deliver Web services, thus creating security "silos." At best, implementing security in each application or each Web services platform significantly increases the cost of application development and environment administration. At worst, it increases the risk of a security breach due to possible inconsistencies in the level of security implemented in each application.

Introducing TransactionMinder

TransactionMinder provides a policy-based platform for securing the content of XML documents and messages used in Web services and market-leading, XML-based, business-to-business integration (B2Bi) infrastructures.

TransactionMinder builds upon Netegrity's shared-services vision embodied in SiteMinder, Netegrity's flagship product for securing access to Web-based documents and business applications within a single enterprise and across multiple partners.

TransactionMinder is a product in its own right. In addition, TransactionMinder is designed to be installable on an existing SiteMinder environment to support XML security.

TransactionMinder Customer Benefits

- **Shared security services –** Centralized, policy-based authentication and authorization, profile management, and auditing services.
- **Integration of Web services frameworks and environments –** Single platform to securely manage a wide variety of Web services environments. Support for market-leading Web servers and application servers, as well as custom environments.
- **Industry-standard content-level authentication schemes** – Support for user credentials embedded in XML documents, XML Digital Signature, and the Security Assertion Markup Language (SAML).
- **Central management of user information and entitlements** – Support for user and trading-partner directories stored in Lightweight Directory Access Protocol (LDAP) environments or relational databases. Support for delegated administration through Netegrity's forthcoming IdentityMinder product.
- **Fine-grained access control** – Allows for authorization policies based on information provided at any layer of the XML message (transport, envelope, or business payload).
- **Seamless integration with existing SiteMinder-enabled sites** – Provides single sign-on between Web sites and Web services through a common agent framework.

Web Services In Action

Note: This section describes Web services concepts necessary to understand the security challenges and the solution provided by TransactionMinder. Feel free to skip this section if you're already familiar with the technologies used in Web services implementation, deployment, and access.

As mentioned earlier, a Web service can be simply defined as an interface to business logic exposed by a company for use over the Internet by its employees, customers, suppliers, and partners.

Typically, a Web service provides three functions:

1. Listen for requests from clients (or "consumers")
2. Translate requests for internal use by the application (or component) that's going to process the request
3. Return the result of the request in a response to the client

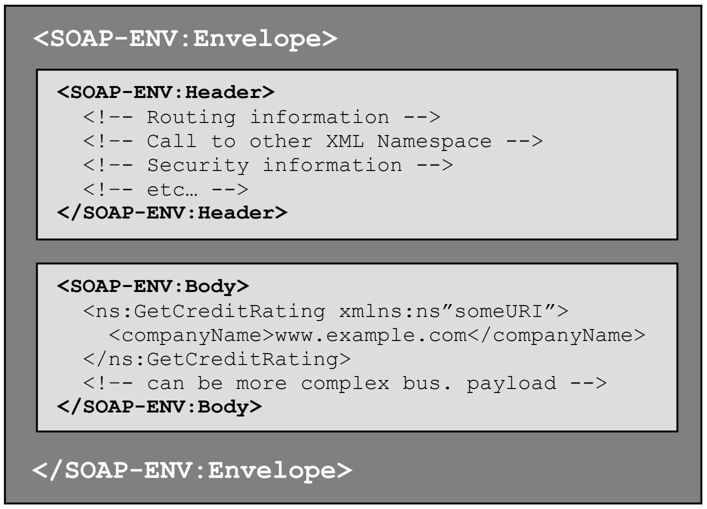
Before being exposed, a Web service must first be deployed. In other words, the request listener at the Web service provider's site must know what types of requests are expected from clients, and how to route those requests to the backend application for processing.

Deploying Web Services

Web services use XML for communication between consumers and providers. This means that requests to services and responses to clients are carried out using XML documents. These XML documents can be "packaged" in a messaging framework allowing Web services consumers and Web services providers to exchange information over standard transport protocols.

The metaphor used for describing the messaging framework is an envelope that includes information about the XML document that it contains. The messaging framework that is emerging as an industry standard is called Simple Object Access Protocol (SOAP). SOAP is itself an XML application (i.e., SOAP is defined by an XML Schema).

The following figure describes the structure of a SOAP message.



```
<SOAP-ENV:Envelope>

<SOAP-ENV:Header>
  <!-- Routing information -->
  <!-- Call to other XML Namespace -->
  <!-- Security information -->
  <!-- etc… -->
</SOAP-ENV:Header>

<SOAP-ENV:Body>
  <ns:GetCreditRating xmlns:ns"someURI">
    <companyName>www.example.com</companyName>
  </ns:GetCreditRating>
  <!-- can be more complex bus. payload -->
</SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

SOAP messages can be transmitted between consumers and providers over several standard transport protocols. The HyperText Transport Protocol (HTTP) is the most popular transport protocol used to exchange SOAP messages over the Internet. The SOAP specification defines a `SOAPAction` HTTP header for a SOAP request, used by Web servers to locate SOAP requests.

Following is an example of a SOAP request / response transmitted over HTTP using a POST method.

