Oracle® Adaptive Access Manager

Developer's Guide

Release 10*g* (10.1.4.5)

E12052-03

May 2009

Oracle Adaptive Access Manager Developer's Guide, Release 10*g* (10.1.4.5)

E12052-03

Primary Author: Priscilla Lee

Contributors: Mandar Bhatkhande, Sree Chitturi, Josh Davis, Bosco Durai, Luke Harris, Prakash Hegde, Daniel Joyce, Mark Karlstrand, Derick Leo, Karl Miller, Valarie Moore, Srinivas Nagandla, Madhan Neethiraj, Paresh Raote, Jim Redfield, Uday Sambhara, Kamal Singh, Nandini Subramani, Vidhya Subramanian, Sachin Vanungare, and Saphia Yunaeva

Contents

Preface

This Developer's Guide provides information about Oracle Adaptive Access Manager integrations.

The Preface covers the following topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

Audience

This guide is intended for administrators and developers who are responsible for integrating Oracle Adaptive Access Manager.

This guide assumes that you are familiar with your Web servers, Oracle Adaptive Access Manager, .net and Java, and the product that you are integrating.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at `http://www.fcc.gov/cgb/consumerfacts/trs.html`, and a list of phone numbers is available at `http://www.fcc.gov/cgb/dro/trsphonebk.html`.

Related Documents

For more information, see the following documents in the Oracle Adaptive Access Manager 10.1.4.5 documentation set:

- *Oracle Adaptive Access Manager Release Notes*
- *Oracle Adaptive Access Manager Administrator's Guide*
- *Oracle Adaptive Access Manager Reference Guide*
- *Oracle Adaptive Access Manager Developer's Guide*
- *Oracle Adaptive Access Manager Concepts*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Supported Integrations

This chapter contains a brief introduction to the deployment integration options and the feature integration options available in Oracle Adaptive Access Manager.

1.1 Integration Deployment Options

All the integration deployment options are listed below. For detailed information, please refer to the chapters in this guide.

Native and Web Services Integration

The web application communicates with Adaptive Risk Manager Online using the Adaptive Risk Manager Online Native Client API or through Web Services.

For an introduction to API integration, refer to Chapter 2, "API Integration."

For information on native and web services integration, refer to Chapter 3, "Native Integration .net," and Chapter 4, "Native Integration Java."

Static Linked Integration

Most of the native API are wrappers over the SOAP API that is published by the Adaptive Risk Manager Online Server and written in the client's native application language. The static-linked integration is another option available for integrations using Java language. There will be no SOAP calls to Adaptive Risk Manager in this option; instead the API implementation runs within the client application itself.

For an introduction to API integration, refer to Chapter 2, "API Integration."

For information on static linked integration, refer to Chapter 4, "Native Integration Java."

Universal Installation Option Integration

Oracle Adaptive Access Manager's Universal Installation Option (UIO) is a proxy-based deployment of Adaptive Risk Manager and Adaptive Strong Authenticator that requires little or no integration with client web applications.

A proxy intercepts site traffic and routes it through Adaptive Risk Manager Online for Strong Authentication.

For information on the integration of the Oracle Adaptive Access Manager's Installation Option, refer to the following chapters:

- Chapter 5, "Oracle Adaptive Access Manager Proxy"
- Chapter 6, "Configuring Adaptive Strong Authenticator"
- Chapter 7, "Authenticator Properties"

Oracle Access Manager

Oracle Adaptive Access Manager is integrated with Oracle Access Manager to allow Adaptive Strong Authenticator's authentication pads to identify users attempting to access Oracle Access Manager's protected applications.

For information on integrating Oracle Adaptive Access Manager with Oracle Access Manager, refer to Chapter 8, "Oracle Access Manager Integration."

1.2 Features Integration Options

The available features integration options are documented below.

Auto-learning

Auto-learning is a set of features that perform adaptive evaluations of risk.

For information on how to integrate it with Adaptive Risk Manager, refer to Chapter 9, "Auto-learning."

Configurable Actions

Oracle Adaptive Access Manager provides Configurable Actions, a feature which allows users to create new supplementary actions that are triggered based on the result action and/or based on the risk scoring after a Runtime execution.

For information on how to add a Configurable Action and integrate it with the Adaptive Risk Manager software, refer to Chapter 10, "Configurable Actions."

Cases

The expiry/overdue behavior can be configured for cases.

For information on how to change the default behavior, refer to Chapter 11, "Configuring Expiry/Overdue for Cases."

Part I

Native and SOAP Integrations

Part I contains the following chapters:

2 API Integration

This chapter contains the guidelines to natively integrate the client portion of Adaptive Risk Manager Online solutions. In an API integration, the client application invokes the Adaptive Risk Manager Online APIs directly and manages the authentication and challenge flows.

Note: Adaptive Strong Authenticator is not used in this integration; all of the related screens need coding into the client application using the API.

Oracle provides the APIs to fingerprint the devices, collect authentication/transaction logs, run security and business rules, and challenge the user by using Adaptive Risk Manager Online's KBA. Adaptive Risk Manager Online also provides the utility APIs to generate authentication pads like KeyPad, TextPad, and QuestionPad.

API integration of Adaptive Risk Manager Online provides various advantages--some of which are highlighted below:

- Flexibility in managing and controlling the authentication process flow.
- Ability to change the default user registration flow.
- Capability to share session data between existing applications and the Adaptive Risk Manager Online application. For example, the existing login session ID can be passed on to Adaptive Risk Manager Online API calls.

This chapter contains the guidelines for integrating:

- Only the Adaptive Risk Manager
- Adaptive Risk Manager and KBA
- Adaptive Risk Manager, KBA and Authentication devices
- AuthentiPad (Keypad, PinPad, and other pads)
- Customer Care API

2.1 Application (API) Integration

Adaptive Risk Manager Online's components are software- and hardware-independent when deployed in a stand-alone environment using the published Web services API over SOAP. Support is also available for native (Java/.NET) environments.

Basic familiarity with SOAP, Java, or .NET is the skill set requirement for integration.

API integration is available in two flavors:

- SOAP Service
- Native APIs
  - SOAP Service wrapper (in Java or .NET)
  - Static-linked libraries (in Java)

2.1.1 SOAP Services

Adaptive Risk Manager Online SOAP services consists of five major modules:

- **VCryptCommon** contains the common APIs.
- **VCryptTracker** contains the APIs for fingerprinting and collecting authentication and transaction logs.
- **VCryptAuth** contains the APIs for accessing the Adaptive Strong Authenticator and KBA modules.
- **VCryptRulesEngine** contains the APIs for running the rules.
- **VCryptCC** contains the APIs for invoking customer-care-related requests.

Using direct SOAP services is preferred if the client does not want to include any of the Adaptive Risk Manager Online client jars or DLL within their application. However, to use the Adaptive Strong Authenticator functionality, native Java or .NET you must use the native Java or .NET integration.

2.1.2 Native API

The native API consists of a wrapper over the SOAP API that is published by the Adaptive Risk Manager Online Server and written in the client's native application language. The native APIs construct the SOAP bodies for the Adaptive Risk Manager Online request and also invoke the SOAP requests.

API integration can be done using the SOAP as the underlying mechanism or statically linking (available only for Java integration) the Adaptive Risk Manager Online jars.

2.1.2.1 Adaptive Risk Manager Online Native Client API - Web Services/ SOAP

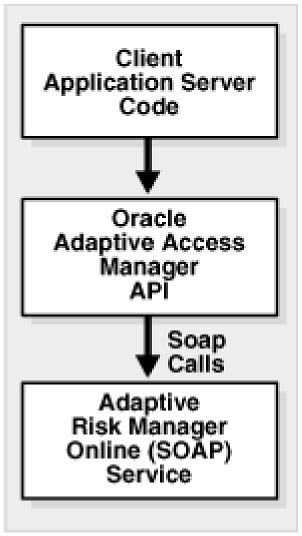
The construction of SOAP bodies and the SOAP calls help in abstracting the SOAP WSDL and other Web Services related issues.

Using native API, which is preferred over making direct SOAP calls, has a lot of advantages. A few advantages are listed below:

- The client library constructs the SOAP body and abstracts the SOAP nuances from the client application developer.
- Changes to any SOAP API signature does not require any code change from the application developer.
- Higher-level utility methods are available to extract parameters required by Adaptive Risk Manager Online directly from the HTTP Request and HTTP Session objects.
- APIs for encoding and decoding of some fingerprint data are available in native integration.

API libraries are available in Java, .NET and C++. In the Web Service configuration, these libraries have utility methods which make direct SOAP calls. The option requires lightweight client libraries (jars or dll) to be added to the client library part.

Figure 2–1 Web Services/SOAP Integration


Client Application Server Code
Oracle Adaptive Access Manager API
Soap Calls
Adaptive Risk Manager Online (SOAP) Service

2.1.2.2 Adaptive Risk Manager Online Native Client API - Static Linking

Clients using Java have the option to choose static linking. In static linking, the API doesn't make SOAP calls, instead they statically call the Adaptive Risk Manager Online engine APIs. With the static linking option, the client must include the Adaptive Risk Manager Online server jars and configure appropriate properties.

Although this option may provide slightly better performance, it may not be suitable for all clients.

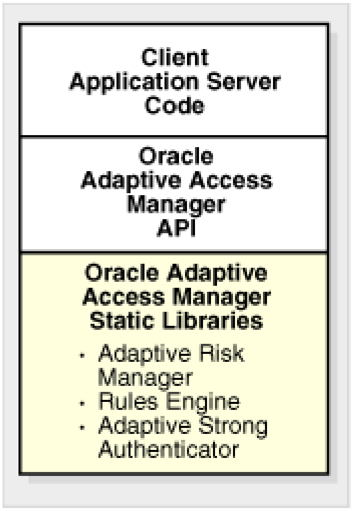
Advantages of static linking are

- No SOAP calls; eliminates creating and tearing down of TCP/IP connections.
- No network latencies.
- Load balancer not required.

Disadvantages of static linking are

- The client server/application server has to accommodate the extra resource required by the Adaptive Risk Manager Online engine.
- The client server/application server may not be able to load balance the requests.

Figure 2–2 Static linking



2.2 Integration Options

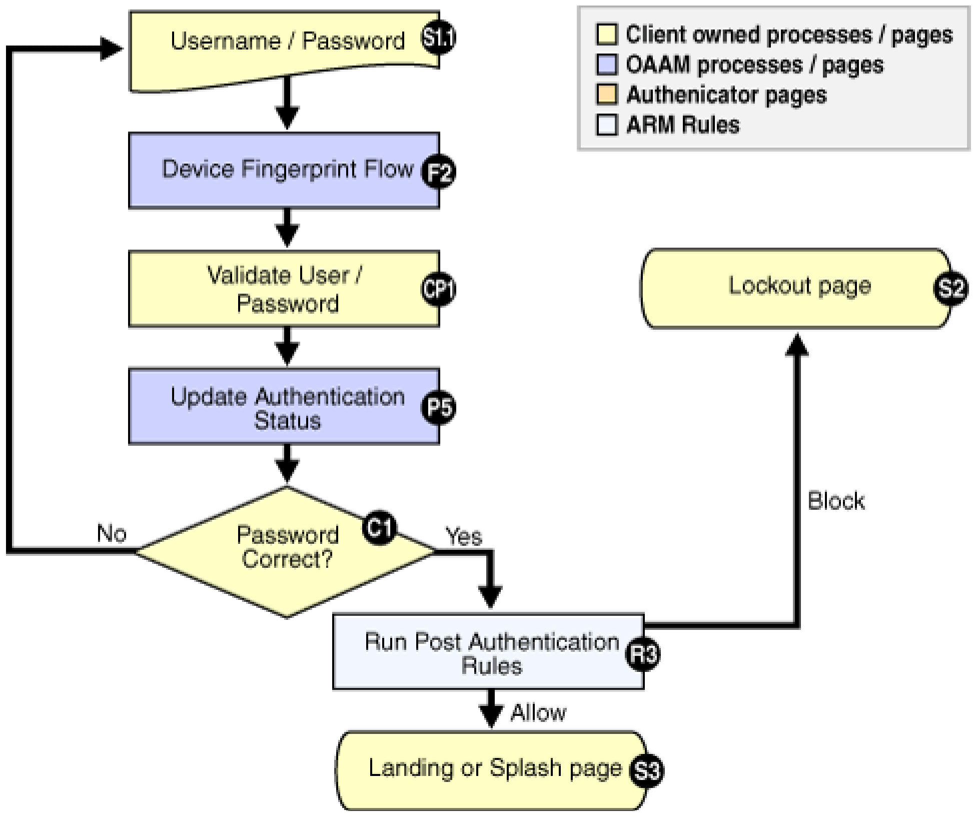
Clients can integrate Adaptive Risk Manager in a relatively short time frame and have their site secure from most fraud attacks. Integrating only the Adaptive Risk Manager doesn't require any change to the user experience.

Note: In this integration, Adaptive Strong Authenticator is not used.

2.2.1 Adaptive Risk Manager Only Scenario

A diagram of the Adaptive Risk Manager only scenario is shown below.

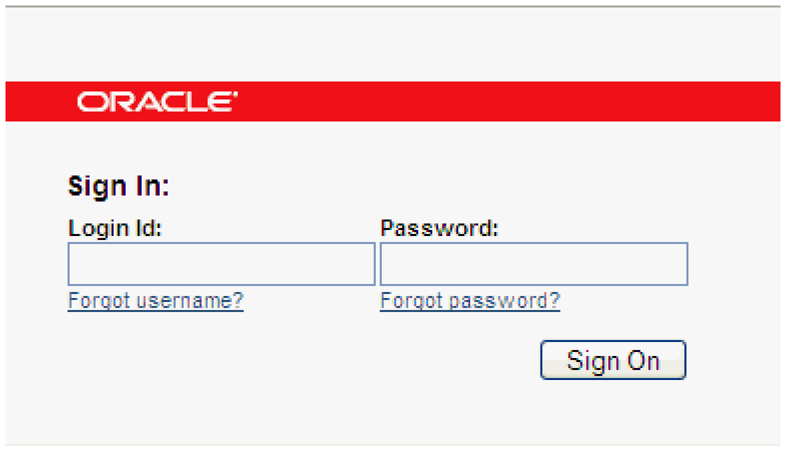
Figure 2–3 Adaptive Risk Manager Only Scenario



2.2.1.1 User/Password Page (S1.1)

The User/Password Page is the existing page currently used by the client. It contains the text box for both the username and password. There are no changes required for this page; however, the post from this page should display a transient (intermediate) refresh page.

Figure 2–4 Username/Password Page



2.2.1.2 Device Fingerprint Flow (F2)

This is the flow for fingerprinting the device.

Table 2–1 Device fingerprint Flow (F2) Reference APIs

Module	APIs
Server	VCryptTracker::updateLog()
Sample	handleJump.jsp and handleFlash.jsp

updateLog(): For information on updateLog(), the parameters, and how to use the parameters, refer to Chapter 4, "Native Integration Java."

handleJump.jsp:

- sets the client's time zone
- sets a secure cookie
- sets the browserfingerprint, sets status to "pending"
- calls the pre-auth rules. Expects an "allow" action to proceed, else "block" or "error" due to unrecognized action or server error
- stores bharosaSession
- forwards to password.jsp

handleFlash.jsp sets the flashCookie if browser is flash enabled

2.2.1.3 Validate User/ Passwd (CP1)

This is the client's existing process. The client invokes the local API to validate the user. The result of the authentication is passed on to the Adaptive Risk Manager Online Server.

Table 2–2 Validate User/ Passwd (CP1) Reference API

Module	API
Sample	handlePassword.jsp

handlePassword.jsp

- retrieves the password from the pad
- decodes the password
- validates the user

2.2.1.4 Update Authentication Status (P5)

After validating the user password, the status is updated in the Adaptive Risk Manager.

Table 2–3 Update Authentication Status (P5) Reference APIs

Module	APIs
Sample	handlePassword.jsp
BharosaHelper	BharosaHelper::updateStatus()

handlePassword.jsp:

- updates the status to "success" if user is valid, else to "invalid," "error," or "bad password"

2.2.1.5 Password Status (C1)

Based on the authentication status, the user is either taken to the retry page or to post authentication rule processing.

2.2.1.6 Post Authentication Rules (R3)

The post authentication rules are run after the user password is authenticated. The post authentication runtime contains security rules.

For example, common actions returned are

- **Allow**: Allow the authentication.
- **Block**: Block the user.
- **Challenge**: Challenge is returned if the user has registered questions. The option may not be available for Adaptive Risk Manager Only deployments.

Table 2–4 Post Authentication Rules (R3) Reference APIs

Module	APIs
Server	VCryptRulesEngine::processRules()
Sample	handlePassword.jsp
BharosaHelper	BharosaHelper::runPostAuthRules()

processRules(): For information on processRules(), the parameters, and how to use the parameters, refer to Chapter 4, "Native Integration Java."

handlePassword.jsp

- runs post-auth rules and one of the following actions:
  - REGISTER_USER_OPTIONAL
  - REGISTER_QUESTIONS
  - REGISTER_USER
  - CHALLENGE
  - BLOCK
  - ALLOW
  - SYSTEM_ERROR
- forwards to registerImageandPhrase during registration or challenge if the user is registered

2.2.1.7 Lock Out Page (S2)

The Lock Out Page is the page that the user is generally redirected to if he is blocked from authentication or if he is carrying out a transaction.

2.2.1.8 Landing or Splash Page (S3)

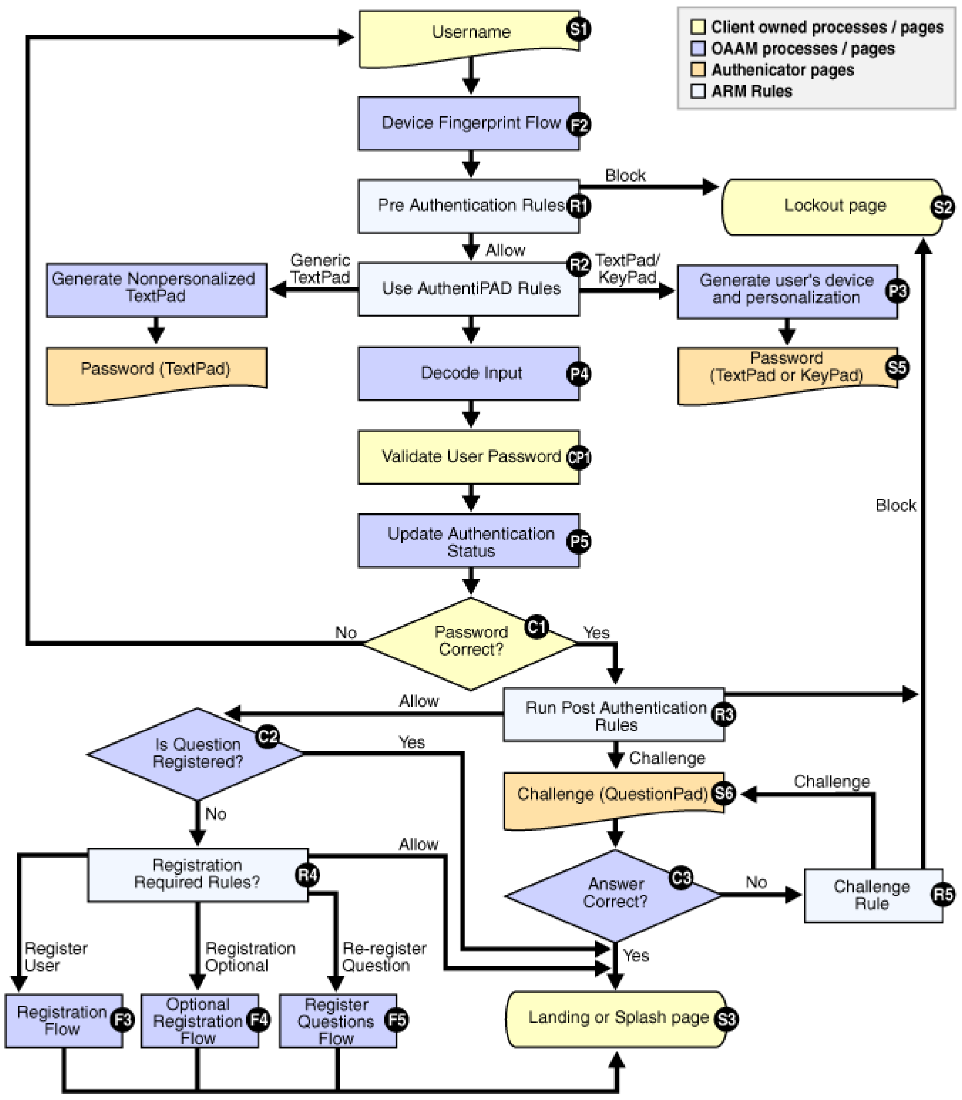
The Landing or Splash Page is the page where the user lands after a successful login.

2.2.2 Adaptive Risk Manager, Adaptive Strong Authenticator and KBA Scenario

This flow is a consolidation of the Adaptive Risk Manager, AuthentiPads and KBA solutions. The flows are determined by the rules that are run at different runtimes.

Note: The flows with models suggested in this section are example scenarios; there are other models and rules and other scenarios available.

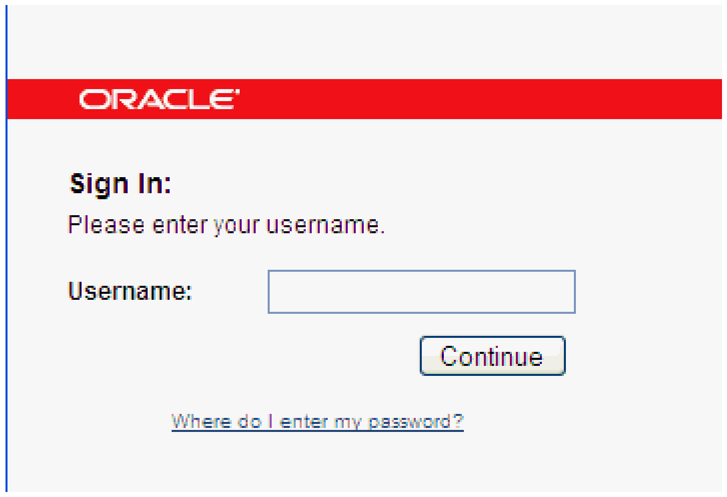
Figure 2–5 Adaptive Risk Manager, Adaptive Strong Authenticator and KBA Scenario



2.2.2.1 Username Page (S1)

When personalization (image and/or phrase) is used, the login page must be split into two pages. The username (login ID) is accepted from the first page and stored in the HTTP session. The username page is followed by a transient page for capturing the flash and secure cookies and for fingerprinting the device.

Figure 2–6 Username Page



2.2.2.2 Device Fingerprint Flow (F2)

This is the flow for fingerprinting the device.

Table 2–5 Device Fingerprint Flow (F2) Reference APIs

Module	APIs
Server	VCryptTracker::updateLog()
Sample	handleJump.jsp and handleFlash.jsp

updateLog(): For information on updateLog(), the parameters, and how to use the parameters, refer to Chapter 4, "Native Integration Java."

handleJump.jsp:

- sets the client's time zone
- sets a secure cookie
- sets the browserfingerprint, sets status to "pending"

handleFlash.jsp sets the flashCookie if browser is flash enabled

2.2.2.3 Pre Authentication Rules (R1)

Pre Authentication rules are run before the user is authenticated or shown his personal device and/or phrase.

The common actions are

- **Allow**: Allow the authentication flow to proceed.
- **Block**: Block the user.

Table 2–6 Pre Authentication Rules Reference APIs

Module	APIs
Server	VCryptRulesEngine::processRules()
Sample	handleJump.jsp
BharosaHelper	BharosaHelper::runPreAuthRules()

processRules(): For information on processRules(), the parameters, and how to use the parameters, refer to Chapter 4, "Native Integration Java."

handleJump.jsp:

- calls the pre-auth rules. Expects an "allow" action to proceed, else "block" or "error" due to unrecognized action or server error
- stores bharosaSession
- forwards to password.jsp

2.2.2.4 Use AuthentiPad Rules (R2)

This runtime runs the rules for determining which AuthentiPad is used. If the user has not registered, the rule returns the Generic TextPad. If the user is registered with Adaptive Risk Manager Online, either the personalized TextPad or KeyPad action will be returned.

The common actions are

- **Generic TextPad**: Use default generic TextPad.
- **TextPad**: Use personalized TextPad with phrase.
- **KeyPad**: Use personalized KeyPad with phrase.

Table 2–7 Use AuthentiPad Rules (R2) Reference APIs

Module	APIs
Server	VCryptRulesEngine::processRules()
Sample	password.jsp
BharosaHelper	BharosaHelper::getAuthentiPad()

processRules(): For information on processRules(), the parameters, and how to use the parameters, refer to Chapter 4, "Native Integration Java."

password.jsp:

- invokes rules to identify user's pad type, else uses the default KeyPad
- creates the pad, names it, and sets all initial background frames, buttons, and so on
- invokes kbimage.jsp as configured in the bharosa_client.properties
- forwards to handlePassword.jsp

2.2.2.5 Generate Non-Personalized TextPad (P2)

Generic TextPad and non-personalized TextPad are used for users who have not yet registered with Adaptive Risk Manager Online. A non-personalized textpad is shown below.

Figure 2–7 Non-Personalized TextPad

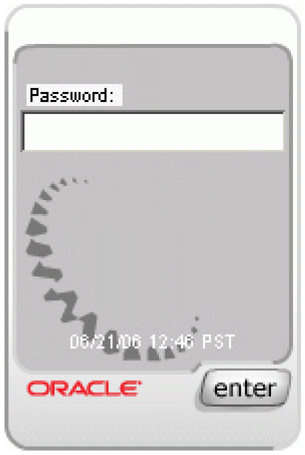


Table 2–8 Generate Non-Personalized TextPad (P2) Reference APIs

Module	APIs
Server	VCryptAuth::getUserByLoginId()
Sample	Password.jsp
BharosaHelper	BharosaHelper:: createPersonalizedAuthentiPad () BharosaHelper::createAuthentiPad()
Client	AuthentiPad::getHTML()

getUserByLoginId(): For information on getUserByLoginId(), the parameters, and how to use the parameters, refer to Chapter 4, "Native Integration Java."

password.jsp:

- invokes rules to identify user's pad type, else uses the default KeyPad
- creates the pad, names it, and sets all initial background frames, buttons, and so on
- invokes kbimage.jsp as configured in the bharosa_client.properties
- forwards to handlePassword.jsp

2.2.2.6 Generate Personalized TextPad or KeyPad (P3)

A textpad and a keypad are shown below.

Figure 2–8 Personalized TextPad

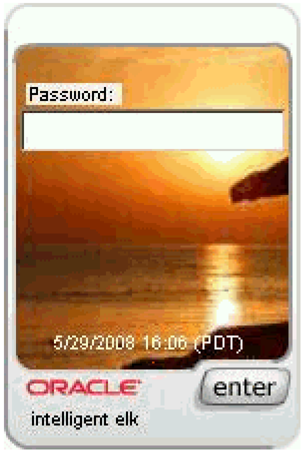
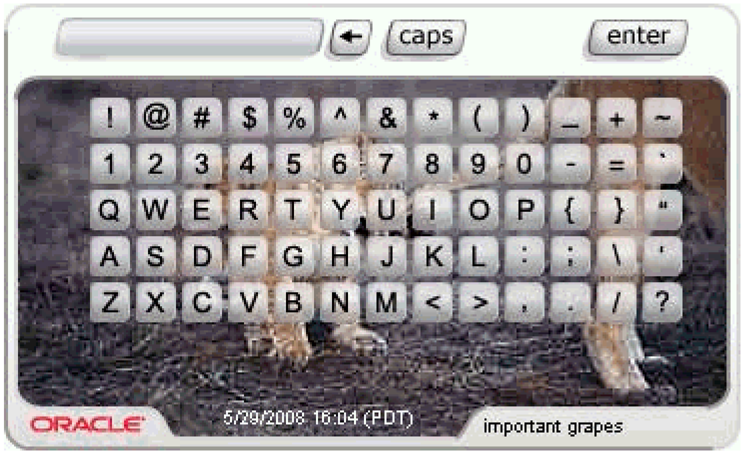


Figure 2–9 Personalized KeyPad



Personalized KeyPad or TextPad are similar to the Generic TextPad, except they use user-selected phrases and background images.

Table 2–9 Generate Personalized TextPad or KeyPad Reference APIs

Module	APIs
Server	VCryptAuth::getUserByLoginId()
Sample	password.jsp
BharosaHelper	BharosaHelper:: createPersonalizedAuthentiPad () BharosaHelper::createAuthentiPad()
Client	AuthentiPad::getHTML()

getUserByLoginId(): For information on getUserByLoginId(), the parameters, and how to use the parameters, refer to Chapter 4, "Native Integration Java."

password.jsp:

- invokes rules to identify user's pad type, else uses the default KeyPad
- creates the pad, names it, and sets all initial background frames, buttons, and so on
- invokes kbimage.jsp as configured in the bharosa_client.properties

- forwards to handlePassword.jsp

2.2.2.7 Display TextPad or KeyPad (S4 and S5)

The HTML snippet should be embedded in the password page. The HTML renders the TextPad or KeyPad using Javascript. There is a <img> tag, which makes a HTTP request to the server to get the TextPad or KeyPad image.

Table 2–10 Display TextPad or KeyPad Reference APIs

Module	APIs
Server	VCryptAuth::getUserByLoginId()
Sample	password.jsp kbimage.jsp

BharosaHelper	BharosaHelper:: createPersonalizedAuthentiPad ()
	BharosaHelper::createAuthentiPad()
	BharosaHelper::imageToStream()

Client	AuthentiPad::getHTML()
	KeyPadUtil::encryptImageToStream()

getUserByLoginId(): For information on getUserByLoginId(), the parameters, and how to use the parameters, refer to Chapter 4, "Native Integration Java."

password.jsp:

- invokes rules to identify user's pad type, else uses the default KeyPad
- creates the pad, names it, and sets all initial background frames, buttons, and so on
- invokes kbimage.jsp as configured in the bharosa_client.properties
- forwards to handlePassword.jsp

kbimage.jsp is the page that outputs the authenticator

2.2.2.8 Decode AuthentiPad Input (P4)

The data entered by the user is decoded by the Adaptive Risk Manager Online Utility API. The decoded value is in raw text format. The AuthentiPad, which had been used to generate the image, is used during the decoding. It is recommended that it be saved in the HTTP Session. The AuthentiPad object is serializable and can be stored in the database or file system.

Table 2–11 Decode AuthentiPad Input Reference APIs

Module	APIs	Notes
Sample	handlePassword.jsp	
BharosaHelper	BharosaHelper::decodePadInput()	This method removes the AuthentiPad object from the HTTP Session
Client	KeyPadUtil::decodeKeyPadCode	

handlePassword.jsp

- retrieves the password from the pad
- decodes the password

- validates the user

2.2.2.9 Validate User/ Passwd (CP1)

This is the client's existing process. The client invokes the local API to validate the user. The result of the authentication is passed on to the Adaptive Risk Manager Online Server.

