Preauth - Zimbra :: Tech Center

Archived: 2026-04-05 17:08:12 UTC

Preauth

KB 2218	Last updated on 2023-03-16	
0.00 (0 votes)		

What is preauth?

Preauth stands for pre-authentication, and is a mechanism to enable a trusted third party to "vouch" for a user's identity. For example, if a user has already signed into a portal and wants to enter the mail application, they should not have to be prompted again for their password.

This can be accomplished by having the mail link they click on in the portal construct a special URL and redirect the user to the Zimbra server, which will then verify the data passed in the URL and create authentication token (the standard mechanism within Zimbra to identify users), save it in a cookie, and redirect the user to the mail app.

How does it work?

It works by having a key that is shared between the third party and Zimbra. Knowing this key, the third party specifies the desired username, a timestamp (to ensure the request is "fresh"), optionally an expiration time, and then computes a SHA-1 HMAC over that data using secret key.

The server computes the HMAC using the supplied data, and its key, to verify that it matches the HMAC sent in the request. If it does, the server will construct an auth token, save it in a cookie, and redirect the user to the mail application.

Preparing a domain for preauth

In order for preauth to be enabled for a domain, you need to run the zmprov command and create a key:

```
zmprov generateDomainPreAuthKey domain.com
preAuthKey: 4e2816f16c44fab20ecdee39fb850c3b0bb54d03f1d8e073aaea376a4f407f0c
```

Make note of that key value, as you'll need to use it to generate the computed-preauth values below. Also make sure you keep it secret, as it can be used to generate valid auth tokens for any user in the given domain!

Behind the scenes, this command is simply generating 32 random bytes, hex encoding them, then setting the "zimbraPreAuthKey" attr for the specified domain with the value of the key.

To see the key value after it's generated again [replace YOUR_DOMAIN with the actual domain name you need to check]:

```
zmprov gd YOUR_DOMAIN zimbraPreAuthKey
```

After generating the key, you'll probably need to restart the ZCS server so it picks up the updated value.

Redirect security setting

Zimbra 9.0.0 Kepler Patch 30 and 8.8.15 James Prescott Joule Patch 37, Zimbra Pre-Auth will only work when it redirects to the zimbraPublicServiceHostname and that means your DNS domain should match zimbraPublicServiceHostname. In case you have not configured this correctly or use multiple redirection domains, refer to https://wiki.zimbra.com/wiki/Fix_preauth_redirection.

Verify you logging

PreAuth when implemented via REST API will not log the originating IP of the user, in addition you need to configure originating IP so it can be logged via the SOAP API. See: https://wiki.zimbra.com/wiki/Secopstips#Pre-authentication and https://wiki.zimbra.com/wiki/Log_Files#Logging_the_Originating_IP

What are the interfaces?

There are two interfaces. One interface is URL-based, for ease of integration. The other interface is SOAP-based, for cases where the third party wants more control over generating the auth token and redirecting the user.

We'll describe the URL interface first, followed by the SOAP interface.

URL Interface

The URL interface uses the /service/preauth URL:

```
/service/preauth?
        account={account-identifier}
        by={by-value}
        timestamp={time}
        expires={expires}
        [&admin=1]
        preauth={computed-preauth}
```

The values are as follows:

```
{account-identifier}   depends on the value of the "by" attr. If "by" is not
                        specified, it is the name (i.e., john.doe@domain.com).

{by-value}             name|id|foreignPrincipal, same as AuthRequest. defaults
                        to name.

{timestamp}            current time, in milliseconds. The timestamp must
                        be within 5 minutes of the server's time for the
                        preauth to work.

{expires}              expiration time of the authtoken, in milliseconds.
                        set to 0 to use the default expiration time for the
                        account. Can be used to sync the auth token expiration
                        time with the external system's notion of expiration
                        (like a Kerberos TGT lifetime, for example).

{admin}                set to "1" if this preauth is for admin console. This
                        only works if given account is an admin, and the request
                        comes in on the admin port (https 7071 by default).
                        If admin is specified, then include its value while computing the
                        HMAC below, after the "account" value, and before the "by" value.

{computed-preauth}      the computed pre-auth value. See below for details.
```

