CHAPTER 16

AAA

Now at last we focus on security. Somehow, it seems that this is always the way—security is an afterthought, that thing that is added two weeks before production in order to keep the auditors happy. And yet, DataPower is somehow different. These devices are built from the ground up with security in mind; at every level, security underlies everything you do with the appliance. It should come as no surprise, then, that there is very powerful out-of-the-box functionality to perform complex Authentication, Authorization, and Auditing, all with a few mouse clicks. This chapter examines those mouse clicks in more detail, exploring the powerful and flexible AAA framework and explaining how you can make it integrate into your particular environment.

AAA: The American Automobile Association?

In computer security, the term AAA (usually said as "triple A") is used to refer to the tripartite interaction pattern of user or entity security:

- **Authentication**—Ascertaining who the entity is
- **Authorization**—Determining what the entity can do
- **Accounting**—Tracking what a given entity has done

These concepts brought together are commonly referred to as Access Control. In DataPower, there is a strong framework for implementing this user interaction pattern, called the AAA framework.

Authentication

Authentication is the process of ascertaining who the user is. In order for a user or other entity to authenticate to the appliance, they must present some form of credential—be that a token, a username and password, a digital certificate, or any of the multitude of credential formats that DataPower supports (and DataPower can be extended to support more, as demonstrated in Chapter 17, "Advanced AAA"). On their own, such credentials are nothing more than an assertion—the equivalent of someone saying "I am Bob." In order for authentication to be trusted, the credentials presented have to be validated—perhaps the person claiming to be Bob could give you his passport, where the government of his home country certifies to you that he is Bob.

Credentials in the digital world are checked against a form of identity repository, and a decision needs to be made as to whether the credentials are valid. This might be as simple as checking a username and password combination against a list of valid usernames and passwords, or as complex as verifying the digital signature of a piece of signed XML, transforming the XML into a completely different credential format, and then passing that credential to an external authentication engine that will make the decision.

Authorization

After we know who the user is, we look at the resource the user is trying to access, and decide whether he is allowed or authorized to access it. DataPower is flexible enough to make authorization decisions in many different ways. It might be as simple as allowing any client who has successfully authenticated. (This is a valid authorization decision—different from simply allowing any request, for which there may also be a valid use case.) It might involve examining a list of users from a file, or checking for membership in an LDAP group, or using more sophisticated methods such as delegating the entire authorization decision to an external service.

Audit

After we know who the user is and what he can do, there are important actions that need to be taken in terms of monitoring and logging what happens during the AAA policy. This includes functionality to log successful and unsuccessful requests, and to maintain a counter of authorized and rejected requests, which can later be used as an input into more advanced accounting.

The Accounting stage is referred to in the context of a DataPower AAA policy as Audit. This makes sense in a DataPower context—the Accounting capabilities of the appliance go far beyond security. DataPower's accounting capabilities include fine-grained usage logging if desired, prioritization of services, dynamic service level management (SLM), and throttling responses based on many trigger points, including consumption of resources. The AAA framework in DataPower focuses only on the security specific Accounting capabilities, such as monitoring and logging issues relating to the earlier AAA steps. It is explicitly to differentiate this from all the other powerful Accounting functionality on the device that the AAA-specific accounting is known in the AAA framework as Audit.

Post Processing

Post Processing (PP) doesn't begin with an A? Perhaps the framework should be named AAAPP, because this final step in the DataPower AAA framework is one of the most important. PP enables DataPower, after it has completed its verification of who the user is and whether he is allowed to access the resource he tried to access, to perform any other relevant processing of the request, the contents, the tokens, the credentials, the identity, or anything else, such that it integrates into your environment and works within the enterprise security idiom of your organization. Out of the box, the PP stage can perform powerful actions such as generating a SAML assertion or calling out to Tivoli Federated Identity Manager (TFIM); by using XSLT you can perform almost any action imaginable.

Clearly, there is much more to security than AAA. Other chapters in this book focus on aspects of security, including network security (Chapter 4, "Advanced DataPower Networking"), SSL (Chapter 18, "DataPower and SSL"), Web Services Security (Chapter 19, "Web Services Security"), XML threats (Chapter 20, "XML Threats"), and Security Integration with WebSphere Application Server (Chapter 21, "Security Integration with WebSphere Application Server"). These other chapters discuss transport and message processing, which are important parts of the appliance's functionality. This chapter and Chapter 17 focus explicitly on access control and the AAA framework.

AAA Policy Stages

The AAA framework actually does substantially more than "just" authentication, authorization, and audit. The process is broken up into a number of sequential stages, with the output of each stage feeding in to the input of the next. Every stage of the process is fully customizable, both with multiple out of the box options for each stage selectable with a simple mouse click and parameters, and by writing custom XSLT (we cover this in Chapter 17).

Figure 16-1 shows a flow diagram of the way a AAA policy works.

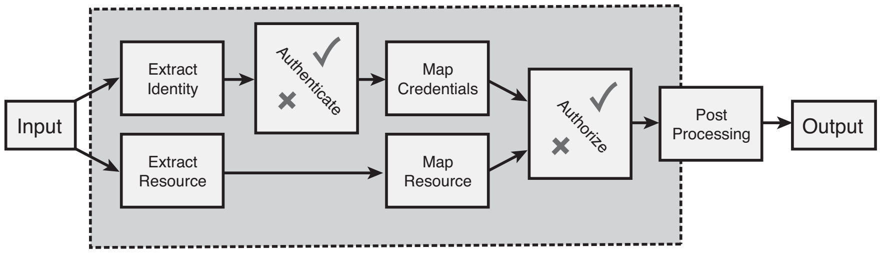


Figure 16-1 Processing phases of a AAA Policy.

The various stages are shown with the order in which they are called. This section examines each stage in depth. Each stage is also referred to by an abbreviation, which is shown in the section headings; for example, Extract Identity is abbreviated to EI, and the authentication stage is commonly abbreviated to AU (in contrast with the authorization stage, which is shortened to AZ).

Extract Identity (EI)

To perform authentication, the entity wanting to authenticate needs to provide a form of credentials. For the vast majority of Internet-facing network servers, this is a static concept. The server is configured so that it accepts one or more specific forms of authentication. In DataPower, however, we can explicitly choose what part of the incoming message we will use as the identity. This stage is known as Identity Extraction, and is commonly referred to as the EI (Extract Identity) phase.

Because a AAA policy is usually deployed as part of an overall Processing Policy, which is started with a match action, we can make a dynamic runtime decision about what we use for the identity, based on the request itself. This gives us a huge amount of flexibility; the result of the match action can determine the type of credentials we decide to use. For example, imagine a business requirement where we have to accept either a Kerberos AP-REQ[1] via SPNEGO (from an internal network where people must be authenticated to a Kerberos KDC), or a specific SSL certificate (for connections that are not expected to be authenticated via Kerberos). The selection might be based on the IP address of the incoming request, a specific HTTP header signed by a reverse proxy, or any other kind of match, which can be specified within the scope of a Processing Policy. In addition, a single AAA policy can actually support multiple methods of identity extraction, although this will have implications on later stages if those stages cannot support one or other of the methods desired!

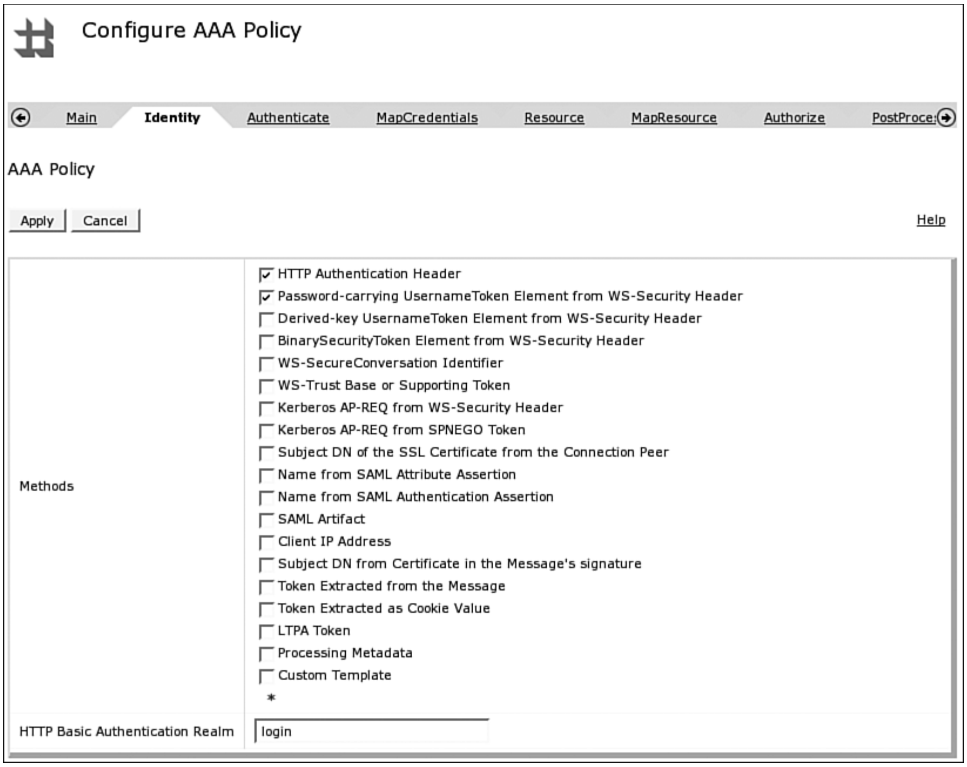
In the example in Figure 16-2, the AAA policy is set up to extract identity from either:

- **HTTP Basic Authentication**—This is a special HTTP header containing a base-64 encoded username/password combination.
- **A UserNameToken (UNT) in the WS-Security Header**—This also contains a username and password, albeit in an XML token format.

In this instance, the appliance extracts any of these that are available and passes it on to the next stage, Authentication.

There are many options available out of the box. Don’t get overwhelmed by all the options! This is going to be a common pattern; all the stages have many possible configurations out of the box, and more with customization. IBM tries to interoperate with as many standards, large market share proprietary solutions, and real-world systems as possible, which means that there will be a lot of choices at each step, most of which you will likely not use. Simply think about your specific infrastructure, and then concentrate on the parts that are relevant to you, and you won’t go wrong!

[1] Kerberos is an established trusted third-party security system, and SPNEGO is a protocol for passing a Kerberos token as part of a negotiation over HTTP. More information on these excellent security protocols is available in product documentation, especially in Appendix D of the WebGUI Guide for firmware versions 3.6.1 and above.



AAA Policy

Methods	[✓] HTTP Authentication Header [✓] Password-carrying UsernameToken Element from WS-Security Header [ ] Derived-key UsernameToken Element from WS-Security Header [ ] BinarySecurityToken Element from WS-Security Header [ ] WS-SecureConversation Identifier [ ] WS-Trust Base or Supporting Token [ ] Kerberos AP-REQ from WS-Security Header [ ] Kerberos AP-REQ from SPNEGO Token [ ] Subject DN of the SSL Certificate from the Connection Peer [ ] Name from SAML Attribute Assertion [ ] Name from SAML Authentication Assertion [ ] SAML Artifact [ ] Client IP Address [ ] Subject DN from Certificate in the Message's signature [ ] Token Extracted from the Message [ ] Token Extracted as Cookie Value [ ] LTPA Token [ ] Processing Metadata [ ] Custom Template *
HTTP Basic Authentication Realm	login

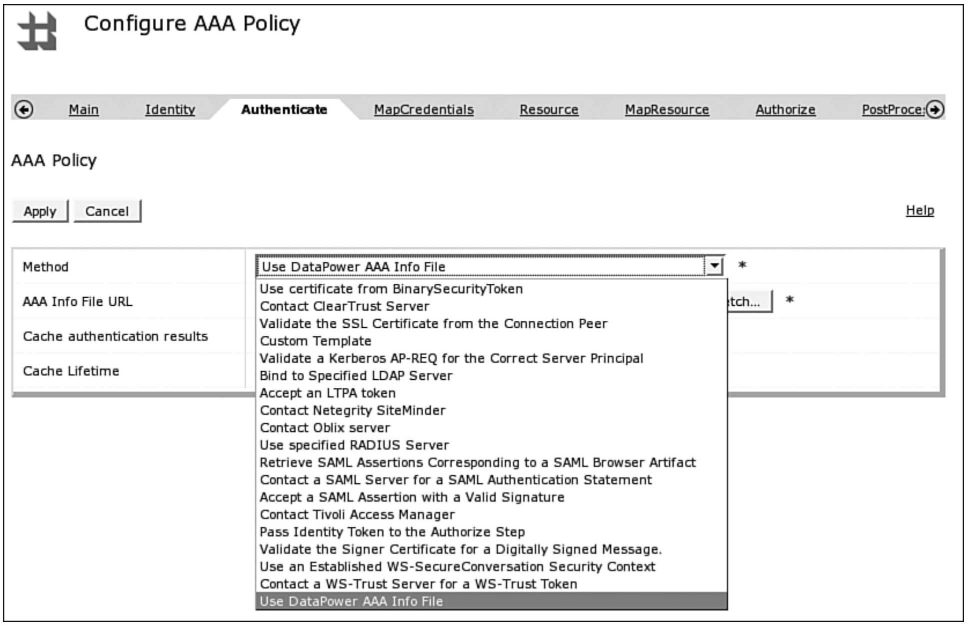
Figure 16-2 Identity Extraction options.

Authentication (AU)

The Authenticate stage takes as its input the output of the identity extraction stage, and defines the method used to authenticate the credentials provided. The number of methods available out of the box is staggering; a list is shown in Figure 16-3.

Different configuration options display on the screen depending on the method chosen from the drop-down list. The options include sending the credentials to a variety of supported external servers including Tivoli Access Manager, Siteminder, ClearTrust, and RADIUS, processing them directly by validating digital signatures or other tokens, using a custom AAA Info file that will be explained later in this chapter, and even simply passing the credentials as-is to the authorize step.

The result of authentication is either an authenticated identity or, in the case of a failure, a lack of identity; the identity is successfully authenticated or it isn't. However, processing continues irrespective of whether authentication is successful. This is because there might be some specific processing that needs to be done even for unsuccessful authentication. For instance, we might want to write an audit record, or create a generic "guest" credential, or redirect to a different service that provides a more detailed form of login.



Method: Use DataPower AAA Info File *

Use certificate from BinarySecurityToken
Contact ClearTrust Server
Validate the SSL Certificate from the Connection Peer
Custom Template
Validate a Kerberos AP-REQ for the Correct Server Principal
Bind to Specified LDAP Server
Accept an LTPA token
Contact Netegrity SiteMinder
Contact Oblix server
Use specified RADIUS Server
Retrieve SAML Assertions Corresponding to a SAML Browser Artifact
Contact a SAML Server for a SAML Authentication Statement
Accept a SAML Assertion with a Valid Signature
Contact Tivoli Access Manager
Pass Identity Token to the Authorize Step
Validate the Signer Certificate for a Digitally Signed Message.
Use an Established WS-SecureConversation Security Context
Contact a WS-Trust Server for a WS-Trust Token
Use DataPower AAA Info File

Figure 16-3 Available authentication methods.