Table 2–12 Validate User/Passwd Reference APIs

Module	APIs
Sample	handlePassword.jsp

handlePassword.jsp

- retrieves the password from the pad
- decodes the password
- updates the status to "success" if user is valid, else to "invalid," "error," or "bad password"
- runs post-auth rules and one of the following actions:
  - REGISTER_USER_OPTIONAL
  - REGISTER_QUESTIONS
  - REGISTER_USER
  - CHALLENGE
  - BLOCK
  - ALLOW
  - SYSTEM_ERROR
- forwards to registerImageandPhrase during registration or challenge if the user is registered

2.2.2.10 Update Authentication Status (P5)

After validating the user password, the status is updated in the Adaptive Risk Manager.

Table 2–13 Update Authentication Status Reference APIs

Module	APIs
Server	VCryptTracker::updateAuthStatus()
Sample	handlePassword.jsp
BharosaHelper	BharosaHelper::updateStatus()

updateAuthStatus(): For information on updateAuthStatus(), the parameters, and how to use the parameters, refer to Chapter 4, "Native Integration Java."

handlePassword.jsp

- retrieves the password from the pad
- decodes the password
- validates the user

- runs post-auth rules and one of the following actions:
  - REGISTER_USER_OPTIONAL
  - REGISTER_QUESTIONS
  - REGISTER_USER
  - CHALLENGE
  - BLOCK
  - ALLOW
  - SYSTEM_ERROR
- forwards to registerImageandPhrase during registration or challenge if the user is registered

2.2.2.11 Password Status (C1)

Based on the authentication status, the user is either taken to the retry page or to post authentication rule processing.

2.2.2.12 Post Authentication Rules (R3)

The post authentication rules are run after the user password is authenticated. The post authentication runtime contains security rules.

Some common actions returned are

- **Allow**: Allow the authentication.
- **Block**: Block the user.
- **Challenge**: Challenge is returned if the user has registered questions. The option may not be available for Adaptive Risk Manager Only deployments.

Table 2–14 Post Authentication Rules Reference APIs

Module	APIs
Server	VCryptRulesEngine::processRules()
Sample	handlePassword.jsp
BharosaHelper	BharosaHelper::runPostAuthRules()

processRules(): For information on processRules(), the parameters, and how to use the parameters, refer to Chapter 4, "Native Integration Java."

handlePassword.jsp

- run post-auth rules which might return one of the following actions:
  - REGISTER_USER_OPTIONAL
  - REGISTER_QUESTIONS
  - REGISTER_USER
  - CHALLENGE
  - BLOCK
  - ALLOW
  - SYSTEM_ERROR

- forwards to registerImageandPhrase during registration or challenge if the user is registered

2.2.2.13 Check Question Registration for User (C2)

If the user is already verified, it is not necessary to continue to Registration Required Rules.

2.2.2.14 Registration Required Rules (R4)

This runtime runs the rules for determining whether the user is asked to register. The registration requirement is based on the business and security requirements. The business requirement dictates whether the registration is mandatory or optional.

The common actions are

- **Register**: Force the user to register.
- **Registration Optional**: Make the registration optional for the user.
- **Skip Registration**: Skip registration for this session.

Table 2–15 Registration Required Rules Reference APIs

Module	APIs
Server	VCryptRulesEngine::processRules()
Sample	password.jsp
BharosaHelper	BharosaHelper::getAuthentiPad()

processRules(): For information on processRules(), the parameters, and how to use the parameters, refer to Chapter 4, "Native Integration Java."

password.jsp:

- invokes rules to identify user's pad type, else uses the default KeyPad
- creates the pad, names it, and sets all initial background frames, buttons, and so on
- invokes kbimage.jsp as configured in the bharosa_client.properties
- forwards to handlePassword.jsp

2.2.2.15 Challenge (QuestionPad) (S6)

Challenge pad is similar to TextPad, only the question is embedded in the pad.

Figure 2–10 Question Pad

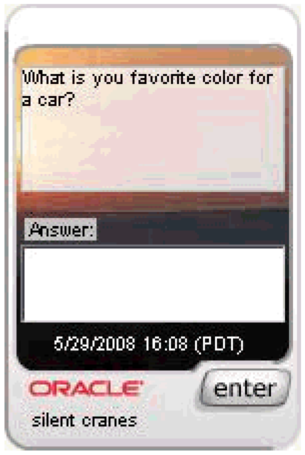


Table 2–16 Challenge Reference APIs

Module	**APIs**
Server	VCryptAuth::getSecretQuestions()
Sample	Challenge.jsp
BharosaHelper	BharosaHelper:: createPersonalizedAuthentiPad () BharosaHelper::createAuthentiPad()
Client	AuthentiPad::getHTML()

getSecretQuestions(): For information on getSecretQuestions(), the parameters, and how to use the parameters, refer to Chapter 4, "Native Integration Java."

Challenge.jsp

- gets the question
- gets the QuestionPad
- displays the Question on the QuestionPad
- submits user answer to handleChallenge.jsp

2.2.2.16 Check Challenge Question Answer (C3)

This step calls the server to validate whether the answer provided by the user is correct.

Table 2–17 Check Challenge Question Answer Reference APIs

Module	**APIs**

Server	VCryptAuth::authenticateQuestion()
	VCryptRulesEngine::processRules()
	VCryptTracker::updateAuthStatus()

Sample	handleChallenge.jsp
BharosaHelper	BharosaHelper:: validateAnswer()

authenticateQuestion(), processRules(), and updateAuthStatus(): For information on the APIs, the parameters, and how to use the parameters, refer to Chapter 4, "Native Integration Java."

handleChallenge.jsp

- validates answer
- if answer is "valid," updates status to "success" and forwards to success
- else if answer is "invalid," runs Challenge Rules to determine what needs to be done

2.2.2.17 Run Challenge Rules (R5)

If the user fails to answer the question correctly, this runtime is invoked to determine whether the user is given another chance to answer the question or to block the user.

Common rule actions are

- **Challenge**: Challenge the user again.
- **Block**: Block the user.

Table 2–18 Run Challenge Rules Reference APIs

Module	APIs
Server	VCryptRulesEngine::processRules()
Sample	handleChallenge.jsp
BharosaHelper	BharosaHelper::validateAnswer()

processRules(): For information on processRules(), the parameters, and how to use the parameters, refer to Chapter 4, "Native Integration Java."

handleChallenge.jsp

- validates answer
- if answer is "valid," updates status to "success" and forwards to success
- else if answer is "invalid," runs Challenge Rules to determine what needs to be done

2.2.2.18 Lock Out Page (S2)

The Lock Out Page is the page the user is generally redirected to if he is blocked from authentication or if he is carrying out a transaction.

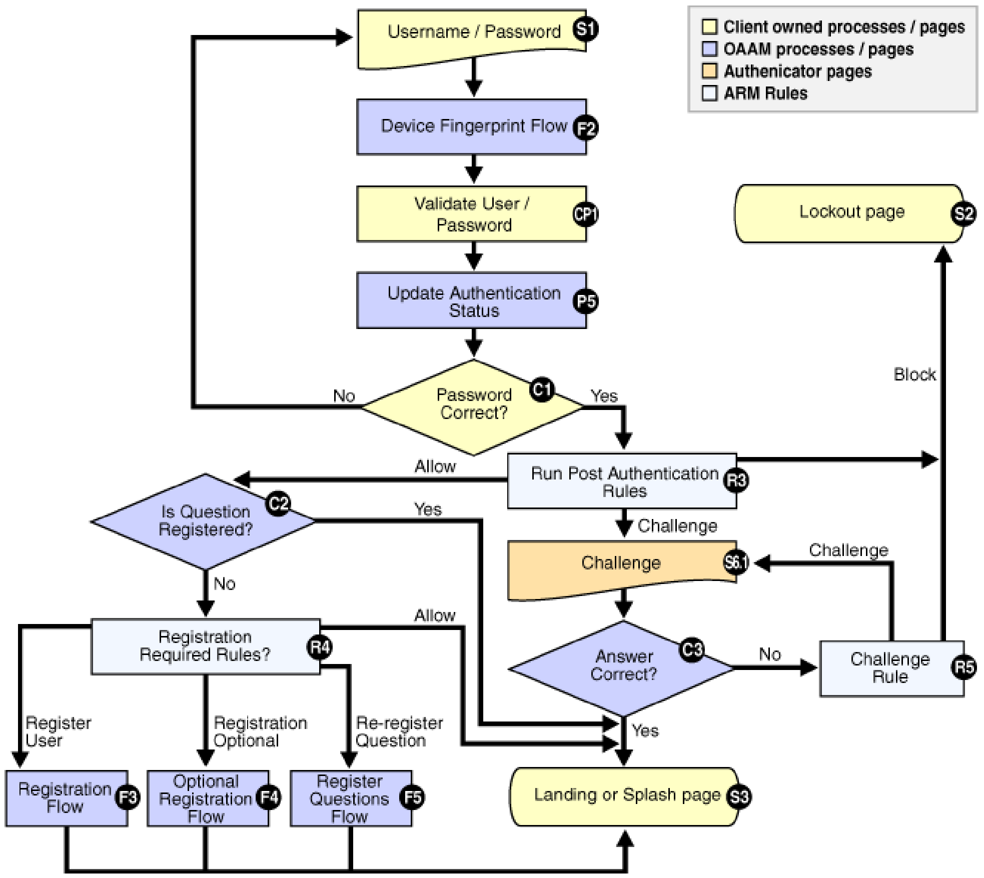
2.2.2.19 Landing or Splash Page (S3)

The Landing or Splash Page is the page the user lands on after a successful login.

2.2.3 Adaptive Risk Manager and KBA Scenario

This scenario is the same as the previous scenario, except it doesn't have a split login flow and there are no personalizations or AuthentiPads.

Figure 2–11 Adaptive Risk Manager and KBA Scenario


Client owned processes / pages
OAAM processes / pages
Authenicator pages
ARM Rules
Username / Password S1
Device Fingerprint Flow F2
Validate User / Password CP1
Update Authentication Status P5
Password Correct? C1
No
Yes
Lockout page S2
Block
Run Post Authentication Rules R3
Allow
Is Question Registered? C2
Yes
Challenge
Challenge S6.1
Challenge
No
Registration Required Rules? R4
Allow
Answer Correct? C3
No
Challenge Rule R5
Register User
Registration Optional
Re-register Question
Yes
Registration Flow F3
Optional Registration Flow F4
Register Questions Flow F5
Landing or Splash page S3

2.3 Troubleshooting

This section describes the steps to take if you experience any problems with Adaptive Risk Manager after the integration.

1. Confirm that bharosa_properties is in the classes directory.
2. Confirm that you have customized bharosa_client.properties.
3. Make sure only one copy of the bharosa_client.properties appears in the classpath. If multiple property files are needed, ensure that they are all are identical.
4. Make sure the directory specified in log4j.xml for logfiles is present and write-accessible.
5. Update log4j.xml to different levels of logging for troubleshooting application issues.
6. Check the log levels in log4j.xml for recommended levels in case of space issues.

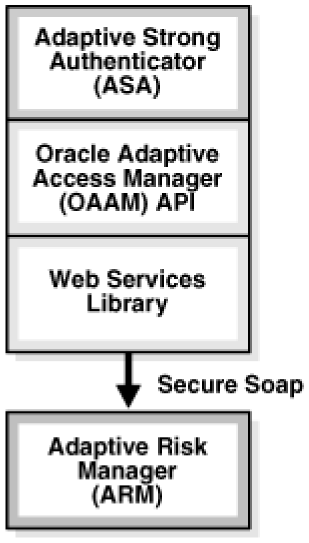
3 Native Integration .net

ASP.NET applications can integrate with Oracle Adaptive Access Manager using the .NET API provided by Oracle Adaptive Access Manager to add multi-factor authentication.

3.1 Architecture

The Oracle Adaptive Access Manager .NET APIs can be called by the client applications natively in the .NET language of their choice. The Oracle Adaptive Access Manager .NET APIs communicate with the Adaptive Risk Manager server using the SOAP protocol. Details on installing and configuring Oracle Adaptive Access Manager .NET APIs are covered later in this chapter.

Figure 3–1 .Net API Communication


Adaptive Strong Authenticator (ASA)
Oracle Adaptive Access Manager (OAAM) API
Web Services Library
Secure Soap
Adaptive Risk Manager (ARM)

3.2 Installing SDK

Oracle Adaptive Access Manager .NET API contents are packaged in a zip file, `Bharosa_SDK_DotNet2.0.zip`. Contents of this zip file should be extracted to the virtual directory of the web application in which Oracle Adaptive Access Manager should be integrated. The application (project files) must be updated to add references to the SDK DLLS.

3.3 Application Configuration

`Web.config`, located in the virtual/root directory of the web application, is the configuration file for web applications. Update `Web.config` to include the BharosaSOAPURL setting in the <appSettings> section. The BharosaSOAPURL setting should be set to the SOAP URL to access the Adaptive Risk Manager SOAP services, as shown below:

```
<appSettings>
  <add key="BharosaSOAPURL" value="http://localhost:9090/fauio/services"/>
</appSettings>
```

3.4 Properties

The Oracle Adaptive Access Manager .NET SDK contains properties files that contain the default values. Each deployment can update certain properties to specify deployment specific values.

The Oracle Adaptive Access Manager .NET API uses properties to read configurable values at runtime, like the location of images for authenticators, and others. Properties are read and cached from a list of files at startup and also whenever one of the loaded properties files is updated. Details of how the properties files are loaded by Oracle Adaptive Access Manager .NET API is given below:

- First the `lookup.properties` file will be loaded, if it exists.
- If the properties.filelist property is defined, all the files listed in this property will be added to the queue.
- Now, all the files in the queue will be loaded in the order they are listed in the `properties.filelist` property.
- When any of the properties files loaded is changed, the properties will be reloaded using the above algorithm

The properties files, including `lookup.properties`, will be searched in the following directories, in the given order. The search will stop when the file is found in one of these directories.

Table 3–1 Directories Searched for Properties Files

Directory description	Example
<ApplicationDirectory>/	c:/Inetpub/wwwroot/MyApp/
<CallingAssemblyDirectory>/	c:/Windows/System32/
<CurrentAssemblyDirectory>/	c:/Inetpub/wwwroot/MyApp/bin/
<CurrentAssemblyDirectory>/../	c:/Inetpub/wwwroot/MyApp/
<CurrentDirectory>/	c:/Windows/System32/
<ApplicationDirectory>/bharosa_properties/	c:/Inetpub/wwwroot/MyApp/bharosa_properties/
<CallingAssemblyDirectory>/bharosa_properties/	c:/Windows/System32/bharosa_properties/
<CurrentAssemblyDirectory>/bharosa_properties/	c:/Inetpub/wwwroot/MyApp/bin/bharosa_properties/
<CurrentAssemblyDirectory>/../bharosa_properties/	c:/Inetpub/wwwroot/MyApp/bharosa_properties/
<CurrentDirectory>/bharosa_properties/	c:/Windows/System32/bharosa_properties/

3.5 User-Defined Enumeration

User-defined enums are a collection of properties that represent a list of items. Each element in the list may contain several different attributes. The definition of a user-defined enum begins with a property ending in the keyword ".enum" and has a value describing the use of the user-defined enum. Each element definition then starts with the same property name as the enum, and adds on an element name and has a value of a unique integer as an ID. The attributes of the element follow the same pattern, beginning with the property name of the element, followed by the attribute name, with the appropriate value for that attribute.

API Usage sample

```
UserDefEnumFactory factory = UserDefEnumFactory.getInstance();

UserDefEnum statusEnum = factory.getEnum("auth.status.enum");

int statusSuccess       = statusEnum.getElementValue("success");
int statusWrongPassword = statusEnum.getElementValue("wrong_password");
```

3.6 Users

Oracle Adaptive Access Manager stores various details of users in its database and uses this information for example, to determine the risk rules to run for a user, to find user specific authenticator attributes, to present the user with a challenge question and validate the answer, and so on.

The client application is responsible for populating Oracle Adaptive Access Manager with the user details at runtime. For example, when a user logs in, the client application should first determine whether the user record exists in Oracle Adaptive Access Manager; if the user record does not exist, appropriate APIs should be called to create the user record and set the user's status. The sample below demonstrates an application calling Oracle Adaptive Access Manager APIs to deal with the user data.

For more usage scenarios and the details, please refer to the sample applications at the end of this chapter.

API Usage sample:

```
string loginId = "testuser";  // loginId of the user logging in

IBharosaProxy proxy = BharosaClientFactory.getProxyInstance();

//
// find the user record in OAAM
//
VCryptAuthUser user = proxy.getUserByLoginId(loginId);

//
// if user does not exist in OAAM - create
//
if(user == null || StringUtil.IsEmpty(user.LoginId))
{
    string customerId  = loginId;
    string userGroupId = "PremiumCustomer";
    string password    = "_"; // this value is not used for now

    user = new VCryptAuthUser(loginId, customerId,
                              userGroupId, password);
    user = proxy.createUser(user);
    //
    // set the status of the new user to Invalid; once the user is
    // authenticated, set the status to PendingActivation; after the
    // user succssfully completes registration, set the status to Valid
    //
    proxy.setUserStatus(user.CustomerId, (int)UserStatus.Invalid);
}

//
// save the user record in the session for later reference
//
AppSessionData sessionData = AppSessionData.GetInstance(Session);

sessionData.CurrentUser = user;
```

The Proxy is the interface to access Adaptive Risk Manager SOAP server services.

3.7 Adaptive Risk Manager

Adaptive Risk Manager is a component of Oracle Adaptive Access Manager. Adaptive Risk Manager provides APIs to capture user login information, user login status, and various attributes of the user session to determine device and location information. Adaptive Risk Manager also provides APIs to collect transaction details.

API Usage sample:

```
//
// record a user login attempt in OAAM
//
string   requestId      = sessionData.RequestId;
string   remoteIPAddr   = Request.UserHostAddress;
string   remoteHost     = Request.UserHostName;
bool     isFlashRequest = Request.Params["client"].Equals("vfc");
string   secureCookie   = (Request.Cookies["vsc"] != null)
                             ? Request.Cookies["vsc"].Value : null;
string   digitalCookie  = isFlashRequest
                             ? Request.Params["v"] : null;
object[] browserFpInfo = HttpUtil.GetBrowserFingerPrint();
object[] flashFpInfo   = HttpUtil.GetFlashFingerPrint();

int browserFingerPrintType =
               browserFpInfo == null ? 0 : (int) browserFpInfo [0];
string browserFingerPrint =
               browserFpInfo == null ? "" : (string) browserFpInfo [1];
int flashFingerPrintType =
               flashFpInfo == null ? 0 : (int) flashFpInfo[0];
string flashFingerPrint =
               flashFpInfo == null ? "" : (string) flashFpInfo[1];

//
// if username/password has been validated by now, set the status
// below to appropriate value: success/wrong_Password/invalid_user.
// if username/password has not yet been validated, set the status to
// pending; after validation is done call updateLog() again with
// the updated status
//
int status = statusEnum.getElementValue("pending");

//
```

```
// Call updateLog() to record the user login attempt
//
CookieSet cs = proxy.updateLog(requestId, remoteIPAddr, remoteHost,
                     secureCookie, digitalCookie, user.CustomerGroupId,
                     user.CustomerId, user.LoginId, false,
                     status, ClientTypeEnum.Normal,
                     "1.0", browserFingerPrintType, browserFingerPrint,
                     flashFingerPrintType, flashFingerPrint);

//
// Update secure cookie in the browser with the new value from OAAM
//
if (cs != null)
{
    HttpUtil.UpdateSecureCookie(Response, cs);
}
```

3.8 Rules Engine

The Rules Engine component of Adaptive Risk Manager provides APIs for policy execution and enforcement. The Rules Engine can be viewed as an enforcement component of the Adaptive Risk Manager. Based on the calling context, various policies and models are evaluated and the results are provided. The policies and models are configured by the administrators.

API Usage sample:

The following sample code demonstrates the use of APIs to run rules in post-auth runtime and process the rule result.

```
AppSessionData  sessionData = AppSessionData.GetInstance(Session);
IBharosaProxy      proxy    = BharosaClientFactory.getProxyInstance();
UserDefEnumFactory factory  = UserDefEnumFactory.getInstance();
UserDefEnum profileTypeEnum = factory.getEnum("profile.type.enum");

string             requestId    = sessionData.RequestId;
BharosaStringList  profileTypes = new BharosaStringList();
BharosaStringTable contextList  = new BharosaStringTable();

int postAuthType = profileTypeEnum.getElementValue("postauth");

profileTypes.Add(postAuthType.ToString());

//
// Run postauth rules
//
VCryptRulesResult res = proxy.processRules(requestId,
                                           profileTypes, contextList);

//
// process the rule result
//
if (StringUtil.EqualsIgnoreCase(res.Result, "Allow"))
{
// Allow the user login
}
else if (StringUtil.EqualsIgnoreCase(res.Result, "Block"))
{
// Block the user login
}
```

```
else if (res.Result.StartsWith("Challenge"))
{
// Take the user through challenge question flow
}
else if (res.Result.StartsWith("RegisterUser"))
{
// Take the user through registration flow
}
```

3.8.1 Device ID Evaluation

Along with rule results, the Rules Engine component can return a Device Id. The Device Id that is returned by the Rules Engine API is an internal identifier within Adaptive Risk Manager. It is the same Id shown in Adaptive Risk Manager for the user session.

A sample algorithm to get the Device Id is shown below:

```
VCryptRulesResult rulesResult = new
VCryptRulesEngineImpl().processRules(<params..>);

If (!rulesResult.getVCryptResponse().isSuccess()) {

     Logger.error("Error running rules " +
rulesResult.getVCryptResponse().getErrorMessage());

}

Long deviceId = VCryptRulesResult#getDeviceId();
```

Troubleshooting:

- Make sure the Adaptive Risk Manager version is 10.1.4.5 or above
- The following property must be set to true for the Device Id to be captured.

  ```
  bharosa.tracker.send.deviceId=true
  ```

3.8.2 Create Transactions in Bulk

The Rules Engine component can create transactions in bulk.

A sample algorithm to create transactions in bulk is shown below:

```
/**
     * API To create a Transactions in bulk
     * Return response object for each create request
     *
     * @param transactionCreateRequestData requestId / Session Id, required
     * @return VCryptResponse Array of response objects. Each request will have
corresponding response. Check isSuccess on each response object
     * @since 10.1.4.5.2
     */
    public VCryptResponse[] createTransactions(TransactionCreateRequestData[]
transactionCreateRequestData);
```

3.8.3 Update Transactions in Bulk

The Rules Engine component can update transactions in bulk.

A sample algorithm to update transactions in bulk is shown below:

```
/**
     * API to update Transactions in bulk
     * If there are errors in any update, will proceed with next transaction and
return response for each request
     *
     * @param transactionUpdateRequestData array of update Request object
     * @return VCryptResponse Array of response objects. Each request will have
corresponding response. Check isSuccess on each response object
     * @since 10.1.4.5.2
     */
    public VCryptResponse[] updateTransactions(TransactionUpdateRequestData[]
transactionUpdateRequestData);
```

3.9 Challenge Questions

Oracle Adaptive Access Manager can be configured to challenge the user with pre-registered questions during high risk/suspicious scenarios. In a typical deployment, users will be asked to pick three questions from a given set of questions and register answers for them. Oracle Adaptive Access Manager expects the user to enter a correct answer for one of the registered questions at the time of the challenge.

The following sample code shows the API to be used during the registration and challenge process. Please refer to the sample application for more details on using various Oracle Adaptive Access Manager APIs to support Knowledge Based Authentication (KBA) scenarios.

API Usage sample:

```
//
//  Retrieve a question-pickset, containing groups of questions from
//  which the user would pick one question from each group for
//  registration
//
VCryptQuestionList[] groups = proxy.getSignOnQuestions(
                                                    user.CustomerId);

//
// Please refer to the sample application for details on displaying the
// questions in the UI and processing the user input; let us assume
// that user selected questions and answers are now in questions object
//

//
// Register the questions and answers with OAAM
//
VCryptResponse response = proxy.addQuestions(
                                       user.CustomerId, questions);


//
// Retrive the question to show the user during challenge
//
VCryptQuestion secretQuestion = proxy.getSecretQuestion(
                                                     user.CustomerId);
//
// Create QuestionPad authenticator to display the question text.
// Please refer to the sample application for details; let us here
// assume that the user entered answer is stroed in string answer
```

```
//

//
// Validate the user entered answer
//
VCryptAuthResult res = proxy.authenticateQuestion(customerId, answer);

bool isValid = (res != null && res.ResultCode == 0);
```

3.10 Reset Challenge Failure Counters

The API to reset failure counters for a given user or user and question combination is shown below. It is applicable when the KBA module is enabled. Adaptive Risk Manager stores the failure counters for the user questions. Failure counters are used to enforce lock. If a question id is sent, failure counters associated with the question are reset. If no question id is sent, failure counters for all questions are reset.

```
/**

     * Reset Challenge failure counters for the given customerid

     * @param requestId to track the request

     * @param customerId external customer id, required and used to identify
customer uniquely, it is not login Id

     * @param questionId optional, if sent, failure counters for the given
question id are reset

     * @return The response from the server. check isSuccess() for success

     */

    public VCryptResponse resetChallengeFailureCounters(final String requestId,
final String customerId, final Long questionId);
```

3.11 Authenticators

This section gives details on creating and using the authenticators in ASP.NET applications.

3.11.1 Creating an Authenticator

Use `BharosaClient.getAuthentiPad()` to generate the .NET authenticator object, as shown below:

```
IBharosaClient client = BharosaClientFactory.getClientInstance();

String padName = "passwordPad";

if (! IsPostBack)
{
    AuthentiPadType padType     = AuthentiPadType.TYPE_ALPHANUMERICPAD;
    String          bgFile      = proxy.getImage(user.CustomerId);
    String          captionText = proxy.getCaption(user.CustomerId);
    String          frameFile   = BharosaConfig.get(
"bharosa.authentipad.alphanumeric.frame.file",
"alphanumpad_bg/kp_v2_frame_nologo.png");
```

```
    AuthentiPad authPad = client.getAuthentiPad(padType, padName,
                                                frameFile, bgFile,
                                                captionText, false,
                                                true, true);

    //
    // save the authenticator object in sessData; this will be needed
    // - in GetImage.aspx.cs to generate the authenticator image
    // - while decoding the user input
    //
    sessionData[padName] = authPad;
}
```

3.11.2 Embedding an Authenticator in a Web Page

The `.ASPX` file and the code behind file (`ASPX.CS` for C#) need to be updated, as shown below, to display the authenticator created in the previous section:

1. Include JavaScript `bharosa_web/js/bharosa_pad.js` in the ASPX file.
2. Create a label in the ASPX file where the authenticator is to be displayed, as shown below:

   ```
   <asp:Label ID="authenticator" runat="server"></asp:Label>
   ```

3. Generate the HTML in the code behind file (ASPX.CS) from the authenticator object and assign to the label, as shown below:

   ```
   this.authenticator.Text = client.getAuthentiPadHTML(authPad,false, false);
   ```

3.11.3 Decoding User Input

The user-entered input in the authenticator is posted to the application in the HTTP parameter named padName + "DataField". This input should be decoded using the authenticator as shown below:

```
if (IsPostBack)
{
    AuthentiPad authPad       = sessionData[padName];
    String      encodedPasswd = Request.Params[padName + "DataField"];
    String      passwd        = authPad.decodeInput(encodedPasswd);

    // validate the password..
}
```

3.12 Specifying Credentials to Access Adaptive Risk Manager SOAP Services

The credentials to access Adaptive Risk Manager Soap services can be specified in one of the following ways.

- By adding the following settings to application configuration file (web.config for web applications)

  ```
  <appSettings>
        <add key="BharosaSOAPUser"     value="soapUser"/>
        <add key="BharosaSOAPPassword" value="soapUserPassword"/>
        <add key="BharosaSOAPDomain"   value="soapUserDomain"/>
  </appSettings>
  ```

- By adding the following properties to one of the properties files used by the application:

```
BharosaSOAPUser=soapUser
BharosaSOAPPassword=soapUserPassword
BharosaSOAPDomain=soapUserDomain
```

When the second option above is used to specify SOAP credentials, the encrypted value can be specified. For details on encrypting the property values, please refer to the section below.

3.13 Encrypting Property Values

Property value in the properties files used by Oracle Adaptive Access Manager .NET DLLs can be encrypted using command-line utility BharosaUtils.exe, which is included in the Oracle Adaptive Access Manager .NET API package. A user-selected encryption key is required to encrypt and decrypt the properties. The key used to encrypt the properties must be available to Oracle Adaptive Access Manager .NET API through the property, "bharosa.cipher.client.key," that is, this property must be set in one of the properties files used by the application. BharosaUtil.exe prompts the user to interactively enter the encryption key and the value to be encrypted. The encrypted value is output to the console. The encrypted value can be used in the properties file, instead of the plain text value.

```
C:\> BharosaUtil.exe -enc
Enter key (min 14 characters len):
Enter key again:
Enter text to be encrypted:
Enter text to be encrypted again:
vCCKC19d14a39hQSKSirXSiWfgbaVG5SKIg==
```

3.14 Troubleshooting

The Oracle Adaptive Access Manager .NET API can be configured to print trace messages of various levels using diagnostics switches in `Web.config`. The tracing can be configured to print to a file by configuring appropriate listeners. The following XML demonstrates the trace switches supported by the Oracle Adaptive Access Manager .NET API and a sample listener configuration that writes to a file.

```
  <system.diagnostics>
    <switches>
      <add name="debug" value="0"/>
      <add name="info" value="0"/>
      <add name="soap" value="0"/>
      <add name="perf" value="0"/>
      <add name="warning" value="1"/>
      <add name="error" value="1"/>
      <add name="traceTimestamp" value="1"/>
      <add name="traceThreadId" value="1"/>
    </switches>
    <trace autoflush="true" indentsize="2">
      <listeners>
        <add name="BharosaTraceListener"
             type="System.Diagnostics.TextWriterTraceListener, System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089"
             initializeData="BharosaTrace.log"/>
      </listeners>
    </trace>
  </system.diagnostics>
```

3.15 ASP.NET Applications

The following four ASP.NET applications are included in the sample package to demonstrate the integration of various Oracle Adaptive Access Manager features in ASP.NET based applications.

Table 3–2 ASP.NET applications samples

Application name	Description
SampleWebApp	This is a plain ASP.NET application with no Oracle Adaptive Access Manager integration. This application is provided so that the reader can easily see incremental changes required to integrate various Oracle Adaptive Access Manager features - like Adaptive Risk Manager, authenticator, KBA.
SampleWebAppWithTracker	This application demonstrates the integration of the Oracle Adaptive Access Manager-Adaptive Risk Manager functionality with SampleWebApp, which is listed above.
SampleWebAppWithAuthTracker	This application demonstrates the integration of the Oracle Adaptive Access Manager-Adaptive Risk Manager and authenticator functionalities with SampleWebApp, which is listed above.
SampleWebAppWithKBATracker	This application demonstrates the integration of the Oracle Adaptive Access Manager-Adaptive Risk Manager and KBA functionalities with SampleWebApp, which is listed above.

The source code for each application is placed in a directory of its own. Visual Studio Solution files for each of these applications can be found in the root directory. The solutions file, "SampleWebApps," can be used to load and view all applications together. This section highlights the content of each application.

A few tips are listed below to set up the environment to successfully run the samples.

Ensure that the SOAP URL to access the Adaptive Risk Manager server is set correctly in the `web.config` file of the application, as per your deployment configuration. A sample is shown below:

```
<appSettings>
  <add key="BharosaSOAPURL"
       value="http://localhost:9090/fauio/services"/>
</appSettings>
```

Make sure that the Adaptive Risk Manager server used with these applications is loaded with the models contained in `models.zip`.