```
POST /CreditRating HTTP/1.1
Content-Type: text/xml
Content-Length: nnnn
SOAPAction:"someURI:CreditRating#GetCreditRating"

<SOAP-ENV:Envelope>
  <!-- request -->
</SOAP-ENV:Envelope>

HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn

<SOAP-ENV:Envelope>
  <!-- response -->
</SOAP-ENV:Envelope>
```

Web services are deployed using tools designed for specific environments. For example, in the Java world, the Apache implementation of the SOAP specification is very popular. Apache SOAP is a servlet that runs in a Java Web Container (such as Apache's Tomcat).

Three parts are necessary to deploy a Web service in a Java environment: The Web service server (the functionality exposed by the Web service provider), the deployment descriptor (an XML file defining information for the SOAP server to handle client requests), and the Web service client (a Java class designed to invoke the Web service).

Consuming and Publishing Web Services

Once Web services are deployed by Web services providers, two standards-based technologies are used to bring Web services consumers and Web services providers together: Description and Discovery.

Description

When you deploy Web services to consumers, you need to provide information on the functionality that the Web service exposes. Before the advent of Web services, this information had to be encoded in the application itself, or in proprietary protocols used to access it. Web services support an industry-standard XML framework known as the Web Service Description Language (WSDL), which standardizes the way of describing what a Web service does. For example, a WSDL document can give information about:

- what method of the business application needs to be invoked to fulfill the request of the Web service consumer (e.g., the `GetCreditRating` method of the CreditRating application, in our earlier example),
- how the method is implemented (we saw earlier in the document that a SOAP message requires a SOAP action that specifies where the method invoked is located).

WSDL documents are generated automatically by the Web services toolkit that you use to deploy the Web service. However, like SOAP, WSDL has no built-in security capabilities. In other words, no security elements are defined in the WSDL XML schema, so security has to be handled elsewhere in the process.

Discovery

Now that we have described the Web service using WSDL, we need to let consumers know that the Web service exists. Very much like we use the telephone *Yellow Pages* when we are looking for professional services such as plumbing or roofing, Web services consumers look up registries that include information about the Web services offered on the Internet, together with their location. This is known as the discovery step.

Discovery is defined by an XML framework called Universal Description, Discovery, and Integration (UDDI). A UDDI registry allows companies to make their Web services known to the public using XML documents.

- A UDDI business entity document includes information about the company together with a list of the Web services exposed by the company.
- A UDDI business service describes each Web service listed in the business entity document, based on the WSDL information.

UDDI services provide SOAP interfaces used by consumers (to query Web services) and providers (to publish Web services).

UDDI uses a lot of information described by WSDL, so it’s important that they both work well together to avoid inconsistencies or redundancies.

Web Services Security

As seen above, XML-based Web services technologies were not designed with security as a primary requirement.

As mentioned earlier in this document, Web services platforms are limited to basic transport-level security, such as the Secure Socket Layer (SSL) used over HTTP (HTTPS). HTTPS provides technologies based on cryptography (or key infrastructures) that support:

- Confidentiality: The sender encrypts the message with a public key, the receiver decrypts it with a private key.
- Authentication: The sender encrypts the message with a private key, the receiver decrypts it with a public key (thus identifying the sender by its unique private key).
- Data Integrity: The data sent arrives at its destination unaltered using digital hashing, i.e., a “digest” representing the full document.

Web services are predicated on the use of the XML specification, in particular the SOAP messaging framework, and the XML documents contained in SOAP envelopes. As a result, the XML messages that are exchanged in Web services requests and responses must include security information that goes beyond the transport layer and applies to the content of each XML message.

XML security is based on the same principles of confidentiality and data integrity described above, and is defined in two seminal XML specifications:

- XML Encryption: Represents the encrypted content of XML data (a whole document or selected parts thereof) and the information that enables a recipient to decrypt it (keys involved in the encryption). Using the `<EncryptedData>` element (defined in the XML Encryption specification), one can define which data in the XML document needs to be encrypted, as well as the encryption value.