The preauth value is computed as follows:

```
 1. concat the values for account, by, expires, timestamp together
    (in that order, order is definitely important!), separated by "|"
```

```
 2. compute the HMAC on that value using the shared key
    (the zimbraPreAuthKey value for the account's domain, generating one
     is described below).
```

```
 3. convert the HMAC value into a hex string.
```

For example, given the following values:

```
key: 6b7ead4bd425836e8cf0079cd6c1a05acc127acd07c8ee4b61023e19250e929c
account: john.doe@domain.com
by: name
expires: 0
timestamp: 1135280708088
```

You would concat the account/by/expires/timestamp values together to get:

```
john.doe@domain.com|name|0|1135280708088
```

You would then compute the SHA-1 HMAC on that string, using the key:

```
preauth = hmac("john.doe@domain.com|name|0|1135280708088", "6b7ead4bd425836e8cf0079cd6c1a05acc127acd
```

Finally, you would take the returned hmac (which is most likely an array of bytes), and convert it to a hex string:

```
preauth-value: b248f6cfd027edd45c5369f84901252047 72f844
```

The resulting URL would be:

```
/service/preauth?account=john.doe@domain.com&expires=0&timestamp=1135280708088&preauth=b248f6cfd027e
```

Hitting that URL on the ZCS server will cause the preauth value to be verified, followed by a redirect to /zimbra/mail with the auth token in a cookie.

If a URL other then /zimbra/mail is desired, then you can also pass in a redirectURL:

```
...&redirectURL=/zimbra/h/
```

If you are pre-authing an admin, the value to use with the hmac would be:

john.doe@domain.com|1|name|0|1135280708088

and you would include "&admin=1" in the URL.

If you are not pre-authing an admin, then you must *NOT* include any value for it in string passed to the hmac.

SOAP Interface

The SOAP interface uses the standard AuthRequest message, but instead of passing in a password, you specify <preauth> data.

For example:

```
<AuthRequest xmlns="urn:zimbraAccount">
<account by="name|id|foreignPrincipal">{account-identifier}</account>
<preauth timestamp="{timestamp}" expires="{expires}">{computed-preauth}</preauth>
</AuthRequest>
```

The values are exactly the same as they were for the URL case:

```
<AuthRequest xmlns="urn:zimbraAccount">
<account>john.doe@domain.com</account>
<preauth timestamp="1135280708088" expires="0">b248f6cfd027edd45c5369f8490125204772f844</preauth>
</AuthRequest>
```

The auth token will be return in the AuthResponse. At which point, you can "inject" it into the app via the URL interface:

https://server/service/preauth?isredirect=1&authtoken={...}

Going to this URL will set the cookie and redirect to /zimbra/mail. If a URL other then /zimbra/mail is desired, then you can also pass in a redirectURL:

```
...&redirectURL=/zimbra/h/
```