Make sure that both the Adaptive Risk Manager server and the web applications are configured to look for authenticator images in the same directory paths. The following properties should be set:

```
bharosa.authentipad.full.background.dirlist
bharosa.authentipad.alphanumeric.background.dirlist
bharosa.authentipad.numeric.background.dirlist
bharosa.authentipad.textpad.background.dirlist
bharosa.authentipad.textpadreset.background.dirlist
bharosa.authentipad.questionpad.background.dirlist
```

3.16 SampleWebApp

This application contains the following pages, which demonstrates a web application before an Oracle Adaptive Access Manager integration.

- LoginPage.aspx
  - Collects the username and password using a simple HTML form
  - Validates the login and password information
  - Depending upon the validation result, the user will be redirected to either `Success.aspx` or to `LoginPage.aspx` with the appropriate error message
- Success.aspx
  - Displays the "Successfully logged in" message with a link for logout
- LogoutPage.aspx
  - Logs out the user session and redirects to login page

3.17 SampleWebAppWithTracker

This application contains the following pages, which demonstrate the integration of Oracle Adaptive Access Manager-Adaptive Risk Manager functionality with the sample application listed above.

This application requires the integration of the Oracle Adaptive Access Manager .NET APIs found in the SDK package, `Bharosa_SDK_DotNet2.0.zip`. The content of the archive needs to be extracted to the root directory of the web application.

- LoginPage.aspx
  - Collects the username and password using a simple HTML form
  - Saves the login and password in the session
  - Redirects the user to `LoginJumpPage.aspx` to collect the flash finger print of the user device
- LoginJumpPage.aspx
  - Loads the user from Adaptive Risk Manager by calling `AppUtil.InitUser()` (`AppUtil` is included in the SDK package). If the user is not found, a new user record will be created
  - Returns HTML to load flash object `bharosa_web/flash/bharosa.swf` in the browser. The flash object calls `CookieManager.aspx` (included in the SDK package) with flash finger print details. `CookieManager.aspx` records the finger print in Adaptive Risk Manager and in return sets a flash cookie on the user's device.
  - After a brief wait (to allow time to get the flash cookie from Adaptive Risk Manager), redirects the browser to `LoginHandlerPage.aspx`
- LoginHandlerPage.aspx
  - Records the user login attempt with Adaptive Risk Manager by calling `AppUtil.InitTracker()`
  - Validates the login and password information
  - Updates Adaptive Risk Manager with the password validation status (success/wrong user/wrong password/disabled user, etc) by calling `AppUtil.UpdateAuthStatus()`
  - If password validation succeeds, runs post-authentication rules by calling `AppUtil.RunPostAuthRules()`

  - If the post-authentication rules return `block`, blocks the user login after updating Adaptive Risk Manager with this information
  - Depending upon the validation result and/or the rules result, redirects the user to either `Success.aspx` or to `LoginPage.aspx` with appropriate error message
- Success Page
  - Displays "Successfully logged in" message with a link for logout
- Logout Page
  - Logs out the user session and redirects to login page

3.18 SampleWebAppAuthTracker

This application contains the following pages that demonstrate integration of Oracle Adaptive Access Manager authenticator and Adaptive Risk Manager functionalities to the sample application listed above. This application collects the password using authenticators offered by Oracle Adaptive Access Manager.

This application requires the integration of the Oracle Adaptive Access Manager .NET APIs found in the SDK package, `Bharosa_SDK_DotNet2.0.zip`. The content of the archive needs to be extracted to the root directory of the web application.

- LoginPage.aspx
  - Collects the username using simple HTML form
  - Saves the login in the session
  - Redirects the user to `LoginJumpPage.aspx` to collect the flash finger print of the user device
- LoginJumpPage.aspx
  - Loads the user from Adaptive Risk Manager by calling `AppUtil.InitUser()` (`AppUtil` is included in the SDK package). If the user is not found, a new user record will be created
  - Returns HTML to load flash object `bharosa_web/flash/bharosa.swf` in the browser. The flash object calls `CookieManager.aspx` (included in the SDK package) with flash finger print details. `CookieManager.aspx` records the finger print in Adaptive Risk Manager and in return sets a flash cookie on the user's device
  - After a brief wait (to allow time to get the flash cookie from Adaptive Risk Manager), redirects the browser to `LoginHandlerPage.aspx`
- LoginHandlerPage.aspx
  - Records the user login attempt with Adaptive Risk Manager by calling `AppUtil.InitTracker()`
  - Redirects the user to `PasswordPage.aspx` to collect the password using Oracle Adaptive Access Manager authenticator.
- PasswordPage.aspx

  On Load:

  1. Sets the session authentication status to "Pending" in Adaptive Risk Manager.

2. Runs pre-authentication rules by calling the `AppUtil.RunPreAuthRules()`.
3. If the pre-authentication rules return `block`, blocks the user login after updating Adaptive Risk Manager with this information
4. If the pre-authentication rules return `allow`, runs another set of rules to determine the authenticator to use for this user, by calling `AppUtil.RunAuthentiPadRules()`
5. Creates appropriate authenticator by calling `AppUtil.CreateAuthentiPad()` and renders the authenticator into HTML by using `AppUtil.getAuthentiPadHTML()`. The authenticator HTML would fetch the authenticator image by calling `GetImage.aspx` (included in the SDK package)
6. Stores the authenticator object in the session for later use during image generation and password decode

On PostBack

1. Decodes the password using the authenticator object stored in the session
2. Validates the login and password information
3. Updates Adaptive Risk Manager with the password validation status (success/wrong user/wrong password/disabled user, etc) by calling `AppUtil.UpdateAuthStatus()`
4. If password validation succeeds, runs post-authentication rules by calling `AppUtil.RunPostAuthRules()`
5. If the post-authentication rules return `block`, blocks the user login after updating Adaptive Risk Manager with this information
6. Depending upon the validation result and/or the rules result, redirects the user to either `Success.aspx` or to `LoginPage.aspx` with appropriate error message

- Success Page
  - Displays "Successfully logged in" message with a link for logout
- Logout Page
  - Logs out the user session and redirects to login page

3.19 SampleKBATracker

This application contains the following pages that demonstrate integration of Oracle Adaptive Access Manager authenticator, Adaptive Risk Manager and KBA (Knowledge Based Authentication) functionalities to the sample application listed above. This application shows authentication mechanisms using password and KBA authenticators offered by Oracle Adaptive Access Manager.

This application requires the integration of the Oracle Adaptive Access Manager .NET APIs found in the SDK package `Bharosa_SDK_DotNet2.0.zip`. The content of the archive needs to be extracted to the root directory of the web application.

- LoginPage.aspx
  - Collects the username using simple HTML form
  - Saves the login in the session

  - Redirects the user to `LoginJumpPage.aspx` to collect the flash finger print of the user device

- LoginJumpPage.aspx
  - Loads the user from Adaptive Risk Manager by calling `AppUtil.InitUser()` (`AppUtil` is included in the SDK package). If the user is not found, a new user record will be created
  - Returns HTML to load flash object `bharosa_web/flash/bharosa.swf` in the browser. The flash object calls `CookieManager.aspx` (included in the SDK package) with flash finger print details. `CookieManager.aspx` records the finger print in Adaptive Risk Manager and in return sets a flash cookie on the user's device
  - After a brief wait (to allow time to get the flash cookie from Adaptive Risk Manager), redirects the browser to `LoginHandlerPage.aspx`
- LoginHandlerPage.aspx
  - Records the user login attempt with Adaptive Risk Manager by calling `AppUtil.InitTracker()`
  - Redirects the user to `PasswordPage.aspx` to collect the password using Oracle Adaptive Access Manager authenticator
- PasswordPage.aspx

  On Load:

  1. Sets the session authentication status to "Pending" in Adaptive Risk Manager
  2. Runs pre-authentication rules by calling the `AppUtil.RunPreAuthRules()`
  3. If the pre-authentication rules return `block`, blocks the user login after updating Adaptive Risk Manager with this information
  4. If the pre-authentication rules return `allow`, runs another set of rules to determine the authenticator to use for this user, by calling `AppUtil.RunAuthentiPadRules()`
  5. Creates appropriate authenticator by calling `AppUtil.CreateAuthentiPad()` and renders the authenticator into HTML by using the `AppUtil.getAuthentiPadHTML()`. The authenticator HTML would fetch the authenticator image by calling `GetImage.aspx` (included in the SDK package)
  6. Stores the authenticator object in the session for later use during image generation and password decode

  On PostBack:

  1. Decodes the password using the authenticator object stored in the session
  2. Validates the login and password information
  3. Updates Adaptive Risk Manager with the password validation status (success/wrong user/wrong password/disabled user, etc) by calling `AppUtil.UpdateAuthStatus()`
  4. If the password validation fails, the user will be redirected to `LoginPage.aspx` with appropriate error message
  5. If password validation succeeds, runs post-authentication rules by calling `AppUtil.RunPostAuthRules()`

6. The user will be taken through different flows, as shown below, depending upon the action from post-authenticator rules result:

Post-Auth Action	**Target URL**
Block	LoginPage.aspx
Allow	Success.aspx
ChallengeUser	ChallengeUser.aspx
RegisterQuestions	RegisterQuestionsPage.aspx
RegisterUser	PersonalizationPage.aspx
RegisterUserOptional	PersonalizationPage.aspx

- PersonalizationPage.aspx
  - Introduces the user to device personalization explaining the steps that would follow to create a new Security Profile for the user
  - If the post authentication rule returns `RegistrationOptional`, the user is allowed to skip the registration process by clicking the "Skip" button to proceed to the `Success.aspx` page directly
  - If registration is not optional, the user must register by clicking "Continue" to proceed to the `RegisterImagePhrase.aspx` page
- RegisterImagePhrase.aspx
  - Allows the user to customize the randomly generated background image, caption and the type of security device used during authentication
  - A new background image and caption is assigned by calling `AppUtil.AssignNewImageAndCaption()`
  - The user selected security device is assigned by calling `AppUtil.SetAuthMode()`
- RegisterQuestionsPage.aspx
  - Displays sets of questions which the user can choose and register the correct answer for each.
  - The sets of questions are fetched by calling `proxy.getSignOnQuestions()`
- ChallengeUser.aspx
  - Challenges the user by displaying a question-pad with one of the questions already registered by the user
  - The answer is validated by calling `proxy.authenticateQuestion()` and the result is updated in Adaptive Risk Manager by calling `AppUtil.UpdateAuthStatus()`
  - If the answer is wrong, a call to `AppUtil.RunChallengeUserRules()` is made and based on the result of which, the user will either be allowed to reenter the answer or be redirected to the block page after updating the block status in Adaptive Risk Manager
  - The number of attempts that a user gets to answer a question correctly is set by the rule administrator for Adaptive Risk Manager
  - On successfully answering the question correctly, the user is forwarded to the `Success.aspx` page

- Success Page
  - Displays "Successfully logged in" message with a link for logout
- Logout Page
  - Logs out the user session and redirects to login page

4 Native Integration Java

This document provides information on the available APIs for integrations using Java. Refer to Chapter 2, "API Integration" for guidelines for integration options and flows.

4.1 Installation

The installation package will provide the required jars and property files. Sun Java 1.4 and above is supported for the integrations.

Property files of interest are listed below.

Table 4–1 Property Files of Interest

File name	Description
bharosa_app.properties	This file is designated for installation-specific properties. Refer to the *Oracle Adaptive Access Manager Installation and Configuration Guide* for information on configuring Adaptive Risk Manager and Chapter 6, "Configuring Adaptive Strong Authenticator," for configuring Adaptive Strong Authenticator.
bharosa_client.properties	This file is designated for client side properties. The sample file, bharosa_client.properites.sample, is provided with the installation package. Copy bharosa_client.properites.sample to bharosa_client.properites. Update the properties as per your environment. **1.** Set the "vcrypt.tracker.soap.useSOAPServer" property. If SOAP is used, set the value to true. If static linking is used, set the value to false. **2.** Set the "vcrypt.tracker.soap.url" property to the Adaptive Risk Manager URL if "vcrypt.tracker.soap.useSOAPServer" is set to true. For example, vcrypt.tracker.soap.url=http://localhost:9090/oarm/services **3.** Set "bharosa.image.dirlist" if Adaptive Strong Authenticator is used. Refer to the *Oracle Adaptive Access Manager Installation and Configuration Guide* for more information on setting "bharosa.image.dirlist."
bharosa_common_keypad.properties	This file contains properties to customize various aspects of authenticators. This file is not to be modified by the clients.

4.2 Adaptive Risk Manager

Adaptive Risk Manager is a component of Oracle Adaptive Access Manager.

Adaptive Risk Manager provides APIs to

- collect/track information from the client application
- capture user login information, user login status, and various attributes of the user session to determine device and location information
- collect transaction details

The sample application and the BharosaHelper class provide some of the usage scenarios.

For examples of usage scenarios and flows, refer to Chapter 2, "API Integration."

4.2.1 handleTrackerRequest

API handleTrackerRequest: API to capture fingerprint details and identify the device. Variations of the same API is provided to capture details with the time of the request. The time of request can be in the past.

- public CookieSet handleTrackerRequest(String requestId, String remoteIPAddr, String remoteHost, String secureCookie, int secureClientType, String secureClientVersion, String digitalCookie, int digitalClientType, String digitalClientVersion, int fingerPrintType, String fingerPrint, int fingerPrintType2, String fingerPrint2);
- public CookieSet handleTrackerRequest(String requestId, Date requestTime, String remoteIPAddr, String remoteHost, String secureCookie, int secureClientType, String secureClientVersion, String digitalSigCookie, int digitalClientType, String digitalClientVersion, int fingerPrintType, String fingerPrint, int fingerPrintType2, String fingerPrint2);

Table 4–2 API handleTrackerRequest Parameters

Parameter	Description
requestId	The ID for the login session. The same ID is necessary for all the calls to the Oracle Adaptive Access Manager API for the login session.
remoteIPAddr	The IP from where the request came in. This is extracted from the HTTP request.
remoteHost	The host name from where the request came in. This is optional
secureCookie	The secure cookie. This is passed only if it is received from the browser
secureClientType	The secure cookie client type. This is an enum value defined to identify the client used for authentication.
secureClientVersion	The version of the secure cookie client. This is an optional parameter to specify the version of the client used
digitalCookie	The digital signature cookie. This could be the flash cookie. This parameter is sent only if it is sent by the browser
digitalClientType	The digital cookie client type. The type of flash client used. If not available, please use the value 0
digitalClientVersion	The version of the digital cookie client. The version of the flash client.
fingerPrintType	The type of finger printing. The value for defined in the properties file.
fingerPrint	The finger print. If it is browser characteristics, then the header is parsed into this string. This is the name value representation of the browser header information.

Table 4–2 (Cont.) API handleTrackerRequest Parameters

Parameter	Description
fingerPrintType2	This is used in case the same request has multiple finger prints. It is like a flash request. The value is defined in the properties file
fingerPrint2	The second finger print value. If it is from the flash, it is passed as it is. This is an optional parameter.
requestTime	The time when this request was made. Used by simulator

4.2.2 createTransaction

API createTransaction: API to create new transaction

- public VCryptResponse createTransaction(TransactionCreateRequestData transactionCreateRequestData);

Table 4–3 API createTransaction Parameters

Parameter	Description
TransactionCreateRequestData	Data object to create a new transaction. A description of the structure is as follows: ■ requestId is required to identify the user session ■ requestTime request time can be null. The server uses the current time if a null requestTime is sent ■ transactionKey is required data. It is the key to the transaction definition. This is the key used when creating a transaction definition ■ externalTransactionId is optional if there is external transaction Id. It can also be used to update the transaction later ■ status of the transaction can be null ■ The keys should map to the source keys defined in the transaction definition ■ BharosaException is thrown if fails validations
VCryptResponse	Response object. Make sure to check isSuccess() before obtaining the transaction Id. getTransactionResponse() method provides the transaction id.

4.2.3 updateTransaction

API updateTransaction: API to update previously created transaction

- public VCryptResponse updateTransaction(TransactionUpdateRequestData transactionUpdateRequestData);

Table 4–4 API updateTransaction Parameters

Parameter	Description
TransactionUpdateRequestData	Data object to update transaction. Either transaction Id is returned during createTransaction API call or external transaction Id is sent during createTrasnaction API call to get the handle to the existing transaction. The description of the structure is as follows: ■ requestId is required to identify the user session ■ requestTime request time can be null. The server uses the current time if null requestTime is sent ■ transactionId id of the previously created transaction (createTransaction API call) ■ status of the transaction ■ The keys should map to the source keys defined in the transaction definition and can be null ■ analyzePatterns if true triggers pattern analysis ■ externalTransactionId external transaction id should be the same as the id provided during the transaction creation ■ BharosaException is thrown if fails validations
VCryptResponse	Response object. Make sure to check isSuccess() before obtaining the transaction Id. getTransactionResponse() method provides transaction id.

4.2.4 handleTransactionLog

API handleTransactionLog: API to capture transaction details.

Note: Deprecated as of 10.1.4.5.1, use createTransaction API.

- public VCryptResponse handleTransactionLog(String requestId, Map[]);
- public VCryptResponse handleTransactionLog(String requestId, Date requestTime, Map[] contextMap);
- public VCryptResponse handleTransactionLog(String requestId, Date requestTime, Integer status, Map[] contextMap);

Table 4–5 API handleTransactionLog Parameters

Parameter	Description
requestId	The ID for the login session. The same ID is necessary for all the calls to the Oracle Adaptive Access Manager API for the login session.
requestTime	The time when this request was made. Used by simulator
contextMap	The array of contextMap. Multiple transactions can be created with a single API call. It is expected to have "transactionType" key in each context map for the creation of the appropriate transaction.
status	The transaction status for this transaction, client specific transaction status.

4.2.5 updateTransactionStatus

API updateTransactionStatus: API to update the given transaction status and, if appropriate, trigger the pattern data analysis.

Note: Deprecated as of 10.1.4.5.1, use updateTransaction API.

- public VCryptResponse updateTransactionStatus(String requestId, long transactionId, int status);
- public VCryptResponse updateTransactionStatus(String requestId, Date requestTime, long transactionId, int status);
- public VCryptResponse updateTransactionStatus(String requestId, long transactionId, int status, Map[] contextMap);
- public VCryptResponse updateTransactionStatus(String requestId, Date requestTime, long transactionId, int status, Map[] contextMap);
- public VCryptResponse updateTransactionStatus(java.lang.String requestId, long transactionId, int status, boolean analyzePatterns)
- public VCryptResponse updateTransactionStatus(java.lang.String requestId, java.util.Date requestTime, long transactionId, int status, java.util.Map[] contextMap, boolean analyzePatterns)

Table 4–6 API updateTransactionStatus Parameters

Parameter	Description
requestId	The ID for the login session. The same ID is necessary for all the calls to the Oracle Adaptive Access Manager API for the login session.
requestTime	The time when this request was made. Used by simulator
contextMap	The array of contextMap. Multiple transactions can be created with a single API call. It is expected to have "transactionType" key in each context map for the creation of the appropriate transaction.
Status	The transaction status for this transaction, client specific transaction status.
transactionId	The ID of the transaction to be updated. If null, it uses the last transaction in the given session.
analyzePatterns	Boolean to indicate if the pattern analysis should be done. When passed in as true the pattern analysis is done for this transaction.

4.2.6 updateLog

API updateLog: API to update the user node log and, if required, to create the CookieSet also.

- public CookieSet updateLog(String requestId, String remoteIPAddr, String remoteHost, String secureCookie, String digitalCookie, String groupId, String userId, String loginId, boolean isSecure, int result, int clientType, String clientVersion, int fingerPrintType, String fingerPrint, int digFingerPrintType, String digFingerPrint);
- public CookieSet updateLog(String requestId, Date requestTime, String remoteIPAddr, String remoteHost, String secureCookie, String digitalCookie, String groupId, String userId, String loginId, boolean isSecure, int result, int clientType, String clientVersion, int fingerPrintType, String fingerPrint, int fingerPrintType2, String fingerPrint2);

Table 4–7 API updateLog Parameters

Parameter	Description
requestId	The ID for the login session. The same ID is necessary for all the calls to the Oracle Adaptive Access Manager API for the login session.
remoteIPAddr	The IP from where the request came in. This is extracted from the HTTP request.
remoteHost	The host name from where the request came in. This is optional
secureCookie	The secure cookie. This is passed only if it is received from the browser
secureClientType	The secure cookie client type. This is an enum value defined to identify the client used for authentication.
secureClientVersion	The version of the secure cookie client. This is an optional parameter to specify the version of the client used
digitalCookie	The digital signature cookie. This could be the flash cookie. This parameter is sent only if it is sent by the browser
digitalClientType	The digital cookie client type. The type of flash client used. If not available, please use the value 0
digitalClientVersion	The version of the digital cookie client. The version of the flash client.
fingerPrintType	The type of finger printing. The value for defined in the properties file.
fingerPrint	The finger print. If it is browser characteristics, the header is parsed into this string. This is the name value representation of the browser header information.
digFingerPrintType	The type of the digital finger printing
digFingerPrint	The digital fingerprint
requestTime	The time when this request was made. Used by simulator
groupId	The groupId of this user. This is the primary group to which this user belongs to.
userId	The ID of the user. This is the primary key ID of the user. It should be null for users who are invalid.
loginId	The loginId used by the user for login in. This is mandatory parameter.
isSecure	Whether this node is secure and can be registered. This is to indicate the login is from a secure or registered device. If there is no concept of device, then send false value for this parameter.
result	The authentication result. This is the enumeration value of the authentication result.
clientType	This is an enum value defined to identify the client type used for authentication.
clientVersion	The version of the client. This is an optional parameter to specify the version of the client used

4.2.7 updateAuthStatus

API updateAuthStatus: API to update the User node log auth status and, if appropriate, trigger pattern data analysis.

- public VCryptResponse updateAuthStatus(String requestId, int resultStatus, int clientType, String clientVersion);
- public VCryptResponse updateAuthStatus(String requestId, Date requestTime, int resultStatus, int clientType, String clientVersion);
- public VCryptResponse updateAuthStatus(java.lang.String requestId, int resultStatus, int clientType, java.lang.String clientVersion, boolean analyzePatterns)

- public VCryptResponse updateAuthStatus(java.lang.String requestId, java.util.Date requestTime, int resultStatus, int clientType, java.lang.String clientVersion, boolean analyzePatterns)

Table 4–8 API updateAuthStatus Parameters

Parameter	Description
requestId	The ID for the login session. The same ID is necessary for all the calls to the Oracle Adaptive Access Manager API for the login session.
requestTime	The time when this request was made. Used by simulator
resultStatus	The authentication result. This is the enumeration value of the authentication result
clientType	An enum value defined to identify the client type used for authentication.
clientVersion	An optional parameter to specify the version of the client used
analyzePatterns	Boolean to indicate if the pattern analysis should be done. When passed in as true the pattern analysis is done for this transaction.

upateAuthStatus API needs to be called when there is a change in the user's authentication status.

Prerequisite: Make sure to call updateLog before making changes to the authentication status.

The list of possible authentication status values are maintained on the server-side with "auth.status.enum". Make sure to match the status value with one of those values. The list of statuses can be installation-specific. You can add or remove the standard status definitions that came with the product by updating "auth.status.enum".

Scenarios:

1. Send login status along with updateLog call
   a. updateLog API call takes status as one of the parameters, send login status with updateLog API, and avoid calling updateAuthStatus calls.
2. Login and then update authentication status
   a. Set status to "pending" with updateLog API call
   b. Call updateAuthStatus with actual status value.
3. Challenge flow
   a. Challenge is considered a second factor authentication. It is recommended to set the status to
      1. "Pending" before challenging
      2. After the user's response, set the status to "success" or "wrong_answer"
4. Run rules
   a. Usually, there is no need to call update auth status after running the rules API. Run rules API internally handles known actions, "Block" and "Locked," and updates the authentication status appropriately.

4.2.8 processPatternAnalysis

API processPatternAnalysis: API to trigger the processing of data for pattern matching. This call will only trigger the processing of data for pattern matching. The last parameter transactionType can be used by authentication type user interactions, since auth (or login) are not first-class transactions.

- public VCryptResponse processPatternAnalysis(java.lang.String requestId, long transactionId, int status, java.lang.String transactionType)

Table 4–9 processPatternAnalysis

Parameter	Description
requestId	request Id
transactionId	Transaction Id to be updated.
status	New Status
transactionType	String that indicates the type of transaction. Has to be "auth" for authentication type. For other types it can be "bill_pay, ...",; basically the type name of the transaction.

4.2.9 markDeviceSafe

API markDeviceSafe: API to mark the device as safe for the user.

- public boolean markDeviceSafe(String requestId, boolean isSafe);

Table 4–10 API markDeviceSafe Parameters

Parameter	Description
requestId	The ID for the login session. The same ID is necessary for all the calls to the Oracle Adaptive Access Manager API for the login session.
isSafe	Is this device safe for the user

4.2.10 IsDeviceMarkedSafe

API IsDeviceMarkedSafe: API to verify that the device associated with this request is safe

- public VCryptBooleanResponse IsDeviceMarkedSafe(String requestId);

Table 4–11 API IsDeviceMarkedSafe Parameters

Parameter	Description
requestId	The ID for the login session. The same ID is necessary for all the calls to the Oracle Adaptive Access Manager API for the login session.

4.2.11 clearSafeDeviceList

API clearSafeDeviceList: API to clear the safe device list of the user associated with this request

- public VCryptBooleanResponse IsDeviceMarkedSafe(String requestId);

Table 4–12 API clearSafeDeviceList Parameters

Parameter	Description
requestId	The ID for the login session. The same Id is necessary for all the calls to Bharosa API for the login session.

4.3 Rules Engine

The Rules Engine component of Adaptive Risk Manager provides APIs for policy execution and enforcement.

The Rules Engine can be viewed as an enforcement component of the Adaptive Risk Manager. Based on the calling context, various policies and models configured by the administrators are evaluated and the results are provided.

4.3.1 processRules

API processRules: API to process policy sets for the given runtimes.

A Policy Set consists of policies and rules. The API processes the policy set for each runtime and returns VCryptRulesResult. There has to be at least one updateLog call before calling processRules.

- public VCryptRulesResult processRules(String requestId, List runtimeTypes, Map contextMap);
- public VCryptRulesResult processRules(String requestId, Date requestTime, List runtimeTypes, Map contextMap);
- public VCryptRulesResult processRules(String requestId, Long transactionId, String extTransactionId, Date requestTime, List runtimeTypes, Map contextMap);

Table 4–13 API processRules Parameters

Parameter	Description
requestId	The ID for the login session. The same ID is necessary for all the calls to the Oracle Adaptive Access Manager API for the login session.
runtimeTypes	The list of runtimes to be evaluated. Each runtime in the given list is evaluated and result is returned.
requestTime	The time when this request was made. Used by simulator
transactionId	Transaction Id, to be treated as the current transaction for this process rules call. Can be null, if no transaction Id is sent, API will check for extTransactionId, else will use the last transaction in the login session as current transaction.
extTransactionId	Handle to the external transaction id to be used as the current transaction. Optional, will use last transaction in the login session as the current transaction, if no transactionId or extTransactionId are sent.
contextMap	The data for the context. Rules in models can make decisions based on this data. Key, value pairs

4.3.1.1 Device ID Evaluation

Along with rule results, the Rules Engine component can return a Device Id. The Device Id that is returned by the Rules Engine API is an internal identifier within Adaptive Risk Manager. It is the same Id shown in Adaptive Risk Manager for the user session.

A sample algorithm to get the Device Id is shown below:

```
VCryptRulesResult rulesResult = new
VCryptRulesEngineImpl().processRules(<params..>);

If (!rulesResult.getVCryptResponse().isSuccess()) {

     Logger.error("Error running rules " +
rulesResult.getVCryptResponse().getErrorMessage());
}

Long deviceId = VCryptRulesResult#getDeviceId();
```

Troubleshooting:

- Make sure the Adaptive Risk Manager version is 10.1.4.5 or above
- The following property must be set to true for the Device Id to be captured.

  ```
  bharosa.tracker.send.deviceId=true
  ```

4.4 Customer Care

Customer Care provides APIs for limited customer care functionality, typically used in customer care portals. These APIs will not use audit feature and access control.

4.4.1 getFinalAuthStatus

API getFinalAuthStatus: API to return the final authentication status of a user given the user ID of the user. This method can only go back up to 30 days.

- public VCryptIntResponse getFinalAuthStatus(String requestId, String userId);

Table 4–14 API getFinalAuthStatus Parameters

Parameter	Description
requestId	requestId for tracking purpose
userId	Unique identifier to the user, can't be null

4.4.2 setTemporaryAllow

API setTemporaryAllow: This API call sets a temporary allow for the given user. A temporary allow can override the final rule action.

- public VCryptResponse setTemporaryAllow(String customerId, int tempAllowType, Date expirationDate);

Table 4–15 API setTemporaryAllow Parameter

Parameter	Description
customerId	Id of the customer
tempAllowType	Type of the temporary allow. The User Defined Enum for this type is customercare.case.tempallow.level.enum
expirationDate	Expiration date if the tempAllowType is "userset". Otherwise it can be null or empty.

4.4.3 cancelAllTemporaryAllows

API cancelAllTemporaryAllows: API to cancel any temporary allows associated with the customer ID.

- public VCryptResponse cancelAllTemporaryAllows(String customerId);

Table 4–16 API cancelAllTemporaryAllows Parameters

Parameter	Description
customerId	The ID of the customer

4.4.4 resetUser

API resetUser: API to reset all the profiles set for the user. This includes registration, questions, images and phrases, selected or assigned to the user

- public VCryptResponse resetUser(String customerId);

Table 4–17 API resetUser Parameters

Parameter	Description
customerId	The ID of the customer

4.4.5 getRulesData

API getRulesData: API to return all the rules executed for the given session ID, and provide basic information on what rules got triggered. It does not provide complete hierarchy information.

- public VCryptSessionRuleData getRulesData(String requestId);

Table 4–18 API getRulesData Parameters

Parameter	Description
requestId	requestId for logging and tracing with client requests in case of errors

4.4.6 getActionCount

API getActionCount: API to get the action count for the given actionEnumId and consult the configuration for the available action enums

- public VCryptIntResponse getActionCount(String requestId, String customerId, Integer);

Table 4–19 API getActionCount Parameters

Parameter	Description
requested	requestId for logging and tracing with client requests in case of errors
customerId	The unique identifier to the user, required
actionEnumId	actionEnum, required, rule.action.enum to be counted

Part II

Universal Installation Option and Related Integrations

Part II contains the following chapters:

5 Oracle Adaptive Access Manager Proxy

The Oracle Adaptive Access Manager's Universal Installation Option (UIO) offers multi-factor authentication to Web applications without requiring any change to the application code.

The purpose of this chapter is to explain the Oracle Adaptive Access Manager Proxy, one of the components in the Oracle Adaptive Access Manager UIO deployment. This chapter provides programming information and instructions on the installation and configuration of the Oracle Adaptive Access Manager Proxy.

The Oracle Adaptive Access Manager Proxy is available for the Apache Web server and Microsoft Internet Security and Acceleration (ISA) Server.

Instructions for both Microsoft ISA and Apache implementations are provided in this chapter.

The chapter is intended for integrators who configure the Oracle Adaptive Access Manager Proxy to add multi-factor authentication to Web applications. An understanding of HTTP request/response paradigm is required to understand the material presented in this document.