- XML Signature: Provides detailed elements supporting data integrity, signature assurance, and non-repudiation for Web services data.

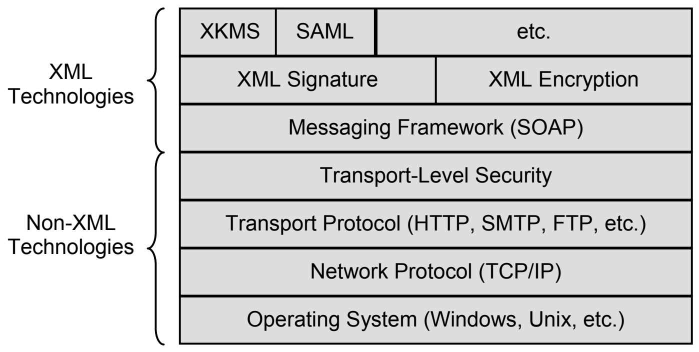
Additional XML-based frameworks use XML Encryption and XML Signature to provide higher levels of security.

- XML Key Management Specificiation (XKMS): Provides a standard interface to proprietary PKI implementations, thus promoting interoperability between PKI vendors. XKMS is designed to be used in conjunction with the XML Signature and XML Encryption specifications.

- Security Assertion Markup Language (SAML): Describes authentication and authorization objects (or “assertions”) exchanged between partners or between applications within the same enterprise. SAML can be used in Web services transactions to define security information relative to the SOAP envelope body (or “payload”). SAML information is inserted in a SOAP envelope header. SAML works in conjunction with the XML Signature specification.

WS-Security is another emerging Web-services security effort spearheaded by Microsoft and IBM. WS-Security is a set of specifications that provides integrity and confidentiality to SOAP-based messages by bringing together security technologies that were not designed to be compatible, such as public key infrastructures (PKI), Kerberos, SAML, etc. Basically, WS-Security specifes how security information is defined and inserted in a SOAP envelope `<security>` header. The security information can be based on various security mechanisms such as digital certificates, Kerberos, or (conceivably) SAML assertions.

The XML security stack is summarized in the figure below.


XML Technologies
Non-XML Technologies
XKMS
SAML
etc.
XML Signature
XML Encryption
Messaging Framework (SOAP)
Transport-Level Security
Transport Protocol (HTTP, SMTP, FTP, etc.)
Network Protocol (TCP/IP)
Operating System (Windows, Unix, etc.)

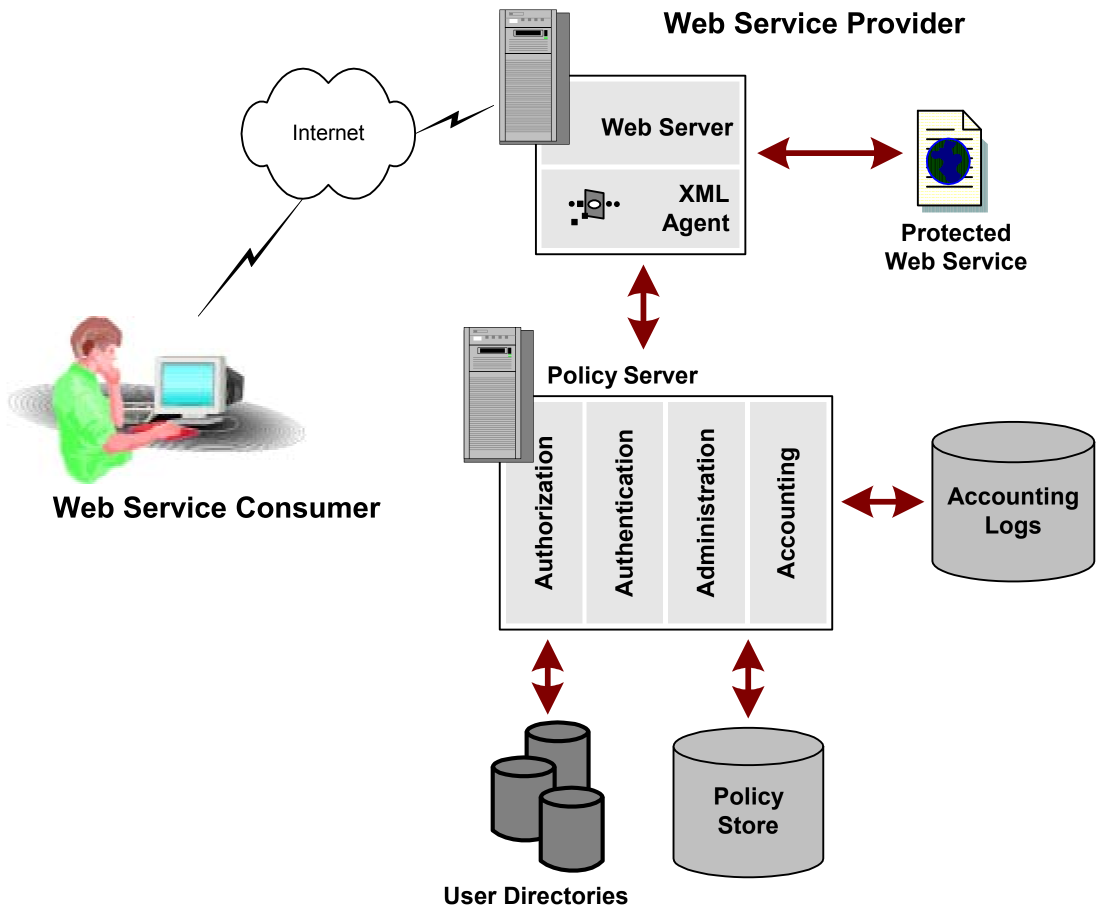
Netegrity provides a new platform supporting content-level, XML-based security called TransactionMinder.

The TransactionMinder Solution

TransactionMinder provides a policy-based platform for securing the XML documents and messages used in Web services and (later) market-leading, XML-based, business-to-business integration (B2Bi) infrastructures. TransactionMinder is designed to be independent of the transport protocol and messaging framework used.

TransactionMinder Overview

The figure below shows TransactionMinder at the Web service provider's site. TransactionMinder is built on SiteMinder's infrastructure, using specific XML agents in conjunction with the SiteMinder Policy Server.



XML Agents

The TransactionMinder XML agent is a component built upon the existing SiteMinder Web agent. The XML Agent is tightly integrated with a variety of Web servers and (later) B2Bi environments. The XML agent intercepts the XML messages sent to a Web service protected by TransactionMinder, and interacts with the Policy Server to execute a set of shared services for protecting and managing the Web service.

The XML Agent uses an XML application programming interface (API) exposing an extensible architecture for dealing with various XML message standards. In addition, the XML API allows for code reuse across all of the XML agents, as well as integration with custom Web services environments.

Policy Server

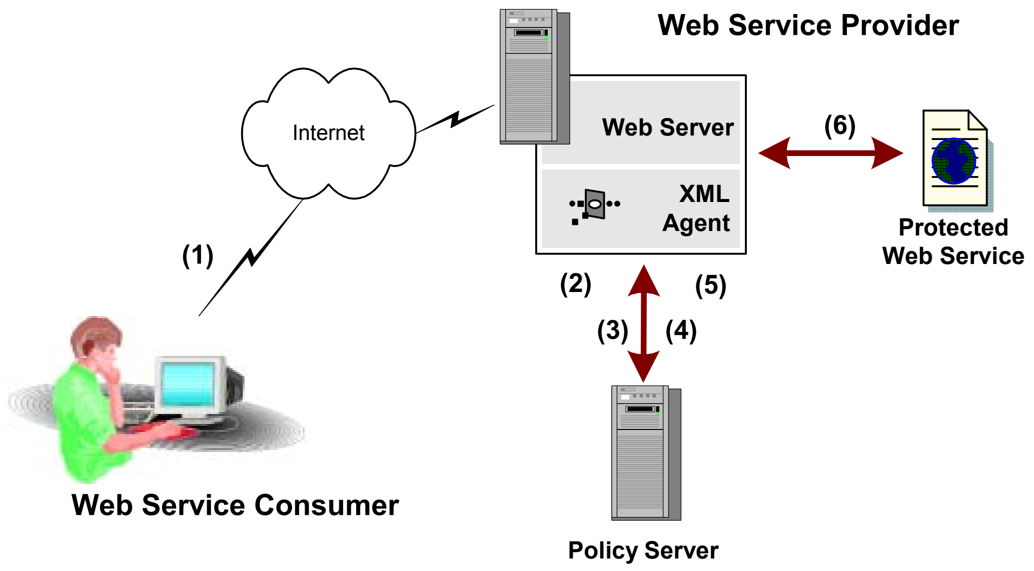
The Policy Server is the centerpiece of the centralized, policy-based management platform. TransactionMinder uses the same Policy Server as SiteMinder, with additional features designed to support TransactionMinder-specific functionality. The Policy Server integrates with TransactionMinder's XML agents as well as Netegrity's other products and agent types to provide a single platform for securely managing every aspect of a company's e-business.

The Policy Server hosts the set of shared services delivered as part of the TransactionMinder solution. Its extensible and scalable architecture allows services to be added and enhanced as the security and management needs for Web services evolve.

The Policy Server integrates with industry-standard Lightweight Directory Access Protocol (LDAP) implementations as well as relational database systems for centralized management of user identity and entitlement information. This information is used by the Policy Server for authentication and authorization.

TransactionMinder Process Flow

This section describes an example of the steps for securing Web services with TransactionMinder.



1. The Web service consumer uses the TransactionMinder Client Toolkit to create the SOAP request message (see details on the TransactionMinder Client Toolkit in the next section)
2. At the Web service provider's site, the TransactionMinder XML agent intercepts the request based on the content type of the HTTP header (`text/xml`) as shown in the code snippet below.