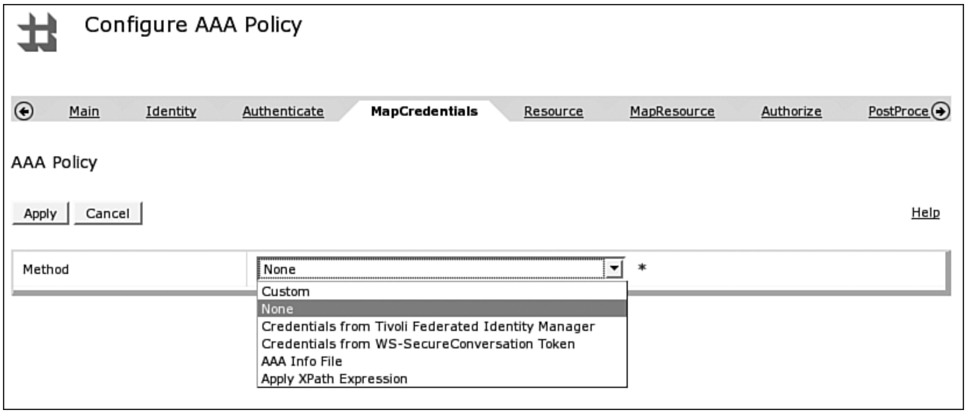
After the identity is authenticated, a credential is output and passed to the next step, for credential mapping.

Map Credentials (MC)

The next stage, Map Credentials (MC), is optional; many AAA policies will not use a map credentials step at all. The purpose of the map credentials step is to take the credentials that have been validated in the authentication step and map them to a different format by performing some kind of processing on them.

An example use of this might be where the authentication step is done against one kind of external service, but the authorization stage is against another. For example, the original authentication might be done using a custom signed token, which is then needed to be mapped into a specific distinguished name so that an authorization service based on LDAP could be used to determine group memberships.

The credential mapping options available are shown in Figure 16-4. The number of possible transformations and mappings is immense and can be used for everything from mapping external users to internal users, to transformation of token types such that a different authorization system can understand them. Some credential mapping implementations will of necessity use custom stylesheets or an XPath expression to map the credentials according to an implementation-specific algorithm. For simpler requirements, it is also possible to use a AAA-Info file as described later in this chapter to provide a simple static mapping.



Method: None *

- Custom
- None
- Credentials from Tivoli Federated Identity Manager
- Credentials from WS-SecureConversation Token
- AAA Info File
- Apply XPath Expression

Figure 16-4 Available credential mapping methods.

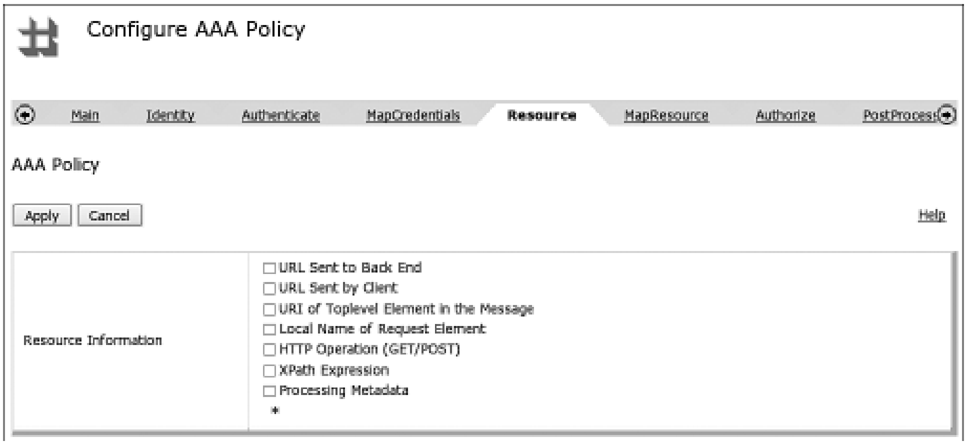
There are two special cases for credential mapping that are worthy of note. The first of these has to do with the WS-SecureConversation specification. A part of that specification, WS-Trust, can contain a "context token," which is the token representing the user or entity carrying out the conversation. The map credentials step has out-of-the-box functionality to use this context token as the identity. More information on WS-SecureConversation can be found in Chapter 19.

The second special case is that of TFIM, which can be used to perform external token transformation; DataPower will pass the incoming credentials to the TFIM trust service, and it in turn will return appropriate credentials to be used further.

Extract Resource (ER)

The next step identifies the resource upon which an authorization decision should be made. As with identity extraction, DataPower is far more flexible with this than the majority of "normal" servers. For example, typical Web application servers perform authorization based on the URL that the client requests. DataPower can do the same, or it can base the resource on any part of the incoming request. This can include HTTP headers, the type of operation or method (HTTP GET or HTTP POST), or even an XPath expression representing a value within the incoming structure. The options available are shown in Figure 16-5.

For Web Services traffic, by far the most common Web service resource to specify is the one named "Local Name of the Request Element." This will usually refer to the operation name in the Web service, which allows for granular authorization of a particular operation.



Resource Information

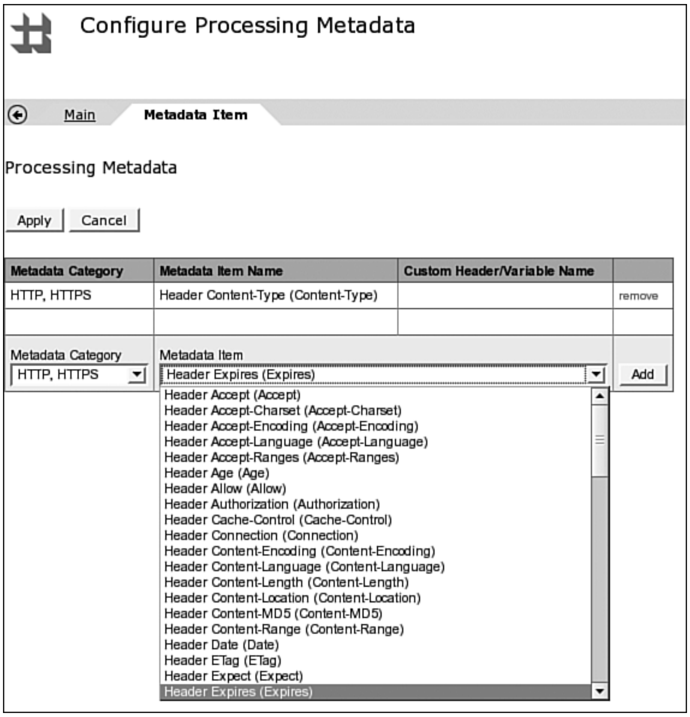
- ☐ URL Sent to Back End
- ☐ URL Sent by Client
- ☐ URI of Toplevel Element in the Message
- ☐ Local Name of Request Element
- ☐ HTTP Operation (GET/POST)
- ☐ XPath Expression
- ☐ Processing Metadata

Figure 16-5 Available resource identification methods.

Of particular interest here are the HTTP Operation and Processing Metadata options. HTTP Operation actually refers to the HTTP protocol request method invoked by the client. By choosing the HTTP Operation as the resource, it is possible to authorize a read operation differently than a write operation; this is a relatively common requirement in some situations. An HTTP GET is always a read operation, and an HTTP POST is always a write operation. Note that HTTP GET is not common with Web services, and indeed is disabled by default on the DataPower HTTP FSH, but is often used in other XML remote procedure call implementations such as REST, as well as in plain Web traffic. In addition, this option can be used on the XI50 for non-HTTP traffic; for instance, with an FTP Server FSH choosing the HTTP Operation for the resource allows authorization of FTP uploads differently (PUT) than FTP downloads (GET).

The Processing Metadata option allows for the resource to be defined based on what is known as "transport metadata." This metadata, also available for the identity extraction step, is simply a set of preconfigured medium-specific fields, which might be interesting for processing requests of that type. These can include information extracted from the incoming message, such as header values specific to the protocol used, or internal information from the appliance processing scope, such as system variables.

This is a powerful concept. For instance, let's say that we want to choose as our resource a specific HTTP header in the request, the Content-Type header. The rationale for this might be that, if a client is requesting a double-byte character set (DBCS) content type, we want to perform authorization differently. To extract this resource, we can write a complicated XPath expression or XSLT to identify the specific value from the request, or we can configure a processing metadata item as shown in Figure 16-6.


Configure Processing Metadata

Main | Metadata Item

Processing Metadata

Apply | Cancel

Metadata Category	Metadata Item Name	Custom Header/Variable Name	
HTTP, HTTPS	Header Content-Type (Content-Type)		remove

Metadata Category: HTTP, HTTPS
Metadata Item: Header Expires (Expires)
Add

```
Header Accept (Accept)
Header Accept-Charset (Accept-Charset)
Header Accept-Encoding (Accept-Encoding)
Header Accept-Language (Accept-Language)
Header Accept-Ranges (Accept-Ranges)
Header Age (Age)
Header Allow (Allow)
Header Authorization (Authorization)
Header Cache-Control (Cache-Control)
Header Connection (Connection)
Header Content-Encoding (Content-Encoding)
Header Content-Language (Content-Language)
Header Content-Length (Content-Length)
Header Content-Location (Content-Location)
Header Content-MD5 (Content-MD5)
Header Content-Range (Content-Range)
Header Date (Date)
Header ETag (ETag)
Header Expect (Expect)
Header Expires (Expires)
```

Figure 16-6 Processing metadata.

This metadata item is preconfigured to extract the exact header we need, specific to the protocol. It could just as easily be configured to extract a WebSphere MQ header, as shown in Figure 16-7.

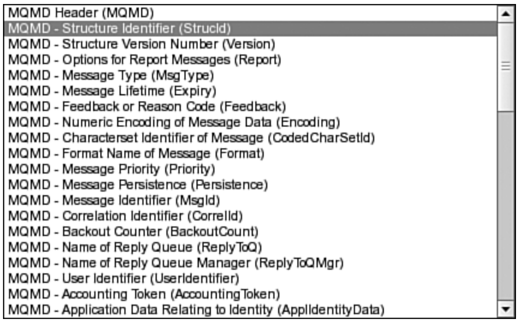

```
MQMD Header (MQMD)
MQMD - Structure Identifier (StrucId)
MQMD - Structure Version Number (Version)
MQMD - Options for Report Messages (Report)
MQMD - Message Type (MsgType)
MQMD - Message Lifetime (Expiry)
MQMD - Feedback or Reason Code (Feedback)
MQMD - Numeric Encoding of Message Data (Encoding)
MQMD - Characterset Identifier of Message (CodedCharSetId)
MQMD - Format Name of Message (Format)
MQMD - Message Priority (Priority)
MQMD - Message Persistence (Persistence)
MQMD - Message Identifier (MsgId)
MQMD - Correlation Identifier (CorrelId)
MQMD - Backout Counter (BackoutCount)
MQMD - Name of Reply Queue (ReplyToQ)
MQMD - Name of Reply Queue Manager (ReplyToQMgr)
MQMD - User Identifier (UserIdentifier)
MQMD - Accounting Token (AccountingToken)
MQMD - Application Data Relating to Identity (ApplIdentityData)
```

Figure 16-7 MQ options for processing metadata.

Other options include a Tibco EMS header, a WebSphere JMS header, and many other kinds of information about the incoming request. For instance, we could specify the specific Frontside Service Address over which the request came, allowing us to authorize differently based on the Ethernet interface over which the request was retrieved. We encourage you to explore this useful and powerful functionality.

Map Resource (MR)

In addition to extracting resources in several ways, DataPower can further refine the input into the authorization step. The resource mapping stage allows you to examine the existing resources for a pattern and map multiple different, yet similar, resources into a single logical resource, such that you can then perform a more coarse-grained authorization based on that logical resource.

For example, let's say that we have extracted the simplest type of resource for an inbound Web service—the URL sent by the client. This is a common use case when integrating with external security systems that perform access control based on URL, of which there are many. In this instance, we know that the URLs will represent the required action, because our application happens to follow the same pattern for all URLs. The URL pattern is `/Brand/AccountType/Operation,` such that all the URLs shown in Listing 16-1 are effectively the `apply` operation.

Listing 16-1 URL Patterns to Process

```
/redbank/premium/apply
/redbank/standard/apply
/bluebank/premium/apply
/bluebank/standard/apply
```

The map resource step could be used to translate all these URLs into a single resource identifier—perhaps simply the word `APPLY`—which will then be much easier to perform authorization against. How do we actually do this? The options are shown in Figure 16-8.

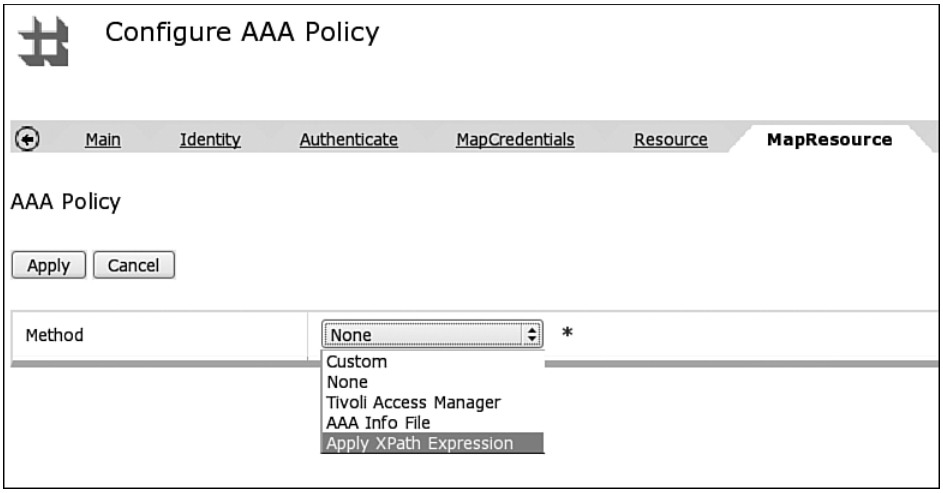
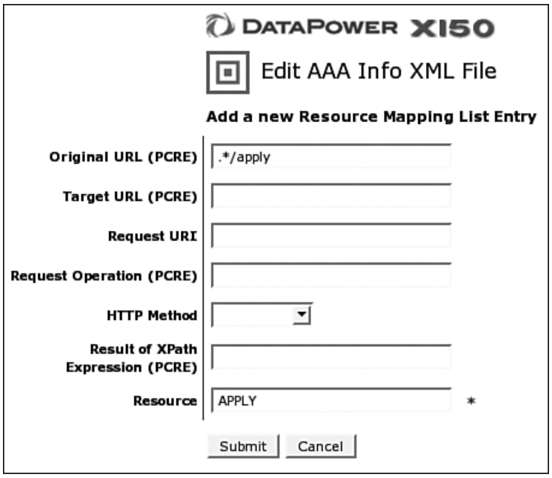


Figure 16-8 Resource mapping options.

