Capítulo 3. Zend_Auth

Traduzido por Flávio Gomes da Silva Lisboa

Sumário

3.1. Introdução

Zend_Auth fornece uma API para autenticação e inclui adaptadores de autenticação concretos para cenários de casos de uso comuns.

Zend_Auth trata somente de **autenticação** e não de **autorização**. Autenticação é vagamente definida como determinar se uma entidade efetivamente é o que afirma ser (isto é, identificação), baseado em um conjunto de credenciais. Autorização, o processo de decidir se permite a uma entidade acessar, ou executar operações sobre outras entidades, está fora do escopo de Zend_Auth. Para mais informações sobre autorização e controle de acesso com Zend Framework, por favor, veja Zend_Acl.

Nota

A classe `Zend_Auth` implementa o padrão de projeto Singleton – somente uma instância da classe está disponível – através do método estático `getInstance()`. Isso significa que usar o operador `new` e a palavra-chave `clone` não funcionará com a classe `Zend_Auth` class; use `Zend_Auth::getInstance()`.

3.1.1. Adaptadores

Um adaptador Zend_Auth é usado para autenticar contra um tipo particular de serviço de autenticação, tal como LDAP, RDBMS, ou armazenamento baseado em arquivos. Diferentes adaptadores costumam ter vastas opções e comportamentos diferentes, mas algums coisas básicas são comuns entre adaptadores de autenticação. Por exemplo, aceitar credenciais de autenticação (incluir uma identidade proposta), executar consultas contra o serviço de autenticação, e retornar resultados são comportamentos comuns aos adaptadores Zend_Auth.

Cada classe adaptadora Zend_Auth implementa `Zend_Auth_Adapter_Interface`. Essa interface define um método, `authenticate()`, que uma classe adaptadora deve implementar para executar uma consulta de autenticação. Cada classe adaptadora deve estar preparada previamente para chamar `authenticate()`. Tal preparação do adaptador inclui configurar credenciais (por exemplo, nome de usuário e senha) e definir valores para opções de configuração específicas de cada adaptador, tais como configurações de conexão de banco de dados para um adaptador de tabela de banco de dados.

A seguir há um exemplo de adaptador de autenticação que requer um nome de usuário e senha sejam configurados por autenticação. Outros detalhes, tais como a forma pela qual o serviço de autenticação é consultado, tem sido omitidos para brevidade:

```
<?php
require_once 'Zend/Auth/Adapter/Interface.php';
class MyAuthAdapter implements Zend_Auth_Adapter_Interface
{
    /**
     * Configura nome de usuário e senha para autenticação
     *
     * @return void
     */
    public function __construct($username, $password)
    {
        // ...
    }
    /**
     * Executa uma tentativa de autenticação
     *
     * @throws Zend_Auth_Adapter_Exception Se a autenticação não pode ser
executada
     * @return Zend_Auth_Result
     */
    public function authenticate()
    {
        // ...
    }
}
```

Como indicado no bloco de documentação, `authenticate()` deve retornar uma instância de `Zend_Auth_Result` (ou de uma classe derivada de `Zend_Auth_Result`). Se por alguma razão executar uma consulta de autenticação for impossível, `authenticate()` deve lançar uma exceção que deriva de `Zend_Auth_Adapter_Exception`.

3.1.2. Resultados

Adaptadores Zend_Auth retornam uma instância de `Zend_Auth_Result` com `authenticate()` de forma a representar os resultados de uma tentativa de autenticação. Adaptadores populam o objeto `Zend_Auth_Result` em construção, de modo que os seguintes quatro métodos forneçam um conjunto básico de operações do lado do usuário que são comuns para os resultados dos adaptadores Zend_Auth:

- `isValid()` - retorna true se e somente se o resultado representa uma tentativa de autenticação feita com sucesso.
- `getCode()` - retorna um identificador constante `Zend_Auth_Result` para determinar o tipo de falha de aurtenticação ou se ela tem ocorrido com sucesso. Isso pode ser usado em situações onde o desenvolvedor deseja distinguir entre diversos tipos de resultado de autenticação. Isso permite que os desenvolvedores mantenham estatísticas de resultado de autenticação detalhadas, por exemplo. Outro uso dessa característica é fornecer mensagens específicas e customizadas para usuários por razões de usabilidade, mesmo que os desenvolvedores sejam encorajados a considerar os riscos de fornecer tais razões detalhadas para os usuários, ao invés de uma mensagem de falha de autenticação geral. Para mais informações, veja as notas abaixo.
- `getIdentity()` - retorna a identidace de uma tentativa de autenticação
- `getMessages()` - retorna um array de mensagens a respeito de uma tentativa de autenticação falha.