Sample Java code for computing the preauth value

The following Java Code (1.5) will compute the pre-auth value.

```
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.HashMap;
import java.util.Map;
import java.util.TreeSet;
import javax.crypto.Mac;
import javax.crypto.SecretKey;

public class test {

 public static void main(String args[]) {
     HashMap<String,String> params = new HashMap<String,String>();
     params.put("account", "john.doe@domain.com");
     params.put("by", "name"); // needs to be part of hmac
     params.put("timestamp", "1135280708088");
     params.put("expires", "0");
     String key =
        "6b7ead4bd425836e8cf0079cd6c1a05acc127acd07c8ee4b61023e19250e929c";
     System.out.printf("preAuth: %s\n", computePreAuth(params, key));
  }

  public static  String computePreAuth(Map<String,String> params, String key)
  {
```

```
        TreeSet<String> names = new TreeSet<String>(params.keySet());
        StringBuilder sb = new StringBuilder();
        for (String name : names) {
            if (sb.length() > 0) sb.append('|');
            sb.append(params.get(name));
        }
        return getHmac(sb.toString(), key.getBytes());
    }

    private static String getHmac(String data, byte[] key) {
        try {
            ByteKey bk = new ByteKey(key);
            Mac mac = Mac.getInstance("HmacSHA1");
            mac.init(bk);
            return toHex(mac.doFinal(data.getBytes()));
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException("fatal error", e);
        } catch (InvalidKeyException e) {
            throw new RuntimeException("fatal error", e);
        }
    }

    static class ByteKey implements SecretKey {
        private byte[] mKey;

        ByteKey(byte[] key) {
            mKey = (byte[]) key.clone();;
        }

        public byte[] getEncoded() {
            return mKey;
        }

        public String getAlgorithm() {
            return "HmacSHA1";
        }

        public String getFormat() {
            return "RAW";
        }
    }

    public static String toHex(byte[] data) {
        StringBuilder sb = new StringBuilder(data.length * 2);
        for (int i=0; i<data.length; i++ ) {
           sb.append(hex[(data[i] & 0xf0) >>> 4]);
           sb.append(hex[data[i] & 0x0f] );
```

```
        }
        return sb.toString();
    }

    private static final char[] hex =
       { '0' , '1' , '2' , '3' , '4' , '5' , '6' , '7' ,
         '8' , '9' , 'a' , 'b' , 'c' , 'd' , 'e' , 'f'};
}
```

You can also use zmprov while debugging to generate preAuth values for comparison:

```
prov> gdpa domain.com john.doe@domain.com name 1135280708088 0
account: john.doe@domain.com
by: name
timestamp: 1135280708088
expires: 0
preAuth: b248f6cfd027edd45c5369f8490125204772f844
prov>
```

Sample preauth.jsp

Here is a sample JSP that does everything needed for preauth. It has a hardcoded username, which should of course be changed to match the username of the user you have "pre-authenticated".

```
File Edit Options Buffers Tools Help
<!--

To configure:

1) Generate a preauth domain key for your domain using zmprov:

zmprov gdpak domain.com
preAuthKey:  ee0e096155314d474c8a8ba0c941e9382bb107cc035c7a24838b79271e32d7b0

Take that value, and set it below as the value of DOMAIN_KEY

2) restart server (only needed the first time you generate the domain pre-auth key)

3) redirect users to this (this, as in *this* file after you install it) JSP page:

http://server/zimbra/preauth.jsp

And it will construct the preauth URL
```

```
-->
<%@ page import="java.security.InvalidKeyException" %>
<%@ page import="java.security.NoSuchAlgorithmException" %>
<%@ page import="java.security.SecureRandom" %>
<%@ page import="java.util.HashMap" %>
<%@ page import="java.util.Map" %>
<%@ page import="java.util.Iterator" %>
<%@ page import="java.util.TreeSet" %>
<%@ page import="javax.crypto.Mac" %>
<%@ page import="javax.crypto.SecretKey" %>
<%!
 public static final String DOMAIN_KEY =
        "ee0e096155314d474c8a8ba0c941e9382bb107cc035c7a24838b79271e32d7b0";


 public static String generateRedirect(HttpServletRequest request, String name) {
     HashMap params = new HashMap();
     String ts = System.currentTimeMillis()+"";
     params.put("account", name);
     params.put("by", "name"); // needs to be part of hmac
     params.put("timestamp", ts);
     params.put("expires", "0"); // means use the default

     String preAuth = computePreAuth(params, DOMAIN_KEY);
     return request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+"/service/
           "account="+name+
           "&by=name"+
           "&timestamp="+ts+
           "&expires=0"+
           "&preauth="+preAuth;
  }

    public static  String computePreAuth(Map params, String key) {
        TreeSet names = new TreeSet(params.keySet());
        StringBuffer sb = new StringBuffer();
        for (Iterator it=names.iterator(); it.hasNext();) {
            if (sb.length() > 0) sb.append('|');
            sb.append(params.get(it.next()));
        }
        return getHmac(sb.toString(), key.getBytes());
    }

    private static String getHmac(String data, byte[] key) {
        try {
            ByteKey bk = new ByteKey(key);
            Mac mac = Mac.getInstance("HmacSHA1");
            mac.init(bk);
```

```
            return toHex(mac.doFinal(data.getBytes()));
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException("fatal error", e);
        } catch (InvalidKeyException e) {
            throw new RuntimeException("fatal error", e);
        }
    }


    static class ByteKey implements SecretKey {
        private byte[] mKey;

        ByteKey(byte[] key) {
            mKey = (byte[]) key.clone();;
        }

        public byte[] getEncoded() {
            return mKey;
        }

        public String getAlgorithm() {
            return "HmacSHA1";
        }

        public String getFormat() {
            return "RAW";
        }
   }

    public static String toHex(byte[] data) {
        StringBuilder sb = new StringBuilder(data.length * 2);
        for (int i=0; i<data.length; i++ ) {
           sb.append(hex[(data[i] & 0xf0) >>> 4]);
           sb.append(hex[data[i] & 0x0f] );
        }
        return sb.toString();
    }

    private static final char[] hex =
       { '0' , '1' , '2' , '3' , '4' , '5' , '6' , '7' ,
         '8' , '9' , 'a' , 'b' , 'c' , 'd' , 'e' , 'f'};


%><%

String redirect = generateRedirect(request, "user1@slapshot.liquidsys.com");
response.sendRedirect(redirect);
```

```
%>
<html>
<head>
<title>Pre-auth redirect</title>
</head>
<body>

You should never see this page.

</body>
</html>
```