We could write a stylesheet to perform the resource mapping and specify that stylesheet using the Custom method. Or we could use XPath to point at the specific information we need. The resource mapping could be externalized to Tivoli Access Manager, where it performs a mapping to add the resource into the TAM Object Space (or TAMBI or TFIM) such that subsequent authorization against TAM is possible. Or the AAA Info File provides an internal method of performing resource mapping—this last option is shown in Figure 16-9.



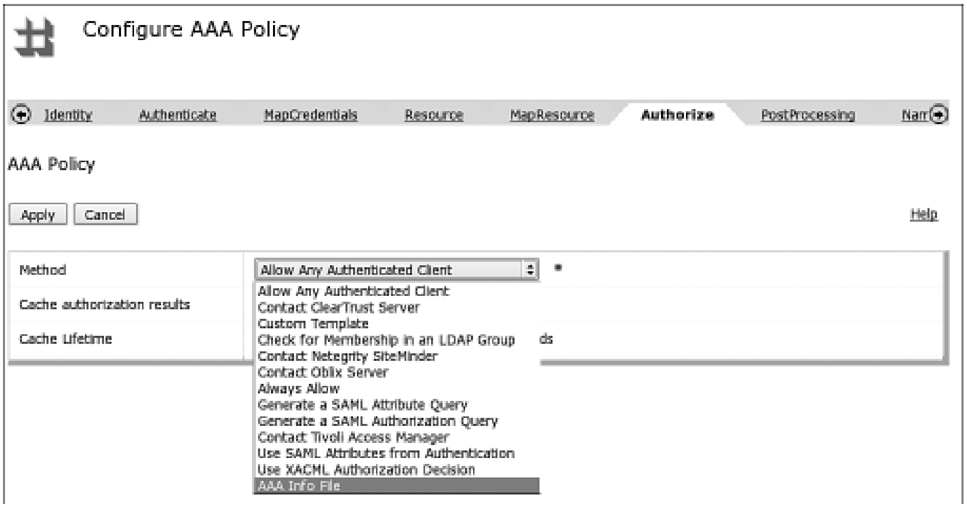
Original URL (PCRE)	.*/apply
Target URL (PCRE)	
Request URI	
Request Operation (PCRE)	
HTTP Method	
Result of XPath Expression (PCRE)	
Resource	APPLY *

Figure 16-9 Resource mapping using AAA info.

Authorization (AZ)

Now that we finally have an identity (which has been extracted from the incoming message, has either passed or failed authentication, and has optionally been mapped to a more meaningful credential) and a resource (which has been identified flexibly from within the incoming message and optionally mapped to a coarser grained resource), we can make an authorization decision. The decision is relatively simple: Can this specific identity access this specific resource, yes or no? The process by which the decision is reached can be implemented in many ways. Figure 16-10 shows the options available.

The simplest option is Allow Any Authenticated Client. This means that if the request has successfully passed the Authenticate stage, it will automatically be allowed through irrespective of which resource it is trying to access. This is a valid use case for surprisingly many scenarios—quite often, simply authenticating an incoming request is considered to be "enough" at this level. (Although this is usually combined with more fine-grained authorization later down the line on a backend server.)


Configure AAA Policy
Identity
Authenticate
MapCredentials
Resource
MapResource
Authorize
PostProcessing
Nam
AAA Policy
Apply
Cancel
Help
Allow Any Authenticated Client
Contact ClearTrust Server
Custom Template
Check for Membership in an LDAP Group
Contact Netegrity SiteMinder
Contact Oblix Server
Always Allow
Generate a SAML Attribute Query
Generate a SAML Authorization Query
Contact Tivoli Access Manager
Use SAML Attributes from Authentication
Use XACML Authorization Decision
AAA Info File

Method: Allow Any Authenticated Client *
Cache authorization results
Cache Lifetime ds

Figure 16-10 Authorization options.

Note that this option is very different from the option named Always Allow. The difference might not be clear at first, but the subtlety is that if you select Always Allow, all requests will be allowed through, even those that failed authentication. It is possible that there are some valid use cases for this, but it is likely not to be what you want. After all, if you intend to allow all connections through, why put in a AAA policy at all? One possible scenario might be where user identity is used for personalization and not access control—if we know who the user is, we can give them personalized content, but if not then they simply get a generic page.

A number of the authorization methods allow for externalizing the authorization decision. That is, rather than making the decision on the device itself, the information about the authenticated user and the resource they are trying to access is sent to an external server, which then makes the decision. External servers supported out of the box in this manner are Tivoli Access Manager, ClearTrust, and SiteMinder.

Other common methods of making an authorization decision based on external input are also supported out of the box. For example, we can simply check for membership in a specific LDAP group. This works by looking up the specific group that users must be a member of in the LDAP server, retrieving a list of all the members of that group, and checking whether the authenticated user is included in that list. This is not quite the same as externalization, since the appliance makes the authorization decision itself based on input from the external server, rather than relying on the external server to make the decision. We can also use SAML to make an authorization or attribute query, and we can make a call to an external XACML PDP (XML Access Control Markup Language Policy Decision Point).

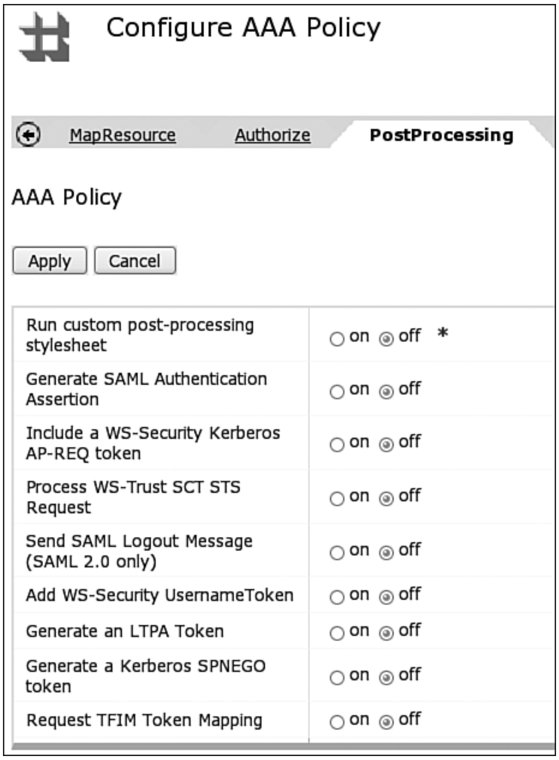
If the preceding isn't enough, we can also make the authorization decision on box, including using a AAA Info file, a local XACML PDP, or even a custom piece of XSLT, which can do almost anything we need it to do (including make calls out to external services if required).

Although authorization is a simple yes or no decision, DataPower offers substantial integration options and flexibility to enable the decision to be made based on whatever is meaningful to your specific situation.

Post Processing (PP)

The final stage, Post Processing (PP), is optional, but at the same time is possibly the most important value-add from the AAA processing in DataPower. PP can be used to perform any required action after the AAA Policy has completed. Most commonly this involves creating credentials, which will be used on the call to the backend server. In these use cases, DataPower acts as a simple token changer, exchanging one set of frontend credentials for an entirely new backend credential supported by the backend server. This allows us to use a frontend credential mechanism of our choice while integrating with various standard credential mechanisms on the backend—out of the box for many standards!

Out of the box, IBM provides a number of options for creating backend credentials. These are shown in Figure 16-11.



Run custom post-processing stylesheet	on [✓ off] *
Generate SAML Authentication Assertion	on [✓ off]
Include a WS-Security Kerberos AP-REQ token	on [✓ off]
Process WS-Trust SCT STS Request	on [✓ off]
Send SAML Logout Message (SAML 2.0 only)	on [✓ off]
Add WS-Security UsernameToken	on [✓ off]
Generate an LTPA Token	on [✓ off]
Generate a Kerberos SPNEGO token	on [✓ off]
Request TFIM Token Mapping	on [✓ off]

Figure 16-11 PP options.

Some of these options exist to propagate the authenticated credentials to the backend server. For example, the Add WS-Security UsernameToken generates a UNT, which contains the username and password output from the Map Credentials stage. In the same way, Generate an LTPA Token creates a Lightweight Third-Party Authentication (LTPA) token for the authenticated user, and Generate a SAML Authentication Assertion provides an assertion in SAML format to state who the authenticated user is.

The Kerberos options (AP-REQ token and SPNEGO) provide a modicum of extra security by passing a static credential for the backend to consume, potentially providing a level of trust between DataPower and the backend. These options are only capable of creating Kerberos credentials for a specific user whose keytab is contained on the appliance. This has to do with the way Kerberos works—it is impossible to generate a Kerberos credential for a user whose password or keytab we do not have, and we might not have it depending on how they authenticated to us! Thus, this use of Kerberos is to represent the appliance as a system user, rather than propagating actual user credentials. (Although this identity might be used to establish trust over which actual user credentials might be asserted, for instance.)

Other options include participating in a WS-SecureConversation by processing a Security-ContextToken request and acting as a WS-Trust Secure Token Service, and requesting a token mapping from Tivoli Federated Identity Manager.

Finally, as with all the other stages of AAA processing, we can run our own custom XSLT to perform any processing we choose. This processing has access to all the outputs of the previous stages, and with some assembly required, enough work can be used to provide almost any form of backend integration required.

Configuring AAA in DataPower

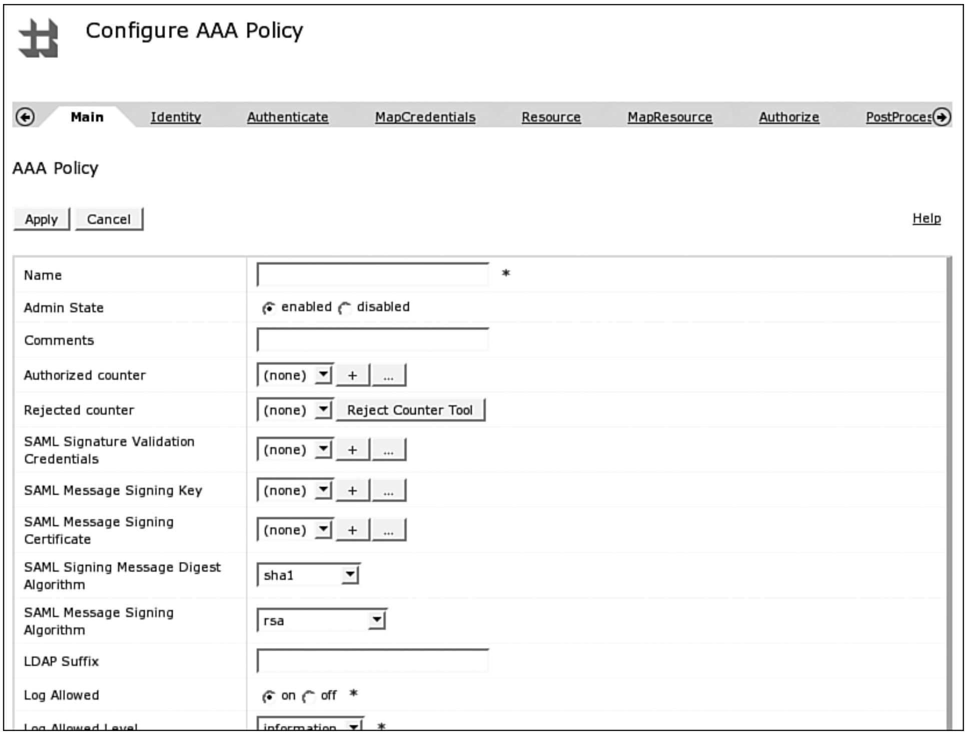
DataPower's AAA framework is built around one main configuration object called the AAA Policy. This policy is used in many contexts to provide a common configuration idiom for AAA functionality—it can be run as an action in a Processing Policy, attached directly to specific FSHs, and even specified in the profiles of a Web Application Firewall. This common configuration object means that often there are many fields that are not specified, because they are not relevant for a specific configuration, but it also means that the entire required configuration is easy to find irrespective of the deployment context.

There are two ways to create a AAA policy object via the WebGUI: using a sequential wizard, which allows for context-sensitive creation of policies that support the majority of use cases, or by using the object page, which gives access to configuration options not available via the wizard.

An often asked question is: Which of these methods should be used? In most cases, when prototyping, creating processing policies, and generally developing configurations, the wizard is used. It is there when needed, and enables you to quickly create a AAA policy without leaving the context of what you are doing in the GUI. This functionality is valuable. However, there is also much to be said for developing the AAA policy independently of the surrounding configuration, by using the object page. This approach allows for a more considered design of AAA policy, and more often leads to creation of policies that are reused in multiple configurations where the authentication and authorization processes are the same. Thus there are advantages to both methods.

The AAA Policy Object Menu

The previous figures in this chapter all show the AAA Policy Object. To use this for creating a AAA policy, go directly to the object page, using Objects→XML Processing→AAA Policy in the left-side menu. This option gives access to all possible configuration settings, but provides a far more complex configuration screen with multiple tabs and many more options on each tab. The AAA policy object page is shown in Figure 16-12.



AAA Policy

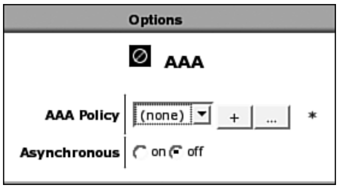
Name	*
Admin State	[✓ enabled] disabled
Comments	
Authorized counter	(none) + ...
Rejected counter	(none) Reject Counter Tool
SAML Signature Validation Credentials	(none) + ...
SAML Message Signing Key	(none) + ...
SAML Message Signing Certificate	(none) + ...
SAML Signing Message Digest Algorithm	sha1
SAML Message Signing Algorithm	rsa
LDAP Suffix	
Log Allowed	[✓ on] off *
Log Allowed Level	information *

Figure 16-12 A AAA Policy can also be configured directly from the objects page.

The majority of settings can also be configured from within the wizard, and which method you use ends up to be a mixture of personal preference and the order in which you configure the various objects on the device. Many sites will mix the two, perhaps using the wizard for initial configuration followed by manual changes on the object page.

The AAA Policy Wizard

Wherever a AAA policy is required, there is the typical drop-down list of existing policies, and a plus sign or ellipsis button that can be clicked on to launch a wizard. For example, when adding a AAA action to a Processing Policy, the option shown in Figure 16-13 appears; clicking on the plus sign launches the wizard to add the AAA action, which takes you through a configuration process.



AAA Policy: (none)
Asynchronous: on [✓ off]

Figure 16-13 The AAA Policy wizard can be launched from anywhere a AAA policy is needed.

The wizard provides a step-by-step process for configuring a AAA policy. This is a less complicated configuration method, as can be expected from a wizard. Some advanced options that are perhaps less common are not shown in the wizard, in order to present a simpler view, which covers the majority of use cases. In addition, because the wizard is sequential, it encourages the user to think about each step of AAA separately.

