Ticket authentication

Overview

Ticket authentication is an authentication method that is often used with an existing, external authentication server to allow remote users to connect to EZproxy to access resources. With ticket authentication, a user authenticates with an authentication method, like CGI, and then the ticket script allows users authenticated with the remote server to access short-lived URLs that EZproxy will automatically recognize as being authorized to log in. This URL will permit access to a resource with no need for EZproxy to check back with the program that creates the URL within the ticket's lifetime. A sample URL looks like this:

```
http://ezproxy.yourlib.org:2048/login?user=rdoe&
ticket=a6911a5d021f428b33e190a80818625%24c2004122220203%24e&url=http://www.somedb.com/
```

The ticket parameter on the URL contains a digital signature that EZproxy uses to verify that the URL was created by an authorized program. The ticket contains a time-stamp of when it was created, and EZproxy can be configured to determine how old a ticket can be before it is considered expired.

Ticket generating code

Sample code for generating tickets is available for ASP, Cold Fusion, Perl, and PHP. You may need to use your browser's "View Source" command to view the code behind these examples.

Sample code:

**ASP**

- Examples for ASP are located under the [Ticket authentication with ASP](#) heading below.

**Cold Fusion**

```
<cfset ezproxy="http://ezproxy.yourlib.org/login">
<!--- Set secret to the shared secret value that appears after MD5 in user.txt/ezproxy.usr --->
<cfset secret="shhh">

<cfparam name="user" type="string" default="">
<cfparam name="pass" type="string" default="">
<cfparam name="qurl" type="string" default="">
<cfif user is not ">
  <cfquery name="checkusers" datasource="UserDSN">
    Select * from Users
  </cfquery>
```
Where LoginName = 
    <cfqueryparam cfsqltype="cf_sql_varchar" value="#user#"/>
AND
Password =
    <cfqueryparam cfsqltype="cf_sql_varchar" value="#pass#"/>
</cfquery>
<cfif #checkusers.RecordCount# eq 0>
    <cfset error="<p>Invalid username or password, please try again.</p>"/>
<cfelse>
    <cfset rightNow = Now()>
    <cfset formattedDate = DateFormat(rightNow,"yyyymmdd") & TimeFormat(rightNow,"HHmmss")>
    <cfset packet = "c" & formattedDate & "e">
    <cfset hashin = secret & user & packet>
    <cfset ticket = hashin & user & packet & packet>
    <cfset destUrl = ezproxy & "user=" & URLEncodedFormat(user) & 
        
        "ticket=" & 
        URLEncodedFormat(ticket) & 
        "qurl=" & 
        URLEncodedFormat(qurl)>
    <cflocation url="#destUrl#" addToken="no">
</cfif>
<cfif>

<html>
<body>
<p><b>You have chosen a resource that requires you to log in for remote access.</b></p>
Please enter your Forums login and password to access the resource. </b>
</p>
<p>
<hr />
<form action="#CGI.SCRIPT_NAME#" method="post">
Please enter your username: <input type="text" name="user" /><br />
Please enter your password: <input type="password" name="pass" /><br />
<input type="hidden" name="qurl" value="#qurl#" />
<input type="submit" value="Login" />
</form>
</cfoutput>
</body>
</html>

Perl
#!/usr/bin/perl
use strict;
use CGI;
use MD5;

my $EZproxyStartingPointURL = "";

sub EZproxyURLInit
{
    my ($EZproxyServerURL, $secret, $user, $groups) = @_; 
    my ($packet, $EZproxyTicket);
if ($secret eq "") {
    die "EZproxyURLInit secret missing";
}

$packet = "$u" . time();
if ($groups ne "") {
    $packet .= "$g" . $groups;
}
$packet .= "$e";
$EZproxyTicket = CGI::escape(MD5->hexhash($secret . $user . $packet) . $packet);
$EZproxyStartingPointURL = $EZproxyServerURL . "/login?user=" . CGI::escape($user) . "&ticket=" . $EZproxyTicket;

sub EZproxyURL {
    my ($url) = @_;

    if ($EZproxyStartingPointURL eq "") {
        die "EZproxyURLInit must be called before EZproxyURL";
    }

    return $EZproxyStartingPointURL . "&qurl=" . CGI::escape($url);
}
EZproxyURLInit("http://ezproxy.yourlib.org", "shhh", "someuser");

print "Content-Type: text/html

"
print "<a href="", EZproxyURL("http://www.somedb.com"), "">Some Database</a><br />
"<a href="", EZproxyURL("http://www.otherdb.com"), "">Other Database</a><br />
";

PHP

- Examples for PHP are located under the Ticket authentication with PHP heading below.