Um desenvolvedor pode querer estender isso baseando-se no tipo de resultado da autenticação, de modo a executar operações mais específicas. Alguns desenvolvedores podem achar útil bloquear contas depois de várias tentativas infrutíferas de acesso, sinalizar um endereço IP depois de várias tentativas de validar uma identidade não existente e fornecer, especificamente, mensagens de resultado de autenticação customizadas para o usuário. Os seguintes códigos de resultado estão disponíveis:

```
Zend_Auth_Result::SUCCESS
Zend_Auth_Result::FAILURE
Zend_Auth_Result::FAILURE_IDENTITY_NOT_FOUND
Zend_Auth_Result::FAILURE_IDENTITY_AMBIGUOUS
Zend_Auth_Result::FAILURE_CREDENTIAL_INVALID
Zend_Auth_Result::FAILURE_UNCATEGORIZED
```

O seguinte exemplo ilustra como um desenvolvedor pode estender o código de resultado:

```
<?php
// dentro de AuthController / loginAction
$result = $this->_auth->authenticate($adapter);
switch ($result->getCode()) {
    case Zend_Auth_Result::FAILURE_IDENTITY_NOT_FOUND:
        /** faz alguma coisa para identidade não existente **/
        break;
```

```
    case Zend_Auth_Result::FAILURE_CREDENTIAL_INVALID:
        /** faz alguma coisa para credencial inválida **/
        break;
    case Zend_Auth_Result::SUCCESS:
        /** faz alguma coisa para autenticação realizada com êxito **/
        break;
    default:
        /** faz alguma coisa para outra falha **/
        break;
}
```

3.1.3. Persistência de Identidade

Autenticar uma requisição que inclui credenciais de autenticação é útil por si só, mas também é importante suportar a manutenção da identidade autenticada sem ter de apresentar as credenciais de autenticação a cada requisição.

HTTP é um protocolo sem estados, entretanto, e técnicas tais como cookies e sessões foram desenvolvidas de modo a facilicar a manutenção de estados ao longo de múltiplas requisições em aplicações web do lado do servidor.

3.1.3.1. Persistência Padrão na Sessão PHP

Por padrão, `Zend_Auth` fornece armazenamento persistente da identidade de uma tentativa de autenticação frutífera usando a sessão PHP. Sobre uma tentativa frutífera de autenticação, `Zend_Auth::authenticate()` armazena a identidade do resultado de autenticação dentro de um armazenamento persistente. A menos que a configuração diga o contrário, `Zend_Auth` usa uma classe de armazenagem chamada `Zend_Auth_Storage_Session`, que, por sua vez, usa Zend_Session. Uma classe customizada pode, ao invés disso, ser usada para fornecer um objeto que implemente `Zend_Auth_Storage_Interface` para `Zend_Auth::setStorage()`.

Nota

Se o armazenamento persistente automático da identidade não for apropriado para um caso de uso particular, então os desenvolvedores podem desistir do uso da classe `Zend_Auth` completamente, ao invés de usar uma classe adaptadora diretamente.

Example 3.1. Modificar o Namespace da Sessão

`Zend_Auth_Storage_Session` usa um namespace de sessão de `'Zend_Auth'`. Esse namespace pode ser sobrescrito pela passagem de valor diferente para o construtor de `Zend_Auth_Storage_Session`, e esse valor é internamente passado para o construtor de `Zend_Session_Namespace`. Isso deve ocorrer antes da tentativa de autenticação, uma vez que `Zend_Auth::authenticate()` executa o armazenamento automático da identidade.

```
<?php
// Grava uma referência para instância Singleton de Zend_Auth
require_once 'Zend/Auth.php';
$auth = Zend_Auth::getInstance();
// Usa 'someNamespace' ao invés de 'Zend_Auth'
require_once 'Zend/Auth/Storage/Session.php';
$auth->setStorage(new Zend_Auth_Storage_Session('someNamespace'));
/**
 * @todo Configue o adaptador de autenticação, $authAdapter
```

```
 */
// Autentica, grava o resultado e persiste a identidade em caso de sucesso
$result = $auth->authenticate($authAdapter);
```

3.1.3.2. Implementando Armazenamento Customizado

Algumas vezes os desenvolvedores podem necessitar do uso de diferentes comportamentos de persistência de identidade do que os fornecidos por `Zend_Auth_Storage_Session`. Para tais casos os desenvolvedores podem simplesmente implementar `Zend_Auth_Storage_Interface` e fornecer uma instância da classe para `Zend_Auth::setStorage()`.