Let's look at the creation of a simple AAA policy using the wizard. For now, we will shoot through and examine the process; the rest of the chapter will go into the details of what the options mean. It is vital at this stage not to be overwhelmed with the number of options available! In all likelihood you will use only a small subset of them, but because the appliance supports so many options out of the box, you are probably in the nice position that the specific subset applicable to your particular configuration is supported simply on the device.

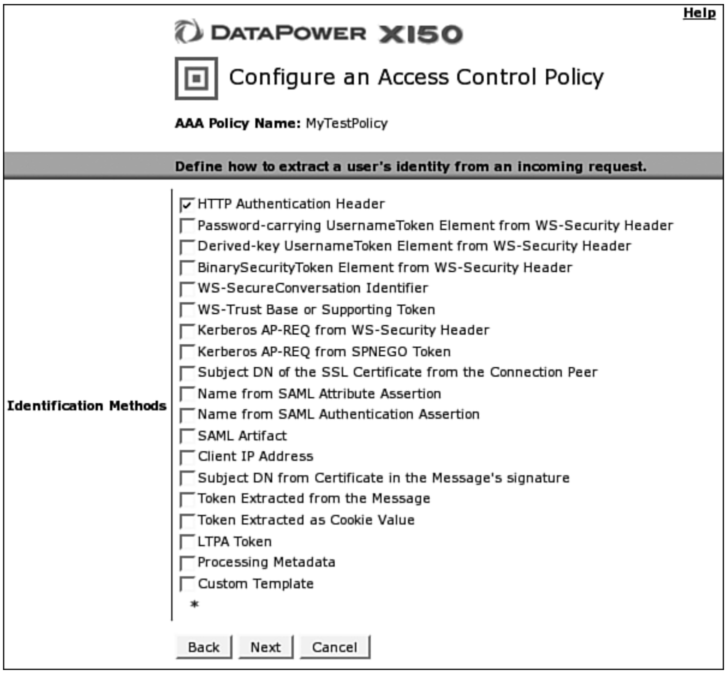
Let's say we want to create a simple policy to protect access to specific URLs. We enforce security through the use of HTTP Basic Authentication, with a set of usernames and passwords that we can define in a file, along with a list of which user is allowed to access which URL. This simple requirement can easily be implemented by clicking a few buttons in the wizard.

Start with clicking the plus sign (see Figure 16-13) to start the wizard. The first screen of the wizard asks for a name for the policy; after this is supplied, the next screen pops up and asks you how you would like to identify the users, as shown in Figure 16-14.

To enforce the use of HTTP Basic Authentication, we have selected "HTTP Authentication Header" from the wizard. Ignore the other options for now; we will get to them in time.

TIP—ONLINE HELP

The online help for AAA is particularly useful. By clicking the Identification Methods hotlink shown on the left side of Figure 16-14, we are taken to a screen that explains each of the methods in turn—the same is true for all the stages in the wizard.



Configure an Access Control Policy

AAA Policy Name: MyTestPolicy

Define how to extract a user's identity from an incoming request.

Identification Methods

- [✓] HTTP Authentication Header
- [ ] Password-carrying UsernameToken Element from WS-Security Header
- [ ] Derived-key UsernameToken Element from WS-Security Header
- [ ] BinarySecurityToken Element from WS-Security Header
- [ ] WS-SecureConversation Identifier
- [ ] WS-Trust Base or Supporting Token
- [ ] Kerberos AP-REQ from WS-Security Header
- [ ] Kerberos AP-REQ from SPNEGO Token
- [ ] Subject DN of the SSL Certificate from the Connection Peer
- [ ] Name from SAML Attribute Assertion
- [ ] Name from SAML Authentication Assertion
- [ ] SAML Artifact
- [ ] Client IP Address
- [ ] Subject DN from Certificate in the Message's signature
- [ ] Token Extracted from the Message
- [ ] Token Extracted as Cookie Value
- [ ] LTPA Token
- [ ] Processing Metadata
- [ ] Custom Template

*

Figure 16-14 How do I identify my clients?

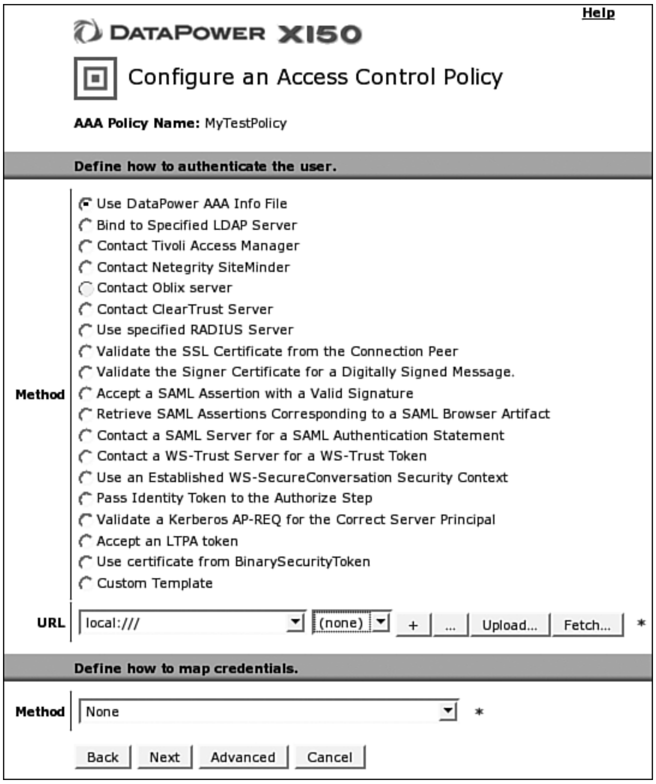
Clicking on Next takes us to the third wizard screen, where we decide how to perform authentication. In this case, we want to specify a simple set of users with passwords and create rules about which URLs each of them can access. The rules we define are shown in Table 16-1.

Table 16-1 Users, Passwords, and Access Rights

User	Password	/public	/private	/secret
Alice	password1	Yes	No	No
Bob	password2	Yes	Yes	No
Charlie	password3	Yes	Yes	Yes

These rules are simple and are designed only to demonstrate the functionality; obviously no organization's access control would be this simplistic! Likewise, we use HTTP Basic Auth for simplicity here; it is rarely used for direct user access to applications, where a login form and form parameters would more likely be presented to users.

Figure 16-15 shows the third screen.



Configure an Access Control Policy

AAA Policy Name: MyTestPolicy

Define how to authenticate the user.

Method:
- [✓ Use DataPower AAA Info File]
- Bind to Specified LDAP Server
- Contact Tivoli Access Manager
- Contact Netegrity SiteMinder
- Contact Oblix server
- Contact ClearTrust Server
- Use specified RADIUS Server
- Validate the SSL Certificate from the Connection Peer
- Validate the Signer Certificate for a Digitally Signed Message.
- Accept a SAML Assertion with a Valid Signature
- Retrieve SAML Assertions Corresponding to a SAML Browser Artifact
- Contact a SAML Server for a SAML Authentication Statement
- Contact a WS-Trust Server for a WS-Trust Token
- Use an Established WS-SecureConversation Security Context
- Pass Identity Token to the Authorize Step
- Validate a Kerberos AP-REQ for the Correct Server Principal
- Accept an LTPA token
- Use certificate from BinarySecurityToken
- Custom Template

URL: local:/// (none) *

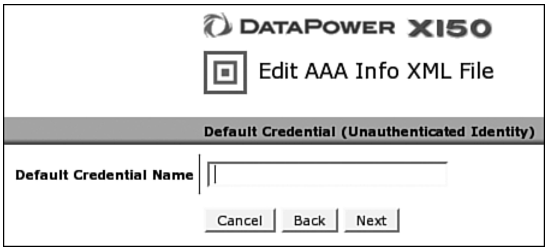
Define how to map credentials.

Method: None *

Figure 16-15 How do I authenticate my users?

Note that the option "Use DataPower AAA Info File" is selected. This is an on-device file format, which allows us to provide input into a AAA policy—a simple place to store information such as that shown in Table 16-1. More detail of the structure of the AAA Info file is provided later, but for now we simply use the helpful wizard to create one. At the bottom, under "URL," is where we specify the file to use; we can click on the + sign to create a new file.

The first page of the AAA Info wizard is shown in Figure 16-16.



Edit AAA Info XML File

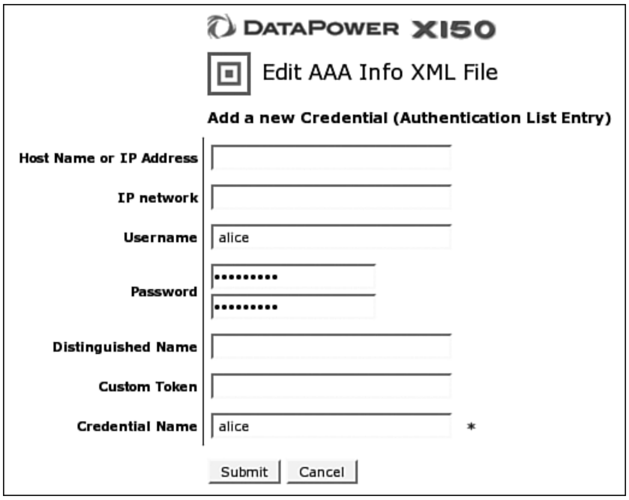
Default Credential (Unauthenticated Identity)

Default Credential Name:

Figure 16-16 Default credential name.

Default Credential is the name used to identify people who try to access protected resources but do not provide a valid credential. This can be a powerful concept; it allows you to explicitly supply a username, which can later be used for making authorization decisions. By leaving it blank, we will state that any requests that do not provide valid credentials should not have any credential passed to the authorization step, and thus should likely be denied.

The next page of the AAA Info wizard shows our list of authenticated identities. These must be populated by clicking the Add button. Figure 16-17 shows us adding the entry for our first user, Alice.



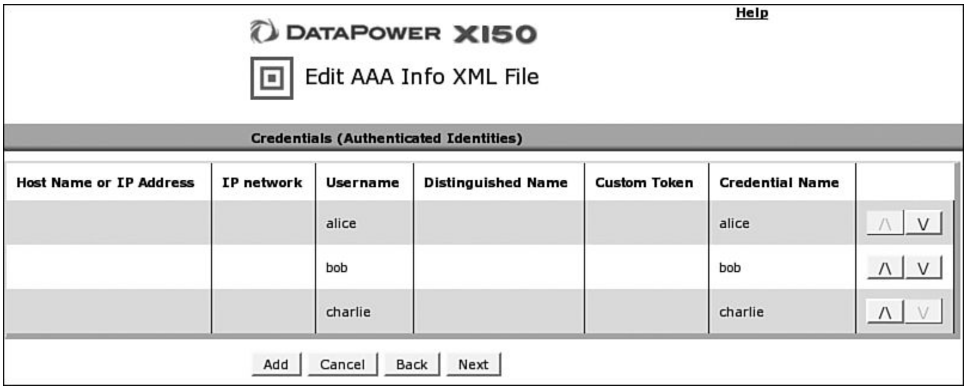
Add a new Credential (Authentication List Entry)

Host Name or IP Address	
IP network	
Username	alice
Password	••••••••• •••••••••
Distinguished Name	
Custom Token	
Credential Name	alice *

Figure 16-17 Add a new credential.

Note that we have put the word alice twice into Figure 16-17: first as the username, and second as the credential. The username is the name that alice must actually type in for HTTP basic authentication to work; the credential is what the appliance uses to identify her after the authentication has been checked. In our example, we use the same word, but the credential name is arbitrary. We could also specify a number of users with the same "credential" to write access control rules for them all at once—for instance, credentials called "validuser" and "superuser" might be used to define two different groups of users, for whom we can then write specific access control rules. Figure 16-18 shows the final list of users and the credentials they will use.

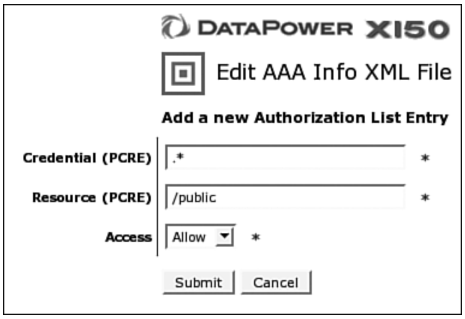
After clicking next, we get to the next page of the wizard, which is used for mapping credentials; we will not use this page for our sample. Following this is another page we will not use for the sample, regarding mapping of resources. The next page is used for authorization of access to resources. This is where we specify our actual access control rules: which user can access what URL. Again rules must be specified individually, so click Add to see how to add these, as shown in Figure 16-19.



Credentials (Authenticated Identities)

Host Name or IP Address	IP network	Username	Distinguished Name	Custom Token	Credential Name	
		alice			alice	
		bob			bob	
		charlie			charlie	

Figure 16-18 All users and their credentials.



Add a new Authorization List Entry

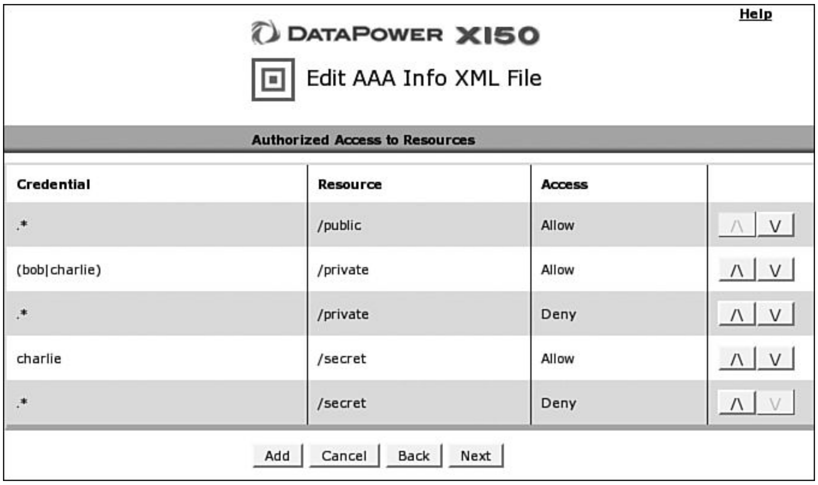
Credential (PCRE)	.* *
Resource (PCRE)	/public *
Access	Allow *

Figure 16-19 Adding an access control rule for /public.

Figure 16-19 shows the adding of a rule for the resource `/public`. This rule is somewhat special, in that the credential required to access public is defined as the Perl Compatible Regular Expression `.*`, which matches anything. Thus, all users can access the URL `/public`. The final list of access control rules is shown in Figure 16-20.

TIP—Removing Entries

To remove an entry from the list, click the credential, resource, or access level to enter the edit screen for that entry, from which you can subsequently remove it by clicking the Delete button. The Delete button is available only on the edit screen, not on the original entry screen!