For information on configuring the Adaptive Strong Authenticator, the client-facing multi-factor authentication Web application specific to the UIO deployment, refer to Chapter 6, "Configuring Adaptive Strong Authenticator."

5.1 Introduction

The Introduction section of this chapter contains the following topics:

- Important Terms
- Architecture
- References

5.1.1 Important Terms

For your reference, important terms are defined below.

Microsoft ISA

From the Microsoft Web site: "the Internet Security and Acceleration (ISA) Server is the integrated edge security gateway that helps protect IT environments from Internet-based threats while providing users with fast and secure remote access to applications and data."

Universal Installation Option (UIO)

The Universal Installation Option (UIO) is the Oracle Adaptive Access Manager integration strategy that does not require any code modification to the protected Web applications. The Universal Installation Option involves placing the Oracle Adaptive Access Manager Proxy in front of the protected Web applications

Proxy

A proxy is a server that services the requests of its clients by forwarding requests to other servers. This chapter is concerned with the Web proxy, where the proxy handles Web Protocols, mainly HTTP.

Forward Proxy

A forward proxy is an intermediate server that sits between the client and the origin server. To get content from the origin server, the client sends a request to the proxy naming the origin server as the target, and the proxy then requests the content from the origin server and returns it to the client. The client must be specially configured to use the forward proxy to access other sites.

Reverse Proxy

A reverse proxy appears to the client just like an ordinary Web server. No special configuration on the client is necessary. The client makes ordinary requests for content in the name-space of the reverse proxy. The reverse proxy then decides where to send those requests and returns the content as if it were itself the origin. The Oracle Adaptive Access Manager Proxy running in the Microsoft Internet Security and Acceleration (ISA) Server is an example of a reverse proxy.

Adaptive Strong Authenticator

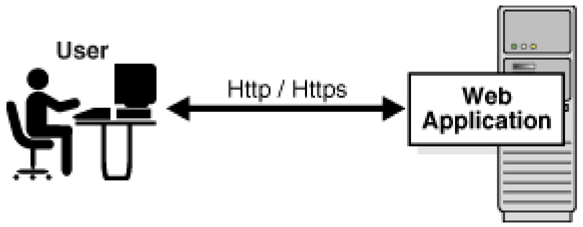
The Adaptive Strong Authenticator is the Web application component of Oracle Adaptive Access Manager. The Oracle Adaptive Access Manager Proxy redirects the client browser to Adaptive Strong Authenticator for tracking and authentication purposes as defined by the Oracle Adaptive Access Manager UIO Proxy XML configuration.

5.1.2 Architecture

The following diagrams show a typical Oracle Adaptive Access UIO deployment.

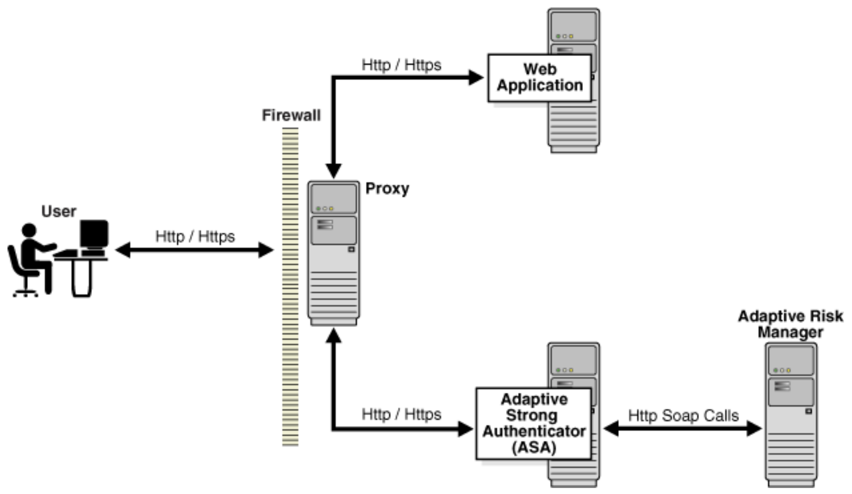
The first diagram shows a Web application before the Oracle Adaptive Access UIO is deployed to provide multi-factor authentication.

Figure 5–1 Before the Oracle Adaptive Access UIO



The second diagram shows various components added after the Oracle Adaptive Access UIO deployment.

Figure 5–2 After UIO Deployment



The Oracle Adaptive Access Manager Proxy intercepts the HTTP traffic between the client (browser) and the server (Web application) and performs the appropriate actions, such as redirecting the traffic to the Adaptive Strong Authenticator to provide multi-factor authentication and authorization. The Adaptive Strong Authenticator, in turn, communicates with Adaptive Risk Manager to assess the risk, and then takes the appropriate actions, such as permitting the login, challenging the user, blocking the user, and other actions.

The Oracle Adaptive Access Manager Proxy is available for the Apache Web server and Microsoft Internet Security and Acceleration (ISA) Server.

5.1.3 References

For detailed information on installing and configuring the Microsoft ISA server, refer to the Microsoft ISA Server setup documentation. Web publishing rule creation and listener creation are explained further in this document.

For more information about the Apache HTTP Server, refer to the Apache HTTP Server 2.2 documentation (http://httpd.apache.org/docs/2.2).

5.2 Oracle Adaptive Access Manager Proxy for Microsoft ISA Installation

The Oracle Adaptive Access Manager Proxy for Microsoft ISA uses the API provided by Microsoft ISA Server to monitor the HTTP traffic and perform various actions. Microsoft ISA Server 2006 Standard Edition should be installed before installing the Oracle Adaptive Access Manager Proxy for Microsoft ISA. Please refer to the Microsoft ISA Server setup documentation for the details on installing and configuring the ISA server. For a successful installation of the proxy, a .NET framework 2.0 or better should to be installed. We also advise you to install all the recommended updates from Microsoft on the machine.

Microsoft ISA Server 2006 Standard Edition should be installed and Web publishing rules for Web applications should be created before installing the Oracle Adaptive Access Manager Proxy.

This section provides:

- information on creating Web publishing rules and listeners so that Web applications and Adaptive Strong Authenticator can be accessible from the Internet.

- programming information and instructions on the installation of the Oracle Adaptive Access Manager Proxy for Microsoft ISA.

5.2.1 Proxy Web Publishing Configuration

The purpose of this section is to explain the creation of Web publishing rules and listeners in Microsoft ISA for Adaptive Access Manager Applications. It is intended for integrators who install and configure Microsoft ISA to support multiple Web applications.

5.2.1.1 Web Listener Creation

For details on creating a Web listener, refer the Microsoft Web site.

1. For the Web Listener Name, enter Bharosa Proxy Listener.
2. Select SSL secure connection as the type of connection the Web listener will establish with clients.
3. For the Web Listener IP Addresses, choose external, internal, and local host.
4. Choose to use a single certificate for the Web Listener and select the certificate.
5. Select no authentication for how clients will validate their credentials.

5.2.1.2 Web Publishing Rule Creation

In a typical deployment, Web applications and the Adaptive Access Manager Adaptive Strong Authenticator run on machines in an internal network and are not directly accessible from the Internet. In the case of the Oracle Adaptive Access Manager Proxy for Microsoft ISA, only the Oracle Adaptive Access Manager Proxy machine, which runs Microsoft ISA Server, will be accessible from the Internet. Web applications, including Adaptive Strong Authenticator, should be published via Web publishing rules in the Microsoft ISA Server.

To create a new Web publishing rule you will need to access Microsoft ISA Server's Web publishing rule wizard and follow the on-screen instructions.

1. For the name of the rule, enter a name such as "Bharosa Adaptive Strong Authenticator" or "Online Banking Application."
2. Choose to allow incoming requests matching the rule conditions.
3. Choose to publish a single Web site or a load balancer in front of several servers.
4. Choose SSL as a connection option if the Web application is listening on SSL; otherwise, choose the non-secured connection option.
5. For the internal site name, provide the machine name where the Web server runs.
6. If the IP address or the machine name of the Web application to be published is known, select the option to use the computer name or IP address and provide that information.
7. If you want to include all files and subfolders within a folder, enter /* for the name of the file or folder you want to publish. If you need to publish more than one file or folder, enter only the first file/folder instead. The remaining files can be entered later by editing the rule. Later you will enter the path you entered here for your public details.
8. For your Web listener, select Bharosa Proxy Listener.
9. For authentication delegation, select no delegation and that client cannot authenticate directly.

10. Make sure you're able to apply the rule to requests from all users.

Check the properties for your newly created rule by accessing the rule properties.

1. If more than one file or folders need to be published, add all paths.
2. If you have more than one domain name to access the application, add all the domain names.

5.2.2 Registering the Oracle Adaptive Access Manager Proxy for Microsoft ISA DLL

The Oracle Adaptive Access Manager Proxy for Microsoft ISA is installed as a Web filter in Microsoft ISA Server. Please follow the steps below to install the Oracle Adaptive Access Manager Proxy for Microsoft ISA:

1. Copy the BharosaProxy.dll to the Microsoft ISA Server installation directory, which is by default, %ProgramFiles%\Microsoft ISA Server
2. Open the command prompt and navigate to the Microsoft ISA Server installation directory
3. Register the BharosaProxy.dll with the following command:

   ```
   regsvr32 .\BharosaProxy.dll
   ```

5.2.3 Settings

Various aspects of the Oracle Adaptive Access Manager Proxy for Microsoft ISA can be controlled using the registry values. All Oracle Adaptive Access Manager Proxy for Microsoft ISA settings are stored under HKLM\SOFTWARE\Bharosa\Proxy key. Changes to most of the registry values are picked up by the proxy immediately without requiring a proxy restart.

5.2.3.1 Configuration files

During startup (and during config reload), the proxy loads the configuration from the files listed under the HKLM\SOFTWARE\Bharosa\Proxy\ConfigFiles key.

- The type of each value under this key should be REG_DWORD.
- The name of each value under this key should be the filename containing the proxy configuration.
- The filename can either be fully qualified or relative to the location of the BharosaProxy.dll
- The proxy will load a configuration file only if the data has a nonzero value. This can be used to dynamically load and unload proxy configuration files.
- The files will be loaded in the lexicographic order of the filenames in the registry.
- Changes under the ConfigFiles key will not be effective until either the proxy is restarted or HKLM\SOFTWARE\Bharosa\Proxy\ReloadConfig is set to 1.

5.2.3.2 Configuration Reload

The proxy configuration can dynamically be changed while the proxy is running; new configuration files can be added and currently loaded files can be updated or removed. These changes will not be applied until ReloadConfig registry value is set to a nonzero value. Upon setting ReloadConfig to a nonzero value, the proxy will load configuration files. After loading the files, the proxy will reset the ReloadConfig value to 0.

Please note that the new configuration will be used only for new client (browser) connections. Clients already connected will continue to use the previous configuration.

5.2.3.3 Session Id Cookie

The Oracle Adaptive Access Manager Proxy for Microsoft ISA uses a cookie to associate multiple requests from a client. The name of this cookie can be configured in the registry value, SessionIdCookieName (of type REG_SZ). If this value is not present or empty, the Oracle Adaptive Access Manager Proxy for Microsoft ISA will use the cookie name, BharosaProxy_SessionId.

5.2.3.4 Session Inactive Interval

Sessions in the Oracle Adaptive Access Manager Proxy for Microsoft ISA will be removed after a certain period of inactivity. This period, in number of seconds, is specified in the MaxSessionInactiveInterval registry value. If this value is not specified, the Oracle Adaptive Access Manager Proxy for Microsoft ISA will remove a session after 1200 seconds (20 minutes) of inactivity. This value should be set to at least a few seconds higher than the Web application session timeout value.

5.2.3.5 Settings for Troubleshooting

Trace messages from the Oracle Adaptive Access Manager Proxy for Microsoft ISA can be used for trouble shooting any issues with the proxy configuration and operation. Trace settings, like trace level and destinations, can be controlled using the following registry values under HKLM\SOFTWARE\Bharosa\Proxy:

Table 5–1 Settings for Troubleshooting

Name	Type	Description
TraceFilename	REG_SZ	Full path to the file in which the trace messages should be written to
TraceFileMaxLength	REG_DWORD	Maximum length of the trace file in bytes. Once the trace file reaches this size, the proxy will rename the file by adding the timestamp to the filename and create a new trace file to write subsequent trace messages.
TraceToFile	REG_DWORD	Trace messages will be written to file only if this value is nonzero.
TraceToDebugTerminal	REG_DWORD	Trace messages will be written to debug the terminal only if this value is nonzero. Tools like DbgView can be used to view these trace messages in real time.

Table 5–1 (Cont.) Settings for Troubleshooting

Name	Type	Description
TraceLevel	REG_DWORD	Each trace level (debug, info, warning, error) has an integer value associated. The registry value should be set to the sum of desired the trace level values. FATAL 0x1, ERROR 0x2, WARN 0x4 INFO 0x8, DEBUG 0x10, HTML 0x80, FLOW 0x80000
IgnoreUrlMappings	REG_DWORD	If this value is nonzero, the proxy will ignore all the interceptors defined in the proxy configuration. Essentially this will put the Oracle Adaptive Access Manager Proxy for Microsoft ISA in "pass-through" mode.
CaptureTraffic	REG_DWORD	The proxy does not handle (save, inspect) the HTTP traffic for URLs that do not have interceptors defined in the configuration. But during application discovery process, it will be necessary to get a dump of all the HTTP traffic thorough the proxy. On such occasion, this registry value should be set to nonzero.

5.3 Oracle Adaptive Access Manager Proxy for Apache

To install the Oracle Adaptive Access Manager Proxy for Apache, a new Apache httpd has to be installed into which the Oracle Adaptive Access Manager Proxy for Apache will be installed. This Apache httpd will use mod_proxy to reverse-proxy to the backend application that has to be protected.

The Installation section contains information for installing the Oracle Adaptive Access Manager Proxy for Apache for Windows and Linux platforms.

The procedure involves:

- ensuring that the Apache httpd requirements are met
- copying the proxy dlls and supported dlls to specific directories in Apache
- creating a new user to run the UIO Proxy Apache process (on Linux only)
- configuring memcache (for Linux only)
- editing the httpd.conf to activate the proxy
- modifying the settings of the proxy using application configuration xml files
- Optionally install the mod_proxy_html, which is needed to rewrite the HTML links in a proxy situation, to ensure that links work for the users outside the proxy

5.3.1 Package Contents

The Oracle Adaptive Access Manager Proxy for Apache package binaries for Windows and Linux are different. Since the proxy is in C/C++, the same binary will not work on different platforms (unlike Java).

The typical package names are:

- oaam_win_apache_uio.zip
- oaam_rhel4_apache_uio.zip

5.3.1.1 Windows

For Windows, the Oracle Adaptive Access Manager Proxy for Apache Package, oaam_win_apache_uio.zip, contains the binary files listed below.

Name	Description
mod_uio.so	Oracle Adaptive Access Manager Proxy for Apache module
log4cxx.dll	Apache Log4cxx library
libxml2.dll	XML/HTML Parser
apr_memcache.dll	APR Memcache client library. For 10.1.4.5.1 and onward.

The package contains the following data files.

Name	Description
UIO_Settings.xml	Oracle Adaptive Access Manager Proxy for Apache Settings XML file
UIO_log4j.xml	Oracle Adaptive Access Manager Proxy for Apache Log4j (log4cxx) configuration XML file
TestConfig.xml	Oracle Adaptive Access Manager Proxy for Apache Sample application configuration file
UIO_Settings.rng	Relax NG grammar for UIO_Settings.xml
UIO_Config.rng	Relax NG grammar for application configuration XML files

5.3.1.2 Linux

For Linux, the Oracle Adaptive Access Manager Proxy for Apache Packages contain the binary files listed below.

Name	Description
mod_uio.so	Oracle Adaptive Access Manager Proxy for Apache module
liblog4cxx.so.0.10.0.0	Apache Log4cxx library
libxml2.so.2.6.32	XML/HTML parser
libapr_memcache.so.0.0.1	APR Memcache client library. For 10.1.4.5.1 and later.

It contains the following data files.

Name	Description
UIO_Settings.xml	Oracle Adaptive Access Manager Proxy for Apache Settings XML file
UIO_log4j.xml	Oracle Adaptive Access Manager Proxy for Apache Sample Log4j (log4cxx) configuration XML file
TestConfig.xml	Oracle Adaptive Access Manager Proxy for Apache Sample application configuration files
UIO_Settings.rng	Relax NG grammar for UIO_Settings.xml
UIO_Config.rng	Relax NG grammar for application configuration XML files

5.3.2 Apache httpd Requirements

The pre-installation steps involve for downloading or building the Apache httpd, depend on whether you are on the Windows or Linux platform, and on whether certain requirements are met.

5.3.2.1 Windows

You can download the latest Apache httpd (2.2.8) build for Windows from the Apache Web site.

Ensure that:

- the Apache httpd (2.2.8) build is version 2.2.8
- the mod_proxy support is enabled (the standard installation contains the mod_proxy)
- the mod_ssl support is enabled (required for 10.1.4.5.1 and above)

5.3.2.2 Linux

Instructions to build the Apache httpd are available on the Apache Web site. When you build Apache, ensure that

- the Apache httpd (2.2.8) build is version 2.2.8
- the mod_so is enabled (for dynamically loading modules)
- the mod_proxy is enabled
- the mod_ssl support is enabled (required for 10.1.4.5.1 and above)

5.3.3 Copying the Oracle Adaptive Access Manager Proxy for Apache and Supported Files to Apache

Instructions are provided below for copying the Oracle Adaptive Access Manager Proxy for Apache and support files to specific directories in Apache for both Windows and Linux platforms.

5.3.3.1 Windows

The table shown below summarizes:

- the directories you have to copy the Oracle Adaptive Access Manager Proxy for Apache files to after installation
- the tree structure of the Oracle Adaptive Access Manager Proxy for Apache libraries and configuration files, assuming that you installed the files in C:\Apache2.2
- the directories the Oracle Adaptive Access Manager Proxy for Apache binary files go into

Directories	File Descriptions
C:\Apache2.2\modules\mod_uio.so	Oracle Adaptive Access Manager Proxy for Apache module
C:\Apache2.2\bin\log4cxx.dll	Apache Log4cxx library
C:\Apache2.2\bin\libxml2.dll	XML/HTML Parser

Directories	File Descriptions
C:\Apache2.2\bin\apr_memcache.dll	APR Memcache library. For 10.1.4.5.1 and onward.

The data files will go in the directories summarized in the table below.

Directories	File Descriptions
C:\OAAMUIO\UIO_Settings.xml	Oracle Adaptive Access Manager Proxy for Apache settings XML file
C:\OAAMUIO\UIO_log4j.xml	Oracle Adaptive Access Manager Proxy for Apache Log4j (log4cxx) configuration XML file
C:\OAAMUIO\TestConfig.xml	Oracle Adaptive Access Manager Proxy for Apache application configuration files (any number)
C:\OAAMUIO\UIO_Settings.rng	Relax NG grammar for UIO_Settings.xml
C:\OAAMUIO\UIO_Config.rng	Relax NG grammar for application configuration XML files
C:\OAAMUIO\logs\uio.log	Oracle Adaptive Access Manager Proxy for Apache log

If you want to change the location of the various configuration files, refer to the "Configuring httpd.conf" section.

5.3.3.2 Linux

After the installation of the Apache httpd, you will have to copy the Oracle Adaptive Access Manager Proxy for Apache binary files into (assuming Apache httpd is installed in /usr/local/apache2) the directories shown below.

Directories	Description
/usr/local/apache2/modules/mod_uio.so	Oracle Adaptive Access Manager Proxy for Apache Module
/usr/local/apache2/lib/liblog4cxx.so.0.10.0.0	Apache Log4cxx Library
/usr/local/apache2/lib/libxml2.so.2.6.32	XML/HTML Parser
/usr/local/apache2/lib/libapr_memcache.so.0.0.1	APR Memcache client library. For 10.1.4.5.1 and later

Then, create soft links to the libraries as follows:

```
cd /usr/local/apache2/lib
ln -s liblog4cxx.so.10.0.0 liblog4cxx.so.10
ln -s libxml2.so.2.6.32 libxml2.so.2
ln -s libapr_memcache.so.0.0.1 libapr_memcache.so.0
```

Also, ensure that the binary files have executable permission.

Apache httpd is typically run as root so that it creates a parent process that listens on port 80, and it spawns handler processes that run as the user given in the User directive in httpd.conf

For this reason, create a user called oaamuio that will be the runtime user for the Oracle Adaptive Access Manager UIO for Apache. The Oracle Adaptive Access Manager UIO configuration and log files will be accessible by this user. Ensure that only this user can access the log files. Assuming /home/oaamuio is the home directory for this user, the directory structure will look like the one presented in the table below.

The Oracle Adaptive Access Manager UIO for Apache data files should follow the directory structure shown in the table below.

Directories	**Description**
/home/oaamuio/uio/UIO_Settings.xml	Oracle Adaptive Access Manager Proxy for Apache settings XML file
/home/oaamuio/uio/UIO_log4j.xml	Oracle Adaptive Access Manager Proxy for Apache Log4j (log4cxx) configuration XML file
/home/oaamuio/uio/TestConfig.xml	Oracle Adaptive Access Manager Proxy for Apache application configuration files (any number)
/home/oaamuio/uio/UIO_Settings.rng	Relax NG grammar for UIO_ Settings.xml
/home/oaamuio/uio/UIO_Config.rng	Relax NG grammar for application configuration XML files
/home/oaamuio/uio/logs/uio.log	Oracle Adaptive Access Manager Proxy for Apache log

If you want to change the location of the various configuration files, refer to the "Configuring httpd.conf" section.

The run-time user of httpd should have the appropriate permissions to access all these files.

5.3.4 Configuring Memcache (for Linux only)

On Linux, httpd can run with two different MPMs (the httpd kernel): httpd with prefork MPM (httpd kernel) or with worker MPM. The MPM is built into the httpd and is not a run-time option. Usually, the binary distribution of Apache httpd is with prefork MPM. If you need to use worker MPM, you will have to build Apache httpd using the instructions from the Apache Web site.

With prefork MPM, httpd maintains a pool of single-threaded process, where each request is handled by a single process. With worker MPM, httpd maintains a pool of multi-threaded processes, where every process could be handling multiple requests at a time. (On Windows, httpd MPM is always in multi-threading mode with a single process.)

On Linux, in the case where the httpd runs multiple processes, the Oracle Adaptive Access Manager Proxy for Apache session data must be maintained in a common store (db or cache) so that multiple processes can access the session data. The Oracle Adaptive Access Manager Proxy uses memcache (a memory based very fast cache) to store the session data.

At startup, the Oracle Adaptive Access Manager Proxy auto-detects whether httpd is running with a single process or multiple processes. If httpd is running with multiple processes (which is the case with prefork or worker mpm on Linux), it will try to connect to the memcache daemon using default connection parameters (that are defined below in the "UIO_Settings.xml" section). On Windows, by default, the Oracle Adaptive Access Manager Proxy will use local sessions and not try to connect to the memcache daemon; however it can be configured to maintain session data in memcache daemon (explained in the "UIO_Settings.xml" section).

For the scenarios where the Oracle Adaptive Access Manager Proxy for Apache will be connecting to memcache daemon, you will have to install memcache on your system using the instructions from the memcache Web site and run the memcache daemon(s) before running the Apache httpd.

Install memcache using instructions from the Web site, http://www.danga.com/memcached/. You may already have a binary installation available from your Linux distribution. The Oracle Adaptive Access Manager Proxy for Apache has been tested with version 1.2.5 of memcache.

In the simple configuration, you can run a single memcache daemon on the machine that is running your Apache httpd.

You can choose to have a highly scalable installation, where you run more than one memcache daemon-- all of which are accessed by multiple machines running Apache httpds.

5.3.5 Configuring httpd.conf

This section provides information on how to edit the httpd.conf file to activate the Oracle Adaptive Access Manager Proxy for Apache.

5.3.5.1 Basic Configuration without SSL

In the sample installation, the Apache httpd has been installed in c:\ProgramFiles\Apache2.2 or /usr/local/apache2.

Also, in the sample installation, BigBank40 and BharosaUIO40 are running on test.dummy.com.

Follow the procedure below to ensure that http.conf is correctly set up in your environment.

1. Ensure that the lines shown below are uncommented to enable mod_proxy.

   ```
   LoadModule proxy_module modules/mod_proxy.so
   LoadModule proxy_http_module modules/mod_proxy_http.so
   ```

2. Add the following line to the end of the LoadModule group of lines to activate the Oracle Adaptive Access Manager Proxy for Apache.

   ```
   LoadModule uio_module modules/mod_uio.so
   ```

3. Add a line to point to the UIO_Settings.xml file that has the settings for the Oracle Adaptive Access Manager Proxy for Apache.

   Note: This should be an absolute path to the UIO_Settings.xml file.

   On Windows (all paths should be with forward slashes),

   ```
   UioProxySettingsFile c:/OAAMUIO/UIO_Settings.xml
   ```

   On Linux,

   ```
   UioProxySettingsFile /home/oaamuio/uio/UIO_Settings.xml
   ```

4. Disable mod_proxy's forward-proxy-ing capability since it is not needed.

   ```
   ProxyRequests Off
   <Proxy *>
         Order deny,allow
         Allow from all
   </Proxy>
   ```

5. Enable the mod_proxy configuration to reverse-proxy to the protected applications (BigBank40 and BharosaUIO40 in our sample installation).

   ```
   ProxyPass / http://uio-dev.oracle.com:9090/
   ProxyPassReverse / http://uio-dev.oracle.com:9090/
   ```

6. Set the user/group of httpd using User and Group directives to oaamuio.

The actual settings for #4 and #5 above are installation-specific. They are only examples of the settings you must set. For information on setting details, refer to the Apache Web site.

With the above changes and by properly setting up UIO_Settings.xml, you should be able to access BigBank40 and run Phase One scenarios. The URL for BigBank40 is

```
http://<apache-host>:<apache-port>/bigbank40
```

So far in this chapter, we have performed the configuration to the proxy without using SSL.

5.3.5.2 Configuration with SSL

To enable SSL, refer to the Apache Web site for Tomcat and for Apache procedures.

Note that the Oracle Adaptive Access Manager Proxy for Apache requires mod_ssl to be part of httpd. This ensures that the OpenSSL library is linked in and is properly configured for the Oracle Adaptive Access Manager Proxy for Apache to generate session ids. You need to ensure that mod_ssl is loaded and you do not need to do any configuration if you are not using SSL.

mod_proxy_html module (optional)

Optionally, you may need to install the mod_proxy_html (http://apache.webthing.com/mod_proxy_html/) Apache module. This module is needed only if the protected application has web pages that have hard-coded URL links to itself. If the application has relative URLs, you do not need this module.

From their Web site, the executive summary of this module is as follows:

"mod_proxy_html is an output filter to rewrite HTML links in a proxy situation, to ensure that links work for users outside the proxy. It serves the same purpose as Apache's ProxyPassReverse directive does for HTTP headers, and is an essential component of a reverse proxy.

For example, if a company has an application server at appserver.example.com that is only visible from within the company's internal network, and a public webserver www.example.com, they may wish to provide a gateway to the application server at http://www.example.com/appserver/. When the application server links to itself, those links need to be rewritten to work through the gateway. mod_proxy_html serves to rewrite <a href="http://appserver.example.com/foo/bar.html">foobar</a> to <a href="http://www.example.com/appserver/foo/bar.html">foobar</a> making it accessible from outside."

5.3.6 Modifying the Oracle Adaptive Access Manager Proxy for Apache Settings

5.3.6.1 UIO_Settings.xml

```
<UIO_ProxySettings xmlns="http://bharosa.com/">

       <Log4jProperties location="C:/OAAMUIO/UIO_log4j.xml"/>

Or

       <Log4jProperties location="/home/oaamuio/uio/UIO_log4j.xml"/>

       <GlobalVariable name="one" value="something"/>

       <ConfigFile location="/home/oaamuio/uio/TestConfig1.xml" enabled="false"/>
       <ConfigFile location="/home/oaamuio/uio/TestConfig.xml" enabled="false"/>

       <ConfigFile location="C:/OAAMUIO/TestConfig1.xml" enabled="false"/>
       <ConfigFile location="C:/OAAMUIO/TestConfig.xml" enabled="true"/>

       <Setting name="GarbageCollectorInterval_ms" value="5"/>
       <Setting name="MaxSessionInactiveInterval_ms" value="5"/>
       <Setting name="SessionIdCookieName_str" value="UIOSessionId"/>

       <Setting name="IgnoreUrlMappings" value="0"/>
       <Setting name="CaptureTraffic" value="0"/>

       </UIO_ProxySettings>
```

Log4jProperties

Set the location of log4j.xml file that defines the logging configuration for the Oracle Adaptive Access Manager Proxy for Apache. The location should be an absolute path; it cannot be ServerRoot relative. On Linux, you have to ensure that the httpd process can access the directory.

When using httpd in a multi-processing mode, do not use FileAppender; use SocketAppender instead to log the logs from the different processes. Refer to the log4j documentation on the Internet for more information.

GlobalVariable

GlobalVariable is a global variable that is used in the application configuration. You can have any number of such name-value pairs.

ConfigFile

ConfigFile is the absolute path to an application configuration. You can have any number of such configurations. Again, you need to make sure, on Linux, that the httpd process has the permissions to access these files. Refer to "Oracle Adaptive Access Manager Proxy Configuration" to understand how to perform a configuration for an application.

Memcache

Memcache has the IP address and port of a memcache server. You can have multiple Memcache elements in the settings file if you have multiple memcache servers running. If you have a single local memcache running, you do not need to have this element at all. By default, the Oracle Adaptive Access Manager Proxy for Apache will try to connect to memcache on IP address 127.0.0.1 and port 11211.

Settings

These are flags to control the behavior of the Oracle Adaptive Access Manager Proxy for Apache. Various settings are listed in the table below.

Flags	Description
MaxSessionInactiveInterval_sec	Session expiry time in sec (default = 30 minutes) For example, <Setting name="MaxSessionInactiveInterval_sec" value="1800"/>
GarbageCollectorInterval_ms	Interval for running session expiry thread (default = 5 minutes) For example, <Setting name="GarbageCollectorInterval_ms" value="300000"/>
FileWatcherInterval_ms	Interval for checking if the settings or any config file has changed (default = 1minute) For example, <Setting name="FileWatcherInterval_ms" value="60000"/> (After modifying the configuration XML file, even though the proxy will update the configuration on the fly, it is advisable to restart the httpd server.)
SessionIdCookieName_str	Name of the cookie used by UIO to maintain its session (default = OAAM_UIOProxy_ SessionId For example, <Setting name="SessionIdCookieName_str" value="SessionId"/>
SessionCookie_DomainLevelCount	Domain level for the Sessions cookie For example, <Setting name="SessionCookie_DomainLevelCount" value="2"/>
IgnoreUrlMappings	Ignore the application configuration XML files; the proxy behaves as a flow-through proxy For example, <Setting name="IgnoreUrlMappings" value="0"/>. The value of 0 disables this mode and the value of 1 enables capture traffic mode. The value of 1 will make the proxy act as flow-through and the value of 0 will enable the configuration XML interceptors.