```
POST /CreditRating HTTP/1.1
Content-Type: text/xml
Content-Length: nnnn
SOAPAction:"someURI:CreditRating#GetCreditRating"

<SOAP-ENV:Envelope>
  <!-- request -->
</SOAP-ENV:Envelope>
```

3. The XML agent gathers the credentials from the SOAP message and authenticates the user based on the required authentication scheme.

4. The XML agent checks the sender's authorization for the payload request (for example a purchase order).
5. If the sender is authorized, the XML agent may optionally insert authorization information (entitlements) into the SOAP message.
6. The authorized message is passed on to the backend business application for processing. The backend application can optionally return a response to the Web service requester with the status of the payload (for example, indicating that the purchase order has been accepted and is being processed).

TransactionMinder Authentication Schemes

TransactionMinder supports three message-level authentication schemes specifically designed to secure the XML documents used in Web services and other business-to-business transactions:

- XML document credential collector
- XML Signature
- SAML Authentication

XML Document Credential Collector

In this authentication scheme, the security information may be contained in the envelope header or the business payload itself. This is the case when two business partners have agreed on a specific XML vocabulary for exchanging business documents. For example, a large online computer vendor may place orders for computer components with its partners using a specific XML format for purchase orders.

TransactionMinder's XML document credential collector allows the Web service provider to define how the Web service consumer's credentials are gathered and processed. The Policy Server provides the query to search for security information, encoded in an XML technology called XML Path (or XPath). Once collected, the credentials are checked against the user store used by the Policy Server.

XML Signature

Digitally-signed XML messages require a user directory containing identifications and public keys. The Policy Server uses these keys to verify that the message is unaltered and authenticated through the private key of the Web service requester.

Digitally-signed messages may also use an X.509 certificate (possibly provided at the transport level in the case of an HTTPS connection). In this case, the certificate provides a third-party guarantee that the binding of the requester's identity to the public/private key pair is accurate.