Exemplo 3.2. Usando uma Classe de Armazenagem Customizada

De modo a usar uma casse de armazenamento de persistência de identidade que não seja `Zend_Auth_Storage_Session`, um desenvolvedor implementa `Zend_Auth_Storage_Interface`:

```
<?php
require_once 'Zend/Auth/Storage/Interface.php';
class MyStorage implements Zend_Auth_Storage_Interface
{
    /**
     * Retorna true se e somente se o armazenamento está vazio
     *
     * @throws Zend_Auth_Storage_Exception Se é impossível determinar se o
armazenamento está vazio
     * @return boolean
     */
    public function isEmpty()
    {
        /**
         * @todo implementação
         */
    }
    /**
     * Retorna o conteúdo do armazenamento
     *
     * O comportamento é indefinido quando o armazenamento está vazio.
     *
     * @throws Zend_Auth_Storage_Exception Se a leitura do conteúdo do
armazenamento for impossível
     * @return mixed
     */
    public function read()
    {
        /**
         * @todo implementação
         */
    }
    /**
     * Escreve $contents no armazenamento
     *
     * @param  mixed $contents
     * @throws Zend_Auth_Storage_Exception Se escrever $contents no
```

```
armazenamento for impossível
     * @return void
     */
    public function write($contents)
    {
        /**
         * @todo implementação
         */
    }
    /**
     * Limpa o conteúdo do armazenamento
     *
     * @throws Zend_Auth_Storage_Exception Se a limpeza do conteúdo do
armazenamento for impossível
     * @return void
     */
    public function clear()
    {
        /**
         * @todo implementation
         */
    }
}
```

De modo a usar essa classe de armazenagem customizada, `Zend_Auth::setStorage()` é invocado antes de uma tentativa de consulta de autenticação:

```
<?php
// Instrui Zend_Auth a usar a classe de armazenagem customizada
Zend_Auth::getInstance()->setStorage(new MyStorage());
/**
 * @todo Configure o adaptador de autenticação, $authAdapter
 */
// Autentica, grava o resultado, e persiste a identidade em caso de sucesso
$result = Zend_Auth::getInstance()->authenticate($authAdapter);
```

3.1.4. Usando Zend_Auth

Há dois modos possíveis de usar adaptadores Zend_Auth:

1. indiretamente, através de `Zend_Auth::authenticate()`
2. diretamente, através do método `authenticate()` do adaptador

O seguinte exemplo ilustra como usar um adaptador Zend_Auth indiretamente, por meio do uso da classe `Zend_Auth`:

```
<?php
// Obtém uma referência a instância Singleton de Zend_Auth
require_once 'Zend/Auth.php';
$auth = Zend_Auth::getInstance();
// Configura o adaptador de autenticação
$authAdapter = new MyAuthAdapter($username, $password);
// Tenta autenticar, gravando o resultado
```

```
$result = $auth->authenticate($authAdapter);
if (!$result->isValid()) {
    // Falha na autenticação; imprime as razões da mesma
    foreach ($result->getMessages() as $message) {
        echo "$message\n";
    }
} else {
    // Autenticação realizada com êxito; a identidade ($username) é armazenada
na sessão
    // $result->getIdentity() === $auth->getIdentity()
    // $result->getIdentity() === $username
}
```

Uma vez que a autenticação tenha sido feita em uma requisição, como no exemplo acima, é uma simples questão de verificar se uma identidade autenticada com sucesso existe:

```
<?php
$auth = Zend_Auth::getInstance();
if ($auth->hasIdentity()) {
    // Identidade existe; captura-a
    $identity = $auth->getIdentity();
}
```

Para remover uma identidade de um armazenamento persistente, simplesmente use o método `clearIdentity()`. Isso tipicamente seria usado para implementar uma operação de "logout" na aplicação:

```
<?php
Zend_Auth::getInstance()->clearIdentity();
```