Flags	Description
CaptureTraffic	Capture the HTTP traffic - headers and content in the log files. This mode is for debugging purpose. Note that it captures the headers and contents as is and could contain customer's personal data. Use this mode with caution and only for debugging/test ! For example, <Setting name="CaptureTraffic" value="0"/>. Value of 1 enables capture traffic and 0 disables it.
MaxReqBodyBytes	Maximum request body size to cache while processing requests. This is necessary when the application has POSTs with big files getting uploaded. For example, <Setting name="MaxReqBodyBytes" value="10240"/>
UseMemcache	Force the use of memcache even when httpd is running in single process mode. Has no effect when running in multiple process mode. Applies at startup and requires restarting httpd for change to apply. For example, <Setting name="UseMemcache" value="1"/>". Value of 1 enables use of memcache for a single process httpd. Value of 0 is ignored. For 10.1.4.5.1 and above.
CachedConfigExpiry_sec	Expiry time for unused config XML data in memory, if multiple config XML configurations have been loaded into memory. This happens when config XML files are automatically loaded when they are modified. (Default = 60 minutes). For example, <Setting name="CachedConfigExpiry_sec" value="3600"/> For 10.1.4.5.1 and above.
AutoLoadConfig	Set to 1 to enable auto-loading of config XML files when they are modified by user. Set to 0 to turn this feature off. It is OK to enable this feature when using single-process mode of httpd. Do not enable this feature for multiple process mode of httpd for production use, since individual processes could have different versions of the config XML files. For example, <Setting name="AutoLoadCOnfig" value="1"/>. Value of 1 enables auto-load and 0 disables it. For 10.1.4.5.1 and above.

5.3.6.2 UIO_log4j.xml

For actual log4j format details, refer to log4j manual available on the Internet. Apache::log4cxx is a C++ implementation of the log4j framework and the XML file format is common to log4cxx and log4j.

5.3.6.3 Application configuration XMLs

These XML files are the application configuration files that are defined in the ConfigFile element of UIO_Settings.xml file.

5.4 Setting Up Rules and User Groups

For information on setting up rules and user groups, refer to the *Oracle Adaptive Access Manager Administrator's Guide*.

5.5 Setting Up Models

To set up models for UIO, import the out-of-the-box models. Information about importing models is available in the *Oracle Adaptive Access Manager Administrator's Guide*.

5.6 Oracle Adaptive Access Manager Proxy Configuration

The Oracle Adaptive Access Manager Proxy intercepts all HTTP traffic between the client browser and the Web application and performs actions specified in the configuration files. The configuration files are in XML format that comply with the Oracle Adaptive Access Manager Proxy configuration XML schema 2. The following sections describe various elements of the Oracle Adaptive Access Manager Proxy configuration file.

5.6.1 Interceptors

Interceptors are the most important elements in the Oracle Adaptive Access Manager Proxy configuration. You will see that authoring the Oracle Adaptive Access Manager Proxy configuration file is all about defining interceptors.

There are two types of interceptors: request interceptors and response interceptors. As the names suggest, request interceptors are used when the Oracle Adaptive Access Manager Proxy receives HTTP requests from the client browser and response interceptors are used when the Oracle Adaptive Access Manager Proxy receives HTTP response from the server, that is, Web application or Adaptive Strong Authenticator.

There are four components to an interceptor and all of them are optional.

1. List of URLs - the interceptor will be evaluated if the interceptor URL list contains the current request URL or if the URL list is empty.
2. List of conditions - conditions can inspect the request/response contents, such as checking for the presence of an HTTP header/parameter/cookie, and so on, or testing whether a header/parameter/cookie has a specific value or not. Filters and action defined in the interceptor will be executed only if all the conditions specified are met or if no condition is specified.
3. List of filters - filters perform an action that might modify the request/response contents or modify some state information in the Oracle Adaptive Access Manager Proxy. For example, a filter can add/remove HTTP headers, save HTTP header/parameter/cookie value in a proxy variable, and so on.
4. Action - after executing the filters the interceptor will perform the action, if one is specified. Actions can be one of:
    a. redirect the client to a different URL
    b. send a saved response to the client

c. perform a HTTP get on server
d. perform a HTTP post on server
e. send a saved request to the server

5.6.2 Conditions

Conditions are used in the Oracle Adaptive Access Manager Proxy to inspect HTTP request/response or the state information saved in the proxy (variables). Each condition evaluates to either true or false. Conditions are evaluated in the order they are listed in the configuration file until a condition evaluates to false or all conditions are evaluated. Here is the list of conditions that can be defined in an interceptor:

Table 5–2 Conditions Defined in an Interceptor

Condition name	Attributes	Description
HeaderPresent	id, enabled, name	Checks the presence of the specified header in request/response. The header name should be terminated by a colon (":"). Example: `<HeaderPresent name="userid:"/>`
ParamPresent	id, enabled, name	Checks the presence of the specified parameter in request. Example: `<ParamPresent name="loginID"/>`
QueryParamPresent	id, enabled, name	Checks the presence of the specified query parameter in the URL. Example: `<QueryParamPresent name="TraceID"/>`
VariablePresent	id, enabled, name	Checks whether the specified proxy variable has been set. Example: `<VariablePresent name="$userid"/>`
RequestCookiePresent	id, enabled, name	Checks the presence of the specified cookie in request Example: `<RequestCookiePresent name="SESSIONID"/>`
ResponseCookiePresent	id, enabled, name	Checks the presence of the specified cookie in response Example: `<ResponseCookiePresent name="MCWUSER"/>`
HeaderValue	id, enabled, name, value, mode, ignore-case	Checks whether the specified request/response header value matches the given value. The header name should be terminated by a colon (":"). Example: `<HeaderValue name="Rules-Result:" value="allow"/>`
ParamValue	id, enabled, name, value, mode, ignore-case	Checks whether the specified request parameter value matches the given value. Example: `<ParamValue name="cancel" value="Cancel"/>`

Table 5–2 (Cont.) Conditions Defined in an Interceptor

Condition name	Attributes	Description

QueryParamValue	id, enabled, name, value, mode, ignore-case	Checks whether the specified URL query parameter value matches the given value.
		Example:
		`<QueryParamValue name="requestID"`
		`value="Logout"/>`

VariableValue	id, enabled, name, value, mode, ignore-case	Checks whether the specified proxy variable value matches the given value.
		Example:
		`<VariableValue name="%REQUEST_METHOD"`
		`value="post"/>`

RequestCookieValue	id, enabled, name, value, mode, ignore-case	Checks whether the specified request cookie value matches the given value.
		Example:
		`<RequestCookieValue name="CurrentPage"`
		`value="/onlineserv/"`
		`mode="begins-with"`
		`ignore-case="true"/>`

ResponseCookieValue	id, enabled, name, value, mode, ignore-case	Checks whether the specified response cookie value matches the given value.
		Example:
		`<ResponseCookieValue name="CurrentPage"`
		`value="/onlineserv/"`
		`mode="begins-with"`
		`ignore-case="true"/>`

HttpStatus	id, enabled, status	Checks whether the status code of the response matches the given value. Example: `<HttpStatus status="302"/>`

HtmlElementPresent	id, enabled, name,
	attrib-name, attrib-value,
	attrib-name1, attrib-value1,
	…
	attrib-name9, attrib-value9,
		Checks presence of a html element to match the specified conditions:
		`<name attrib-name="attrib-value" attrib-name1="attrib-value1" …/>`
		Example:
		`<HtmlElementPresent name="form"`
		`attrib-name="name"`
		`attrib-value="signon"/>`

PageContainsText	id, enabled, text	Checks whether the response contains the given text. Example: `<PageContainsText text="You have entered an invalid Login Id"/>`

Table 5–2 (Cont.) Conditions Defined in an Interceptor

Condition name	Attributes	Description

NotVariableValue	id, enabled, name, value, mode, ignore-case	Checks whether the specified proxy variable value does not match the given value.
		Example:
		`<NotVariableValue name="$Login-Status"`
		`        value="In-Session"/>`

And	id, enabled	Evaluates to true only if all the child conditions evaluate to true.
		Example:
		`<And>`
		`  <PageContainsText text="Your password must be"/>`
		`  <PageContainsText text="Please re-enter your password"/>`
		`</And>`

Or	id, enabled	Evaluates to true if one of the child conditions evaluates to true.
		Example:
		`<Or>`
		`  <ParamValue name="register"`
		`        value="Continue"/>`
		`  <ParamValue name="cancel"`
		`        value="Cancel"/>`
		`</Or>`

Not	id, enabled	Reverses the result of the child condition(s).
		Example:
		`<Not>`
		`  <HttpStatus status="200"/>`
		`</Not>`

Attribute "id" is optional and is used only in trace messages. If no value is specified, the condition name (like HeaderPresent) will be used.

Attribute "enabled" is optional and the default value is "true". This attribute can be used to enable/disable a condition. The value of this attribute can be set to the name of a global variable; in such case, the condition will be enabled or disabled according to the value of the global variable.

Attribute "value" can be set to the name of a proxy variable. In such a case, the proxy will evaluate the variable at runtime and use that value in the condition.

Attribute "mode" can be set to one of the following: begins-with, ends-with, contains.

Attribute "ignore-case" can be set to one of the following: true, false.

5.6.3 Filters

Filters are used in the Oracle Adaptive Access Manager Proxy to modify HTTP request/response contents or modify the state information saved in the proxy (variables). Filters are executed in the order they are listed in the configuration file. Here is the list of filters that can be defined in an interceptor:

Table 5–3 Filters Defined in an Interceptor

Filter name	Attributes	Description
AddHeader	id, enabled, name, value	Adds the specified header with a given value to request/response. The header name should be terminated by a colon (":"). Example: <AddHeader name="userid:" value="$userid"/>
SaveHeader	id, enabled, name, variable	Saves the specified request/response header value in the given proxy variable. The header name should be terminated by a colon (":"). Example: <SaveHeader name="userid:" variable="$userid"/>
RemoveHeader	id, enabled, name	Removes the specified header from request/response. The header name should be terminated by a colon (":"). Example: <RemoveHeader name="InternalHeader:"/>
AddParam	id, enabled, name, value	Adds a request parameter with a specified name and value. Example: <AddParam name="loginID" value="$userid"/>
SaveParam	id, enabled, name, variable	Saves the specified request parameter value in to the given proxy variable. Example: <SaveParam name="loginID" variable="$userid"/>
AddRequestCookie	id, enabled, name, value	Adds the specified cookie with a given value to request Example: <AddRequestCookie name="JSESSIONID" value="$JSESSIONID"/>
SaveRequestCookie	id, enabled, name	Saves the specified request cookie value in the given proxy variable
AddResponseCookie	id, enabled, name	Adds the specified cookie with a given value to response Example: <AddResponseCookie name="JSESSIONID" value="$JSESSIONID"/>
SaveResponseCookie	id, enabled, name	Saves the specified response cookie value in the given proxy variable. Example: <SaveResponseCookie name="JSESSIONID" variable="$JSESSIONID"/>

Table 5–3 (Cont.) Filters Defined in an Interceptor

Filter name	Attributes	Description

SaveHiddenFields	id, enabled, form, variable, save-submit-fields	Saves all the hidden, submit fields value, in the given form if form name is specified to the given proxy variable. To not save submit fields, set save-submit-fields attribute to false.
		Example:
		`<SaveHiddenFields form="pageForm"`
		`variable="%lg_HiddenParams"/>`

AddHiddenFieldsParams	id, enabled, variable	Adds request parameters for each hidden field saved in the variable.
		Example:
		`<AddHiddenFieldsParams`
		`variable="%lg_HiddenParams"/>`

SetVariable	id, enabled, name, value	Sets the proxy variable with the given name to the specified value.
		Example:
		`<SetVariable name="$Login-Status"`
		`value="In-Session"/>`

UnsetVariable	id, enabled, name	Removes the proxy variable with the given name. Example: `<UnsetVariable name="$Login-Status"/>`
ClearSession	id, enabled, name	Removes all session variables in the current session. Example: `<ClearSession/>`
SaveQueryParam	id, enabled, name, variable	Saves the specified query parameter in the given proxy variable. Example: `<SaveQueryParam name="search" variable="$search"/>`
SaveRequest	id, enabled, variable	Saves the entire request content in the given proxy variable. Example: `<SaveRequest variable="$billPayRequest"/>`

Table 5–3 (Cont.) Filters Defined in an Interceptor

Filter name	Attributes	Description
SaveResponse	id, enabled, variable	Saves the entire response content in the given proxy variable. Example: <SaveResponse variable="$BillPay-Response"/>
ReplaceText	id, enabled, find, replace	Updates the response by replacing the text specified in "find" attribute with the value given in "replace" attribute. Example: <ReplaceText find="string-to-find" replace="string-to-replace"/>
ProcessString	id, enabled, source, find, action, count, search-str, start-tag, end-tag, ignore-case, replace	This filter can be used to extract a sub-string from a string (like request, response contents) and save it to a proxy variable. This filter can also be used to dynamically format strings. Please see the examples below on how to use this filter.

5.6.4 Filter Examples - ProcessString

Find the sub-string between the given start-tag and end-tag in the source string, extract the sub-string found and save extracted sub-string in the given variable.

```
<ProcessString source="%RESPONSE_CONTENT"
    find="sub-string"
    start-tag="var traceID = '" end-tag="';"
    action="extract"
    variable="$TRACE_ID"/>
```

Find the given search-string in the source string, replace it with the replace string and save the updated string in the given variable.

```
<ProcessString
      source="/bfb/accounts/accounts.asp?TraceID=$TRACE_ID"
      find="string" search-str="$TRACE_ID"
      action="replace"
      replace="$TRACE_ID"
      variable="%POST_URL"/>
```

Find the sub-string between the given start-tag and end-tag in the source string, replace it (including the start and end tags) with the evaluated value of the sub string found and save the updated string in the given variable.

```
<ProcessString
         source="/cgi-bin/mcw055.cgi?TRANEXIT[$UrlSuffix]"
         find="sub-string" start-tag="[" end-tag="]"
         action="eval"
         variable="%LogoffUrl"/>
```

5.6.5 Actions

An interceptor can optionally perform one of the following actions after executing all the filters. No further interceptors will be attempted after executing an action.

redirect-client

Often the proxy would need to redirect the client to load another URL; redirect-client is the action to use in such cases. The proxy will send a 302 HTTP response to request the client to load the specified URL.

If the display-url attribute is specified in the interceptor, the proxy will send a HTTP 302 response to the browser to load the URL specified in display-url attribute. When the proxy receives this request, it will do a HTTP-GET on the server to get the URL specified in "url" attribute.

send-to-client

Often a response from the server would have to be saved in the proxy and sent to the client later after performing a few other HTTP requests; send-to-client is the action to use in such cases. The proxy will send the client the contents of specified variable.

If the display-url attribute is specified in the interceptor, the proxy will send a HTTP 302 response to the browser to load the URL specified in display-url attribute. When the proxy receives this request, it will send the response specified in the interceptor.

get-server

Sometimes the proxy would need to get a URL from the server; get-server is the action to use in such cases. The proxy will send a HTTP-GET request for the specified URL to the server.

If the display-url attribute is specified in the interceptor or if this action is specified in a response interceptor, the proxy will send a HTTP 302 response to the browser. When the proxy receives this request it will do a HTTP-GET on the server to get the URL specified in "url" attribute.

post-server

Sometimes the proxy would need to post to a URL in the server; post-server is the action to use in such cases. The proxy will send a HTTP-POST request for the specified URL to the server.

If display-url attribute is specified in the interceptor or if this action is specified in a response interceptor, the proxy will send a HTTP 302 response to the browser. When the proxy receives this request it will do a HTTP-POST to the server to the URL specified in "url" attribute.

send-to-server

In certain situations the request from client needs to be saved in the proxy and sent to the server later after performing a few other HTTP requests; send-to-server is the action to use in such cases. The proxy will send the contents of the specified variable to the server.

If display-url attribute is specified in the interceptor or if this action is specified in a response interceptor, the proxy will send a HTTP 302 response to the browser. When the proxy receives this request it will send the request specified in the interceptor to the server.

5.6.6 Variables

The proxy variables can store string data in the proxy memory. Variables can be used in conditions, filters and actions. For example, SaveHeader filter can be used to save the value a specific header in the given proxy variable. This variable value could later be used, for example, to add a parameter to the request. Variables can also be used in conditions to determine whether to execute an interceptor or not.

The proxy variables are of 3 types, depending upon the lifespan of the variable. The type of variable is determined by the first letter of the variable name, which can be one of: %, $, @.

All types of variables can be set using filters like SetVariable SaveHeader, SaveParam, SaveResponse, and so on.

All types of variables can be unset/deleted by UnsetVariable filter. ClearSession filter can be used to remove all session variables.

Request variables

Request variables - these variable names start with %. These variables are associated with the current request and are deleted at the completion of the current request. Request variables are used where the value is not needed across requests.

Session variables

Session variables - these variable names start with $. These variables are associated with the current proxy session and are deleted when the proxy session is cleaned up. Session variables are used where the value should be preserved across requests from a client.

Global variables

Global variables - these variable names start with @. These variables associated with the current proxy configuration and are deleted when the proxy configuration is unloaded. Global variables are used where the value needs to be preserved across requests and across clients.

Global variables can be set at the proxy configuration load time using SetGlobal in the configuration file. Global variables can also be set by adding registry values under key HKLM\Software\Bharosa\Proxy\Globals. Name of each entry under this key should be the variable name, starting with @. And the data of the entry should be the value of the variable. The registry-type of the value can be REG_DWORD, REG_SZ or REG_EXPAND_SZ.

Pre-defined variables

The Oracle Adaptive Access Manager Proxy supports the following pre-defined request variables:

Table 5–4   Pre-defined Variables Supported by the Proxy

Variable name	Description
%RESPONSE_CONTENT	This variable contains the contents of the entire response from the Web server for the current request.
%REQUEST_CONTENT	This variable contains the contents of the entire request from the client.
%QUERY_STRING	This variable contains the query string, starting with ?, for the current request URL.
%REQUEST_METHOD	HTTP method verb for the request: GET, POST, etc
%REMOTE_HOST	Hostname of the client or agent of the client
%REMOTE_ADDR	IP address of the client or agent of the client
%HTTP_HOST	The content of HTTP Host header
%URL	URL for the current request

5.6.7 Application

A single Oracle Adaptive Access Manager Proxy installation can be used to provide multi-factor authentication for multiple Web application that run in one or more Web servers. In the Oracle Adaptive Access Manager Proxy configuration, an application is a grouping of interceptors defined for a single Web application.

Request and response interceptors can be defined outside of an application in the Oracle Adaptive Access Manager Proxy configuration file. These interceptors are called "global" interceptors and will be evaluated and executed prior to interceptors defined in applications.

5.7 Interception process

When a request arrives, the Oracle Adaptive Access Manager Proxy evaluates request interceptors defined for the URL in the order they are defined in the configuration file. Similarly when on receiving response from the Web server, the Oracle Adaptive Access Manager Proxy evaluates response interceptors defined for the URL in the order defined in the configuration file.

If the conditions in an interceptor evaluate to true, the Oracle Adaptive Access Manager Proxy will execute that interceptor, that is, execute the filters and action. After executing an interceptor, the Oracle Adaptive Access Manager Proxy will continue with the next interceptor only if the following conditions are met:

- no action is specified for the current interceptor
- post-exec-action attribute for the current interceptor is continue

Even if one of the above conditions is not met the Oracle Adaptive Access Manager Proxy will stop evaluating subsequent interceptors.

It is highly recommended that "post-exec-action" attribute is specified for interceptors that do not define an action. For global interceptors (for example, the interceptors defined outside of any application), the default value of "post-exec-action" attribute is continue. For non-global interceptors, the default value is stop-intercept.

As mentioned earlier the Oracle Adaptive Access Manager Proxy configuration can contain multiple applications. While finding the list of interceptors to evaluate for a URL, only the following interceptors are considered:

- global interceptors that are defined outside of any application
- interceptors defined in the application associated with the current session

Each session will be associated with at most one application. If no application is associated with the current session (yet) when the proxy finds an interceptor in an application for the URL, it will associate the application with the current session.

If the current session already has an application associated, and if no interceptor is found in that application for the URL, the proxy will then look for intercepts in other applications. If an interceptor is found in another application for the URL, a new session will be created and the request will be associated with the new session.

5.8 Adaptive Strong Authenticator Interface

The Oracle Adaptive Access Manager Proxy redirects the user to Adaptive Strong Authenticator pages at appropriate times, for example to collect the password using the Adaptive Strong Authenticator, to run risk rules, and so on. HTTP headers are used to exchange data between the Oracle Adaptive Access Manager Proxy and

Adaptive Strong Authenticator. The following table lists the Adaptive Strong Authenticator pages referenced in the proxy configuration along with the details of HTTP headers used to pass data. It also lists the expected action to be taken by the proxy on the given conditions.

Table 5–5 Adaptive Strong Authenticator Interface

URL	Condition	Action
Any request to Adaptive Strong Authenticator page	On receiving request	Set header "BharosaAppId". Adaptive Strong Authenticator will use this header value to select appropriate customizations (UI, rules, etc).
loginPage.jsp or login.do	On receiving request to application login page	Redirect to this URL to use the Oracle Adaptive Access Manager login page instead of the application's login page.
password.do	Response contains headers userid, password (could be more depending upon the application)	Save the credentials from the response headers and post to the application
login.do	Phase-1 only: After validating the credentials entered by the user.	Redirect to this URL to update the status in the Adaptive Risk Manager and run appropriate risk rules.
login.do	Phase-1 only: On receiving the request.	Set "userid" header to the userid entered by the user. Set "Login-Status" header to one of the following: success, wrong_password, invalid_user, user_disabled, system_error. Set "Adaptive Strong AuthenticatorPhase" header to "one".
updateLoginStatus.do	Phase-2 only: After validating the credentials entered by the user.	Redirect to this URL to update the status in Adaptive Risk Manager and run appropriate risk rules
updateLoginStatus.do	Phase-2 only: On receiving request	Set "Login-Status" header to one of the following: success, wrong_password, invalid_user, user_disabled, system_error
updateLoginStatus.do challengeUser.do registerQuestions.do userPreferencesDone.do	Response header "Rules-Result" has value "allow"	The Oracle Adaptive Access Manager rules evaluated to permit the login. The proxy can permit access to the protected application URLs after this point.
updateLoginStatus.do challengeUser.do registerQuestions.do userPreferencesDone.do	Response header "Rules-Result" has value "block"	Either the application did not accept the login credentials or the Oracle Adaptive Access Manager rules evaluated to block the login. The proxy should logoff the session in the application, if login was successful. Then a login blocked message should be sent to the browser.
changePassword.do	Response contains headers "password", "newpassword" and "confirmpassword"	Save the passwords from the response headers and post to the application
loginFail.do	To display error message in Adaptive Strong Authenticator page, like to display login blocked message	Redirect to this URL with appropriate "action" query parameter, like loginFail.do?action=block

Table 5–5 (Cont.) Adaptive Strong Authenticator Interface

URL	Condition	Action
logout.do	On completion of application session logout	Redirect to this URL to logout Adaptive Strong Authenticator session
logout.do	On receiving response	Redirect to application logout URL to logoff application session, if it is not done already
resetPassword.do	Response contains headers "newpassword" and "confirmpassword"	Save the passwords from the response headers and post to the application
getUserInput.do	Response contains headers "BH_UserInput"	Save the user input and take appropriate action (like post to application, etc)
changeUserId.do	On receiving request	Add "newUserId" header
changeUserId.do	On receiving response	Redirect to appropriate application page or send back saved application response
updateForgotPasswordStatus.do	Phase-2 only: After validating the forgot-password-credentials entered by the user.	Redirect to this URL to update the status in Adaptive Risk Manager and run appropriate risk rules.
updateForgotPasswordStatus.do	Phase-2 only: On receiving request	Set "BH_FP-Status" header to one of the following: success, wrong_password, invalid_user, user_disabled, system_error.
updateForgotPasswordStatus.do challengeForgotPasswordUser.do	Response header "BH_FP-Rules-Result" has value "allow"	The Oracle Adaptive Access Manager rules evaluated to permit the forgot-password flow. The proxy can permit continuation of to forgot-password flow, perhaps to reset the password or allow the user login, depending on the application.
updateForgotPasswordStatus.do challengeForgotPasswordUser.do	Response header "BH_FP-Rules-Result" has value "block"	Either the application did not accept the forgot-password credentials or the Oracle Adaptive Access Manager rules evaluated to block the forgot-password flow. A login blocked message should be sent to the browser.
Any request to Adaptive Strong Authenticator page	If the proxy needs to get a property value from Adaptive Strong Authenticator. On receiving request	"BH_PropKeys" request header should be set to list of property names (separated by comma). Adaptive Strong Authenticator will return the values in multiple response headers, one for each property. The return response header names will be of format: "BH_Property-<name>"

5.9 Application Discovery

Application discovery is the process of studying an existing Web application to author the proxy configuration to add multi-factor authentication using the Oracle Adaptive Access Manager UIO. Few logins attempts to the application would be made via the proxy to capture the HTTP traffic in each attempt. The captured HTTP traffic would be then be analyzed to author the proxy configuration. The Oracle Adaptive Access Manager Proxy should be set up to dump all the HTTP traffic through it to a file. Then a few logins/login attempts to the application should be made via the proxy. The captured HTTP traffic should be then be analyzed to author the proxy configuration.

5.9.1 Application Information

For application discovery process it is preferable to work with the Web application in customer's test environment, rather than the live application being used by users. If the test environment is not available for some reason, the live application can be used.

The following information is needed from the client for the application discovery process:

1. URL to login to the application.
2. Test user account credentials, including the data required in forgot password scenario. It will be best to get as many test accounts as possible, preferably at least 5 accounts, for uninterrupted discovery and testing. Please note that during discovery process some accounts could become disabled, perhaps due to multiple invalid login attempts.
3. Contact (phone, email) to enable/reset test accounts

5.9.2 Setting Up the Oracle Adaptive Access Manager Proxy for Microsoft ISA

The Microsoft ISA server should be set up to publish the Web application under discovery, that is, creating a Web site publishing rule with appropriate parameters. During the application discovery process, the application will be accessed via Microsoft ISA, which hosts the Oracle Adaptive Access Manager Proxy for Microsoft ISA. Please refer to the Microsoft ISA configuration document for details of setting up Microsoft ISA.

The Oracle Adaptive Access Manager Proxy for Microsoft ISA settings (registry values under HKLM\SOFTWARE\Bharosa\Proxy key) should be set as given below for the proxy to capture the HTTP traffic to the specified file. This HTTP traffic captured will later be used for analysis to author the proxy configuration.

Table 5–6 Setting up the proxy

Setting	Value
IgnoreUrlMappings	1
CaptureTraffic	1
TraceFilename	<filename>
TraceLevel	0x87
TraceToFile	1

It might be useful to capture the HTTP traffic for each scenario (like successful login attempt, wrong password, wrong username, disabled user, etc) in separate files. TraceFilename setting should be updated to the desired filename before start of the scenario.

After application discovery is done, the proxy settings should be set as given below to restore the default Oracle Adaptive Access Manager Proxy for Microsoft ISA behavior.

Table 5–7 Proxy settings after application discovery

Setting	Value
IgnoreUrlMappings	0
CaptureTraffic	0
TraceFilename	<filename>

Table 5–7 (Cont.) Proxy settings after application discovery

Setting	Value
TraceLevel	0x7
TraceToFile	1

5.9.3 Setting Up the Oracle Adaptive Access Manager Proxy for Apache

For application discovery, the HTTP traffic needs to be captured through the proxy.

The following table shows the settings (in UIO_Settings.xml) to enable this mode of operation.

Table 5–8 Settings for Capturing HTTP

Settings	Value
IgnoreUrlMappings	1
CaptureTraffic	1

The IgnoreUrlMappings setting is used to disable URL interception of the HTTP traffic through the proxy.

The CaptureTraffic setting captures the HTTP traffic through the logger name http set to log level of info.

It might be useful to capture the HTTP traffic for each scenario (like successful login attempt, wrong password, wrong username, disabled user, and so on) in separate files. The log file name setting should be updated to the desired filename before the start of the scenario.

After application discovery is performed, the proxy settings should be set, as shown below, to restore the default Oracle Adaptive Access Manager Proxy for Apache behavior.

Table 5–9 Settings to restore detault proxy behavior

Settings	Value
IgnoreUrlMappings	0
CaptureTraffic	0

5.9.4 Scenarios

Information should be collected for the following scenarios during the discovery process:

Login

1. URL that starts the login process
2. URL that contains the login form
3. Names of the input fields like username, password used to submit the credentials
4. URL to which the login form submits the credentials
5. Identifying successful login. The HTTP traffic dump needs to be studied carefully to derive this information. Here are few ways applications respond on successful login:

   a. by setting a specific cookie in the credential submit response
   b. by redirecting to a specific URL (like account summary, welcome page)
   c. by responding with specific text

6. Identifying failure login with the reason for failure. This would often be derived by looking for certain text in the response to credential submit request.

Logout

1. URL that starts the logout process
2. URL that completes the logout process. In most cases the logout completes on receiving response to the logout start URL.

Change password

1. URL that starts the change password process
2. URL that contains the change password form
3. Names of the input fields like password, new-password, confirm-password used to submit the change password request
4. URL to which the change password form submits the passwords
5. Identifying the status (success/failure) of the change password request. This would often be derived by looking for certain text in the response.

Reset password

Follow the same process as above for Change password.

Change LoginId

1. URL to which the login-id change is posted to the application
2. Names of the input fields like new-login used to submit the change password request.
3. Identifying the status (success/failure) of the change login-id request. On successful change login-id request, changeUserId.do page in Adaptive Strong Authenticator should be called to update the login-id in the Oracle Adaptive Access Manager database.

Forgot password

Forgot-password options provided by the application should first be understood. Most applications ask for alternate ways to identity the user (account number/pin, SSN/pin, question/answer, etc); some applications provide more than one option. Some applications let the user reset the password on successfully entering alternate credentials; others send a new password to the user by mail/email; and some other applications would require the user to call customer care. For each of the supported scenarios, the following data should be captured:

1. URL that starts the forgot-password process
2. URL that contains the forgot-password form
3. Names of the input fields and URLs to submit the forgot-password request
4. Identifying the status (success/failure) of the forgot-password request.

5.10 Samples

The Oracle Adaptive Access Manager Proxy configuration to add multi-factor authenticator to BigBank Web application is listed below.