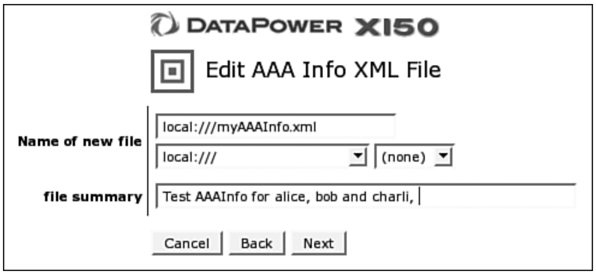


Credential	Resource	Access
.*	/public	Allow
(bob|charlie)	/private	Allow
.*	/private	Deny
charlie	/secret	Allow
.*	/secret	Deny

Figure 16-20 Access control rules.

Note again the use of Perl Compatible Regular Expressions to specify two credentials in a single line, with `(bob|charlie)`. Note also that we have explicitly put in a line denying all credentials other than those that have been explicitly allowed. This is not strictly speaking required (if a credential has no Allow entry it will default to Deny), but does not hurt, and it is a good general practice to explicitly deny.

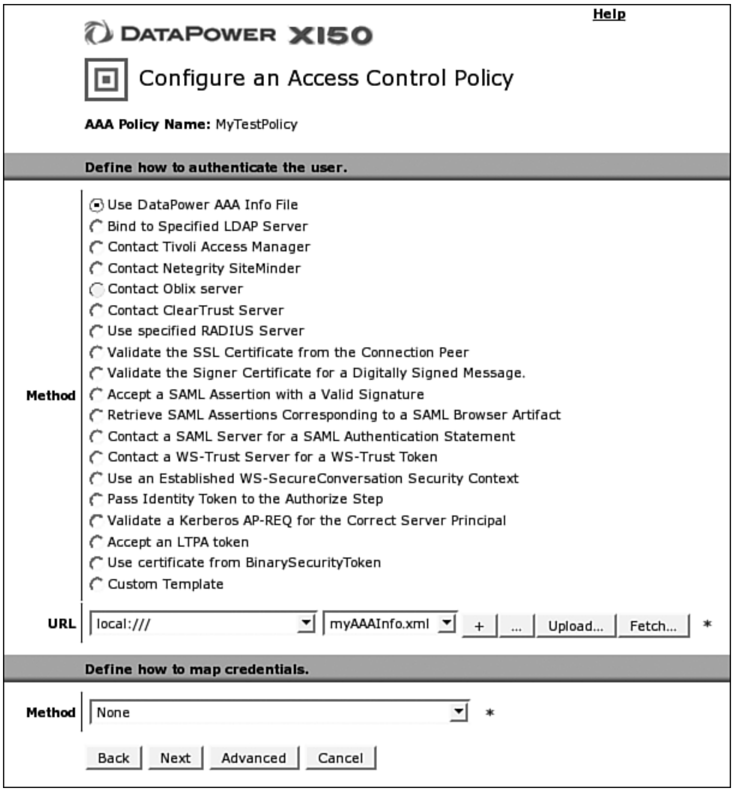
Now that we have our definitions of users and credentials and access control rules, we can click next and specify where on the device to save the resulting file, as depicted in Figure 16-21.



Name of new file	local:///myAAAInfo.xml
	local:/// (none)
file summary	Test AAAInfo for alice, bob and charli,

Figure 16-21 Save the file to the device.

After it is saved, we return to the original AAA Policy wizard, where the new AAA Info file is now selected in the URL field at the bottom of the window as shown in Figure 16-22.


Help

DataPower XI50

Configure an Access Control Policy

AAA Policy Name: MyTestPolicy

Define how to authenticate the user.

Method

[✓ Use DataPower AAA Info File]
Bind to Specified LDAP Server
Contact Tivoli Access Manager
Contact Netegrity SiteMinder
Contact Oblix server
Contact ClearTrust Server
Use specified RADIUS Server
Validate the SSL Certificate from the Connection Peer
Validate the Signer Certificate for a Digitally Signed Message.
Accept a SAML Assertion with a Valid Signature
Retrieve SAML Assertions Corresponding to a SAML Browser Artifact
Contact a SAML Server for a SAML Authentication Statement
Contact a WS-Trust Server for a WS-Trust Token
Use an Established WS-SecureConversation Security Context
Pass Identity Token to the Authorize Step
Validate a Kerberos AP-REQ for the Correct Server Principal
Accept an LTPA token
Use certificate from BinarySecurityToken
Custom Template

URL local:/// myAAAInfo.xml + ... Upload... Fetch... *

Define how to map credentials.

Method None *

Back Next Advanced Cancel

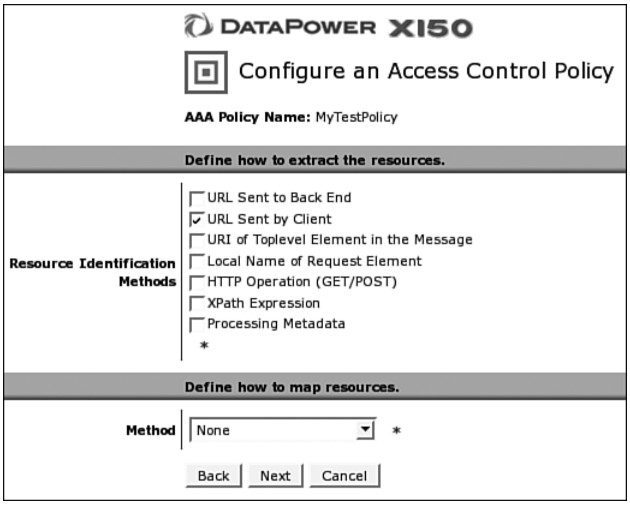
Figure 16-22 Back to the AAA Policy wizard.

For this example, we leave the Map Credentials method at the bottom as None; this step is described in more detail later in the chapter. Clicking on Next goes to the Resource Extraction Methods screen, shown in Figure 16-23.

For our example, the resource we want to protect is the URL that the client requests, as per our definition in Table 16-1. We select URL Sent by Client because this function is available out of the box. Leave the map resources method as “None,” and click Next to access the authorization page of the wizard, as shown in Figure 16-24.

Note that we selected the same AAA Info file that we created earlier. Although the whole file was created in one step with the easy-to-use wizard, the various parts of it match with the stages of the AAA policy; it is at this authorization stage that we actually implement checking of the access control rules for the resources we have defined.

Clicking Next leads to the final screen of the wizard, as depicted in Figure 16-25.


DATAPOWER XI50

Configure an Access Control Policy

AAA Policy Name: MyTestPolicy

Define how to extract the resources.

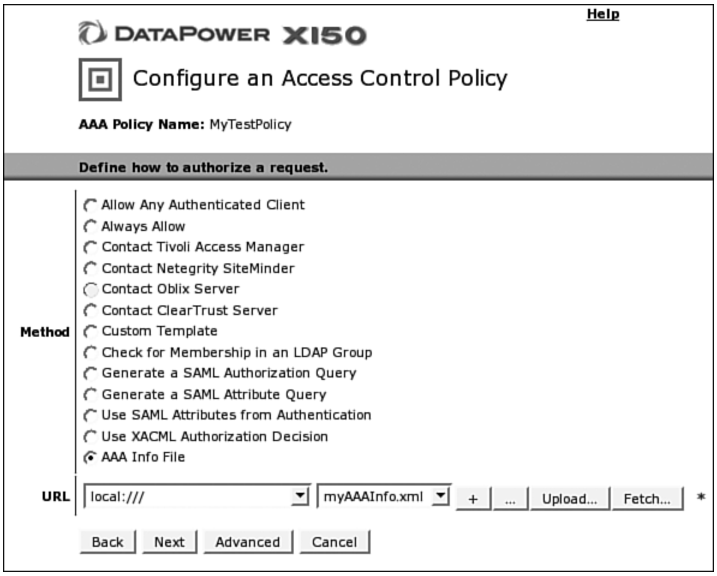
Resource Identification Methods
- [ ] URL Sent to Back End
- [✓] URL Sent by Client
- [ ] URI of Toplevel Element in the Message
- [ ] Local Name of Request Element
- [ ] HTTP Operation (GET/POST)
- [ ] XPath Expression
- [ ] Processing Metadata

*

Define how to map resources.

Method None *

Figure 16-23 Resource identification methods.



DATAPOWER XI50

Configure an Access Control Policy

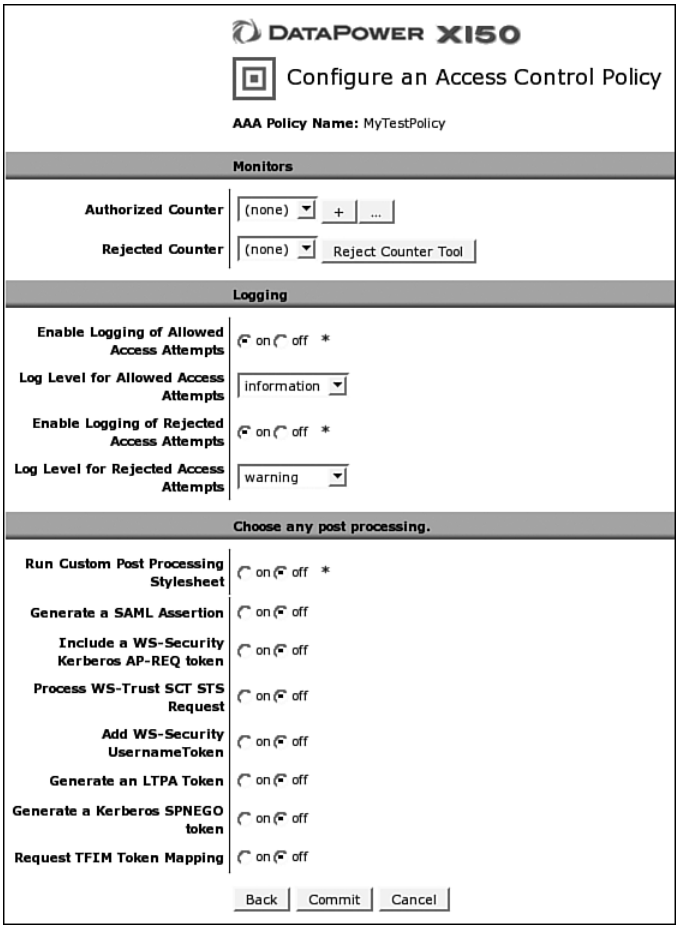
AAA Policy Name: MyTestPolicy

Define how to authorize a request.

Method
- ( ) Allow Any Authenticated Client
- ( ) Always Allow
- ( ) Contact Tivoli Access Manager
- ( ) Contact Netegrity SiteMinder
- ( ) Contact Oblix Server
- ( ) Contact ClearTrust Server
- ( ) Custom Template
- ( ) Check for Membership in an LDAP Group
- ( ) Generate a SAML Authorization Query
- ( ) Generate a SAML Attribute Query
- ( ) Use SAML Attributes from Authentication
- ( ) Use XACML Authorization Decision
- (✓) AAA Info File

URL local:/// myAAAInfo.xml *

Figure 16-24 Authorization page of the wizard.



DATAPOWER X150

Configure an Access Control Policy

AAA Policy Name: MyTestPolicy

Monitors

Authorized Counter	(none)
Rejected Counter	(none)

Logging

Enable Logging of Allowed Access Attempts	[✓ on] off *
Log Level for Allowed Access Attempts	information
Enable Logging of Rejected Access Attempts	[✓ on] off *
Log Level for Rejected Access Attempts	warning

Choose any post processing.

Run Custom Post Processing Stylesheet	on [✓ off] *
Generate a SAML Assertion	on [✓ off]
Include a WS-Security Kerberos AP-REQ token	on [✓ off]
Process WS-Trust SCT STS Request	on [✓ off]
Add WS-Security UsernameToken	on [✓ off]
Generate an LTPA Token	on [✓ off]
Generate a Kerberos SPNEGO token	on [✓ off]
Request TFIM Token Mapping	on [✓ off]

Figure 16-25 Audit and post processing.

This last page allows for setting of counters for successful and failed attempts to access the resources protected by the policy, which can later be used for service level management; definition of logging configuration where successful and failed access attempts can be (and be default are) logged to the DataPower logging infrastructure; and post processing actions, that enable modifying the security aspects of the request before it is passed to the backend. The last of these, post processing, is powerful and important; for instance, it allows generation of tokens to propagate authenticated credentials to the backend. Nonetheless for our simple scenario, nothing except the defaults are needed on this page of the wizard.

And that's it! Clicking Commit on the final page of the wizard creates a fully functional access control policy that can now be used to restrict user access to specific URLs.

Example Scenarios

This section walks you through a number of basic AAA scenarios and explains the configuration required to implement them.

Simple On-Box Authentication/Authorization with AAA Info

Many of the previous stages have mentioned and referenced a file format called AAA Info (pronounced "triple-A info"). This is an on-box custom simple file format for storing information required for AAA processing. An example AAA Info file is shown in Listing 16-2.

Listing 16-2 AAAInfo.xml

```
<?xml version="1.0"?>
<aaa:AAAInfo xmlns:dpfunc="http://www.datapower.com/extensions/functions"
xmlns:aaa="http://www.datapower.com/AAAInfo">
    <aaa:FormatVersion>1</aaa:FormatVersion>
    <aaa:Filename>local:///AAAInfo.xml</aaa:Filename>
    <aaa:Summary/>
    <aaa:Authenticate>
        <aaa:Username>bob</aaa:Username>
        <aaa:Password>passw0rd</aaa:Password>
        <aaa:OutputCredential>bob</aaa:OutputCredential>
    </aaa:Authenticate>
    <aaa:Authenticate>
        <aaa:DN>CN=Bob,OU=Workers,DC=SomeCompany,DC=com</aaa:DN>
        <aaa:OutputCredential>bob</aaa:OutputCredential>
    </aaa:Authenticate>
    <aaa:Authenticate>
        <aaa:Username>alice</aaa:Username>
        <aaa:Password>pa55word</aaa:Password>
        <aaa:OutputCredential>alice</aaa:OutputCredential>
    </aaa:Authenticate>
    <aaa:Authenticate>
        <aaa:Any/>
        <aaa:OutputCredential>guest</aaa:OutputCredential>
    </aaa:Authenticate>
    <aaa:MapCredentials>
        <aaa:InputCredential>bob</aaa:InputCredential>
        <aaa:OutputCredential>VALIDUSERS</aaa:OutputCredential>
    </aaa:MapCredentials>
    <aaa:MapCredentials>
        <aaa:InputCredential>alice</aaa:InputCredential>
        <aaa:OutputCredential>VALIDUSERS</aaa:OutputCredential>
    </aaa:MapCredentials>
    <aaa:MapResource>
        <aaa:OriginalURL>/app3/.*/private</aaa:OriginalURL>
        <aaa:OutputResource>PRIVATE</aaa:OutputResource>
    </aaa:MapResource>
    <aaa:MapResource>
        <aaa:OriginalURL>/private/.*</aaa:OriginalURL>
        <aaa:OutputResource>PRIVATE</aaa:OutputResource>
    </aaa:MapResource>
    <aaa:Authorize>
        <aaa:InputCredential>VALIDUSERS</aaa:InputCredential>
        <aaa:InputResource>PRIVATE</aaa:InputResource>
        <aaa:Access>allow</aaa:Access>
    </aaa:Authorize>
    <aaa:Authorize>
        <aaa:InputCredential>guest</aaa:InputCredential>
        <aaa:InputResource>PRIVATE</aaa:InputResource>
        <aaa:Access>deny</aaa:Access>
    </aaa:Authorize>
</aaa:AAAInfo>
```

This file contains several entries that relate to the various stages. We explore each of them in the following sections.