For assistance in adapting this sample code for use in your application or for creating similar code for other web scripting environments, contact support@oclc.org.

user.txt Example

A basic sample entry in user.txt is:

::Ticket
TimeValid 10
SHA512 somekey
This ticket authentication block should follow the authentication block for the external authentication server that users log in to using their authentication credentials. For example, if your institution uses CGI authentication, the CGI authentication block would come first in user.txt, and would be followed by the ticket block above.

For more details about each of these directives and the meaning, see the following section on Directives.

**Directives**

The following directives are specific to ticket authentication and can be used to customize your configuration in user.txt. All of these directives should fall within the Ticket authentication block:

```::Ticket
All directives and actions here
/Ticket
```

In addition to these directives, you may use additional actions from the [Common conditions and actions](https://help.oclc.org/Library_Management/EZproxy/Authenticate_users/EZproxy_authentication_methods/Ticket_authentication/) page.

**AcceptGroups**

*AcceptGroups* is a directive used when configuring EZproxy Groups. It specifies the groups that are allowed to appear in tickets and the users who can access the resources assigned to those tickets.

**Description**

The following table describes how different configurations in the authentication block, both with and without AcceptGroups, would impact EZproxy's processing of user groups.

<table>
<thead>
<tr>
<th>AUTHENTICATION BLOCK</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>No AcceptGroups included</td>
<td>Any groups specified in the ticket are ignored.</td>
</tr>
<tr>
<td>AcceptGroups is included, but the ticket does not contain any groups</td>
<td>The user is assigned to a group based on the last Group directive that was processed.</td>
</tr>
<tr>
<td></td>
<td>The user is assigned to the Default group if no previous Group directive was processed.</td>
</tr>
<tr>
<td>AcceptGroups is included, and the ticket contains groups</td>
<td>The user is assigned to the groups specified in the ticket that are also allowed by AcceptGroups.</td>
</tr>
</tbody>
</table>

**Qualifiers**

The following qualifiers must be added after AcceptGroups to define the user groups in a ticket.
group

The name of a group that should be allowed to appear in tickets.

Example

This directive must appear before any directives specifying the shared key (e.g. MD5, SHA1, SHA256, and SHA512).

The following example would allow the groups Default, Law, or Medical to appear in a ticket:

```
::Ticket
AcceptGroups Default+Law+Medical
SHA512 secretkey
IfUnauthenticated; Stop
/Ticket
```

**MD5, SHA1, SHA256, SHA512**

MD5, SHA1, SHA256 and SHA512 are directives that specify the cryptographic hashing algorithm that should be used to validate tickets. The directive should be followed by the shared key that should be used to validate tickets. The shared key entered with this directive in user.txt must match the key specified in the ticket generating script.

If the shared key is compromised, any external system can use the shared key to generate a ticket to authorize a user to have access to EZproxy. If this occurs, the shared key should be changed to a new value in the external system and in user.txt.

Qualifiers

<table>
<thead>
<tr>
<th>QUALIFIER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>sharedkey</td>
<td>The key shared between the ticket creating software and EZproxy to authenticate.</td>
</tr>
</tbody>
</table>

Examples

The following table shows examples of each directive in the context of a Ticket authentication block.

<table>
<thead>
<tr>
<th>DIRECTIVE</th>
<th>VERSION</th>
<th>EXAMPLE</th>
</tr>
</thead>
</table>
| MD5       | V5.2 or later | ::Ticket
TimeValid 10
MD5 md5key |
<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>minutes</td>
<td>The number of minutes to add or subtract from the time value included in the ticket. To indicate subtraction, specify a negative number.</td>
</tr>
</tbody>
</table>

**Examples**

This directive must appear before the MD5, SHA1, SHA256 or SHA512 directive. The following table shows how to add or subtract time from the time value specified in the ticket.
Add 60 minutes to the time specified in the ticket.

::Ticket
TimeOffset 60
SSHA256 secretkey
IfUnauthenticated; Stop
/Ticket

Subtract 2 hours (120 minutes) from the time specified in the ticket.