Quando o uso automático da armazenagem persistente for inapropriada para um caso de uso particular, um desenvolvedor pode simplesmente passar adiante o uso da classe `Zend_Auth`, usando uma classe adaptadora diretamente. O uso direto de uma classe adaptadora envolve a configuração e preparo de um objeto adaptador e por fim a chamada a seu método `authenticate()`. Detalhes específicos de adaptador são discutidos na documentação para cada adaptador. O seguinte exemplo utiliza diretamente `MyAuthAdapter`:

```
<?php
// Configura o adaptador de autenticação
$authAdapter = new MyAuthAdapter($username, $password);
// Tenta autenticar, gravando o resultado
$result = $authAdapter->authenticate();
if (!$result->isValid()) {
    // Falha na autenticação; imprime as razões da mesma
    foreach ($result->getMessages() as $message) {
        echo "$message\n";
    }
} else {
    // Autenticação realizada com êxito
    // $result->getIdentity() === $username
}
```

3.2. Autenticação de Tabela de Banco de Dados

3.2.1. Introdução

`Zend_Auth_Adapter_DbTable` fornece a habilidade de autenticar contra credenciais armazenadas em uma tabela de banco de dados. Como `Zend_Auth_Adapter_DbTable` requer que uma instância de `Zend_Db_Adapter_Abstract` seja passada para seu construtor, cada instância é ligada a uma conexão de banco de dados particular. Outras opções de configuração podem ser configuradas através do construtor e através de métodos de instância, um para cada opção.

As opções de configuração disponíveis incluem:

- `tableName`: Este é o nome da tabela do banco de dados que contém as credenciais de autenticação, e contra a qual a consulta de autenticação do banco de dados é executada.
- `identityColumn`: Este é o nome da coluna da tabela do banco de dados usada para representar a identidade. A coluna de identidade deve conter valores únicos, tais como nome de usuário ou endereço de e-mail.
- `credentialColumn`: Este é o nome da coluna da tabela do banco de dados usada para representar a credencial. Sob um simples esquema de autenticação de identidade e senha, o valor da credencial corresponde à senha. Veja a opção `credentialTreatment`.
- `credentialTreatment`: Em muitos casos, senhas e outros dados sensíveis são encriptados, embaralhados, codificados, obscurecidos, ou em outros casos tratados através de alguma função ou algoritmo. Pela especificação de um literal de tratamento parametrizado com este método, tal como `'MD5(?)'` ou `'PASSWORD(?)'`, um desenvolvedor pode aplicar tal SQL arbitrário sobre dados de credencial de entrada. Uma vez que essas funções são específicas para o RDBMS subjacente, verifique o manual do banco de dados para a disponibilidade de tais funções para seu sistema de banco de dados.

Exemplo 3.3. Uso Básico

Como explicado na introdução, o construtor de `Zend_Auth_Adapter_DbTable` requer uma instância de `Zend_Db_Adapter_Abstract` que serve como conexão de banco de dados para a qual a instância do adaptador de autenticação está associada. Primeiro, a conexão com o banco de dados deve ser criada.

O seguinte código cria um adaptador para um banco de dados em memória, cria um esquema de tabela simples, e insere uma linha contra a qual nós podemos executar uma consulta de autenticação mais tarde. Esse exemplo requer que a extensão PDO SQLite esteja disponível:

```
<?php
// Cria uma conexão de banco de dados SQLite em memória
require_once 'Zend/Db/Adapter/Pdo/Sqlite.php';
$dbAdapter = new Zend_Db_Adapter_Pdo_Sqlite(array('dbname' => ':memory:'));
// Constrói uma simples consulta de criação de tabela
$sqlCreate = 'CREATE TABLE [users] ( '
           . '[id] INTEGER  NOT NULL PRIMARY KEY, '
           . '[username] VARCHAR(50) UNIQUE NOT NULL, '
           . '[password] VARCHAR(32) NULL, '
           . '[real_name] VARCHAR(150) NULL)';
// Cria a tabela de credenciais de autenticação
$dbAdapter->query($sqlCreate);
// Constrói uma consulta para inserir uma linha para   a qual a autenticação
```

```
possa ter  êxito
$sqlInsert = 'INSERT INTO users (username, password, real_name) '
           . 'VALUES ("my_username", "my_password", "My Real Name")';
// Insere os dados
$dbAdapter->query($sqlInsert);
```

Com a conexão de banco de dados e os dados da tabela disponíveis, uma instância de `Zend_Auth_Adapter_DbTable` pode ser criada. Os valores de opção de configuração podem ser passados para o construtor ou transferidos como parâmetros para configurar métodos depois da instanciação:

```
<?php
require_once 'Zend/Auth/Adapter/DbTable.php';
// Configura a instância com os parâmetros do construtor...
$authAdapter = new Zend_Auth_Adapter_DbTable($dbAdapter, 'users', 'username', 'p
assword');
// ...ou configura a instância com métodos setter
$authAdapter = new Zend_Auth_Adapter_DbTable($dbAdapter);
$authAdapter->setTableName('users')
            ->setIdentityColumn('username')
            ->setCredentialColumn('password');
```