Authentication

First, we have the authenticate stage. This works by providing authenticate statements, which contain the credentials to be presented, and the output credential, which is to be the authenticated credential from this stage. In this instance, we have four authenticate entries, as shown in Listing 16-3.

Listing 16-3 Authentication in AAA Info

```
<aaa:Authenticate>
    <aaa:Username>bob</aaa:Username>
    <aaa:Password>passw0rd</aaa:Password>
    <aaa:OutputCredential>bob</aaa:OutputCredential>
</aaa:Authenticate>
<aaa:Authenticate>
    <aaa:DN>CN=Bob,OU=Workers,DC=SomeCompany,DC=com</aaa:DN>
    <aaa:OutputCredential>bob</aaa:OutputCredential>
</aaa:Authenticate>
<aaa:Authenticate>
    <aaa:Username>alice</aaa:Username>
    <aaa:Password>pa55word</aaa:Password>
    <aaa:OutputCredential>alice</aaa:OutputCredential>
</aaa:Authenticate>
<aaa:Authenticate>
    <aaa:Any/>
    <aaa:OutputCredential>guest</aaa:OutputCredential>
</aaa:Authenticate>
```

There are two entries for the user bob, one entry for the user alice, and one entry with no required credentials. The bob entries are especially interesting—there are clearly two separate authenticate sections, with both providing the same output credential. This is because the user bob is allowed to authenticate using one of two methods—either by providing a username and password as shown, or by presenting a client SSL certificate with the distinguished name as shown. Note that the client certificate will have been cryptographically validated using Validation Credentials (which are described in Chapter 18), so we can rely on the value to be correct.

The Any entry, which has no required credentials but provides an output credential of guest, is a simple statement saying that authentication should succeed even for users who do not provide credentials to authenticate with, but that they should be assigned a credential of guest. There is nothing special in the term guest here; this is simply the name we have assigned for use later in the Processing Policy where we decide what the guest user (unauthenticated users) is allowed to do.

Map Credentials

The next stage to which we apply our AAA Info file is credential mapping. Here we map all our actual valid users to a single output credential enabling us to make a more coarse-grained authorization decision. The relevant part of the AAA Info file is shown in Listing 16-4.

Listing 16-4 Credential Mapping in AAA Info

```
<aaa:MapCredentials>
    <aaa:InputCredential>bob</aaa:InputCredential>
    <aaa:OutputCredential>VALIDUSERS</aaa:OutputCredential>
</aaa:MapCredentials>
<aaa:MapCredentials>
    <aaa:InputCredential>alice</aaa:InputCredential>
    <aaa:OutputCredential>VALIDUSERS</aaa:OutputCredential>
</aaa:MapCredentials>
```

Notice how both bob and alice as input credentials are mapped to the same output credential, VALIDUSERS. This credential is used later to make authorization decisions where all valid users should be allowed to access. Because we plan to use it in this way, the term VALIDUSERS is appropriate, but this can be any valid string. Notice also that we do not map the guest user, because there is only one of him defined, mapping his credentials would provide no value.

Map Resource

The next part of the file is related to the Map Resource stage. At this point we have two entries as shown in Listing 16-5.

Listing 16-5 Resource Mapping in AAA Info

```
<aaa:MapResource>
    <aaa:OriginalURL>/app3/.*/private</aaa:OriginalURL>
    <aaa:OutputResource>PRIVATE</aaa:OutputResource>
</aaa:MapResource>
<aaa:MapResource>
    <aaa:OriginalURL>/private/.*</aaa:OriginalURL>
    <aaa:OutputResource>PRIVATE</aaa:OutputResource>
</aaa:MapResource>
```

This entry maps all URLs that match either of the specified regular expressions to the specific output resource PRIVATE. This resource is used in the authorization stage.

Authorization

Finally, we come to the authorization stage. At this point, we have already defined our users to be specific mapped credentials, and we have identified and mapped our resources such that they cover the URLs required. We can now make some authorization decisions, as shown in Listing 16-6.

Listing 16-6 Authorization in AAA Info

```
<aaa:Authorize>
    <aaa:InputCredential>VALIDUSERS</aaa:InputCredential>
    <aaa:InputResource>PRIVATE</aaa:InputResource>
    <aaa:Access>allow</aaa:Access>
</aaa:Authorize>
<aaa:Authorize>
    <aaa:InputCredential>guest</aaa:InputCredential>
    <aaa:InputResource>PRIVATE</aaa:InputResource>
    <aaa:Access>deny</aaa:Access>
</aaa:Authorize>
```

From this snippet it is clear that valid users (remember this is someone who has authenticated as either bob, using his username and password or an SSL certificate, or as alice using her username and password, and has therefore been mapped to the VALIDUSERS credential) can access PRIVATE resources (remember these are anything that match the private URL statements earlier), and that guest users (which is anyone who has not authenticated) are explicitly denied access to PRIVATE resources.

AAA Info is a simple yet powerful way to implement a number of the stages of AAA processing. When used with care, it can be an important tool. However, remember that it has limitations. Chief among these is that the file itself must be maintained, distributed to all appliances where it will be used, updated manually, and so on. It also becomes unwieldy when dealing with large numbers of users—although performance is not a problem. (After all DataPower is good at processing XML quickly!)

Because of these limitations, AAA Info should likely not be used for user authentication for large-scale production use at enterprise class customers. Such customers are likely to have a real enterprise directory service or single sign-on solution, and they should work to integrate with these. AAA Info can be used in these environments for learning about the AAA process, and it can be a good beginning to enable quick deployments, with a longer term goal of slowly migrating in part or in whole to integrate with the external enterprise-wide security systems as and when required. In addition, the credential mapping and resource mapping functionality can be used to invoke the powerful matching engine in DataPower for dynamic regular-expression based mapping, which is a relatively common scenario.

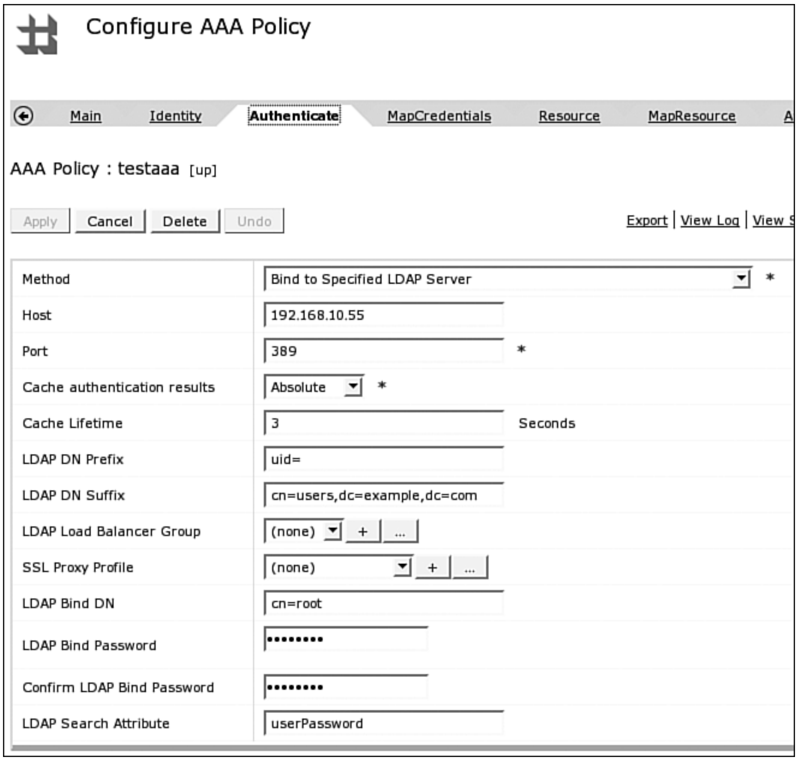
AAA Info is also invaluable in scenarios where the access control required is not end user access control, but rather authentication and authorization of system-to-system calls. In this kind of scenario, the capability to quickly and efficiently define required security credentials for a limited number of known other systems can be extremely useful.

Integration with LDAP

The Authentication and Authorization stages can utilize LDAP—which is a common requirement. The Lightweight Directory Access Protocol has been in use in distributed computing for quite some time, and many enterprises already have such a directory, which contains all their user data. Using that directory makes a lot of sense!

Authentication

Figure 16-26 shows an example of configuring an LDAP server for the authentication stage.



AAA Policy : testaaa [up]

Method	Bind to Specified LDAP Server *
Host	192.168.10.55
Port	389 *
Cache authentication results	Absolute *
Cache Lifetime	3 Seconds
LDAP DN Prefix	uid=
LDAP DN Suffix	cn=users,dc=example,dc=com
LDAP Load Balancer Group	(none)
SSL Proxy Profile	(none)
LDAP Bind DN	cn=root
LDAP Bind Password	••••••••
Confirm LDAP Bind Password	••••••••
LDAP Search Attribute	userPassword

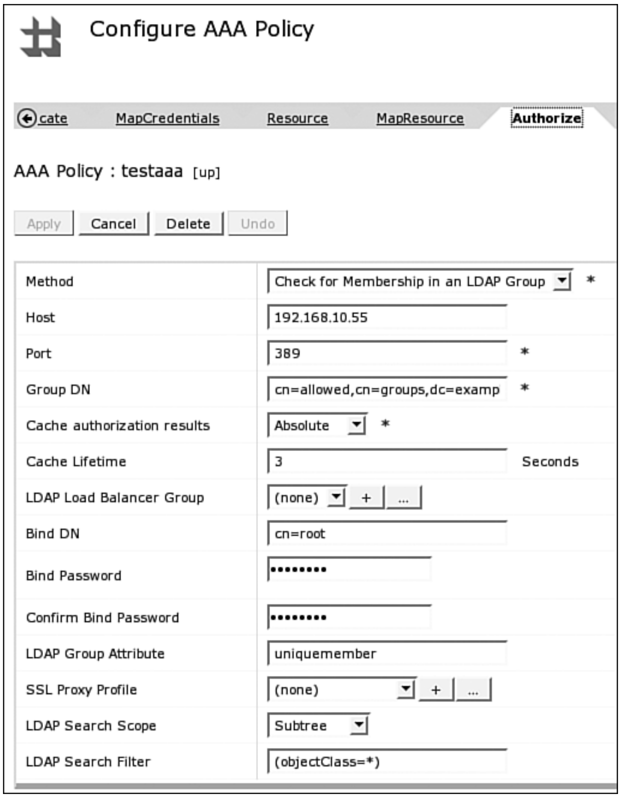
Figure 16-26 LDAP for authentication.

The LDAP DN Prefix and LDAP DN Suffix fields are used to build up a "template" for what the user lookup should look like. The username input from the EI phase will be included in the template, and the resulting distinguished name will be used along with the password from the EI phase. For example, if the username provided was bob, it would create a DN of `uid=bob,cn=users, dc=example,dc=com`. An LDAP bind will be attempted to the server using this distinguished name and the password provided, and if the bind is successful, then the user will be deemed to have authenticated successfully. Otherwise authentication will be deemed to have failed.

The final four fields, the Bind DN and Password and the LDAP Search Attribute fields, are used only in the special circumstance when the password from the Extract Identity stage happened to be a WS-Security UsernameToken PasswordDigest. In that specific case, this specified DN and password are used to bind to the LDAP server so that it can search for a digest, as specified in the LDAP Search Attribute field, to compare the provided digest against. This enables LDAP authentication even when the password is a hash of the plain-text password, which is a common use case in WS-Security.

Authorization

Figure 16-27 shows an example of configuring the same LDAP server as before, but this time for the authorization stage.



AAA Policy : testaaa [up]

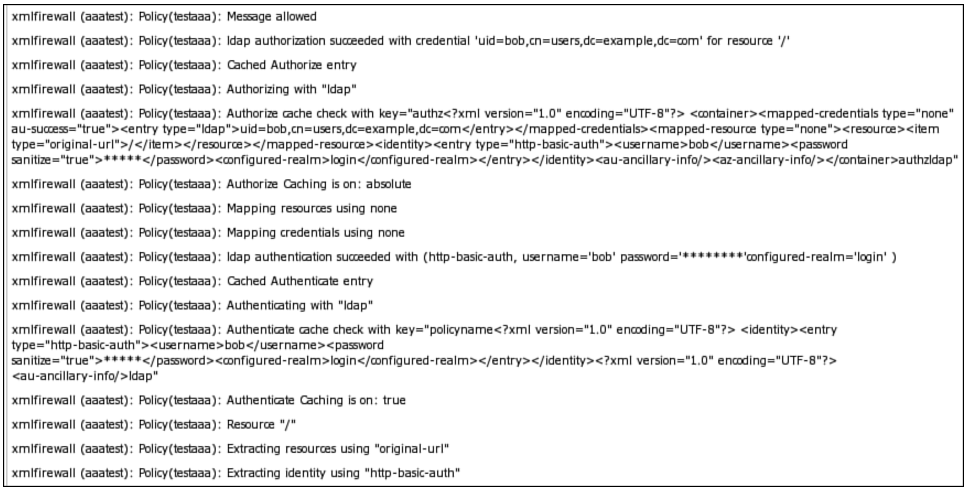
Method	Check for Membership in an LDAP Group *
Host	192.168.10.55
Port	389 *
Group DN	cn=allowed,cn=groups,dc=examp *
Cache authorization results	Absolute *
Cache Lifetime	3 Seconds
LDAP Load Balancer Group	(none)
Bind DN	cn=root
Bind Password	••••••••
Confirm Bind Password	••••••••
LDAP Group Attribute	uniquemember
SSL Proxy Profile	(none)
LDAP Search Scope	Subtree
LDAP Search Filter	(objectClass=*)

Figure 16-27 LDAP for authorization.

The built-in LDAP authorization is relatively simple. It allows for searching the LDAP server for membership of a specific group. That group is specified in the Group DN field. To perform the search, an authenticated bind is made to the LDAP server using the Bind DN and Bind Password provided. The search is then carried out using the LDAP Search Filter and the LDAP Group Attribute. If the authenticated user is found to be a member of the specified group, authorization is deemed to have been successful; if not, then it has failed.