Java Program for Zimbra Autologin from Windows

The following program is based mainly on the sample Java code provided above. This program fetches the Windows login name and uses it to construct the preauth URL, open the default web browser, and autologin to the Zimbra server. The preauth key and server are hard-coded into the source, so you will need to set them before you compile. **You should be aware that this program creates a potential security problem whereby UserA copies the compiled jar file, creates a Windows login account on a computer with UserB's login name, and runs the program under that account. UserA will be authenticated as UserB in this scenario.**

```
/*
 * Main.java
 *
 * Created on October 25, 2007, 1:54 PM
 *
 * Some code originally from: http://wiki.zimbra.com/index.php?title=Preauth
 *
 * For this program to work, you must follow the directions for enabling preauth
 * on your server (available at the URL above). You must also set preAuthKey and
 * preAuthUrl in the main method below.
 *
 * By default, this program uses the Windows login name to set the Zimbra login
 * name. These names must match for this program to work.
 *
 * This method of determining username creates a security problem wherein a user
 * could copy the compiled jar and use it to login as any other user. To do so,
 * he has to create a Windows login account with the username of whomever he
 * wants to login as.
 */

package zimbrapreauthloader;

import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
```

```
import java.security.SecureRandom;
import java.io.IOException;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.TreeSet;
import javax.crypto.Mac;
import javax.crypto.SecretKey;

/**
 *
 */
public class Main {

    static String newline = System.getProperty("line.separator");

    /** Creates a new instance of Main */
    public Main() {
    }

    public static void main(String args[]) {
        String account = getAccount();   // Account name you are trying to login
        String by = "name";         // Specifies how account should be determined
        String timestamp = "";      // Current Unix date and time
        String expires = "0";       // Length of time in milliseconds to expire the auth
        String preAuthKey = "                                                                  "; // S
        String preAuthValue = "";   // Calculated value for preauth
        String preAuthUrl = "https://mail.domain.com/service/preauth?";  // URL to use preauth
        String openBrowser = "rundll32 url.dll,FileProtocolHandler ";   // Command to open browser a

        // Generate timestamp
        Date date = new Date();
        timestamp = "" + date.getTime();

        // Assign values needed to calculate preauth
        HashMap<String,String> params = new HashMap<String,String>();
        params.put("account", account);
        params.put("by", by); // needs to be part of hmac
        params.put("timestamp", timestamp);
        params.put("expires", expires);

        // Compute preAuthValue
        preAuthValue = computePreAuth(params, preAuthKey);

        // Construct preAuthUrl
        preAuthUrl += constructUrl(account, expires, timestamp, preAuthValue);
```

```
        // Open Browser;
        try {
            Runtime.getRuntime().exec(openBrowser + preAuthUrl);
        }
        catch (IOException e) {
            System.out.println("Unable to open browser.");
            System.exit(1);
        }
    }

    /**
     * Constructs the URL needed to authorize
     *
     * Example url:
     * https://mail.domain.com/service/preauth?account=john.doe@domain.com&expires=0&timestamp=11352
     */
    public static String constructUrl(String useAccount, String useExpires, String useTimestamp, Str