Neste ponto, a instância do adaptador de autenticação está pronta para aceitar consultas de autenticação. De modo a formular uma consulta de autenticação, os valores da credencial de entrada são passados para o adaptador precedente para chamara o método `authenticate()`:

```
<?php
// Configura os valores de credencial de entrada (por exemplo, de um formulário
de login)
$authAdapter->setIdentity('my_username')
            ->setCredential('my_password');
// Executa a consulta de autenticação, gravando o resultado
$result = $authAdapter->authenticate();
```

Em adição a disponibilidade do método `getIdentity()` sobre o objeto de resultado de autenticação, `Zend_Auth_Adapter_DbTable` também suporta a a recuperação de linha de tabela sobre autenticação feita com êxito.

```
<?php
// Imprime a identidade
echo $result->getIdentity() . "\n\n";
// Imprime a linha de resultado
print_r($authAdapter->getResultRowObject());
/* Saída:
my_username
Array
(
    [id] => 1
    [username] => my_username
    [password] => my_password
    [real_name] => Meu Nome Real
)
*/
```

Uma vez que a linha da tabela contém o valor da credencial, é importante preservar os valores contra acessos não intencionados.

3.2.2. Uso Avançado: Persistindo um Objeto DbTable Result

Por padrão, `Zend_Auth_Adapter_DbTable` retorna a identidade fornecida anteriormente para o objeto de autenticação em caso de êxito na autenticação. Outro cenário de caso de uso, onde desenvolvedores querem guardar no mecanismo de armazenagem persistente de `Zend_Auth` um objeto de identidade contendo outras informações úteis, é solucionado pelo uso do método `getResultRowObject()` que retornam um objeto da classe `stdClass`. O seguinte trecho de código ilustra esse uso:

```
<?php
// Autenticação com Zend_Auth_Adapter_DbTable
$result = $this->_auth->authenticate($adapter);
if ($result->isValid()) {
    // Armazena a identidade como um objeto onde somente o username e o
real_name serão retornados
    $this->_auth->getStorage()->write($adapter-
>getResultRowObject(array('username', 'real_name')));
    // Armazena a identidade como um objeto onde a coluna de senha tem sido
omitida
    $this->_auth->getStorage()->write($adapter->getResultRowObject(null, 'passwo
rd'));
    /* ... */
} else {
    /* ... */
}
```

3.2.3. Uso Avançado Através de Exemplo

Enquanto o propósito primário de Zend_Auth (e conseqüentemente Zend_Auth_Adapter_DbTable) é primariamente **autenticação** e não **autorização**, há poucas instâncias e problemas que tocam a linha sobre que domínio se encaixa dentro de qual. Dependendo de como você decidiu explicar seu problema, algumas vezes faz sentido resolver o que parece um problema de autorização dentro de um adaptador de autenticação.

Com essa pequena desaprovação fora do caminho, Zend_Auth_Adapter_DbTable tem construído algo em mecanismos que podem ser alavancados para adicionar verificações adicionais em tempo de autenticação para resolver alguns problemas comuns do usuário.

```
<?php
// O valor do campo status de uma conta não é igual a "compromised"
$adapter = new Zend_Auth_Adapter_DbTable($db, 'users', 'username', 'password', '
MD5(?) AND status != "compromised"');
// O valor do campo ativo de uma conta é igual a "TRUE"
$adapter = new Zend_Auth_Adapter_DbTable($db, 'users', 'username', 'password', '
MD5(?) AND active = "TRUE"');
```

3.3. Autenticação Sumária

3.3.1. Introdução

Autenticação sumária é um método de autenticação HTTP que melhora a autenticação básica fornecendo um modo de autenticar sem ter de transmitir a senha em texto claro através da rede.