Under the Covers

Let's examine how the authentication and authorization work here in a little more detail. We have configured a AAA policy, which contains the previous LDAP authentication and authorization stages, and we put it inside a AAA action in the Processing Policy of a simple XML firewall. The AAA policy extracts identity using basic authentication by processing an incoming "Authorization: Basic" HTTP header, and passes the username and password into the LDAP processing stages. Figure 16-28 shows a snippet of the processing log when this policy is run.



```
xmlfirewall (aaatest): Policy(testaaa): Message allowed
xmlfirewall (aaatest): Policy(testaaa): ldap authorization succeeded with credential 'uid=bob,cn=users,dc=example,dc=com' for resource '/'
xmlfirewall (aaatest): Policy(testaaa): Cached Authorize entry
xmlfirewall (aaatest): Policy(testaaa): Authorizing with "ldap"
xmlfirewall (aaatest): Policy(testaaa): Authorize cache check with key="authz<?xml version="1.0" encoding="UTF-8"?> <container><mapped-credentials type="none"
au-success="true"><entry type="ldap">uid=bob,cn=users,dc=example,dc=com</entry></mapped-credentials><mapped-resource type="none"><resource><item
type="original-url">/</item></resource></mapped-resource><identity><entry type="http-basic-auth"><username>bob</username><password
sanitize="true">*****</password><configured-realm>login</configured-realm></entry></identity><au-ancillary-info/><az-ancillary-info/></container>authzldap"
xmlfirewall (aaatest): Policy(testaaa): Authorize Caching is on: absolute
xmlfirewall (aaatest): Policy(testaaa): Mapping resources using none
xmlfirewall (aaatest): Policy(testaaa): Mapping credentials using none
xmlfirewall (aaatest): Policy(testaaa): ldap authentication succeeded with (http-basic-auth, username='bob' password='********'configured-realm='login' )
xmlfirewall (aaatest): Policy(testaaa): Cached Authenticate entry
xmlfirewall (aaatest): Policy(testaaa): Authenticating with "ldap"
xmlfirewall (aaatest): Policy(testaaa): Authenticate cache check with key="policyname<?xml version="1.0" encoding="UTF-8"?> <identity><entry
type="http-basic-auth"><username>bob</username><password
sanitize="true">*****</password><configured-realm>login</configured-realm></entry></identity><?xml version="1.0" encoding="UTF-8"?>
<au-ancillary-info/>ldap"
xmlfirewall (aaatest): Policy(testaaa): Authenticate Caching is on: true
xmlfirewall (aaatest): Policy(testaaa): Resource "/"
xmlfirewall (aaatest): Policy(testaaa): Extracting resources using "original-url"
xmlfirewall (aaatest): Policy(testaaa): Extracting identity using "http-basic-auth"
```

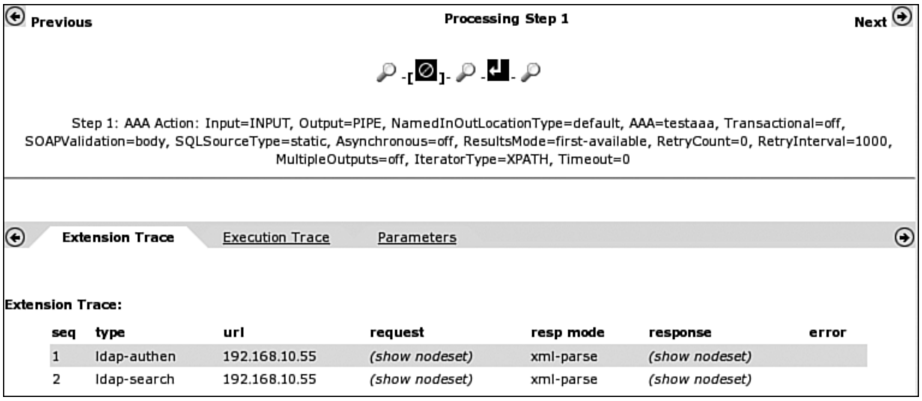
Figure 16-28 Relevant snippet of log.

We can see the processing extract the identity using http-basic-auth, and the resource using "original-url." We then see a search in the authentication cache to see whether we already have a cached authentication entry for these credentials. Because we do not, a new call is made to the LDAP server, which succeeds, and the result is stored in the cache. (So that if this call is repeated within the cache validity lifetime, by default three seconds, a new call to LDAP will not be performed for authentication.)

No mapping of credentials or resources is required, so we check our authorization cache to see if it contains an entry for this user accessing this specific resource. Because again it does not, we call LDAP and check to see whether the user is in the specific group required, and because he is, we deem the authorization to be successful and the message is allowed.

These logs are useful at a high level, but to gain more insight as to what actually happens under the covers, we need to look into the probe. There are two main areas where more information about the LDAP call can be obtained: in the extension trace for the AAA action, and in context variables that specifically exist for debugging purposes when the probe is enabled.

Figure 16-29 shows the extension trace for the AAA action in our Processing Policy.



Step 1: AAA Action: Input=INPUT, Output=PIPE, NamedInOutLocationType=default, AAA=testaaa, Transactional=off, SOAPValidation=body, SQLSourceType=static, Asynchronous=off, ResultsMode=first-available, RetryCount=0, RetryInterval=1000, MultipleOutputs=off, IteratorType=XPATH, Timeout=0

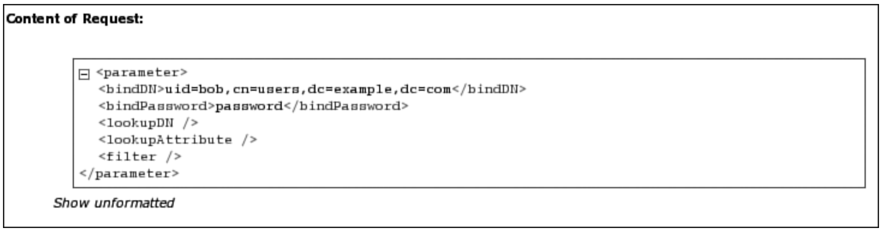
Extension Trace:

seq	type	url	request	resp mode	response	error
1	ldap-authen	192.168.10.55	*(show nodeset)*	xml-parse	*(show nodeset)*	
2	ldap-search	192.168.10.55	*(show nodeset)*	xml-parse	*(show nodeset)*	

Figure 16-29 AAA action extension trace.

We can see from the extension trace that two LDAP calls are made during this AAA action. The first is an authenticate call; that is, DataPower attempted to bind to the LDAP server as a specific user, specifying his password. The second is an LDAP search; that is, DataPower bound as a user authorized to perform searches and ran an LDAP search (in this case, to see whether our user is in a specific group).

By clicking show nodeset, we can see the parameters used by DataPower for the request and the return of the extension function call in the response. The request for the authentication call is shown in Figure 16-30.



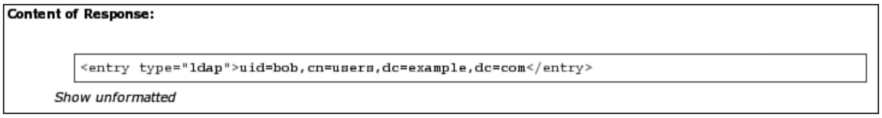
Content of Request:

```
⊟ <parameter>
  <bindDN>uid=bob,cn=users,dc=example,dc=com</bindDN>
  <bindPassword>password</bindPassword>
  <lookupDN />
  <lookupAttribute />
  <filter />
</parameter>
```

Show unformatted

Figure 16-30 Request parameters used for LDAP authentication call.

It is clear from the probe that DataPower tried to bind with a DN of `uid=bob,cn=users, dc=example,dc=com`, and a password of password. This information is made up of the LDAP DN Prefix `uid` and LDAP DN Suffix `cn=users,dc=example,dc=com`, shown previously in Figure 16-26, along with the username and password provided in basic auth (user of `bob`, with a password of `password`). The response to this call is shown in Figure 16-31.


Content of Response:

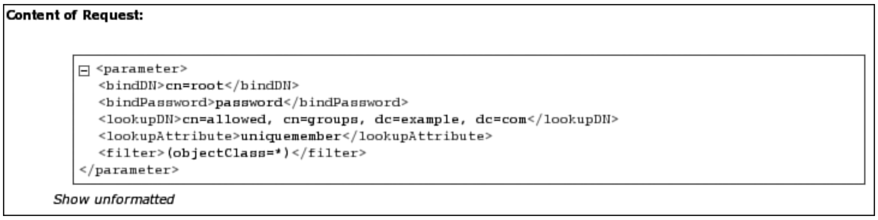
```
<entry type="ldap">uid=bob,cn=users,dc=example,dc=com</entry>
```

Show unformatted

Figure 16-31 Response from LDAP authentication call.

The response shows the whole LDAP distinguished name for the user that DataPower was able to bind as; this becomes the output credential.

Likewise, we can see the details of the call to the LDAP server for authorization. This is shown in Figure 16-32.


Content of Request:

```
⊟ <parameter>
    <bindDN>cn=root</bindDN>
    <bindPassword>password</bindPassword>
    <lookupDN>cn=allowed, cn=groups, dc=example, dc=com</lookupDN>
    <lookupAttribute>uniquemember</lookupAttribute>
    <filter>(objectClass=*)</filter>
</parameter>
```

Show unformatted

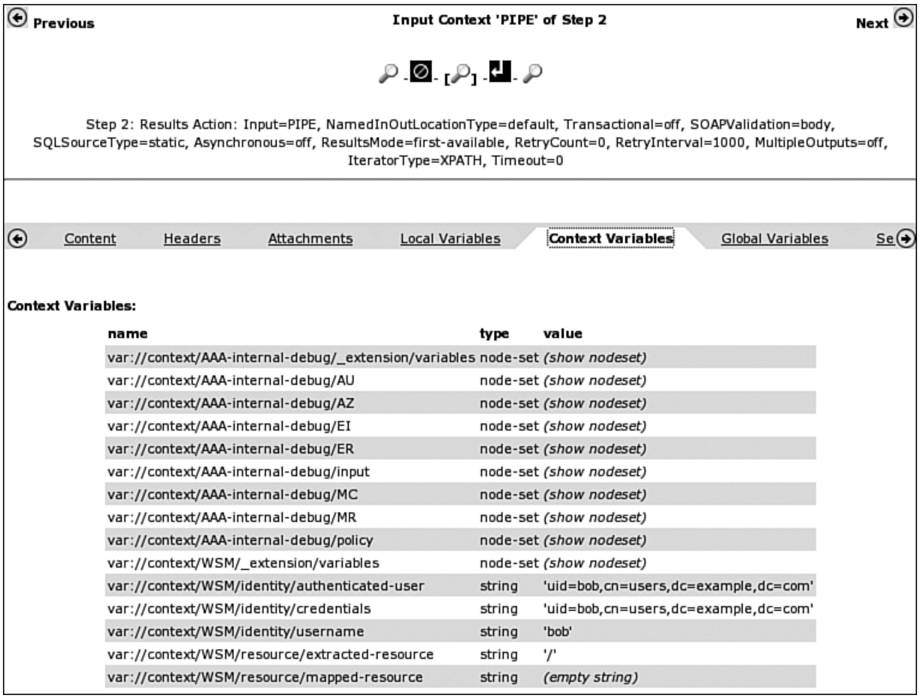
Figure 16-32 Request for LDAP authorization.

The request here is made up of the Bind DN, Bind Password, Group DN, LDAP Group Attribute, and LDAP Search Filter parameters from Figure 16-27. DataPower binds as the user specified in the Bind DN, providing the Bind Password, and then performs a search using the other parameters to ensure that the distinguished name, which was the result of the authentication stage (or the map credential stage if one was defined), is a member of the specified group.

The information from the AAA action extension trace is very useful. Sometimes, however, your chosen authentication or authorization mechanism might not provide any extension trace. In those instances, similar information is also available via specific debug variables in the probe.

Note that these variables *are* normal DataPower variables, and could theoretically be manipulated programmatically if desired. However, the variables exist *only* when the probe is enabled. Because the probe should never be enabled in a production environment, this means that the variables are unlikely to be of use programmatically for anything other than debugging specific issues. Because they are directly viewable in the probe itself, it is unlikely that there is any value in accessing the variables from a stylesheet. The variables as displayed in the probe are shown in Figure 16-33.

Inside these variables is fundamentally the same data as shown in the previous extension trace. However, the variables show the actual structure of how DataPower passes the information around between the various stages. For instance, the AU stage is shown in Figure 16-34.

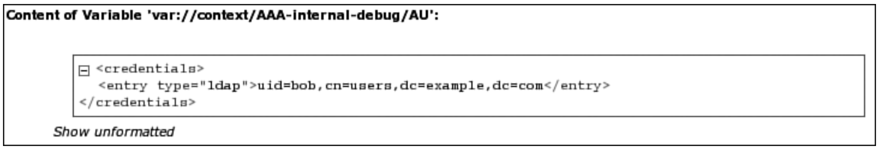

Previous Input Context 'PIPE' of Step 2 Next

Step 2: Results Action: Input=PIPE, NamedInOutLocationType=default, Transactional=off, SOAPValidation=body, SQLSourceType=static, Asynchronous=off, ResultsMode=first-available, RetryCount=0, RetryInterval=1000, MultipleOutputs=off, IteratorType=XPATH, Timeout=0

Context Variables:

name	type	value
var://context/AAA-internal-debug/_extension/variables	node-set	*(show nodeset)*
var://context/AAA-internal-debug/AU	node-set	*(show nodeset)*
var://context/AAA-internal-debug/AZ	node-set	*(show nodeset)*
var://context/AAA-internal-debug/EI	node-set	*(show nodeset)*
var://context/AAA-internal-debug/ER	node-set	*(show nodeset)*
var://context/AAA-internal-debug/input	node-set	*(show nodeset)*
var://context/AAA-internal-debug/MC	node-set	*(show nodeset)*
var://context/AAA-internal-debug/MR	node-set	*(show nodeset)*
var://context/AAA-internal-debug/policy	node-set	*(show nodeset)*
var://context/WSM/_extension/variables	node-set	*(show nodeset)*
var://context/WSM/identity/authenticated-user	string	'uid=bob,cn=users,dc=example,dc=com'
var://context/WSM/identity/credentials	string	'uid=bob,cn=users,dc=example,dc=com'
var://context/WSM/identity/username	string	'bob'
var://context/WSM/resource/extracted-resource	string	'/'
var://context/WSM/resource/mapped-resource	string	*(empty string)*

Figure 16-33 AAA internal debugging variables.

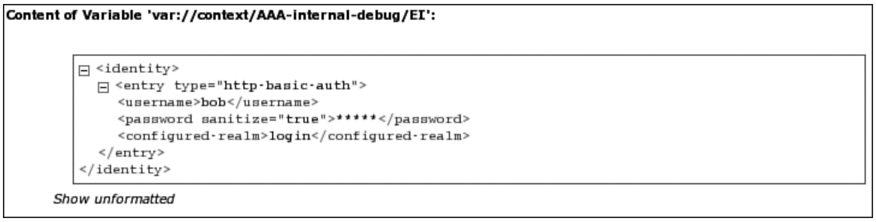

Content of Variable 'var://context/AAA-internal-debug/AU':

```
⊟ <credentials>
   <entry type="ldap">uid=bob,cn=users,dc=example,dc=com</entry>
</credentials>
```

Show unformatted

Figure 16-34 The AU internal debugging variable.