::Ticket
TimeOffset -120
SSHA256 secretkey
IfUnauthenticated; Stop
/Ticket

**TimeValid**

TimeValid is a directive that specifies the window of time in minutes for which a ticket should be considered valid. This window will begin at the time specified in the ticket. For example, if the ticket has a time stamp of 1:00pm EST, and the TimeValid is set to 30, that ticket will provide a remote user with access until 1:30pm EST. If there is no TimeValid directive within the authentication block, the default value of 60 minutes is used.

TimeValid can be used to compensate for time variations between the ticket generating system and EZproxy clocks. A larger window is useful if the external system generates a page of ticket URL links that is intended to remain valid for use for an extended period of time.

**Qualifiers**

<table>
<thead>
<tr>
<th>QUALIFIER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>minutes</td>
<td>The number of minutes that a ticket should be considered valid.</td>
</tr>
</tbody>
</table>

**Examples**

This directive must appear before the MD5, SHA1, SHA256 or SHA512 directive.

The following example will reduce the time a ticket is valid from the default of 60 minutes to 5 minutes.

::Ticket
TimeValid 5
SSHA256 secretkey
/Ticket

**Ticket authentication with ASP**

To create tickets with ASP, download asp.zip and expand the files into a directory on your web server. Once expanded,
refer to ticketdemo.asp for information on how to use ezproxyticket.asp to generate tickets in your own applications.

**Ticket authentication with PHP**

To create tickets with PHP, download `ezproxyticket.txt`, which provides support for generating EZproxy ticket URLs. The .txt extension on this file is only needed for downloading; once you download the file, replace this extension with "php". Place this file in your PHP include directory or place it in the same directory where you will place your scripts that create tickets.

**ezproxyticket.txt**

```php
<?php

// This PHP class can be used to generate EZproxy tickets. See
// ticket URLs. When creating a new EZproxyTicket option,
// the first parameter is the EZproxy server base
// URL, the second is the shared secret that appears in user.txt/ezproxy.usr,
// the third is a username to associate with the user, and the
// last is optional and specifies any EZproxy groups the user
// should be associated with.

// If you use a shared secret of shhh, then in user.txt/ezproxy.usr you
// might use:
//   ::Ticket
//   MD5 shhhh
//   /Ticket
// to allow EZproxy to recognize your tickets.

// Once the object is created, you can call its url method with a
// database URL to generate a ticket URL.

class EZproxyTicket {
    var $EZproxyStartingPointURL;

    function EZproxyTicket( $EZproxyServerURL, $secret, $user, $groups = ""
    ) {
        if (strcmp($secret, ") == 0) {
            echo("EZproxyURLInit secret cannot be blank");
            exit(1);
        }

        $packet = '$u' . time();
        if (strcmp($groups, ") != 0) {
            $packet .= '$g' . $groups;
        }
        $packet .= '$e';
        $EZproxyTicket = urlencode(md5($secret . $user . $packet) . $packet);
    }

    function url($databaseURL) {
        return $EZproxyStartingPointURL . $databaseURL . $EZproxyTicket . '
    }

    function main() {
        $databaseURL = 'http://example.com/app.sql';
        print_r(url($databaseURL));
    }

    public static function main() {
        EZproxyTicket::main();
    }

    function info() {
        // ... additional information ...
    }

    function usage() {
        // ... usage information ...
    }
}

$ezproxyTicket = new EZproxyTicket('http://ezproxy.example.com', 'secret', 'user', 'group1 group2');
$url = $ezproxyTicket->url('http://example.com/app.sql');
```
$this->EZproxyStartingPointURL = $EZproxyServerURL . "/login?user=" . urlencode($user) . "]ticket=" . $EZproxyTicket;
}

function URL($url)
{
    return $this->EZproxyStartingPointURL . "]url=" . $url;
}
}
?>

For an example of using this library to create tickets, see: ticketdemo.txt. The .txt extension on this file is only needed for downloading; once downloaded, replace this extension with "php".

ticketdemo.txt

```php
<?php
    // If ezproxyticket.php isn't on your PHP path, you may need to
    // include a directory reference such as
    // require("./ezproxyticket.php");

    require("ezproxyticket.php");

    // Create an EZproxyTicket object that will be used to generate
    // ticket URLs.  The first parameter is the EZproxy server base
    // URL, the second is the shared secret that appears in user.txt/ezproxy.usr,
    // and the third should be changed to a variable that provides
    // the user's username

    $ezproxy = new EZproxyTicket("http://ezproxy.yourlib.org:2048",
    "shhhh", "someuser");

?>

<br />

<a href="<?php echo($ezproxy->url("http://www.otherdb.com")) ?>">Other Database</a>
<br />
```