Este adaptador permite autenticação contra arquivos de texto contendo linhas com elementos básicos de autenticação sumária:

- nome de usuário, tal como "`joe.user`"
- domínio, tal como "`Administrative Area`"
- hash MD5 do nome de usuário, domínio e senha, separados por dois pontos

Os elementos acima são separados por dois pontos, como no seguinte exemplo (no qual a senha é "`somePassword`"):

```
someUser:Some Realm:fde17b91c3a510ecbaf7dbd37f59d4f8
```

3.3.2. Especificação

O adaptador de autenticação sumária, `Zend_Auth_Adapter_Digest`, requer vários parâmetros de entrada:

- filename – nome do arquivos contra o qual as consultas de autenticação são executadas
- realm – domínio de autenticação sumária
- username – usuário de autenticação sumária
- password – senha para o usuário do domínio

Esses parâmetros devem ser configurados previamente para chamar `authenticate()`.

3.3.3. Identidade

O adaptador de autenticação sumária retorna um objeto `Zend_Auth_Result`, que foi populado com a identidade como um vetor que tem como chaves `realm` e `username`. O respectivos valores do vetor associados com essas chaves correspondem aos valores configurados antes de `authenticate()` ser chamado.

```
<?php
require_once 'Zend/Auth/Adapter/Digest.php';
$adapter = new Zend_Auth_Adapter_Digest($filename, $realm, $username, $password)
;
$result = $adapter->authenticate();
$identity = $result->getIdentity();
print_r($identity);
/*
Array
(
    [realm] => Some Realm
    [username] => someUser
)
*/
```

3.4. Adaptador de Autenticação HTTP

3.4.1. Introdução

`Zend_Auth_Adapter_Http` fornece uma implementação mais complacente de autenticação HTTP sumária, básica e RFC-2617. Autenticação sumária é um método de autenticação HTTP que melhora a autenticação básica por meio de um modo que autentica sem ter de transmitir a senha em texto claro pela rede.

Características Principais:

- Suporta tanto autenticação básica quanto sumária.
- O cliente pode responder com qualquer esquema que suportado.
- Suporta autenticação de proxy.
- Inclui suporte para autenticação contra arquivos de texto e fornece uma interface para autenticação contra outras fontes, tais como bancos de dados..

Há poucas características notáveis de RFC-2617 que não foram implementadas ainda:

- Rastreamento de nonces[1], que permitiria suporte para "seco", e proteção de ataque repetido ampliada.
- Autenticação com verificação de integridade, ou "auth-int".
- Cabeçalho de informações de autenticação HTTP.

3.4.2. Resumo do Projeto

Esse adaptador consiste de dois subcomponentes, a própria classe de autenticação HTTP, e os assim chamados "Dissolventes" (Resolvers). A classe de autenticação HTTP encapsula a lógica para conduzir tanto autenticação básica quanto sumária. Ela usa um Resolver para procurar uma identidade de cliente em algum depósito de dados (arquivo de texto por padrão), e recuperar as credenciais do depósito em questão. As credenciais "dissolvidas" são então comparadas aos valores submetidos pelo cliente para determinar se a autenticação obteve êxito..

3.4.3. Opções de Configuração

A classe `Zend_Auth_Adapter_Http` requer que um vetor de configuração seja passado para seu construtor. Há várias opções de configuração disponíveis, e algumas são requeridas:

Tabela 3.1. Opções de Configuração

Opção	Requerida	Descrição
`accept_schemes`	Sim	Determina quais esquemas de autenticação serão aceitos dos clientes. Deve ser uma lista separada por espaços contendo `'basic'` e/ou `'digest'`.
`realm`	Sim	Configura o domínio (realm) de autenticação; nomes

1 Informação que garante que a mensagem não foi enviada anteriormente e possui a característica de ser sempre diferente para cada solicitação de registro, sendo impossível (?) ser forjada por um intruso.

Opção	Requerida	Descrição
		de usuário devem ser únicos dentro de um domínio dado.
`digest_domains`	Sim, quando `'accept_schemes'` contém `'digest'`	Lista separada por espaços de URIs para os quais a mesma informação de autenticação é válida. As URIs não precisam apontar todas para o mesmo servidor.
`nonce_timeout`	Sim, quando `'accept_schemes'` contém `'digest'`	Configura o número de segundos para o qual um nonce é válido. Veja notas abaixo.
`proxy_auth`	Não	Desabilitada por padrão. Habilita a execução de executar autenticação de Proxy, ao invés da autenticação de servidor de origem normal.

Nota

A implementação atual de `nonce_timeout` tem alguns efeitos interessantes. Essa configuração supostamente determina o tempo de vida válido de um dado nonce, ou efetivamente por quanto tempo uma informação de autenticação de cliente é aceita. Atualmente, se for configurada para 3600 (por exemplo), fará com que o adaptador solicite as credenciais ao cliente a cada hora, na hora. Isso será resolvido em uma futura versão, uma vez que o rastreamento de nonces e suporte seco são implementados.