For ISA proxy use:

```
<?xml version="1.0" encoding="utf-8"?>
<BharosaProxyConfig xmlns="http://bharosa.com/"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://bharosa.com/ BharosaProxy.xsd ">
```

For Apache proxy use:

```
<?xml version="1.0" encoding="utf-8"?>
<BharosaProxyConfig xmlns="http://bharosa.com/">


  <Application id="BigBank">

    <RequestInterceptor id="AddAppIdTobharosauioRequests"
           desc="Add BharosaAppId header to each request to bharosauio"
           post-exec-action="continue">
      <Conditions>
        <VariableValue name="%URL"
                       value="/bharosauio/"
                       mode="begins-with"
                       ignore-case="true"/>
      </Conditions>

      <Filters>
        <AddHeader name="BharosaAppId:" value="BigBank"/>
      </Filters>
    </RequestInterceptor>

    <!-- Phase-1: Use BigBank login form to collect credentials -->
    <!-- Phase-2: Use BharosaUIO login forms to collect credentials -->

    <!-- Disable this interceptor after phase one is retired -->
    <RequestInterceptor id="Phase1BigBankLoginPostRequest"
                        desc="get the loginid from the post parameters"
                        post-exec-action="continue" enabled="true">
      <RequestUrl url="/bigbank/login.do"/>

      <Conditions>
        <VariableValue name="%REQUEST_METHOD" value="post"/>
      </Conditions>

      <Filters>
        <ClearSession/>
        <SetVariable name="$WebUIOPhase" value="one"/>
        <SaveParam   name="userid"       variable="$userid"/>
      </Filters>
    </RequestInterceptor>

    <!-- Enable this interceptor after phase one is retired -->
    <RequestInterceptor id="Phase2RedirectBigBankLoginPageRequest"
          desc="Redirect BigBank login page requests to UIO login page"
          enabled="false">
      <RequestUrl url="/bigbank"/>
      <RequestUrl url="/bigbank/"/>
      <RequestUrl url="/bigbank/loginPage.jsp"/>

      <Target action="redirect-client" url="/bharosauio/login.do"/>
    </RequestInterceptor>

    <RequestInterceptor id="Phase2BharosaLoginPageRequest"
                        desc="Phase-2 loginid post request"
                        post-exec-action="continue">
      <RequestUrl url="/bharosauio/login.do"/>

      <Conditions>
        <VariableValue name="%REQUEST_METHOD" value="post"/>
        <ParamPresent  name="userid"/>
        <Not>
          <ParamPresent name="password"/>
        </Not>
      </Conditions>

      <Filters>
        <ClearSession/>
        <SetVariable name="$WebUIOPhase" value="two"/>
      </Filters>
    </RequestInterceptor>

    <ResponseInterceptor id="Phase2PassowrdPageResponse"
           desc="Save userid, decoded password from Bharosa response">
      <ResponseUrl url="/bharosauio/password.do"/>

      <Conditions>
        <HeaderPresent name="userid:"/>
        <HeaderPresent name="password:"/>
      </Conditions>

      <Filters>
        <SaveHeader name="userid:"   variable="$userid"/>
        <SaveHeader name="password:" variable="$password"/>
      </Filters>

      <Target action="redirect-client"
              url="/bigbank/login.do"
              display-url="/bigbank/GetLoginPage"/>
    </ResponseInterceptor>

    <ResponseInterceptor id="GetBigBankLoginPageResponse"
             desc="Save hidden fields; then post login crdentials">
      <ResponseUrl url="/bigbank/GetLoginPage"/>

      <Filters>
        <SaveHiddenFields variable="%LoginPageHiddenParams"/>

        <AddHiddenFieldsParams variable="%LoginPageHiddenParams"/>
        <AddParam              name="userid"   value="$userid"/>
        <AddParam              name="password" value="$password"/>

        <UnsetVariable name="$password"/>
      </Filters>

      <Target action="post-server" url="/bigbank/login.do"/>
    </ResponseInterceptor>
```

```
<ResponseInterceptor id="InvalidLoginResponse"
                     desc="Invalid login response from BigBank">
  <ResponseUrl url="/bigbank/login.do"/>

  <Conditions>
    <PageContainsText text="You have entered an invalid Login Id"/>
  </Conditions>

  <Filters>
    <SetVariable  name="$Login-Credentials-Status"
                  value="invalid_user"/>
    <SetVariable  name="$Login-Continue-Url"
                  value="%URL"/>
    <SaveResponse variable="$Submit-Credentials-Response"/>
  </Filters>

  <Target action="redirect-client"
          url="/bharosauio/UpdateLoginStatusPage"/>
</ResponseInterceptor>

<ResponseInterceptor id="WrongPasswordResponse"
                     desc="Invalid login response from BigBank">
  <ResponseUrl url="/bigbank/login.do"/>

  <Conditions>
    <PageContainsText text="We do not recognize your password"/>
  </Conditions>

  <Filters>
    <SetVariable name="$Login-Credentials-Status"
                 value="wrong_password"/>
    <SetVariable name="$Login-Continue-Url"
                 value="%URL"/>
    <SaveResponse variable="$Submit-Credentials-Response"/>
  </Filters>

  <Target action="redirect-client"
          url="/bharosauio/UpdateLoginStatusPage"/>
</ResponseInterceptor>

<ResponseInterceptor id="LoginSuccessResponse"
                     desc="Login success response from BigBank">
  <ResponseUrl url="/bigbank/activity.do"/>
  <ResponseUrl url="/bigbank/login.do"/>

  <Conditions>
    <NotVariableValue name="$Login-Status" value="In-Session"/>
    <PageContainsText text="/bigbank/images/success.gif"/>
  </Conditions>

  <Filters>
    <SetVariable name="$Login-Credentials-Status" value="success"/>
    <SetVariable name="$Login-Continue-Url"       value="%URL"/>
    <SaveResponse variable="$Submit-Credentials-Response"/>
  </Filters>

  <Target action="redirect-client"
          url="/bharosauio/UpdateLoginStatusPage"/>
</ResponseInterceptor>
```

```
    <RequestInterceptor id="Phase1UpdateLoginStatusPageRequest"
                 desc="Update Bharosa Tracker with the login status">
      <RequestUrl url="/bharosauio/UpdateLoginStatusPage"/>

      <Conditions>
        <VariableValue name="$WebUIOPhase" value="one"/>
      </Conditions>

      <Filters>
        <AddHeader name="WebUIOPhase:"  value="$WebUIOPhase"/>
        <AddHeader name="userid:"       value="$userid"/>
        <AddHeader name="Login-Status:"
                   value="$Login-Credentials-Status"/>
      </Filters>

      <!-- Any interceptors for /bigbank/login.do will not run because we are
doing get-server. -->
      <Target action="get-server" url="/bharosauio/login.do"/>
    </RequestInterceptor>

    <RequestInterceptor id="Phase2UpdateLoginStatusPageRequest"
                 desc="Update Bharosa Tracker with the login status">
      <RequestUrl url="/bharosauio/UpdateLoginStatusPage"/>

      <Filters>
        <AddHeader name="Login-Status:"
                   value="$Login-Credentials-Status"/>
      </Filters>

      <Target action="get-server"
              url="/bharosauio/updateLoginStatus.do"/>
    </RequestInterceptor>

    <ResponseInterceptor id="AllowLoginResponse"
              desc="Tracker returned 'allow' - continue with login">
      <ResponseUrl url="/bharosauio/UpdateLoginStatusPage"/>
      <ResponseUrl url="/bharosauio/updateLoginStatus.do"/>
      <ResponseUrl url="/bharosauio/challengeUser.do"/>
      <ResponseUrl url="/bharosauio/registerQuestions.do"/>
      <ResponseUrl url="/bharosauio/userPreferencesDone.do"/>

      <Conditions>
        <HeaderValue name="Rules-Result:" value="allow"/>
      </Conditions>

      <Filters>
        <SetVariable name="$Login-Status" value="In-Session"/>
      </Filters>

      <Target action="send-to-client"
              html="$Submit-Credentials-Response"
              display-url="$Login-Continue-Url"/>
    </ResponseInterceptor>

    <ResponseInterceptor id="Phase1FailLoginResponse"
                      desc="BigBank failed the login">
      <ResponseUrl url="/bharosauio/UpdateLoginStatusPage"/>
      <ResponseUrl url="/bharosauio/updateLoginStatus.do"/>
      <ResponseUrl url="/bharosauio/challengeUser.do"/>
      <ResponseUrl url="/bharosauio/registerQuestions.do"/>
```

```
        <ResponseUrl url="/bharosauio/userPreferencesDone.do"/>

        <Conditions>
          <VariableValue name="$WebUIOPhase" value="one"/>
          <NotVariableValue name="$Login-Credentials-Status"
                            value="success"/>
          <HeaderValue name="Rules-Result:" value="block"/>
        </Conditions>

        <Filters>
          <UnsetVariable name="$Login-Status"/>
        </Filters>

        <Target action="send-to-client"
                html="$Submit-Credentials-Response"
                display-url="$Login-Continue-Url"/>
      </ResponseInterceptor>

      <ResponseInterceptor id="FailLoginResponse"
                           desc="BigBank failed the login">
        <ResponseUrl url="/bharosauio/UpdateLoginStatusPage"/>
        <ResponseUrl url="/bharosauio/updateLoginStatus.do"/>
        <ResponseUrl url="/bharosauio/challengeUser.do"/>
        <ResponseUrl url="/bharosauio/registerQuestions.do"/>
        <ResponseUrl url="/bharosauio/userPreferencesDone.do"/>

        <Conditions>
          <HeaderValue name="Rules-Result:" value="block"/>
          <NotVariableValue name="$Login-Credentials-Status"
                            value="success"/>
        </Conditions>

        <Filters>
          <UnsetVariable name="$Login-Status"/>
        </Filters>

        <Target action="redirect-client"
                url="/bharosauio/loginPage.jsp?action=invalid_user"/>
      </ResponseInterceptor>

      <ResponseInterceptor id="BlockLoginResponse"
                desc="BigBank passed login but tracker returned 'block'">
        <ResponseUrl url="/bharosauio/UpdateLoginStatusPage"/>
        <ResponseUrl url="/bharosauio/updateLoginStatus.do"/>
        <ResponseUrl url="/bharosauio/challengeUser.do"/>
        <ResponseUrl url="/bharosauio/registerQuestions.do"/>
        <ResponseUrl url="/bharosauio/userPreferencesDone.do"/>

        <Conditions>
          <HeaderValue name="Rules-Result:" value="block"/>
        </Conditions>

        <Filters>
          <UnsetVariable name="$Login-Status"/>
        </Filters>

        <!-- /bigbank/LoginBlockedPage isn't a real page.  The request will be
intercepted and redirected. -->
        <Target action="redirect-client" url="/bigbank/LoginBlockedPage"/>
      </ResponseInterceptor>
```

```
<RequestInterceptor id="LoginBlockedPageRequest"
                    desc="logoff the session in BigBank">
  <RequestUrl url="/bigbank/LoginBlockedPage"/>

  <Target action="get-server" url="/bigbank/logout.do"/>
</RequestInterceptor>

<ResponseInterceptor id="Phase1LoginBlockedPageResponse"
       desc="BigBank approved; but Bharosa blocked the login"
       post-exec-action="stop-intercept">
  <ResponseUrl url="/bigbank/LoginBlockedPage"/>

  <Conditions>
    <VariableValue name="$WebUIOPhase" value="one"/>
  </Conditions>

  <Filters>
    <ClearSession/>
  </Filters>

  <Target action="redirect-client"
          url="/bharosauio/loginFail.do?action=block"/>
</ResponseInterceptor>

<ResponseInterceptor id="Phase2LoginBlockedPageResponse"
      desc="BigBank approved; but Bharosa blocked the login">
  <ResponseUrl url="/bigbank/LoginBlockedPage"/>

  <Filters>
    <ClearSession/>
  </Filters>

  <Target action="redirect-client"
          url="/bharosauio/loginPage.jsp?action=block"/>
</ResponseInterceptor>

<ResponseInterceptor id="LogoutPageResponse"
       desc="Bharosa logout selected; logoff BigBank session ">
  <ResponseUrl url="/bharosauio/logout.do"/>

  <Target action="redirect-client" url="/bigbank/logout.do"/>
</ResponseInterceptor>

<ResponseInterceptor id="Phase1LogoffPageResponse"
                     desc="Logoff - clear Bharosa proxy session"
                     post-exec-action="stop-intercept">
  <ResponseUrl url="/bigbank/logout.do"/>

  <Conditions>
    <VariableValue name="$WebUIOPhase" value="one"/>
  </Conditions>

  <Filters>
    <ClearSession/>
  </Filters>
</ResponseInterceptor>

<ResponseInterceptor id="Phase2LogoffPageResponse"
                     desc="Logoff - clear Bharosa proxy session">
      <ResponseUrl url="/bigbank/logout.do"/>

      <Filters>
        <ClearSession/>
      </Filters>

      <Target action="redirect-client"
              url="/bharosauio/loginPage.jsp"/>
    </ResponseInterceptor>
  </Application>
</BharosaProxyConfig>
```

5.11 Troubleshooting

This section covers common troubleshooting issues and tips to resolve them.

Microsoft ISA

For Proxy Web publishing:

- .Net2.0 Framework should be installed and enabled to successfully register the Bharosa Proxy DLL.
- Ensure the database access credentials are correct when the Firewall logging properties in Microsoft ISA use SQL Database as the Log Storage Format.
- Define IP Exceptions for Trusted IPs (like Router IP) when Flood Mitigation settings are enabled to mitigate flood attacks and worm propagation.

For Proxy configuring, enable tracing to file and set the trace level to 0x8008f. This will print detailed interceptor evaluation and execution information to the log file.

Apache

- On launching httpd, it gives an error for loading mod_uio.so. Ensure that mod_uio.so and all the libraries are placed in the proper directories. On Linux, use the 'ldd' command to confirm that mod_uio.so can load all the dynamic libraries that it depends upon. On Windows, use Dependency Walker to find out any missing DLLs and in some cases, you may have to install the "Microsoft Visual C++ 2005 Redistributable Package" from the Microsoft Web site, if your server does not have these libraries pre-installed.
- Nothing is working - no logs, and so on. Ensure that the user of httpd has permissions to read the uio directory. Typically httpd is run as a daemon user. Ensure the daemon user has write permissions for the logs directory.
- In case of a parsing error in UIO_Settings.xml or any configuration XML, an error log will be created in httpd's logs directory with the name UIO_Settings.xml.log.
- For errors, look in uio.log. Use log level of error for production use; info for more details; debug for debugging issues and trace for verbose logs.
- Ensure that the config XML and settings XML are conforming to the RNG schema. You can use the UIO_Settings.rng and UIO_Config.rng in any XML editor to edit the UIO_Settings.xml and application configuration XML files.
- You can change the Apache httpd log level to debug for testing, or keep it at info to reduce log file size. The Apache httpd log is separate from Oracle Adaptive Access Manager Proxy for Apache log.
- When migrating ISA config XML to be used with the Apache Universal Installation Option Proxy, you need to do the following:

1. Change the header of the XML file to use

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <BharosaProxyConfig xmlns="http://bharosa.com/">
   ```

2. Run your config XML file through libxml2's xmllint utility.

   For Windows, download from http://www.zlatkovic.com/pub/libxml/ the latest libxml2-2.x.x.win32.zip file and unzip it.

   For Linux, if you have libxml2 installed then xmllint command should be available, or check with your Linux System Administrator.

   Copy the UIO_Config.rng file from the Apache UIO distribution and run following command:

   ```
   xmllint --noout --relaxng UIO_Config.rng <your config xml file>
   ```

   And fix any errors that are reported.

- The Oracle Adaptive Access Manager Proxy for Apache is not working or intercepting request.

  Problem: The following error appears:

  ```
  Failed to create session in memcached, err = 70015(Could not find specified
  socket in poll list.) proxy - Failed to create session, cannot process this
  request distsessions - memcache server localhost create failed 111
  ```

  Possible Solutions:

  - Make sure "memcache" is installed and configured.
  - Make sure "memcache" process is up and running before creating the session.

6

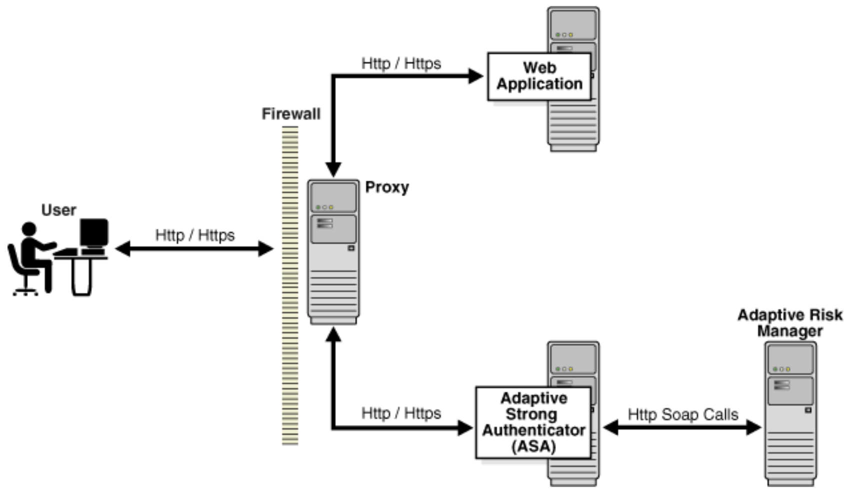
Configuring Adaptive Strong Authenticator

This chapter provides information on customizing the client-facing Adaptive Strong Authenticator Web application. The Oracle Adaptive Access Manager's Universal Installation Option (UIO) offers multi-factor authentication to Web applications without requiring any change to the application code. The Adaptive Strong Authenticator configuration is specific to the UIO deployment. Please refer to the architectural diagram below for the components involved.

The user interface provided by the Adaptive Strong Authenticator Web application can be easily customized to achieve the look-n-feel of the customer applications. This chapter is intended for integrators who install and configure Adaptive Strong Authenticator to support one or more Web application authentication and user registration flows.

6.1 Architecture

The following diagram shows an Adaptive Risk Manager UIO deployment.



The Adaptive Strong Authenticator proxy intercepts the HTTP traffic between the client (browser) and the server (Web application) and performs appropriate actions, such as redirecting to the Adaptive Strong Authenticator, to provide multi-factor authentication and authorization. The Adaptive Strong Authenticator in turn communicates with Adaptive Risk Manager to assess the risk and takes the appropriate actions, such as permitting the login, challenging the user, blocking the user, and other actions.

6.2 Setting Adaptive Strong Authenticator Settings

The Adaptive Strong Authenticator configuration is controlled through property files.

Configuration Files

The property files used to configure the Adaptive Strong Authenticator are listed below:

- The bharosauio_client.properties file contains the client-configured properties (any properties that have been customized for a specific deployment). These client-configured properties will override the default configurations contained in the bharosauio.properties file.
- The bharosauio.properties file contains the default UIO system /device configurations. The file deals with the structural changes in the overall application. It is where the header, footer, and CSS properties are located.
- The asa_msg_resource.properties file contains the default UIO messaging and page content configuration. For example, page titles, links at the bottom of the pages, page messages, error message, and confirmation messages.
- The client_resource_<locale>.properties file contains the client-configured properties that are configurable for each locale being supported. <locale> is the locale string for which you wish to use the custom values (en, es, etc). These client configured properties will override the default configurations contained in the asa_msg_resource.properties and asa_resource.properties files.

In the deployed application, these property files are located in the web-inf/classes directory.

6.3 First Steps

The first steps of Adaptive Strong Authenticator configuration and customization are:

1. Determine the application ID of each application being secured.
2. Assign default user groups for each application being secured.

6.3.1 Determining the Application ID

UIO can be placed in front of multiple applications, and customized to work with each one as required. Determine how many applications are going to be configured, assign each application an Application ID. This Application ID should be the same one used to configure the Proxy (see Chapter 5, "Oracle Adaptive Access Manager Proxy"). In many cases applications are referred to internally by some name or abbreviation, so a person configuring Adaptive Strong Authenticator might want to use that name. For an example if the client has two applications, one wholesale banking application and one retail banking application, the integrator might choose to use "wholesale" and "retail" as the Application IDs for the two applications.

The Proxy will send the AppId to the Adaptive Strong Authenticator as needed via HTTP header. This AppId is then used to determine which configuration is used when displaying pages to the client. Adaptive Strong Authenticator is configured by a set of properties which we will discuss in more detail later. An example of how AppId is used in a property definition is shown below.

```
bharosauio.appId1.default.user.group=app1Group
```

The bold "**appId1**" is the location in the property where the AppId is used to configure application specific values.

6.3.2 Determining Default User Groups

Each application can be configured to have a unique default user group. This is the group that a user of that application will be associated with as their primary user group when first created in the Adaptive Risk Manager Online database. Similarly, it will be the group used to attempt to load user information from the database when a user attempts to log in to the application.

As used in the previous example the property for default user group looks as follows:

```
bharosauio.appId1.default.user.group=app1Group
bharosauio.appId2.default.user.group=app2Group
```

In this case you can see that we have defined two user groups to two different applications. The application with an AppId of "appId 1" has been assigned the default user group of "app1Group" and the application with an AppId of "appId2" has been assigned the default user group of "app2Group".

6.4 Customizing User Interface Branding

The Adaptive Strong Authenticator user interface branding is customized in several ways.

- Custom header / footer files
- Custom CSS file
- Custom properties for page content and messaging

6.4.1 Custom Header / Footer

Adaptive Strong Authenticator provides the ability to create a custom header and / or footer file for applications being secured. The header and footer files are JSP and can contain any HTML or JSP code required to replicate the look of the application being secured. All the customer resources (JSP files, image files, HTML, and others) should be copied into the deployed application directories along with the Adaptive Strong Authenticator Web application.

The header (header.jsp) and footer (footer.jsp) files should contain only content html, all page related tags (<html>, <head>, <body>, etc) are already provided by the Adaptive Strong Authenticator. As a simple example we will create a header and footer that contain a single image each, to be used as the header and footer of an application called "appId1".

Copy the following code into a file called header.jsp for the header.

```
/client/app1/header.jsp
        <img src="/client/app1/images/header.jpg" alt="Welcome to App1"/>
```

Copy the following code into a file called footer.jsp for the footer.

```
/client/app1/footer.jsp
       <img src="/client/app1/images/footer.jsp" alt="App1 Footer"/>
```

These files will be housed in the "/client/app1/" directory within the Web application.

To associate these files with the application we would add the following properties to client_resource_<locale>.properties:

```
bharosa.uio.appId1.header = /client/app1/header.jsp
bharosa.uio.appId1.footer = /client/app1/footer.jsp
```

6.4.2 Custom CSS

The Adaptive Strong Authenticator styles are controlled through a single CSS file, bharosa_uio.css, located in the css directory. These styles can be overridden by including a custom CSS file. Much like the header and footer example above, you can create your own file and include that file on an application or global level through properties (see "How the Properties Work" in this document).

In this example we will override the font-family of the default body style definition.

The body style in bharosa_uio.css is defined as follows:

```
body{
    background-color:#ffffff;
    font-size:12px;
    color:#000000;
    font-family:arial,helvetica,sans-serif;
    margin:0px 0px 0px 0px
}
```

Now to use our newly created file, we will add the following property to bharosauio_ client.properties:

```
bharosa.uio.appId1.custom.css=/client/app1/css/app1.css
```

In this case, all we did was change helvetica to the primary font-family in our "appId1" application. Any style defined in bharosa_uio.css can be overridden in this manner if required.

6.4.3 Custom Content and Messaging

Adaptive Strong Authenticator pages have a variety of content and messaging sections. These sections can be customized by properties; the default values for these are found in asa_msg_resource.properties. Some customizable items, like page title and message, are applicable for each page. While other items, like login blocked message, are specific to a particular page.

To change the page title on the login page in our example "appId1" application, we would add the following line to client_resource_<locale>.properties. <locale> is the locale string for which you wish to use the custom values. (en, es, etc).

bharosa.uio.appId1.signon.page.title=Welcome to App1, please sign in.

Please refer to the asa_msg_resource.properties for additional properties.

The contents of error messages are also controlled in the same way. In the following example we will customize the error message displayed when a user has been blocked by security rules.

```
bharosa.uio.appId1.login.user.blocked = You are not authorized to login. Please
contact customer service at 1-888-555-1234.
```

6.5 How Properties Work

An application in Adaptive Strong Authenticator is made up of a grouping or set of properties. You can configure the Adaptive Strong Authenticator properties on a global or application specific level.

The Adaptive Strong Authenticator property names are prefixed with bharosa.uio. They are followed by the Application ID or "default" if the setting is global.

An "application-level" property is one that only effects a single application when there are more than one application defined in the properties.

For example,

```
# Global or Default header and footer definitions
# - Apply to all applications that do not specifically define their own
bharosa.uio.default.header = /globalHeader.jsp
bharosa.uio.default.footer = /globalFooter.jsp
# Application specific definitions
# - These values override the default settings
bharosa.uio.app1.header = /app1Header.jsp
bharosa.uio.app1.footer = /app1Footer.jsp
bharosa.uio.app2.footer = /app2Footer.jsp
```

In this example, app1 uses an "application-level" defined header and footer file, but app2 uses an "application-level" defined footer but a "global" or "default" defined header file.

The bharosa.uio.default.header property, shown below, defines the location of the header file.

```
bharosa.uio.default.header = /globalHeader.jsp
```

The property is used across all applications of the Adaptive Strong Authenticator installation unless the specific application has another location specified.

In the case shown above, "default" is used instead of the Application ID to designate the property as a global default. If the same property is not defined for an application; then, this value will be used.

6.5.1 Property Extension

In addition to configuring properties for each application, you can configure a set of properties that several applications have in common. You can then extend that set to customize the parameters that differ between the set of applications.

If you were to configure three applications that all use a single footer, but each has a unique header, you can include the following properties:

```
bharosa.uio.myAppGroup.footer = /myAppGroup/header.jsp

bharosa.uio.appId1.extends=myAppGroup
bharosa.uio.appId1.header=/client/app1/header.jsp

bharosa.uio.appId2.extends=myAppGroup
bharosa.uio.appId2.header==/client/app2/header.jsp

bharosa.uio.appId3.extends=myAppGroup
bharosa.uio.appId3.header==/client/app3/header.jsp
```

6.5.2 User-Defined Enums

User-defined enums are a collection of properties that represent a list of items. Each element in the list may contain several different attributes. The definition of a user-defined enum begins with a property ending in the keyword ".enum" and has a value describing the use of the user-defined enum. Each element definition then starts with the same property name as the enum, and adds on an element name and has a value of a unique integer as an ID. The attributes of the element follow the same pattern, beginning with the property name of the element, followed by the attribute name, with the appropriate value for that attribute.

The following is an example of an enum defining credentials displayed on the login screen of an Adaptive Strong Authenticator implementation:

```
bharosa.uio.default.credentials.enum = Enum for Login Credentials
bharosa.uio.default.credentials.enum.companyid=0
bharosa.uio.default.credentials.enum.companyid.name=CompanyID
bharosa.uio.default.credentials.enum.companyid.description=Company ID
bharosa.uio.default.credentials.enum.companyid.inputname=comapanyid
bharosa.uio.default.credentials.enum.companyid.maxlength=24
bharosa.uio.default.credentials.enum.companyid.order=0
bharosa.uio.default.credentials.enum.username=1
bharosa.uio.default.credentials.enum.username.name=Username
bharosa.uio.default.credentials.enum.username.description=Username
bharosa.uio.default.credentials.enum.username.inputname=userid
bharosa.uio.default.credentials.enum.username.maxlength=18
bharosa.uio.default.credentials.enum.username.order=1
```

This set of properties defines one user-defined enum that contains two elements, each of which with five attributes. The "name" and "description" attributes are required to define any user-defined enum, other attributes are defined and used as needed by each individual use of a user-defined enum.

6.5.3 Overriding Existing User-Defined Enums

Overriding existing user-defined enums has some special cases. You may override any existing enum element's attribute value of the default application ID just as you would any other property, but to change the value of an element's attribute in a single application using an appId, you must create the entire enum in that application using the appropriate appId.

For example, using the User Defined Enum defined above, if we wanted to change "Company ID" to "Profile ID" for only one application (appId1), we would need to do the following:

```
bharosa.uio.appId1.credentials.enum = Enum for Login Credentials
bharosa.uio.appId1.credentials.enum.profileid=0
bharosa.uio.appId1.credentials.enum.profileid.name=ProfileID
bharosa.uio.appId1.credentials.enum.profileid.description=Profile ID
bharosa.uio.appId1.credentials.enum.profileid.inputname=profileid
bharosa.uio.appId1.credentials.enum.profileid.maxlength=20
bharosa.uio.appId1.credentials.enum.profileid.order=0
bharosa.uio.appId1.credentials.enum.username=1
bharosa.uio.appId1.credentials.enum.username.name=Username
bharosa.uio.appId1.credentials.enum.username.description=Username
bharosa.uio.appId1.credentials.enum.username.inputname=userid
bharosa.uio.appId1.credentials.enum.username.maxlength=18
bharosa.uio.appId1.credentials.enum.username.order=1
```

6.5.4 Disabling Elements

To disable any already defined element in a user-defined enum, simply add an "enabled" attribute with a value of "false". Using the appId1 credentials enum from above, we would add the following line to remove "Profile ID" from the elements used by the application:

```
bharosa.uio.appId1.credentials.enum.profileid.enabled=false
```

6.6 Authenticator Properties

Each Authenticator interface (device) has its own unique security features. Some of these features can be enabled and disabled by editing a properties file. The following properties can be configured by adding them to client_resource_<locale>.properties. The default properties can be found in asa_resource.properties.

For details on the visual elements for each device, refer to Chapter 7, "Authenticator Properties."

6.6.1 TextPad

The TextPad is a personalized device for entering passwords or PINs using a regular keyboard. Like other Adaptive Strong Authentication devices, the TextPad helps in solving phishing problems.

Feature	Property
Default BG (Can be application specific)	bharosa.uio.<appId>.DeviceTextPad.default.image = textpad_bg/UIO_BG.jpg
Password Frame File (Can be application specific)	bharosa.uio.<appId>.password.DeviceTextPad.frame =
Challenge Frame File (Can be application specific)	bharosa.uio.<appId>.challenge.DeviceTextPad.frame =
Registration Frame File (Can be application specific)	bharosa.uio.<appId>.register.DeviceTextPad.frame = textpad_bg/TP_O_preview.png
User Preferences Frame File (Can be application specific)	bharosa.uio.<appId>.userpreferences.DeviceTextPad.frame= textpad_bg/TP_O_preview.png

6.6.2 KeyPad

The KeyPad is a customizable graphics keyboard that can be used to enter alphanumeric and special character like a traditional keyboard. KeyPad is ideal for entering passwords and other sensitive data. For example, credit card numbers can be entered.

Feature	Property
Default BG (Can be application specific)	bharosa.uio.<appId>.DeviceKeyPadFull.default.image = keypad_bg/UIO_BG.jpg
Password Frame File (Can be application specific)	bharosa.uio.<appId>.password.DeviceKeyPadFull.frame =
Challenge Frame File (Can be application specific)	bharosa.uio.<appId>.challenge.DeviceKeyPadFull.frame =

Feature	Property
Registration Frame File (Can be application specific)	bharosa.uio.<appId>.register.DeviceKeyPadFull.frame = alphapad_bg/kp_O_preview.png
User Preferences Frame File (Can be application specific)	bharosa.uio.<appId>.userpreferences.DeviceKeyPadFull.frame = alphapad_bg/kp_O_preview.png

6.6.3 PinPad

The PinPad is a lightweight authentication device to enter a numeric PIN.

Feature	Property
Default BG (Can be application specific)	bharosa.uio.default.DevicePinPad.default.image = pinpad_bg/UIO_BG.jpg
Password Frame File (Can be application specific)	bharosa.uio.<appId>.password.DevicePinPad.frame =
Challenge Frame File (Can be application specific)	bharosa.uio.<appId>.challenge.DevicePinPad.frame =
Registration Frame File (Can be application specific)	bharosa.uio.<appId>.register.DevicePinPad.frame = pinpad_bg/PP_v02_frame_preview.png
User Preferences Frame File (Can be application specific)	bharosa.uio.<appId>.userpreferences.DevicePinPad.frame = pinpad_bg/PP_v02_frame_preview.png

6.6.4 QuestionPad

The QuestionPad is a personalized device for entering answers to challenge questions using a regular keyboard. The QuestionPad is capable of incorporating the challenge question into the QuesitonPad image. Like other Adaptive Strong Authentication devices, QuestionPad also helps in solving the phishing problem.

Feature	Property
Default BG (Can be application specific)	bharosa.uio.<appId>.DeviceQuestionPad.default.image = textpad_bg/UIO_BG.jpg
Challenge Frame File (Can be application specific)	bharosa.uio.<appId>.challenge.DeviceQuestionPad.frame =

6.7 Enabling Device Registration

Device registration is a feature that allows a user to flag the computer he is using as a safe device. The customer can then configure the rules to challenge a user that is not coming from one of his registered devices.

Device registration is available as a standard feature in Oracle Adaptive Access Manager. The feature can be turned on, although it is off by default in the product.

To enable the device registration features for all applications, add the following lines to the bharosauio_client.properties file