The TransactionMinder XML agent verifies the signature and passes the certificate to the Policy Server. The Policy Server is configured to map the Distinguished Name (DN) field of the certificate to the user store information. The Policy Server then ensures certificate validation.

SAML Authentication

In this case, the incoming XML message intercepted by the TransactionMinder XML agent includes SAML assertions containing the credentials to be used for authenticating the Web service consumer. SAML assertions may or may not be digitally signed.

SAML assertions can be inserted in SOAP envelope headers or HTTP headers (TransactionMinder supports both cases). Typically, SAML assertions are issued by an "asserting authority," i.e., a server that can reliably generate SAML assertions on behalf of the user. When the TransactionMinder XML agent discovers SAML security in the incoming message, it proceeds to authenticate the SAML assertion issuer.

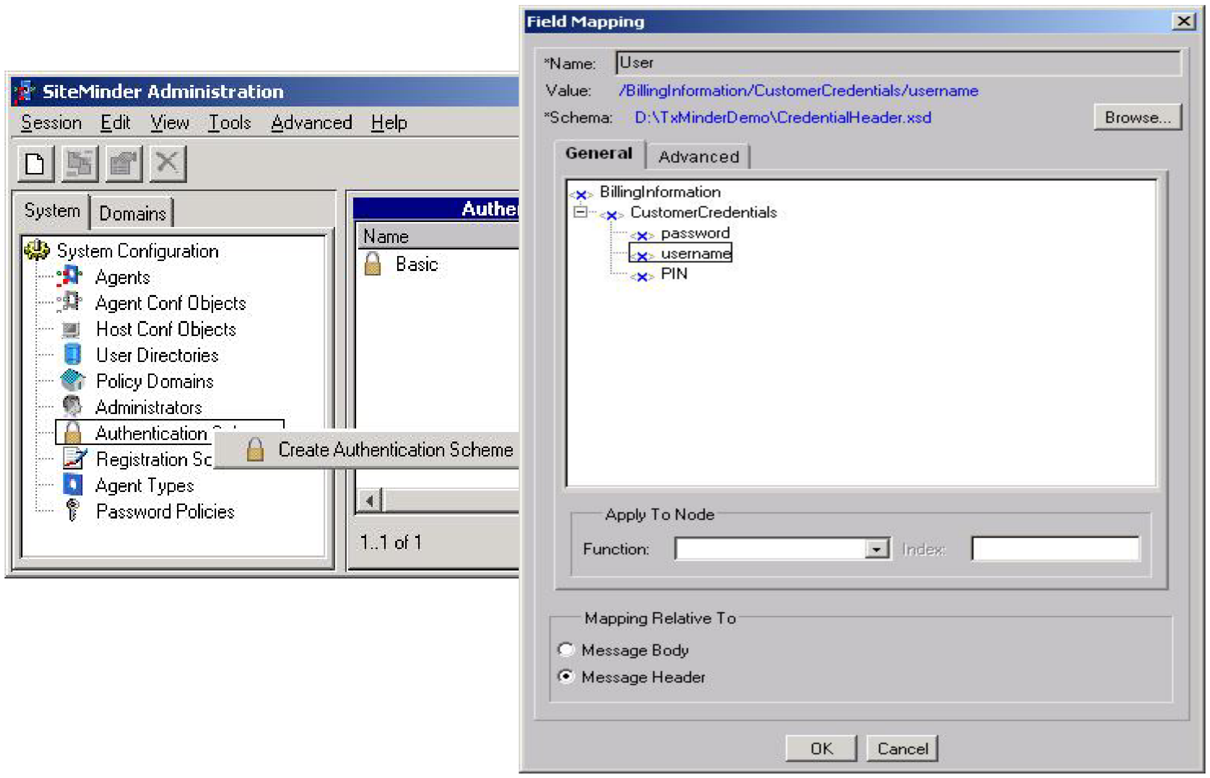
TransactionMinder In Action

This section describes a typical TransactionMinder usage scenario.

- Configuring the authentication scheme from the TransactionMinder administration console
- Setting up an authorization policy based on the SOAP message
- Constructing and POSTing the SOAP message using the TransactionMinder Client Toolkit

Creating The Authentication Scheme

The screenshots below show the creation of an XML Document Credential Collector authentication scheme. We map the credentials that will be used to authenticate the user at runtime.



*Name: User
Value: /BillingInformation/CustomerCredentials/username
*Schema: D:\TxMinderDemo\CredentialHeader.xsd

Mapping Relative To
Message Body [✓ Message Header]

The Web service provider defines the format of a SOAP header that contains the credentials. In this example, we use a username / password authentication scheme, but we could use other variations including a PIN as shown in snapshot above.

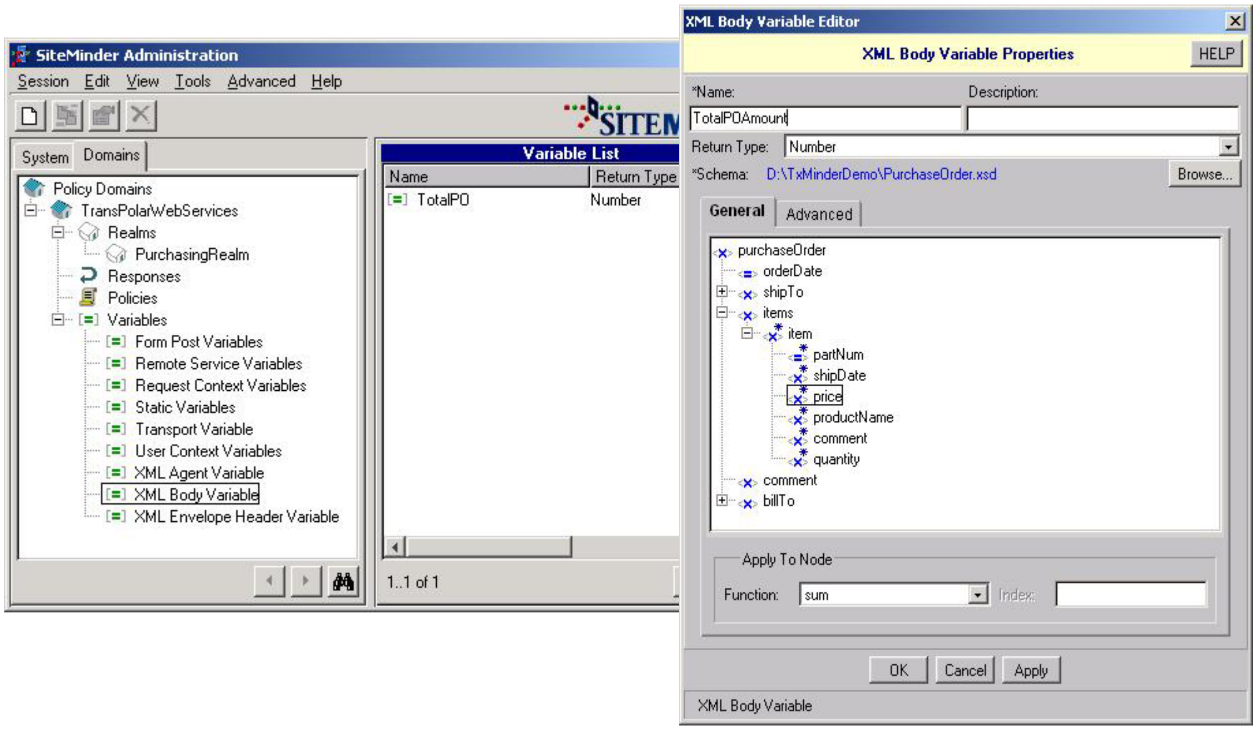
To configure this authentication scheme, you need to open the schema for the SOAP header and choose the particular field that contains the data that will exactly map to the information of the user context (user name / password in this case). In the snapshot above, we choose to put the user context in the SOAP envelope header. That information could also be in the SOAP envelope body.

Creating Authorization Variables

Authorization variables are based on SiteMinder's *eTelligent* rules. *eTelligent* Rules include a set of services designed to enhance the SiteMinder Policy Server's authorization functionality.