This shows the identity that was obtained in the authentication stage, but it is wrapped in a <credentials/> tag. This structure is used as the input to the next stage, Map Credentials. Likewise the EI variable is shown in Figure 16-35.


Content of Variable 'var://context/AAA-internal-debug/EI':

```
⊟ <identity>
  ⊟ <entry type="http-basic-auth">
      <username>bob</username>
      <password sanitize="true">*****</password>
      <configured-realm>login</configured-realm>
    </entry>
</identity>
```

Show unformatted

Figure 16-35 The EI internal debugging variable.

This variable shows the extracted identity as an <entry/> inside an <identity/> tag. This structure is used because there can actually be more than one extracted identity, so multiple entries are required, with each entry containing information relevant to the specific type of identity extracted. (For example the realm used for basic auth is given alongside the username and password.)

This information is invaluable when customizing the AAA process using XSLT. An example of customizing with XSLT is given in Chapter 17, which demonstrates how useful the variables are.

Real-World Policy

The final example for this chapter presents a AAA policy, which is representative of the kind of complexity that might be found in real-world situations. At a high level, the policy can be described as follows:

- Extract identity based on a combination of a specific SSL Certificate and a WS-Security Username Token.
- Authenticate the identity using an on-box AAA Info file, to enforce the multiple identities as extracted.
- Identify the resource based on an XPath expression that extracts application specific routing metadata.
- Authorize the connection with an XACML document using the on-box Policy Decision Point.
- Generate a SAML token for the connection to the back end to propagate the credentials to a SAML capable service. (The backend is assumed in this scenario to explicitly trust the DataPower appliance, with that trust built using a mutually authenticated SSL tunnel; this technique will be described in Chapter 18.)

Each of these is described in the following sections.

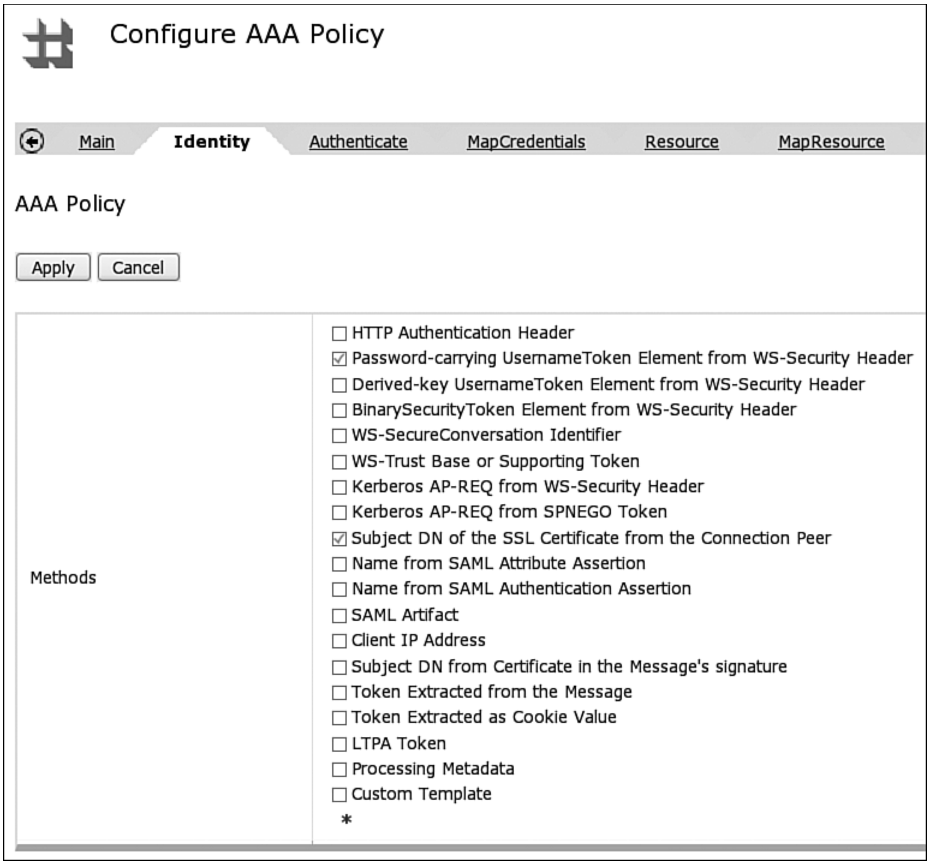
Identity Extraction

The configuration of identity extraction is done by simply selecting from the WebGUI. However in this case, we actually select more than one method, as shown in Figure 16-36.

By selecting more than one method, we are configuring the policy to extract one or both of the identities if they are present; there is no statement here that one or the other is compulsory. (This comes in the authentication stage.)

Authentication

The authentication stage is where we enforce the specific identities required. For this example, the AAA Info file is useful, because it can enforce multiple required credentials in a single authenticate stanza, as shown in Listing 16-7.


Configure AAA Policy

Main | **Identity** | Authenticate | MapCredentials | Resource | MapResource

AAA Policy

Apply | Cancel

Methods

- ☐ HTTP Authentication Header
- ☑ Password-carrying UsernameToken Element from WS-Security Header
- ☐ Derived-key UsernameToken Element from WS-Security Header
- ☐ BinarySecurityToken Element from WS-Security Header
- ☐ WS-SecureConversation Identifier
- ☐ WS-Trust Base or Supporting Token
- ☐ Kerberos AP-REQ from WS-Security Header
- ☐ Kerberos AP-REQ from SPNEGO Token
- ☑ Subject DN of the SSL Certificate from the Connection Peer
- ☐ Name from SAML Attribute Assertion
- ☐ Name from SAML Authentication Assertion
- ☐ SAML Artifact
- ☐ Client IP Address
- ☐ Subject DN from Certificate in the Message's signature
- ☐ Token Extracted from the Message
- ☐ Token Extracted as Cookie Value
- ☐ LTPA Token
- ☐ Processing Metadata
- ☐ Custom Template

*

Figure 16-36 Multiple methods of identity extraction.

Listing 16-7 Multiple Credentials

```
<aaa:Authenticate>
    <aaa:Username>bob</aaa:Username>
    <aaa:Password>passw0rd</aaa:Password>
    <aaa:DN>CN=Bob,OU=Workers,DC=SomeCompany,DC=com</aaa:DN>
    <aaa:OutputCredential>bob</aaa:OutputCredential>
</aaa:Authenticate>
```

This stanza is different from those presented earlier. What it says is that to authenticate as the user bob, specified as the OutputCredential, the user must present *both* the username and password *and* the distinguished name from the certificate. Presenting only one or the other of these is not enough. The AAA Info file is the only authentication method that, out of the box, can enforce that more than one type of credential must be present for a single user within a single AAA Policy. To perform similar functionality for other authentication methods, we would have to either have multiple AAA policies or use custom XSLT.

Authorization

After our user has been authenticated, we need to make an authorization decision. To provide a common way of specifying authorization constraints, the company has decided to specify them in an XACML policy file.

The eXtensible Access Control Markup Language (XACML) is a language that enables generic XML declarations of access control policies. DataPower acts as a Policy Enforcement Point for XACML policies; that is, it is able to take the requests it receives and transform them into XACML requests for access to specific resources. It is also capable of acting as a Policy Decision Point, and actually deciding whether access should be granted by interpreting the declared policies and applying them as relevant to the requests it receives. The on-box PDP functionality is capable of using an XACML policy file, and for this scenario that is enough.

Listing 16-8 XACML Policy

```
<Policy PolicyId="LoginPolicy"
 RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
 rule-combining- algorithm:permit-overrides">
<Rule RuleId="LoginRule" Effect="Permit">
<Target>
 <Subjects>
  <Attribute
   AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
   DataType="http://www.w3.org/2001/XMLSchema#string">
   <AttributeValue>
    bob
   </AttributeValue>
  </Attribute>
 </Subjects>
 <Resources>
  <ResourceMatch
   MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
   <AttributeValue
   DataType="http://www.w3.org/2001/XMLSchema#string">
    /target/url
   </AttributeValue>
   <ResourceAttributeDesignator
   DataType="http://www.w3.org/2001/XMLSchema#string"
   AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
  </ResourceMatch>
 </Resources>
</Target>
</Rule>
<Rule RuleId="FinalRule" Effect="Deny"/>
</Policy>
```

Listing 16-9 XACML Request

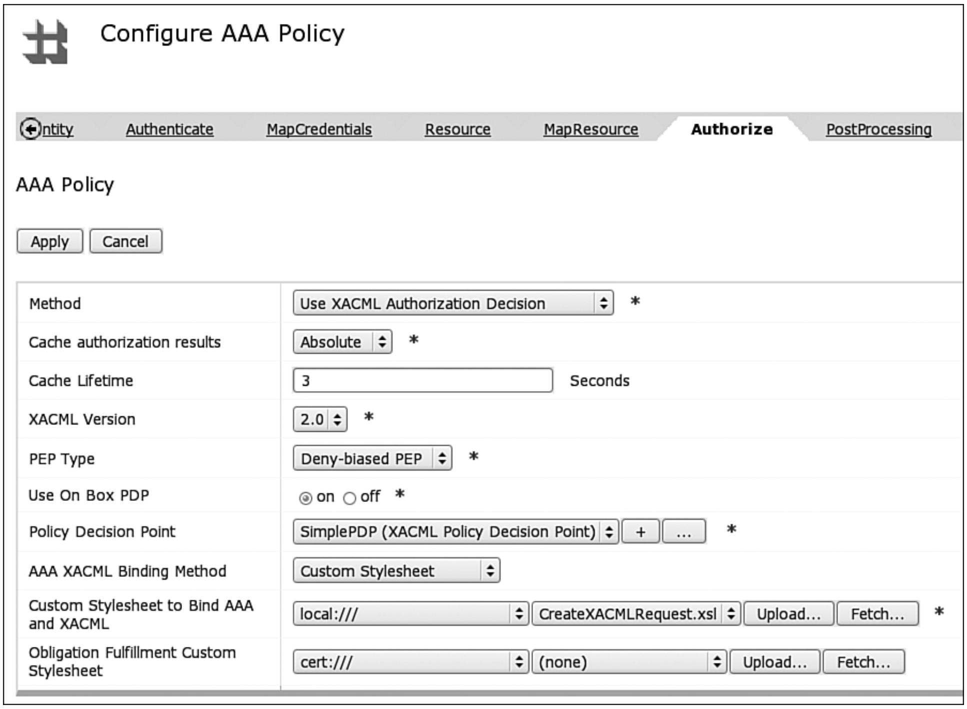
```
<Request>
 <Subject>
  <Attribute
   AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
   DataType="http://www.w3.org/2001/XMLSchema#string">
   <AttributeValue>
    bob
   </AttributeValue>
  </Attribute>
 </Subject>
 <Resource>
  <Attribute
   AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
   DataType="http://www.w3.org/2001/XMLSchema#string">
   <AttributeValue>
    /target/url
   </AttributeValue>
  </Attribute>
 </Resource>
</Request>
```

Chapter 17 goes into more detail about how to actually write XSLT to customize the AAA processing. Meanwhile, the configuration needs to be put together under the Authorize tab, as shown in Figure 16-37.

After this is done, the output of the Resource and the Authenticate stages is converted to a XACML request using the CreateXACMLRequest.xsl stylesheet, and then passed to the SimplePDP on-box Policy Decision Point, which contains the policy shown in Listing 16-8.

Post Processing

Finally, after we have authenticated our request, defined our resource, and authorized the entity to access the resource, we need to propagate that request to the backend. Recall that our backend is capable of understanding SAML, and we want to propagate identity using a SAML assertion. The configuration for this is shown in Figure 16-38.

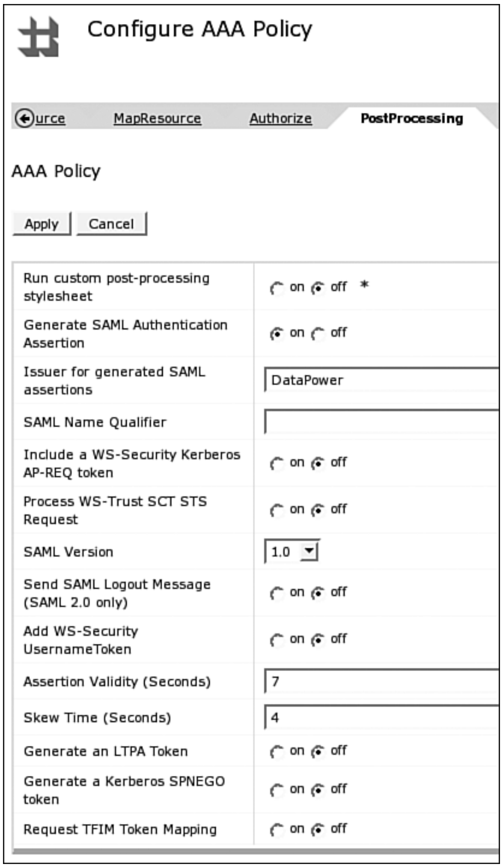

Configure AAA Policy

AAA Policy

Method	Use XACML Authorization Decision *
Cache authorization results	Absolute *
Cache Lifetime	3 Seconds
XACML Version	2.0 *
PEP Type	Deny-biased PEP *
Use On Box PDP	[✓ on] off *
Policy Decision Point	SimplePDP (XACML Policy Decision Point) + ... *
AAA XACML Binding Method	Custom Stylesheet
Custom Stylesheet to Bind AAA and XACML	local:/// CreateXACMLRequest.xsl Upload... Fetch... *
Obligation Fulfillment Custom Stylesheet	cert:/// (none) Upload... Fetch...

Figure 16-37 The XACML on-box configuration.

The issuer of the assertion is configured as DataPower and the assertion is valid for seven seconds with a possible skew of four seconds. (The skew is to allow for possible latencies in date and time settings on different servers.) For short-lived assertions, which are designed to be processed immediately, this should be enough. SAML is covered in more depth in Chapter 19.



Run custom post-processing stylesheet	on [✓ off] *
Generate SAML Authentication Assertion	[✓ on] off
Issuer for generated SAML assertions	DataPower
SAML Name Qualifier	
Include a WS-Security Kerberos AP-REQ token	on [✓ off]
Process WS-Trust SCT STS Request	on [✓ off]
SAML Version	1.0
Send SAML Logout Message (SAML 2.0 only)	on [✓ off]
Add WS-Security UsernameToken	on [✓ off]
Assertion Validity (Seconds)	7
Skew Time (Seconds)	4
Generate an LTPA Token	on [✓ off]
Generate a Kerberos SPNEGO token	on [✓ off]
Request TFIM Token Mapping	on [✓ off]

Figure 16-38 Generation of a SAML assertion.

Summary

This chapter provided an introduction into the DataPower AAA framework. We have shown how flexible and powerful it is, allowing you to customize access control to unprecedented levels and integrate with almost any other security system. Chapter 17 goes through some of the deeper levels of customization, explaining exactly how custom XSLT can be used to have complete control over exactly who can do what with your services. You ain't seen nothin' yet!