3.4.4. Separadores

O trabalho do separador é tomar um nome de usuário e um domínio, e retornar algum tipo de valor de credencial. Autenticação básica espera receber a versão codificada em Base64 da senha do usuário. Autenticação sumária espera receber um hash (embaralhamento) do nome de usuário, do domínio, e de sua senha (cada um separado por dois pontos). Atualmente, o único algoritmo de hash suportado é MD5.

`Zend_Auth_Adapter_Http` confia na implementação de objetos `Zend_Auth_Adapter_Http_Resolver_Interface`. Uma classe separadora de arquivo de texto é incluída com este adaptador, mas qualquer outro tipo de separador pode ser criado pela simples implementação da interface separadora.

3.4.4.1. Separador de Arquivo

O separador de arquivo é uma classe muito simples. Ela tem uma propriedade simples especificando um nome de arquivo, que pode também ser passada para o construtor. Seu método `resolve()` caminha através do arquivo de texto, procurando por uma linha com um nome de usuário e domínio que se encaixem. O formato do arquivo de texto é similar aos arquivos htpasswd do Apache:

```
<username>:<realm>:<credentials>\n
```

Cada linha consiste de três campos – nome de usuário, domínio, e credenciais – cada um separado por dois pontos. O campo credenciais é obscuro para o separador de arquivo; ele simplesmente

retorna aquele valor como é para o chamador. Portanto, esse mesmo formato de arquivo serve tanto autenticação básica quanto sumária. Na autenticação básica, o campo credenciais deve ser a codificação Base64 da senha do usuário. Na autenticação sumária, ele deve ser o hash MD5 descrito acima.

Há dois modos igualmente fáceis de criar um separador de arquivo:

```
<?php
$path     = 'files/passwd.txt';
$resolver = new Zend_Auth_Adapter_Http_Resolver_File($path);
```

ou

```
<?php
$path     = 'files/passwd.txt';
$resolver = new Zend_Auth_Adapter_Http_Resolver_File();
$resolver->setFile($path);
```

Se o caminho dado estiver vazio ou não legível, uma exceção é lançada.

3.4.5. Uso Básico

Primeiro, configure um vetor com os valores da configuração requerida:

```
<?php
$config = array(
    'accept_schemes' => 'basic digest', // autenticação básica e sumária
    'realm'          => 'My Web Site',  // domínio de autenticação
    'digest_domains' => '/members_only /my_account',
    'nonce_timeout'  => 3600,
);
```

Esse vetor fará com que o adaptador aceite autenticação básica ou sumária, e requeira acesso autenticado a todas as áreas do site debaixo de `/members_only` e `/my_account`. O valor do domínio de autenticação é geralmente exibido pelo navegador na caixa de diálogo de senha. O `nonce_timeout`, naturalmente, comporta-se como descrito acima.

A seguir, cria o objeto Zend_Auth_Adapter_Http:

```
<?php
require_once 'Zend/Auth/Adapter/Http.php';
$adapter = new Zend_Auth_Adapter_Http($config);
```

Uma vez que nós suportamos tanto autenticação básica quanto sumária, precisamos de dois diferentes objetos separadores. Note que isso poderia facilmente ser duas classes diferentes:

```
<?php
require_once 'Zend/Auth/Adapter/Http/Resolver/File.php';
$basicResolver = new Zend_Auth_Adapter_Http_Resolver_File();
$basicResolver->setFile('files/basicPasswd.txt');
$digestResolver = new Zend_Auth_Adapter_Http_Resolver_File();
```

```
$digestResolver->setFile('files/digestPasswd.txt');
$adapter->setBasicResolver($basicResolver);
$adapter->setDigestResolver($digestResolver);
```

Finalmente, nós executamos a autenticação. O adaptador precisa de uma referência para ambos objetos de requisição e resposta de modo a realizar seu trabalho:

```
<?php
assert($request instanceof Zend_Controller_Request_Http);
assert($response instanceof Zend_Controller_Response_Http);
$adapter->setRequest($request);
$adapter->setResponse($response);
$result = $adapter->authenticate();
if (!$result->isValid()) {
    // Nome de usuário /senha "ruim" ou solicitação de senha cancelada
}
```

3.5. Autenticação Open ID[2]

3.5.1. Introdução

`Zend_Auth_Adapter_OpenId` permite ao usuário se autenticar usando um servidor remoto OpenID. Tal processo de autenticação assume que o usuário submete à aplicação web somente sua identidade OpenID. Então ele é redirecionado para seu provedor OpenId para fornecer a titularidade da identidade usando senha ou algum outro método. Essa senha nunca é conhecida no loca da aplicação web.