```
# Enables device registration
bharosa.uio.default.registerdevice.enabled=true

# Enables user to be able to unregister current device in user preferences
bharosa.uio.default.userpreferences.unregister.this.enabled=true

# Enables user to be able to unregister all devices in user preferences
```

```
bharosa.uio.default.userpreferences.unregister.all.enabled=true
```

To enable the features on an application-specific bases, "default" can be replaced with the appropriate appId in each of the prior property names.

You do not need to configure models for Device Registration, but there are two rules specific to Device Registration that you will need to set up: "Device: Is registered" and "Device used but not registered."

7 Authenticator Properties

Adaptive Strong Authenticator provides end users a secure method to enter sensitive credentials online. Adaptive Strong Authenticator is comprised of multiple secure interfaces. There are many security technologies employed in the Adaptive Strong Authenticator user interfaces.

Each Adaptive Strong Authenticator interface is a Virtual Authentication Device (VAD). Each VAD has its own unique set of security features that make it much more than a mere image on a web page.

Details on the Authenticator properties are provided in this chapter for your reference only. Changes are not supported.

- Property Files
- Features Configuration
- KeysSets

7.1 Property Files

Authenticator uses the files listed below:

- **authentipad_resource.properties** - contains all default Authenticator properties other than KeySets.
- **authentipad_keyset_enums.properties** - contains default KeySet definitions used in the KeyPad and PinPad devices.
- **bharosa_client.properties** - contains configuration properties that are not localized (translated).
- **client_resource_<locale>.properties** - files to be created by the person customizing the application to contain locale-specific properties such as translated displayed messages. The locale identifier consists of at least a language identifier, and a region identifier (if required). For example, the custom properties file for US English is client_resource_en_US.properties.

Note: Many of the properties related to the authenticators are in resource bundles so that they are capable of being localized. If the default value is in a "resource" file like asa_resource.properties, then the override value should be placed in the client override file for resource bundle values (client_resource_<locale>.properties).

7.2 What Authenticator Interfaces Should My Organization Use?

The business and security units in your organization should work together with Oracle to determine which Authenticator interfaces should be deployed to your end users. This decision should be based on finding a proper balance between usability and security. For recommendations on which Authenticator interfaces might be best for your organization please consult with your Oracle representative.

7.3 What Elements of the Authenticator User Interface Can Be Configured?

The Virtual Authentication Device concept is integral to the Authenticator component and must be preserved in all circumstances. All graphical configurations of Authenticator need to take the VAD concept into account. This document will illustrate some examples to show what may and may not be changed graphically. For the examples in this document we will use the TextPad interface.

7.3.1 The Frame

Each of the Authenticator "Pad" interfaces (TextPad, PinPad, and so on) has a frame. The frame marks the outer boundary of the Authenticator user interface and delineates the VAD from the rest of the page. The frame must always be apparent regardless of the graphical treatment to preserve the appearance of a device. The frame may not blend into the surrounding elements of an HTML page to the point were it disappears visually. The overall size and aspect of each Pad is fixed and may not be altered. All elements of the interface must be contained within the frame. These elements include buttons, fields, personal phrase and personal image. The individual elements of the Authenticator may not have their size or position altered. A single PNG file contains the branding, frame and button images. Oracle can develop a custom frame for you once your requirements are finalized. All configurations of Authenticator are subject to review by Oracle to ensure proper security, usability and product identity.

The frame may be altered only in the following ways:

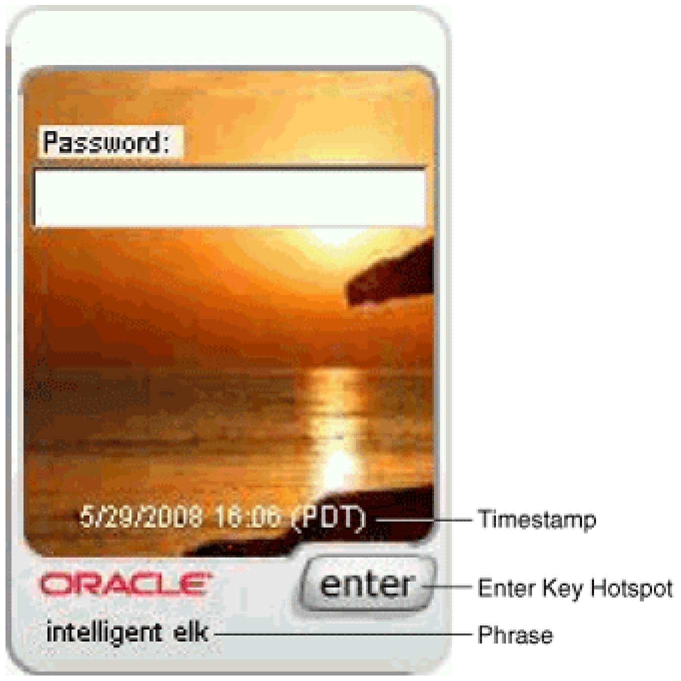
- Colors may be altered for the outline and fill of the frame
- Colors of the buttons on the frame may be altered (enter, back, and so on)
- Branding may be altered

7.3.2 Features Configuration

Each Authenticator interface has its own unique security features. Some of these features can be enabled or disabled by adding/editing properties. For 10.1.4.5 or later, the properties will need to be added to the client_resource_<locale>.properties file. For versions earlier than 10.1.4.5, the property will need to be added to the bharosa_client.properties file.

7.3.2.1 TextPad

TextPad is a personalized device for entering passwords or PIN using a regular keyboard. An example TextPad is shown below.



This section provides information on the visual elements of TextPad.

Phrase (Caption)

```
bharosa.authentipad.textpad.caption.personalize = true
bharosa.authentipad.textpad.caption.x = 14
bharosa.authentipad.textpad.caption.y = 203
bharosa.authentipad.textpad.caption.frame = false
bharosa.authentipad.textpad.caption.wrap = false
bharosa.authentipad.textpad.caption.width = 130
bharosa.authentipad.textpad.caption.height = 16
bharosa.authentipad.textpad.caption.font.name = Arial
bharosa.authentipad.textpad.caption.font.color = 000000
bharosa.authentipad.textpad.caption.font.type= 0
bharosa.authentipad.textpad.caption.font.size = 9
```

Timestamp

```
bharosa.authentipad.textpad.timestamp.x = 25
bharosa.authentipad.textpad.timestamp.y = 165
bharosa.authentipad.textpad.timestamp.width = 132
bharosa.authentipad.textpad.timestamp.height = 16
bharosa.authentipad.textpad.timestamp.frame = false
bharosa.authentipad.textpad.timestamp.wrap = false
bharosa.authentipad.textpad.timestamp.font.name = Arial
bharosa.authentipad.textpad.timestamp.font.color = ffffff
bharosa.authentipad.textpad.timestamp.font.type= 0
bharosa.authentipad.textpad.timestamp.font.size = 9
```

Enter Key Hotspot

```
bharosa.authentipad.textpad.enterkey.x=98
bharosa.authentipad.textpad.enterkey.y=181
bharosa.authentipad.textpad.enterkey.width=45
bharosa.authentipad.textpad.enterkey.height=19
bharosa.authentipad.textpad.enterkey.label=enter
bharosa.authentipad.textpad.enterkey.enable=true
```

Password MaxLength

```
bharosa.authentipad.textpad.datafield.maxLength=25
```

Default Background

```
bharosa.authentipad.textpad.background.file=textpad_bg/UIO_BG.jpg
```

Display Font Size for iPhone

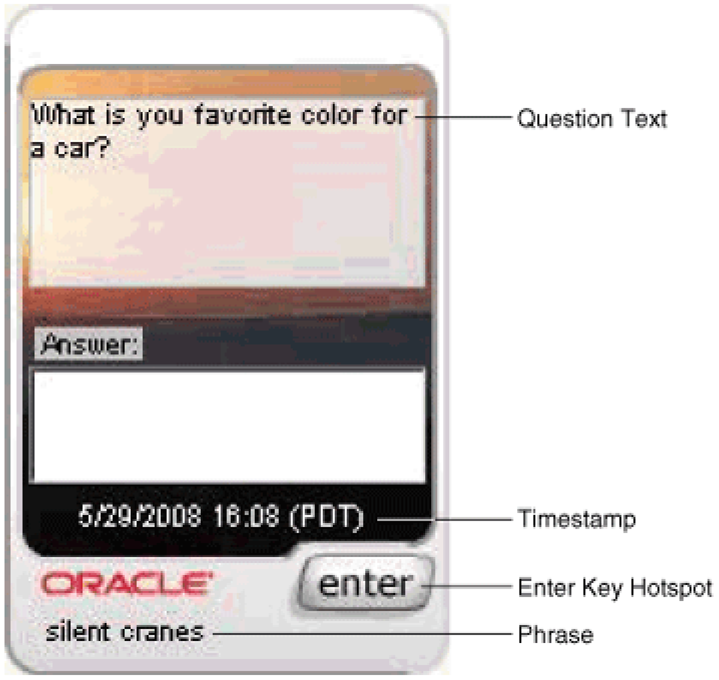
The property to customize the font size for TextPad on an iPhone is shown below:

```
bharosa.authentipad.textpad.datafield.font.size=12
```

For 10.1.4.5 or later, the property will need to be added to the client_resource_<locale>.properties file. For versions earlier than 10.1.4.5, the property will need to be added to the bharosa_client.properties file.

7.3.2.2 QuestionPad

QuestionPad is a personalized device for entering answers to challenge questions using a regular keyboard. An example QuestionPad is shown below.



This section provides information on the visual elements of QuestionPad.

Note: In 10.1.4.5 and above, the QuestionPad is a single line field.

Phrase (Caption)

```
bharosa.authentipad.questionpad.caption.personalize = true
bharosa.authentipad.questionpad.caption.x = 14
bharosa.authentipad.questionpad.caption.y = 203
bharosa.authentipad.questionpad.caption.frame = false
bharosa.authentipad.questionpad.caption.wrap = false
bharosa.authentipad.questionpad.caption.width = 130
bharosa.authentipad.questionpad.caption.height = 16
bharosa.authentipad.questionpad.caption.font.name = Arial
bharosa.authentipad.questionpad.caption.font.color = 000000
bharosa.authentipad.questionpad.caption.font.type= 0
bharosa.authentipad.questionpad.caption.font.size = 9
```

Timestamp

```
bharosa.authentipad.questionpad.timestamp.x = 25
bharosa.authentipad.questionpad.timestamp.y = 165
bharosa.authentipad.questionpad.timestamp.width = 132
bharosa.authentipad.questionpad.timestamp.height = 16
bharosa.authentipad.questionpad.timestamp.frame = false
bharosa.authentipad.questionpad.timestamp.wrap = false
bharosa.authentipad.questionpad.timestamp.font.name = Arial
bharosa.authentipad.questionpad.timestamp.font.color = ffffff
bharosa.authentipad.questionpad.timestamp.font.type= 0
bharosa.authentipad.questionpad.timestamp.font.size = 9
```

Question Text

```
bharosa.authentipad.questionpad.question.x = 9
bharosa.authentipad.questionpad.question.y = 32
bharosa.authentipad.questionpad.question.width = 132
bharosa.authentipad.questionpad.question.height = 62
bharosa.authentipad.questionpad.question.frame = false
bharosa.authentipad.questionpad.question.wrap = true
bharosa.authentipad.questionpad.question.font.name = Arial
bharosa.authentipad.questionpad.question.font.color = 000000
bharosa.authentipad.questionpad.question.font.type= 0
bharosa.authentipad.questionpad.question.font.size = 9
```

Enter Key Hotspot

```
bharosa.authentipad.questionpad.enterkey.x=98
bharosa.authentipad.questionpad.enterkey.y=181
bharosa.authentipad.questionpad.enterkey.width=45
bharosa.authentipad.questionpad.enterkey.height=19
bharosa.authentipad.questionpad.enterkey.label=enter
bharosa.authentipad.questionpad.enterkey.enable=true
```

Visible Text Input or Password (Non-Visible) Input Setting

The following resource bundle property (client_resource_<locale>.properties) in 10.1.4.5 and above determines whether the QuestionPad is set for visible text input or password (non-visible) input.

```
bharosa.authentipad.questionpad.datafield.input.type
```

Valid values are text and password.

Display Font Size for iPhone

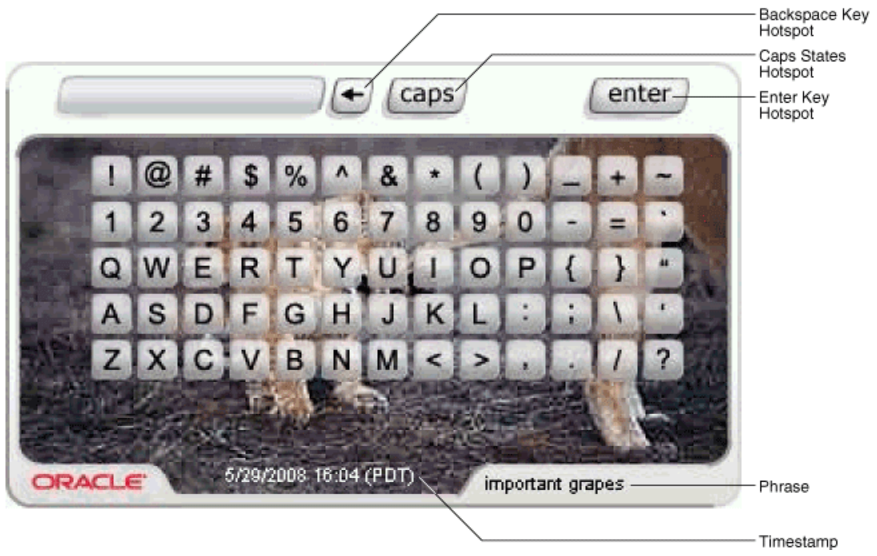
The property to customize the font size for QuestionPad on an iPhone is shown below:

```
bharosa.authentipad.questionpad.datafield.font.size=12
```

For 10.1.4.5 or later, the property will need to be added to the c client_resource_<locale>.properties file. For versions earlier than 10.1.4.5, the property will need to be added to the bharosa_client.properties file.

7.3.2.3 Keypad

KeyPad is a personalized graphics keyboard, which can be used to enter alphanumeric and special character that can be enter using a traditional keyboard. An example KeyPad is shown below.



This section provides information on the visual elements of KeyPad.

Phrase (Caption)

```
bharosa.authentipad.keypad.caption.personalize = true
bharosa.authentipad.keypad.caption.x = 240
bharosa.authentipad.keypad.caption.y = 206
bharosa.authentipad.keypad.caption.frame = false
bharosa.authentipad.keypad.caption.wrap = false
bharosa.authentipad.keypad.caption.width = 130
bharosa.authentipad.keypad.caption.height = 16
bharosa.authentipad.keypad.caption.font.name = Arial
bharosa.authentipad.keypad.caption.font.color = 000000
bharosa.authentipad.keypad.caption.font.type= 0
bharosa.authentipad.keypad.caption.font.size = 9
bharosa.authentipad.full.caption.font.color = 000000
```

Timestamp

```
bharosa.authentipad.keypad.timestamp.x = 110
bharosa.authentipad.keypad.timestamp.y = 202
bharosa.authentipad.keypad.timestamp.width = 132
bharosa.authentipad.keypad.timestamp.height = 16
bharosa.authentipad.keypad.timestamp.frame = false
bharosa.authentipad.keypad.timestamp.wrap = false
bharosa.authentipad.keypad.timestamp.font.name = Arial
bharosa.authentipad.keypad.timestamp.font.color = ffffff
bharosa.authentipad.keypad.timestamp.font.type= 0
bharosa.authentipad.keypad.timestamp.font.size = 9
bharosa.authentipad.full.timestamp.font.color = ffffff
```

Enter Key Hotspot

```
bharosa.authentipad.keypad.enterkey.x=292
bharosa.authentipad.keypad.enterkey.y=8
bharosa.authentipad.keypad.enterkey.width=50
bharosa.authentipad.keypad.enterkey.height=20
bharosa.authentipad.keypad.enterkey.label=enter
bharosa.authentipad.keypad.enterkey.enable=true
```

Backspace Key Hotspot

```
bharosa.authentipad.keypad.backspace.x=164
bharosa.authentipad.keypad.backspace.y=8
bharosa.authentipad.keypad.backspace.width=20
bharosa.authentipad.keypad.backspace.height=20
bharosa.authentipad.keypad.backspace.enable=true
```

Caps States

```
bharosa.authentipad.keypad.capslock.x=188
bharosa.authentipad.keypad.capslock.y=0
bharosa.authentipad.keypad.capslock.width=43
bharosa.authentipad.keypad.capslock.height=29
bharosa.authentipad.keypad.capslock.capsonimg=kp_v2_all_caps.jpg
bharosa.authentipad.keypad.capslock.capsshiftimg=kp_v2_first_caps.jpg
```

Password MaxLength

```
bharosa.authentipad.full.datafield.maxLength=8
```

Jitter

```
bharosa.authentipad.full.encrypt.jitter=true
```

Sub-jitter

```
bharosa.authentipad.full.keyWidthJitter=0
bharosa.authentipad.full.keyHeightJitter=0
```

Scramble

```
bharosa.authentipad.full.randomizeKeys=false
```

Multi-shape key

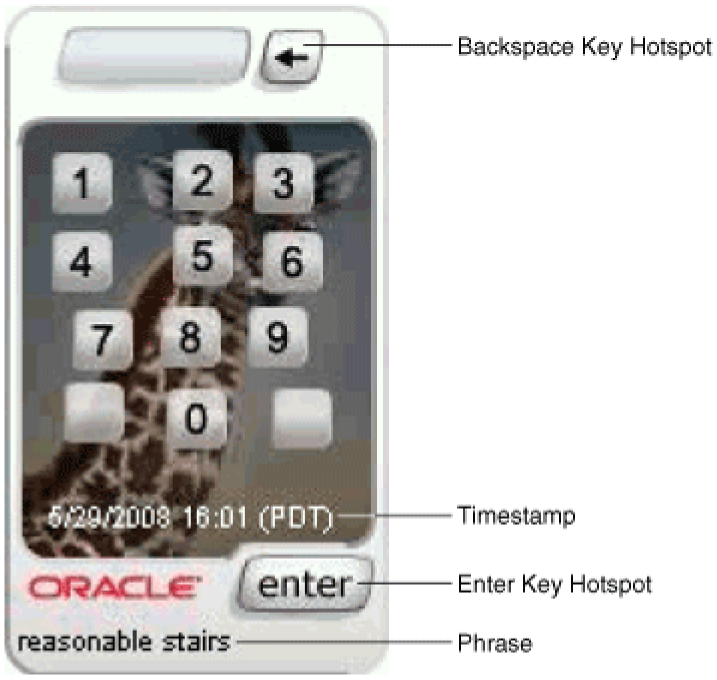
```
bharosa.authentipad.full.skins.dirlist=alphapad_skins/square
```

Default Background

```
bharosa.authentipad.full.background.file=alphapad_bg/UIO_BG.jpg
```

7.3.2.4 PinPad

PinPad is a lightweight authentication device for entering a numeric PIN. An example PinPad is shown below.



This section provides information on the visual elements of PinPad.

Phrase (Caption)

```
bharosa.authentipad.pinpad.caption.personalize = true
bharosa.authentipad.pinpad.caption.x = 5
bharosa.authentipad.pinpad.caption.y = 206
bharosa.authentipad.pinpad.caption.frame = false
bharosa.authentipad.pinpad.caption.wrap = false
bharosa.authentipad.pinpad.caption.width = 130
bharosa.authentipad.pinpad.caption.height = 16
bharosa.authentipad.pinpad.caption.font.name = Arial
bharosa.authentipad.pinpad.caption.font.color = 000000
bharosa.authentipad.pinpad.caption.font.type= 0
bharosa.authentipad.pinpad.caption.font.size = 9
bharosa.authentipad.numeric.caption.font.color = 000000
```

Timestamp

```
bharosa.authentipad.pinpad.timestamp.x = 15
bharosa.authentipad.pinpad.timestamp.y = 165
bharosa.authentipad.pinpad.timestamp.width = 132
bharosa.authentipad.pinpad.timestamp.height = 16
bharosa.authentipad.pinpad.timestamp.frame = false
bharosa.authentipad.pinpad.timestamp.wrap = false
bharosa.authentipad.pinpad.timestamp.font.name = Arial
bharosa.authentipad.pinpad.timestamp.font.color = ffffff
bharosa.authentipad.pinpad.timestamp.font.type= 0
bharosa.authentipad.pinpad.timestamp.font.size = 9
bharosa.authentipad.numeric.timestamp.font.color = ffffff
```

Enter Key Hotspot

```
bharosa.authentipad.pinpad.enterkey.x=78
bharosa.authentipad.pinpad.enterkey.y=182
bharosa.authentipad.pinpad.enterkey.width=49
bharosa.authentipad.pinpad.enterkey.height=20
bharosa.authentipad.pinpad.enterkey.label=enter
bharosa.authentipad.pinpad.enterkey.enable=true
```

Backspace Key Hotspot

```
bharosa.authentipad.pinpad.backspace.x=86
bharosa.authentipad.pinpad.backspace.y=8
bharosa.authentipad.pinpad.backspace.width=20
bharosa.authentipad.pinpad.backspace.height=20
bharosa.authentipad.pinpad.backspace.label=&lt;
bharosa.authentipad.pinpad.backspace.enable=true
```

Pin MaxLength

```
bharosa.authentipad.numeric.datafield.maxLength=8
```

Jitter

```
bharosa.authentipad.numeric.encrypt.jitter =true
```

Sub-jitter

```
bharosa.authentipad.numeric.keyWidthJitter=50
bharosa.authentipad.numeric.keyHeightJitter=15
```

Scramble

```
bharosa.authentipad.numeric.randomizeKeys=false
```

Multi-shape key

```
bharosa.authentipad.numeric.skins.dirlist=pinpad_skins/square,pinpad_skins/oval,pi
npad_skins/hexa
```

Default Background

```
bharosa.authentipad.numeric.background.file=pinpad_bg/UIO_BG.jpg
```

7.4 Authenticator Specifications

Each Authenticator interface has its own specifications.

Interface	Phrase Max	Question Max	Field Max	Size (pixels)
TextPad	21	NA	21 (visible)	148 X 223
KeyPad	21	NA	18 (visible)	368 X 223
PinPad	21	NA	8 (visible)	128 X 223
QuestionPad	21	55	33 (visible)	148 X 223

7.5 Accessibility

End users who access using assistive techniques will need to use the accessible versions of the virtual authentication devices. Accessible versions of the TextPad, QuestionPad, KeyPad and PinPad are not enabled by default. If accessible versions will be needed in a deployment, they can be enabled via properties.

To enable these versions, set the "is ADA compliant" flag to true.

For native integration the property to control the pads is desertref.authentipad.isADACompliant.

The accessible versions of the pads contain tabbing, directions and alt text necessary for navigation via screen reader and other assistive technologies.

7.6 KeysSets

A KeySet is the configuration that defines what character keys are present on the authenticator. KeySets are used by the KeyPad and PinPad authenticators.

7.6.1 User Defined Enums Overview

KeySets are defined by a series user defined enums.

User-defined enums are a collection of properties that represent a list of items. Each element in the list may contain several different attributes. The definition of a user-defined enum begins with a property ending in the keyword ".enum" and has a value describing the use of the user-defined enum. Each element definition then starts with the same property name as the enum, and adds on an element name and has a value of a unique integer as an ID. The attributes of the element follow the same pattern, beginning with the property name of the element, followed by the attribute name, with the appropriate value for that attribute.

The following is an example of an enum defining credentials displayed on the login screen of an Adaptive Strong Authenticator implementation:

```
bharosa.uio.default.credentials.enum = Enum for Login Credentials
bharosa.uio.default.credentials.enum.companyid=0
bharosa.uio.default.credentials.enum.companyid.name=CompanyID
bharosa.uio.default.credentials.enum.companyid.description=Company ID
bharosa.uio.default.credentials.enum.companyid.inputname=comapanyid
bharosa.uio.default.credentials.enum.companyid.maxlength=24
bharosa.uio.default.credentials.enum.companyid.order=0
bharosa.uio.default.credentials.enum.username=1
bharosa.uio.default.credentials.enum.username.name=Username
bharosa.uio.default.credentials.enum.username.description=Username
bharosa.uio.default.credentials.enum.username.inputname=userid
bharosa.uio.default.credentials.enum.username.maxlength=18
bharosa.uio.default.credentials.enum.username.order=1
```

This set of properties defines one user-defined enum that contains two elements, each of which with five attributes. The "name" and "description" attributes are required to define any user-defined enum, other attributes are defined and used as needed by each individual use of a user-defined enum.

7.6.2 KeySet Definition

The first enum defines the rows of the KeySet and points to an another enum describing the keys present in that row.

For example, the following enum defines the rows of keys in a PinPad:

```
bharosa.authentipad.pinpad.default.keyset.enum=Default PinPad Keyset Enum
bharosa.authentipad.pinpad.default.keyset.enum.row1=0
bharosa.authentipad.pinpad.default.keyset.enum.row1.name=Default PinPad Keyset Row
1
bharosa.authentipad.pinpad.default.keyset.enum.row1.description=Default PinPad
Keyset Row 1
bharosa.authentipad.pinpad.default.keyset.enum.row1.keys=bharosa.authentipad.pinpa
d.default.keyset.row1.enum
bharosa.authentipad.pinpad.default.keyset.enum.row1.order=1
bharosa.authentipad.pinpad.default.keyset.enum.row2=1
bharosa.authentipad.pinpad.default.keyset.enum.row2.name=Default PinPad Keyset Row
2
bharosa.authentipad.pinpad.default.keyset.enum.row2.description=Default PinPad
Keyset Row 2
bharosa.authentipad.pinpad.default.keyset.enum.row2.keys=bharosa.authentipad.pinpa
d.default.keyset.row2.enum
bharosa.authentipad.pinpad.default.keyset.enum.row2.order=2

bharosa.authentipad.pinpad.default.keyset.enum.row3=2
bharosa.authentipad.pinpad.default.keyset.enum.row3.name=Default PinPad Keyset Row
3
bharosa.authentipad.pinpad.default.keyset.enum.row3.description=Default PinPad
Keyset Row 3
bharosa.authentipad.pinpad.default.keyset.enum.row3.keys=bharosa.authentipad.pinpa
d.default.keyset.row3.enum
bharosa.authentipad.pinpad.default.keyset.enum.row3.order=3

bharosa.authentipad.pinpad.default.keyset.enum.row4=3
bharosa.authentipad.pinpad.default.keyset.enum.row4.name=Default PinPad Keyset Row
4
bharosa.authentipad.pinpad.default.keyset.enum.row4.description=Default PinPad
Keyset Row 4
bharosa.authentipad.pinpad.default.keyset.enum.row4.keys=bharosa.authentipad.pinpa
d.default.keyset.row4.enum
bharosa.authentipad.pinpad.default.keyset.enum.row4.order=4
```

Each row is made of the following properties:

Table 7–1 Properties of Rows

Property	Description
name	Name of the row.
description	Description of the row.
keys	Enum identifier of the enum that defines the keys in the row.
order	The order the key resides in the row of keys.

In this case, the row1 enum is defined as follows:

```
bharosa.authentipad.pinpad.default.keyset.row1.enum=Default Pinpad Keyset Row 1
bharosa.authentipad.pinpad.default.keyset.row1.enum.key1=0
bharosa.authentipad.pinpad.default.keyset.row1.enum.key1.name=1
bharosa.authentipad.pinpad.default.keyset.row1.enum.key1.description=1
bharosa.authentipad.pinpad.default.keyset.row1.enum.key1.value=1
bharosa.authentipad.pinpad.default.keyset.row1.enum.key1.shiftvalue=1
bharosa.authentipad.pinpad.default.keyset.row1.enum.key1.image=kp_v2_1.png
bharosa.authentipad.pinpad.default.keyset.row1.enum.key1.order=1

bharosa.authentipad.pinpad.default.keyset.row1.enum.key2=1
bharosa.authentipad.pinpad.default.keyset.row1.enum.key2.name=2
bharosa.authentipad.pinpad.default.keyset.row1.enum.key2.description=2
bharosa.authentipad.pinpad.default.keyset.row1.enum.key2.value=2
bharosa.authentipad.pinpad.default.keyset.row1.enum.key2.shiftvalue=2
bharosa.authentipad.pinpad.default.keyset.row1.enum.key2.image=kp_v2_2.png
bharosa.authentipad.pinpad.default.keyset.row1.enum.key2.order=2

bharosa.authentipad.pinpad.default.keyset.row1.enum.key3=2
bharosa.authentipad.pinpad.default.keyset.row1.enum.key3.name=3
bharosa.authentipad.pinpad.default.keyset.row1.enum.key3.description=3
bharosa.authentipad.pinpad.default.keyset.row1.enum.key3.value=3
bharosa.authentipad.pinpad.default.keyset.row1.enum.key3.shiftvalue=3
bharosa.authentipad.pinpad.default.keyset.row1.enum.key3.image=kp_v2_3.png
bharosa.authentipad.pinpad.default.keyset.row1.enum.key3.order=3
```

Each key is made of the following properties:

Table 7–2 Properties of Each Key

Property	Description
name	Name of the key.
description	Description of the key.
value	The character value the key represents when clicked.
shiftvalue	The character value the key represents when in caps mode.
image	The image file name that will be used to display the visual representation of the key.
order	The order the key resides in the row of keys.

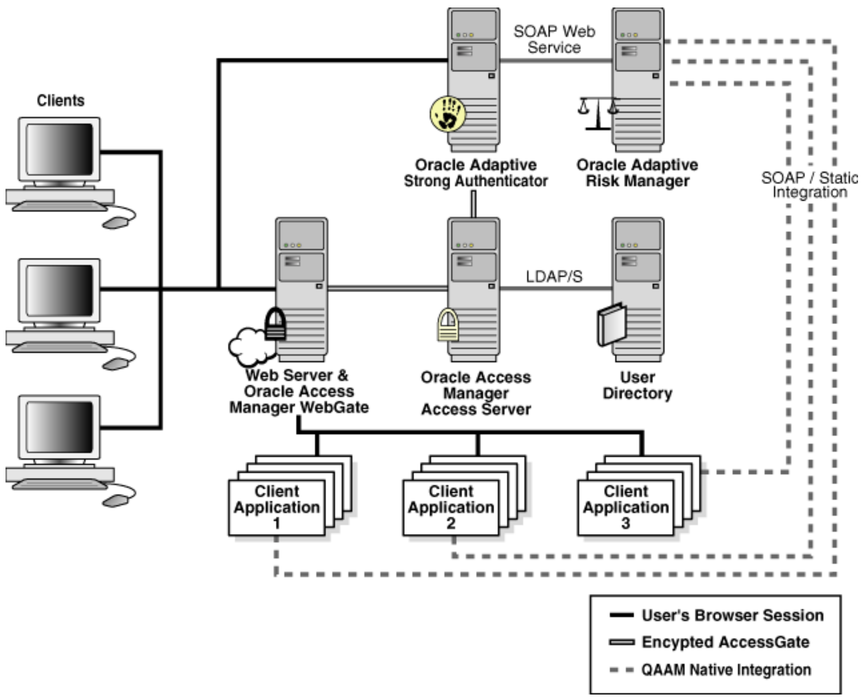
Part III

Integration with Oracle Access Manager

Part III contains a chapter on Oracle Adaptive Access Manager and Oracle Access Manager integration.

8 Oracle Access Manager Integration

This chapter describes the process for integrating Oracle Adaptive Access Manager's Adaptive Strong Authenticator with Oracle Access Manager. Integrating the two will allow you to use Oracle Adaptive Access Manager's Adaptive Strong Authenticator virtual authentication pads to identify users attempting to access Oracle Access Manager's protected applications.



Using these products in combination will allow you fine control over the authentication process and full capabilities of pre-/post- authentication checking against Adaptive Risk Manager models.

8.1 Prerequisites

This integration process assumes that the Oracle Access Manager environment has been configured to protect simple HTML resources using two different authentication schemes. Authentication schemes protect the client application's url.

- The first authentication scheme uses Basic Over LDAP.

- The second authentication scheme is a higher-security level and integrates Adaptive Strong Authenticator by using a custom form-based authentication scheme.

For more information, refer to the *Oracle Access Manager Integration Guide*.

The following set of components is required for this implementation:

- Oracle Adaptive Access Manager
- Oracle Access Manager 10_1_4_01 with Patch 6158232
- an Application Server
- Oracle Access Manager SDK

8.2 Integration Overview

Except where specified, the following procedures are required to complete the integration of Oracle Access Adaptive Manager and Oracle Access Manager.

- Configure Oracle Access Manager AccessGate for Adaptive Strong Authenticator Embedded AccessGate
- Configure Oracle Access Manager AccessGate for Adaptive Strong Authenticator Front-End Web Server
- Configure Oracle Access Manager Authentication Scheme for the Adaptive Strong Authenticator
- Configure Oracle Access Manager Host Identifiers for Adaptive Strong Authenticator (Optional)
- Install ASDK for Adaptive Strong Authenticator
- Configure ASDK AccessGate for Adaptive Strong Authenticator
- Install WebGate for Adaptive Strong Authenticator Front-End Web Server
- Unpack and Install Oracle Adaptive Access Manager Plug-In to Adaptive Strong Authenticator for Oracle Access Manager Integration
- Copy ASDK JAR Files to Adaptive Strong Authenticator
- Add ASDK Library Path to Adaptive Strong Authenticator Application Properties
- Add ASDK Library Path to Adaptive Strong Authenticator Server Properties
- Configure Oracle Access Manager Domain to use Adaptive Strong Authenticator Authentication

8.3 Configure Oracle Access Manager AccessGate for Adaptive Strong Authenticator Embedded AccessGate

Before installing the Access Server SDK (ASDK), you must define the Oracle Access Manager server-side settings for the AccessGate that the ASDK will use for communication.

This section shows you how to define the new AccessGate for the embedded Adaptive Strong Authenticator AccessGate.

Note: This chapter will not explain in detail all of the settings involved with Oracle Access Manager AccessGates.

Steps

1. Launch Internet Explorer.
2. Log in to Oracle Access Manager.

   For example, `http://<oam_hostname>/access/oblix`.
3. Click **Access System Console**.
4. Log in as `<Administrator>`.
5. Click **Access System Configuration**.
6. Click **Add New AccessGate**.
7. Using the oaamAccessGate configuration settings shown below, create a new AccessGate and assign it to an Access Server.

Table 8–1 oaamAccessGate Configuration

Parameter	Value
AccessGate Name	oaamAccessGate
Description	AccessGate for Oracle Adaptive Access Manager-Adaptive Strong Authenticator authentication
Hostname	<hostname>
Port	<port>
AccessGate Password	<passwd>
Debug	<Off>
Maximum user session time (seconds)	3600
Idle Session Time (seconds)	3600
Maximum Connections	1
Transport Security	<Open>
IP Validation	<On>
IP Validation Exception	<leave blank>
Maximum Client Session Time (hours)	24
Failover Threshold	1
Access server timeout threshold	<leave blank>
Sleep for (seconds)	60
Maximum elements in cache	10000
Cache timeout (seconds)	1800
Impersonation Username	<leave blank>
Impersonation Password	<leave blank>
Access Management Service	<On>
Preferred HTTP Cookie Domain	<domain_name>
Preferred HTTP Host	<hostname>:<port>
Deny on not protected	<Off>
CachePragmaHeader	no-cache
CacheControlHeader	no-cache

Table 8–1 (Cont.) oaamAccessGate Configuration

Parameter	Value
LogOutURLs	<leave blank>
User Defined Parameters	<leave blank>
Assign An Access Server (Primary	<hostname>:<port>
Number of Connections	1

8.4 Configure Oracle Access Manager AccessGate for Adaptive Strong Authenticator Front-End Web Server

The Oracle Adaptive Access Manager's Adaptive Strong Authenticator/Oracle Access Manager integration involves two Oracle Access Manager AccessGates: one for fronting the Web server (a traditional WebGate) to Adaptive Strong Authenticator and one for the embedded AccessGate. This section explains how to configure the Oracle Access Manager AccessGate that fronts the Web server to Adaptive Strong Authenticator.

Steps

1. Click **Add New AccessGate**.
2. Use the settings in the table below to create a new AccessGate and assign it an Access Server

Note: The Adaptive Strong Authenticator AccessGate settings (described in the Configure Oracle Access Manager AccessGate for Adaptive Strong Authenticator Embedded AccessGate section) and the OHS WebGate settings are identical (except for the AccessGate names) because OHS is also a server for the Adaptive Strong Authenticator application. In some deployments, these might differ.

Table 8–2 ohsWebGate Configuration

Parameter	Value
AccessGate Name	ohsWebGate
Description	AccessGate for Web server hosting Adaptive Strong Authenticator application
Hostname	<hostname>
Port	<port>
AccessGate Password	<passwd>
Debug	<Off>
Maximum user session time (seconds)	3600
Idle Session Time (seconds)	3600
Maximum Connections	1
Transport Security	<Open>
IP Validation	<On>

Table 8–2 (Cont.) ohsWebGate Configuration

Parameter	Value
IP Validation Exception	<leave blank>
Maximum Client Session Time (hours)	24
Failover Threshold	1
Access server timeout threshold	<leave blank>
Sleep for (seconds)	60
Maximum elements in cache	10000
Cache timeout (seconds)	1800
Impersonation Username	<leave blank>
Impersonation Password	<leave blank>
Access Management Service	<On>
Preferred HTTP Cookie Domain	.<domain_name>
Preferred HTTP Host	<hostname>:<port>
Deny on not protected	<Off>
CachePragmaHeader	no-cache
CacheControlHeader	no-cache
LogOutURLs	<leave blank>
User Defined Parameters	<leave blank>
Assign An Access Server (Primary)	<oam_hostname>:<port>
Number of Connections	1

3. Click **AccessGate Configuration**.
4. Click **OK** to search for all AccessGates.

   The new AccessGate is now listed

8.5 Configure Oracle Access Manager Authentication Scheme for the Adaptive Strong Authenticator

To leverage Adaptive Strong Authenticator as an authentication mechanism, Oracle Access Manager must have a defined Authentication Scheme to understand how to direct authentications to Adaptive Strong Authenticator.

Steps

1. Click **Authentication Management**.
2. Click **New**.

3. Using the settings in the table below, begin creating the new Adaptive Strong Authenticator authentication scheme:

Table 8–3 OAAM ASA Authentication Scheme Configuration

Parameter	Value
Name	Adaptive Strong Authentication
Description	Oracle Adaptive Access Manager-Adaptive Strong Authenticator virtual authentication pad auth scheme
Level	3
Challenge Method	Form
Challenge Parameter(s)	form:/oasa/loginPage.jsp
	creds:userid password
	action:/oasa/dummy.jsp
SSL Required	<No>
Challenge Redirect	<Redirect Url>
Enabled	<Disabled/Greyed Out>

Note: For the challenge parameter, do not use "action:/oasa". Use "action:/oasa/dummy.jsp". If you do not do this, you will receive a "technical error" message from Oracle Adaptive Access Manager authentication. "dummy.jsp" does not need to exist.

4. Click **Save**.
5. Click **Ok** to confirm the saved operation.
6. Click **Plugins**.
7. Click **Modify**.
8. Click **Add**.
9. Create the plugin configurations using the information presented in the table below.

Table 8–4 OAAM ASA Authentication Scheme Configuration - Plugins

Plugin Name	Plugin Parameters
credential_mapping	obMappingBase="dc=<domain>,dc=com",obMappingFilter="(uid=%userid%)"
validate_password	obCredentialPassword="password"

10. Click **Save**.
11. Click **General**.
12. Click **Modify**.
13. Set **Enabled** to **Yes**.
14. Click **Save**.

8.6 Configure Oracle Access Manager Host Identifiers for Adaptive Strong Authenticator (Optional)

The AccessGates used by Adaptive Strong Authenticator must have host identifier entries. Use the Host Identifiers feature to enter the official name for the host, and every other name by which the host can be addressed by users.

A request sent to any address on the list is mapped to the official host name, and applicable rules and policies are implemented. This is primarily used in virtual site hosting environments.

8.7 Install ASDK for Adaptive Strong Authenticator

Install the ASDK that will be used by the Adaptive Strong Authenticator for communication with the Oracle Access Manager Access Server.

Adaptive Strong Authenticator requires ASDK to communicate with the Oracle Access Manager Access Server.

8.8 Configure ASDK AccessGate for Adaptive Strong Authenticator

After installing the ASDK for the Adaptive Strong Authenticator, the ASDK must be configured for use.

Use a command-line tool (`configureAccessGate`) to specify the settings for the ASDK to use for communication with the Oracle Access Manager Access Server.

Steps

1. Navigate to the `configureAccessGate` directory at <ASDK install dir>\AccessServerSDK\oblix\tools\configureAccessGate.
2. Run following command and press **Enter**.

   ```
   configureAccessGate -i < Installation directory of the AccessServerSDK>  -t
   AccessGate -w <Enter the name of the defined oaamAccessGate> -p <port> -h
   <hostname> -a <Name of the Access Server> -m open
   ```

   For example:

   ```
   configureAccessGate -i E:\oracle\oaam\AccessServerSDK -t AccessGate -w
   oaamAccessGate -p 6021 -h www.otherdomain.com -a accessSvr1 -m open
   ```

8.9 Install Web Server to Implement WebGate

Install an Apache HTTP server 2.x and configure it with the WebLogic Server Plug-in.

For instructions on installing and configuring the Apache HTTP Server Plug-In, refer to:

http://e-docs.bea.com/wls/docs92/plugins/apache.html

8.10 Install WebGate for Adaptive Strong Authenticator Front-End Web Server

To correctly handle the cookies for authentication and the required HTTP headers for the Adaptive Strong Authenticator application, Adaptive Strong Authenticator must be protected with a standard WebGate and Web server.

Steps

1. Stop the application server (and Web server).
2. Run the WebGate installation program
3. For the WebGate configuration, use the following settings:

Table 8–5 ohsWebGate Configuration

Attribute	Value
WebGate ID	ohsWebGate
Password for WebGate	<password>
Access Server ID	<Access ServerId>
Host Name	<hostname>
Port	<port

Note: Oracle Application Server installs an Oracle HTTP Server (OHS) with the application server and OC4J container.

If a different application server or servlet container (for example, BEA WebLogic, JBoss, or Tomcat) is used for Adaptive Strong Authenticator/Adaptive Risk Manager, a front-end Web server with the appropriate proxy plug-in (for example, mod_wl_20.so or mod_jk) would be necessary before installing the WebGate on the Web server.

Installation instructions for "mod_wl_20.so" is documented at:

`http://e-docs.bea.com/wls/docs92/plugins/apache.html`

8.11 Unpack and Install Oracle Adaptive Access Manager Plug-In to Adaptive Strong Authenticator for Oracle Access Manager Integration

Unpack the Adaptive Strong Authenticator plug-in for Oracle Access Manager from the oaam_plugins folder and copy the required files to the Adaptive Strong Authenticator installation.

Steps

1. Copy oasa_oam_override.jar from ... `oaam_plugins\oaam_oam_plugin\oasa\war\WEB-INF\lib` to `<OASA_HOME>\WEB-INF\lib`.
2. Copy the client folder to `<OASA_HOME>\`.
3. Rename `<OASA_HOME>\WEB-INF\struts-config.xml` to `<OASA_HOME>\WEB-INF\struts-config.xml.bak`.
4. Copy `struts-config.xml` from ... `oaam_plugins\oaam_oam_plugin\oasa\war\WEB-INF` to `<OASA_HOME>\WEB-INF\`.
5. Copy `bharosauio_client.properties` from ... `oaam_plugins\oaam_oam_plugin\oasa\war\WEB-INF\classe` to `<OASA_HOME>\WEB-INF\classes\bharosauio_client.properties`.
6. Copy `bharosauio_client.properties` from plugin.zip to <OASA_HOME>\WEB-INF\classes\bharosauio_client.properties
7. Check `lookup.properties` under `<OASA_HOME>\WEB-INF \classes` to verify that `bharosauio_client.properties` is listed.

8.12 Copy ASDK JAR Files to Adaptive Strong Authenticator

Copy the key Java AccessGate library file from the ASDK to the Adaptive Strong Authenticator installation for use.

For example, copy <ASDK install>oblix\lib\jobaccess.jar to `<OASA_HOME>\WEB-INF\lib`.

If the jar files are not copied, the Adaptive Strong Authenticator installation will not identify the ASDK Java Access Gate library.

8.13 Add ASDK Library Path to Adaptive Strong Authenticator Application Properties

Modify `bharosa_client.properties` under `<OASA_HOME>\WEB-INF\classes` to include the path of the Oracle Access Manager Java AccessGate (`jobaccess.jar`). The application properties for Adaptive Strong Authenticator must be updated to locate the AccessGate configuration information you specified with the configureAccessGate utility previously.

For example

```
bharosa.accesserversdk.path=E:\\oracle\\oaam\\AccessServerSDK
```

Note: There are 2 s's in a row in "accesserversdk" not 3 s's.

If we do not have this in our path, Adaptive Strong Authenticator will not be able to located the Access Gate configuration.

8.14 Add ASDK Library Path to Adaptive Strong Authenticator Server Properties

The Oracle Adaptive Access Manager AccessGate used by Adaptive Strong Authenticator must use the supporting library files from the ASDK directories. Please update your Application Server PATH variable to include the libraries from the ASDK.

For example,

Add `E:\oracle\oaam\AccessServerSDK\oblix\lib` to your Environment Variables

If this setting is not there, Adaptive Strong Authenticator will not be able to identify the AccessGate libraries during startup.

8.15 Configure Oracle Access Manager Domain to use Adaptive Strong Authenticator Authentication

The Adaptive Strong Authenticator authentication should now be operable for Oracle Access Manager policy domains. Please modify your application Oracle Access Manager policy domain to use the Adaptive Strong Authenticator authentication scheme (Adaptive Strong Authentication).

Steps

1. Log in to the Oracle Access Manager host. For example, `http://<hostname>/access/oblix`.

2. Click **Policy Manager**.
3. Log in as an admin user
4. Click **My Policy Domains**
5. Click **<ApplicationPolicy >**.
6. Click **Default Rules**.
7. Click **Modify**
8. From the Authentication Scheme drop-down selector, select **Adaptive Strong Authentication**.
9. Click **OK** to confirm the change in authentication schemes.
10. Ensure that **Update Cache** is checked.
11. Click **Save**.
12. Close Internet Explorer.

8.16 Testing Oracle Adaptive Access Manager-Oracle Access Manager Integration

To test the configuration, try accessing your application. The Oracle Access Manager will intercept your un-authenticated request and redirect you to the Adaptive Strong Authenticator to challenge for credentials.

Part IV

Features Integrations

Part IV contains the following chapters:

9 Auto-learning

Auto-learning is the application of a number of Oracle Adaptive Access Manager features to dynamically profile behavior of user, device, locations, and transaction entities. Patterns are defined by an administrator to automatically capture behavior. These patterns are in turn used by Adaptive Risk Manager to dynamically create and populate buckets based on the pattern parameters. Adaptive Risk Manager automatically records/maintains the bucket memberships of the users/devices/locations/entities over time so that the overall profile can be used to evaluate risk. As well, dynamic actions are used to populate groups based on rule outcomes to further profile behavior. The memberships of these automatically managed groups are also used to evaluate risk.

The Auto-learning feature profiles transactions and authentications being performed by different actors (entities). This process establishes what is normal or average behavior for an individual or a population. In turn this allows evaluations to be made that can determine if a situation is an anomaly and therefore potentially fraudulent. The task is accomplished by

- capturing the transaction and authentication data and passing it through various patterns thereby creating/populating various buckets to profile behavior in a granular way.

  An example scenario is when we want to find out what the typical log in times (time of the day) are for users. To find out the login times, we would set up an Auto-learning pattern that has the user as a member and time as a parameter. We would choose to have a multi-bucket pattern and set start time=00 and endtime=23 (these are hours of the day).

  When users log in, we will have profiles for each user on log in time. When we gather this data for a few days or months, we can use this data to challenge the user if he logs in at a time at which he does not usually log in.

- capturing the behavioral and transaction data, based on the actors (entities), and then creating the statistics for the entities based on their memberships to various patterns and hour/day/month/year time samples.

  An example: we will have several patterns in the system.

  – one capturing the login name (user) and the time of log in

  – another capturing the login device and the city of the log in

  When the data is analyzed and tabulated, we will have created records to tell us

  – user "U1" exhibited the behavior of Pattern A. The user "U1" is a member of Pattern A

  - if the user, or IP, or device exhibited a behavior that matched the configured pattern, then that user (entity) is a member of the pattern.

This document provides information on pattern data processing and the APIs for triggering pattern data processing.

9.1 Pattern Data Processing (On-Line and Scheduled)

If the system load is light and if the pattern is configured, the data will be processed as soon as the clients calls the API that is used for triggering the data processing. The system load is the number of authentication, transaction, rule processing (and other) reports and requests served by the Oracle Adaptive Access Manager server.

The logic for processing the data is as follows.

For each (successful) transaction record, the following process occurs:

1. Get all the attributes of the transaction from the database.
2. Determine the transaction type and if any of the patterns have the same transaction type as the one we have at hand.
3. If there are no patterns having the same transaction type as the one at hand, the process is stopped at this point and returns to the caller with nothing.
4. If there are patterns that have the same transaction type as the one at hand, then the following process is performed for each pattern.
   a. Get the parameters for that pattern and determine if the parameter values for the transaction at hand satisfy the requirements (like range for example). If not, move to next pattern.
   b. If the parameters satisfy the requirements, then go to the fingerprint table.
   c. If the fingerprint exists for such a combination, then go ahead and update the counters in workflow tables (hour, day, month, year).
   d. If the fingerprint does not exist, then create a fingerprint and create entries in the workflow table for that fingerprint and put the count there.
   e. After this determine if the pattern is configured to capture the one-time or lifetime values for the parameters, if set to do so. Then go and update the correct profile table. While doing this, if the profile table does not have an entry for this entity, create the entry. Data1 thru Data10 fields from entity profile tables will be used to capture the pattern membership and the values.
   f. Repeat steps a through e for rest of the patterns.
5. Repeat steps 1 through 4 for each transaction.

9.2 APIs for Triggering Pattern Data Processing

The APIs for triggering patterning data processing are

- updateTransactionStatus
- updateAuthStatus
- processPatternAnalysis

9.2.1 updateTransactionStatus

API to update the given transaction status and trigger the pattern data analysis if appropriate. A nonzero value of analyzePatterns will result in triggering the pattern analysis if not already done for this transaction.

- public VCryptResponse updateTransactionStatus(java.lang.String requestId, long transactionId, int status, boolean analyzePatterns)
- public VCryptResponse updateTransactionStatus(java.lang.String requestId, java.util.Date requestTime, long transactionId, int status, java.util.Map[] contextMap, boolean analyzePatterns)

Table 9–1 updateTransactionStatus Parameters

Parameter	Description
requestId	Request Identifier
requestTime	The time when this request was made.
transactionId	Transaction Id to be updated.
status	New Status
contextMap	array of contextMap
analyzePatterns	Boolean to indicate if the pattern analysis should be done. When passed in as true the pattern analysis is done for this transaction.

9.2.2 updateAuthStatus

API to update the user node log auth status and trigger the pattern data analysis if appropriate. A nonzero value of analyzePatterns will result in triggering the pattern analysis if not already done for this transaction.

- public VCryptResponse updateAuthStatus(java.lang.String requestId, int resultStatus, int clientType, java.lang.String clientVersion, boolean analyzePatterns)
- public VCryptResponse updateAuthStatus(java.lang.String requestId, java.util.Date requestTime, int resultStatus, int clientType, java.lang.String clientVersion, boolean analyzePatterns)

Table 9–2 updateAuthStatus Parameters

Parameter	Description
requestId	request Id
requestTime	Time of update
resultStatus	The authentication result. This is the enumeration value of the authentication result.
clientType	This is an enum value defined to identify the client type used for authentication.
clientVersion	Optional parameter to specify the version of the client used
analyzePatterns	Boolean to indicate if the pattern analysis should be done. When passed in as true the pattern analysis is done for this transaction.

9.2.3 processPatternAnalysis

API to trigger the processing of data for pattern matching. This call will only trigger the processing of data for pattern matching. The last parameter transactionType can be used by authentication type user interactions, since auth (or login) are not first-class transactions.

public VCryptResponse processPatternAnalysis(java.lang.String requestId, long transactionId, int status, java.lang.String transactionType)

Table 9–3 processPatternAnalysis

Parameter	Description
requestId	request Id
transactionId	Transaction Id to be updated.
status	New Status
transactionType	String that indicates the type of transaction. Has to be "auth" for authentication type. For other types it can be "bill_pay, ...",; basically the type name of the transaction.

10 Configurable Actions

Oracle Adaptive Access Manager provides Configurable Actions, a feature which allows users to create new supplementary actions that are triggered based on the result action and/or based on the risk scoring after a Runtime execution. This section describes how to integrate a Configurable Action with the Adaptive Risk Manager software.

10.1 Integration

To add a new Configurable Action, please perform the tasks listed below.

1. Develop the Configurable Action by implementing the com.bharosa.vcrypt.tracker.dynamicactions.intf.DynamicAction java interface.

   Note: In this step, implementing means writing java code based on the contract specified by the Java interface com.bharosa.vcrypt.tracker.dynamicactions.intf.DynamicAction.

   While implementing the com.bharosa.vcrypt.tracker.dynamicactions.intf.DynamicAction java interface, the following two methods have to be coded:

   - getParameters() - In this method, the code has to be written that returns the parameters used by the Configurable Action. Make sure that the size of the parameters array returned is the same as the number of parameters. Look at the sample configurable actions java code in Sample application.
   - execute() - In this method, code has to be written that performs the logic required by the Configurable Action. Configurable Action parameter values are passed in actionParamValueMap where the parameter name is the key and the RuntimActionParamValue object is the value. Use the appropriate getXXXValue() method to get the parameter value.

2. Make sure the java code is JDK 1.4-compliant. Do not use any language features of JDK 1.5 or JDK 6.

   For sample code, look at the java files in the sample application.

3. Test the implementation of the Configurable Action thoroughly.

   Since Configurable Actions are standalone java classes, they can be tested with Unit Testing Methodology using JUnit framework.

   For sample JUnit code for testing dynamic action, refer to the "Sample JUnit Code" section.

4. Compile the java class and create a jar file of the compiled class files.
5. Copy the jar file along with any other required libraries to the classpath of the Adaptive Risk Manager Web application.
6. Restart the Adaptive Risk Manager application if required.
7. Log into Adaptive Risk Manager web application and create an action definition entry for the newly deployed Configurable Action.
8. Make sure all the parameters required for the Configurable Action are displayed in the user interface.
9. Use the newly available Configurable Action by adding it to the required runtimes. For more information on configuring Configurable Actions, please refer to the *Oracle Adaptive Access Manager Administrator's Guide*.

10.2 Executing Configurable Actions in a Particular Order and Data Sharing

Configurable Actions can be used to implement chaining in such a way that

- they execute in a particular order
- data can be shared across these actions

Note: Sharing data across Configurable Actions involves writing java code and requires more effort than just a configuration task.

To be able to execute Configurable Actions in a particular order and share data:

1. Configure Configurable Actions as synchronous actions with the required order of execution in ascending order.

   Note: A Configurable Action is executed only if the trigger criteria is met; therefore, make sure the trigger criteria is correct.

2. To share data, insert the data into the actionContextMap parameter of the Configurable Action's execute() method. Since the actionContextMap is a Map, it requires a key and value pair that represents the data to be shared.

   Note: it is the implementer's responsibility to ensure that

   - the duplicate keys are not used while inserting data
   - the same key is used when trying to access this shared data from another Configurable Action.

3. Ensure that the code can handle the case where the key is not present in the actionContextMap. This step must be performed to avoid errors or NullPointerException when the other action do not insert the value into the actionContextMap.

10.3 How to Test Configurable Actions Triggering

1. Make sure there is a way to identify if the code in the Configurable Action is executed. This could be as simple as an entry in log file or an entry in database and so on
2. Set the log-level for "com.bharosa.vcrypt.tracker.dynamicactions.ActionExecutor" to "DEBUG" using the Environment->Logging screen.
3. Create an action template for the given Configurable Action.
4. Add the action to a Pre-Authentication runtime with trigger criteria as score between 0 and 1000.
5. Try logging in from the Adaptive Strong Authenticator application or the sample application that is connected to Adaptive Risk Manager.
6. Check the Adaptive Risk Manager logs for the entry "Enter: executeAction(): Executing Action Instance" from the class "com.bharosa.vcrypt.tracker.dynamicactions.ActionExecutor."
7. If there is no error then you will see a related log statement like "Exit: executeAction(): Action Instance".
8. If there is an error, you will see a log statement like "Error: executeAction()"
9. Apart from these, check for a log entry or a database entry that is created by the Configurable Action itself

10.4 Sample JUnit Code

The sample JUnit code for testing dynamic action is provided below:

```
public class TestDynamicActionsExecution extends TestCase {
        static Logger logger =
Logger.getLogger(TestDynamicActionsExecution.class);
        private DynamicAction caseCreationAction = null;


        public void setUp()throws Exception {
                caseCreationAction = new CaseCreationAction();
    }

        public void testDynamicAction() {

                //RequestId
                String requestId = "testRequest";

                //Request Time
                Date requestTime = new Date();

                //Map that contains values passed to the rule/model execution
                Map ruleContextMap = new HashMap();

                //Result from rule execution
                VCryptRulesResultImpl rulesResult = new VCryptRulesResultImpl();
                rulesResult.setResult("Allow");
                rulesResult.setRuntimeType(new Integer(1));

                //Configurable action's parameter values
                Map actionParamValueMap = new HashMap();
                RuntimeActionParamValue caseTypeParamValue = new
RuntimeActionParamValue();
                caseTypeParamValue.setIntValue(CaseConstants.CASE_AGENT_TYPE);

                RuntimeActionParamValue caseSeverityParamValue = new
RuntimeActionParamValue();
                caseSeverityParamValue.setIntValue(1);

                RuntimeActionParamValue caseDescriptionParamValue = new
RuntimeActionParamValue();
                caseDescriptionParamValue.setStringValue("Testing CaseCreation
Action");

                //ActionContext Map for passing data to/from the dynamic action
execution
                Map actionContextMap = new HashMap();

                //Execute the action
                try {
                        caseCreationAction.execute(requestId, requestTime,
ruleContextMap, rulesResult, actionParamValueMap, actionContextMap);
                }catch(Exception e) {
                        Assert.fail("Exception occurred while executing dynamic
action");
                        logger.error("Exception occcurred while executing dynamic
action", e);
                }

                //Write appropriate asserts to check if the configurable action
has executed properly
        }

    public void tearDown() throws Exception {

    }
}
```

11

Configuring Expiry/Overdue for Cases

The expiry/overdue behavior can be configured using the customercare.properties file, located in the WEB-INF\classes folder of Adaptive Risk Manager.

The default setting is for CSR cases to "expire" after 24 hours and Agent cases to become "Overdue" after 24 hours. Once a CSR case expires, CSR Agents cannot access them. CSR Managers have to extend the expiration time so that CSR Agents can access them. Agent Users and their managers can access Agent cases even if they are Overdue. Once accessed the "Overdue" time is automatically increased by 24 hours.

Information to change the default behavior is provided below.

11.1 CSR Cases

The CSR Cases section contains information to

- Set the "Expiry" Behavior for CSR Cases (Default Setting)
- Disable "Expiry/Overdue" Behavior for CSR Cases
- Set "Overdue" Behavior for CSR Cases

11.1.1 Set the "Expiry" Behavior for CSR Cases (Default Setting)

To set "expiry"behavior for CSR cases (default setting), modify the following properties as shown below.

```
customercare.case.expirybehavior.enum.csrcase.behavior = expiry
customercare.case.expirybehavior.enum.csrcase.label = Expired
customercare.case.expirybehavior.enum.csrcase.durationInHrs = 24
customercare.case.expirybehavior.enum.csrcase.resetonaccess = false
```

11.1.2 Disable "Expiry/Overdue" Behavior for CSR Cases

To disable the "expiry/overdue" behavior for CSR cases, modify the following property as shown below.

```
customercare.case.expirybehavior.enum.csrcase.behavior = none
```

Note: You will not need to change the other parameters.

11.1.3 Set "Overdue" Behavior for CSR Cases

To set "overdue" behavior for CSR cases, modify the following properties as shown below.

```
customercare.case.expirybehavior.enum.csrcase.behavior = overdue
customercare.case.expirybehavior.enum.csrcase.label = Overdue
customercare.case.expirybehavior.enum.csrcase.durationInHrs = 24
customercare.case.expirybehavior.enum.csrcase.resetonaccess = true
```

11.2 Agent Cases

The Agent Cases section contains information to

- Set "Overdue" Behavior for Agent Cases (Default Setting)
- Disable "Overdue/Expiry" Behavior for Agent Cases
- Set "Expiry" Behavior for Agent Cases

11.2.1 Set "Overdue" Behavior for Agent Cases (Default Setting)

To set "overdue" behavior for Agent cases, modify the following properties as shown below.

```
customercare.case.expirybehavior.enum.agentcase.behavior = overdue
customercare.case.expirybehavior.enum.agentcase.label = Overdue
customercare.case.expirybehavior.enum.agentcase.durationInHrs = 24
customercare.case.expirybehavior.enum.agentcase.resetonaccess = true
```

11.2.2 Disable "Overdue/Expiry" Behavior for Agent Cases

To disable the "overdue/expiry" behavior for Agent cases, modify the following property as shown below.

```
customercare.case.expirybehavior.enum.agentcase.behavior = none
```

Note: You will not need to change the other parameters.

11.2.3 Set "Expiry" Behavior for Agent Cases

To set "expiry"behavior for Agent cases, modify the following properties as shown below.

```
customercare.case.expirybehavior.enum.agentcase.behavior = expiry
customercare.case.expirybehavior.enum.agentcase.label = Expired
customercare.case.expirybehavior.enum.agentcase.durationInHrs = 24
customercare.case.expirybehavior.enum.agentcase.resetonaccess = false
```

Index