With *eTelligent* Rules, security administrators define specific security logic using graphical editors that don't require custom development. The purpose of such security logic is to dynamically gain access to attributes about the user requesting a protected resource, the resource itself, and the remote services optionally invoked in the authorization-decision process.

Security administrators can use different types of variables in order to create security rules based on Boolean expressions evaluated at runtime. If a Boolean expression is evaluated as true by the Policy Server, the security policy to which it is attached is enforced.



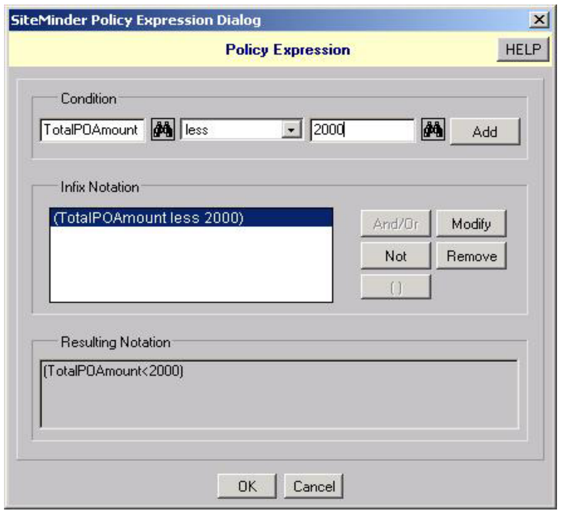
Name	Return Type
TotalPO	Number

*Name: TotalPOAmount
Return Type: Number
*Schema: D:\TxMinderDemo\PurchaseOrder.xsd
Function: sum

In this example we're creating an XML Body Variable. The Web service provider has defined the format of the business payload in an XML schema. In this case, the business payload is a purchase order (PO). We pick the desired field from the PO XML schema ("price", in our example) that will be evaluated at runtime.

We use XPath to select elements in the XML document, very much like database administrators use the Structured Query Language (SQL) to search a relational database. Using the Advanced tab, you can directly type in the XPath expression if it is very complex.

The following screenshot shows a policy expression.



TotalPOAmount less 2000

(TotalPOAmount less 2000)

(TotalPOAmount<2000)

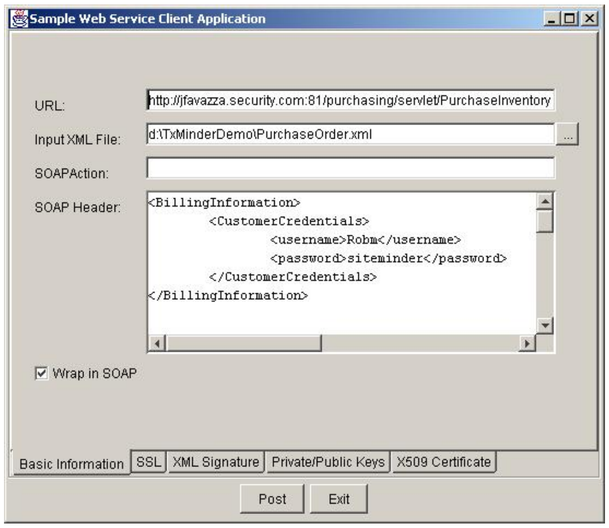
In this example, we use a variable called `TotalPOAmount`. The expression describes the limit of the PO amount ($2,000). If this expression is evaluated to true (at runtime), the policy fires, thus providing the authorization to process the PO. The PO is then handed over to the backend business application that will process it and a message will be returned to the user indicating that the PO has been accepted by the Web service provider.

Defining the Web Service Request

TransactionMinder provides a Client Toolkit (TK) that allows Web service consumers to make requests to the Web service provider.

There are several toolkits available to make Web services requests. You can also build you own utility in-house. The TransactionMinder TK is provided for your convenience and is not required to make TransactionMinder-based Web service requests.

The TransactionMinder TK is a simple Java API with a graphical user interface (we provide the source code for this sample application to help users understand the API).



URL: http://jfavazza.security.com:81/purchasing/servlet/PurchaseInventory

Input XML File: d:\TxMinderDemo\PurchaseOrder.xml

SOAPAction:

SOAP Header:

```
<BillingInformation>
        <CustomerCredentials>
                <username>Robm</username>
                <password>siteminder</password>
        </CustomerCredentials>
</BillingInformation>
```

☑ Wrap in SOAP

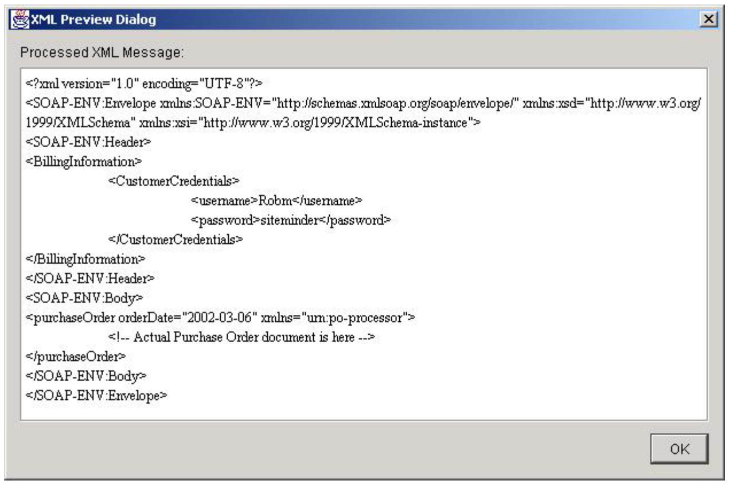
The TransactionMinder TK allows you to apply various Web services consumer technologies within your application. The screenshot above shows the Basic Information tab displaying the username / password information that we used earlier in setting up the environment.

In addition to basic HTTP POSTing, we can use the following:

- SSL: Client-side and server-side Secure Socket Layer (SSL) support.
- XML Signature: TransactionMinder supports the standard specification for digitally signing SOAP documents. Optionally users can include X.509 certificates in the signature, which could be used by the Policy Server for authentication (in this case we would not need the SOAP envelope header shown above).
- Private / Public Keys: The TransactionMinder TK allows you to generate private-public key pairs.
- X.509 Certificate: The TransactionMinder TK helps you generate self-signed X.509 certificates. This feature is designed to be used for testing purposes. Optionally you can generate a certificate signing request (CSR) for the private key and then submit the CSR to a certificate authority (CA).

Note: The Basic Information, SSL, and XML Signature tabs demonstrate features that are used at runtime. The Private / Public Keys and X.509 Certificate tabs are utilities that generate keys and certificates to be used with the Basic Information, SSL, and XML Signature features.

Pressing on the POST button will generate the following window.