A identidade OpenID é apenas uma URL HTTP que aponta para alguma página web com informação adequada sobre o usuário e delimitadores especiais que descrevem qual servidor usar e qual identidade submeter. Você pode ler mais sobre no OpenID official site.

A classe `Zend_Auth_Adapter_OpenId` é um invóluvro no topo do componente `Zend_OpenId_Consumer` que implementa o próprio protocolo de autenticação OpenID.

3.5.2. Especificação

Como qualquer outro adaptador `Zend_Auth` a classe `Zend_Auth_Adapter_OpenId` implementa `Zend_Auth_Adapter_Interface`, que define um método - `authenticate()`. Esse método executa a própria autenticação, mas o objeto deve ser preparado previamente para chamá-lo. Tal preparação do adaptador inclui a configuração da identidade OpenID e algumas outras opções específicas `Zend_OpenId`.

Entretanto, em oposição a outros adaptadores `Zend_Auth` ele executa autenticação em um servidor externo e isso é feito em duas requisições separadas HTTP. Por isso `Zend_Auth_Adapter_OpenId::authenticate()` deve ser chamado duas vezes. Na primeira vez, o método não retornará, mas irá redirecionar o usuário para seu servidor OpenID. Então depois da autenticação no servidor ele serpa redirecionado para trás e o script para essa

2 OpenID é um sistema de identificação desenvolvido por Brad Fitzpatrick do LiveJournal. Se trata de uma rede distribuída na qual a identidade de um usuário é dada por uma URL ou XRI que pode ser verificada por qualquer servidor executando o protocolo.

segunda requisição deve chamar `Zend_Auth_Adapter_OpenId::authenticate()` novamente para verificar a assinatura que vem com requisição redirecionada do servidor e complete o processo de autenticação. Dessa vez o método retornará o objeto `Zend_Auth_Result` como esperado.

O seguinte exemplo mostra o uso de `Zend_Auth_Adapter_OpenId`. Como foi dito antes `Zend_Auth_Adapter_OpenId::authenticate()` é chamado duas vezes. Na primeira vez – depois de submeter um form HTML onde `$_POST['openid_action']` é configurado para `"login"`, e na segunda vez depois do redirecionamento do servidor OpenID onde `$_GET['openid_mode']` ou `$_POST['openid_mode']` é configurado.

```
<?php
require_once "Zend/Auth.php";
require_once "Zend/Auth/Adapter/OpenId.php";
$status = "";
$auth = Zend_Auth::getInstance();
if ((isset($_POST['openid_action']) &&
     $_POST['openid_action'] == "login" &&
     !empty($_POST['openid_identifier'])) ||
    isset($_GET['openid_mode']) ||
    isset($_POST['openid_mode'])) {
    $result = $auth->authenticate(
        new Zend_Auth_Adapter_OpenId(@$_POST['openid_identifier']));
    if (!$result->isValid()) {
        $status = "You are logged-in as " . $auth->getIdentity() . "<br>\n";
    } else {
        $auth->clearIdentity();
        foreach ($result->getMessages() as $message) {
            $status .= "$message<br>\n";
        }
    }
} else if ($auth->hasIdentity()) {
    if (isset($_POST['openid_action']) &&
        $_POST['openid_action'] == "logout") {
        $auth->clearIdentity();
    } else {
        $status = "You are logged-in as " . $auth->getIdentity() . "<br>\n";
    }
}
?>
<html><body>
<?php echo "$status";?>
<form method="post"><fieldset>
<legend>OpenID Login</legend>
<input type="text" name="openid_identifier" value="">
<input type="submit" name="openid_action" value="login">
<input type="submit" name="openid_action" value="logout">
</fieldset></form></body></html>
*/
```

É permitido customizar o processo de autenticação OpenId através da recepção do redirecionamento de um servidor OpenId em página separada e especificando a “raiz” do site web. Neste caso, use `Zend_OpenId_Consumer_Storage` ou `Zend_Controller_Response` customizados. É também possível usar Extensão de Registro Simples para recuperar informação sobre o usuário do servidor OpenID. Todas essas possibilidades são descritas em mais detalhes na referência a `Zend_OpenId_Consumer`.