```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/
1999/XMLSchema" xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
<SOAP-ENV:Header>
<BillingInformation>
		<CustomerCredentials>
				<username>Robm</username>
				<password>siteminder</password>
		</CustomerCredentials>
</BillingInformation>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<purchaseOrder orderDate="2002-03-06" xmlns="urn:po-processor">
		<!-- Actual Purchase Order document is here -->
</purchaseOrder>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The screenshot above shows a preview of the SOAP message prior to being HTTP POSTed.

Personalizing Web services

Once TransactionMinder has successfully identified, authenticated, and authorized a user for a particular Web service, it has the ability to leverage the user identity (centrally managed by the Policy Server) to personalize the behavior of the Web service.

The Web service can be personalized through user entitlement information passed to the Web service by TransactionMinder. The information to be passed is configured as responses to a successful authorization in the Policy Server. When a user has successfully been authenticated and authorized for that Web service, the Policy Server identifies which entitlements should be obtained about that user, retrieves them from the user store, and associates them with the Web service request by binding them to the XML message.

This powerful feature eliminates the need for a Web service to keep its own entitlement database and handle the retrieval of entitlements in the application logic. Through TransactionMinder, these entitlements can be centrally and securely managed and associated directly with the user identity.

Conclusion

TransactionMinder (version 5.5) is designed to provide content-based security for authenticating and authorizing users and applications accessing Web services.

TransactionMinder leverages SiteMinder, Netegrity's flagship platform for managing and securing e-business environments, and extends it to secure the XML documents used in Web services and business-to-business transactions.

TransactionMinder v5.5 functionality will be available in the second half of 2002, used in conjunction with Netegrity PortalMinder, Netegrity's portal management platform.

Appendix: Technology Reference List

The following is a list of references to the various technologies mentioned in this document.

http://www.w3.org/XML/: The W3C's XML 1.0 Second Edition Recommendation describes the eXtensible Markup Language (XML), "the universal format for structured documents and data on the Web."

http://www.w3.org/Security/: This is the W3C's security-resources home page, which includes many links to various aspects of Web and Internet security (cryptography, authentication, authorization, etc.).

http://www.w3.org/DSig/Overview.html: The W3C's XML Signature (XML-DSIG) Recommendation describes digital signatures as applied to XML documents.

http://www.w3.org/Encryption/2001/: The W3C's XML Encryption draft defines a process for encrypting and decrypting XML documents.

http://www.w3.org/TR/xkms/: The W3C's XML Key Information Service Specification (XKMS) Recommendation defines protocols for distributing and registering public keys, used together with XML Signature and XML Encryption.

http://www.oasis-open.org/committees/security/: The OASIS specification for the Security Assertion Markup Language (SAML) defines an XML-based security framework for exchanging authentication and authorization information. Netegrity originally started and chaired the OASIS technical committee that defined the SAML specification.

http://www.w3.org/TR/SOAP/: The W3C's SOAP draft (v1.1) describes the Simple Object Access Protocol, designed as a messaging framework for exchanging XML documents between peers in a platform-neutral environment.

http://www.w3.org/TR/wsdl: The W3C's WSDL submission specifies the Web Services Description Language, a framework providing definitions for network services and the automation of application communication through XML documents.

http://www.uddi.org/about.html: The Universal Description, Discovery and Integration (UDDI) project is an industry initiative designed to create an XML framework for describing Web services providers and a description of the services they provide.

http://www.w3.org/TR/xpath: The W3C's XML Path (or XPath) draft specification defines a language designed to locate XML objects (elements, etc.) in an XML document. XPath is used to build search expressions used in conjunction with other XML technologies, in particular XSLT (designed to transform XML documents into other formats, such as HTML), and XQuery (an XML query language project designed to do for XML what the Structured Query Language (SQL) does for relational databases).