        String url = "";
        // Account part
        url += "account=" + useAccount + "&";
        // Expires part
        url += "expires=" + useExpires + "&";
        // Timestamp part
        url += "timestamp=" + useTimestamp + "&";
        // PreAuth part
        url += "preauth=" + usePreAuthValue;
        return url;
    }

    /**
     * Provides the account name to login as
     *
     * Retrieves the Windows login name and uses that as the Zimbra login name
     */
    private static String getAccount() {
        String username = System.getProperty("user.name");
        return username;
    }

    /**
     * Computes the preauth string
     */
    public static String computePreAuth(Map<String,String> params, String key) {
        TreeSet<String> names = new TreeSet<String>(params.keySet());
        StringBuilder sb = new StringBuilder();
        for (String name : names) {
            if (sb.length() > 0) {
```

```
            sb.append('|');
        }
        sb.append(params.get(name));
    }
    return getHmac(sb.toString(), key.getBytes());
}

/**
 * Computes the preAuth excryption key
 */
private static String getHmac(String data, byte[] key) {
    try {
        ByteKey bk = new ByteKey(key);
        Mac mac = Mac.getInstance("HmacSHA1");
        mac.init(bk);
        return toHex(mac.doFinal(data.getBytes()));
    } catch (NoSuchAlgorithmException e) {
        throw new RuntimeException("fatal error", e);
    } catch (InvalidKeyException e) {
        throw new RuntimeException("fatal error", e);
    }
}

static class ByteKey implements SecretKey {
    private byte[] mKey;

    ByteKey(byte[] key) {
        mKey = (byte[]) key.clone();
    }

    public byte[] getEncoded() {
        return mKey;
    }

    public String getAlgorithm() {
        return "HmacSHA1";
    }

    public String getFormat() {
        return "RAW";
    }
}

public static String toHex(byte[] data) {
    StringBuilder sb = new StringBuilder(data.length * 2);
    for (int i=0; i<data.length; i++ ) {
        sb.append(hex[(data[i] & 0xf0) >>> 4]);
```

```
            sb.append(hex[data[i] & 0x0f] );
        }
        return sb.toString();
    }

    private static final char[] hex = {'0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','

}
```

--Brousch 12:56, 25 October 2007 (PDT)

.NET C# Code For Redirecting to Zimbra Preauth

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using System.Text;
using System.Security.Cryptography;

namespace Zimbra
{
        /// <summary>
        /// Summary description for Zimbra.
        /// </summary>
        public class Zimbra : System.Web.UI.Page
        {
                private void Page_Load(object sender, System.EventArgs e)
                {
                        /*
 *  This page authenticates a user to zimbra using zimbra's preauth method.
 * A user should be authenticated before reaching this page via some external
 * method which populates LOGON_USER, then a one time key is generated and forwarded the user
 * is forwarded to zimbra.  Zimbra validates the key, generates a cookie for the
 * user, and they are then authenticated.
 */


                        //            warning.Text = Request.ServerVariables["ALL_HTTP"];
```

```
 string preauthkey = "3298a239b8983c723874893274..."; //copied from zimbra do
 string preauthurl = "https://zimbra.example.com/service/preauth";
 string account = Request.ServerVariables["LOGON_USER"] + "@zimbra.example.co
 string preauthvalue; //holds the string to be encoded
 string preauthencoded; //holds the preauthvalue after it has been sha1-hmac'
 string timestamp; //millisecs since epoch


 //convert key to byte form, as required for hmac
 System.Text.ASCIIEncoding encoding=new System.Text.ASCIIEncoding();
 HMACSHA1 hmacsha1 = new HMACSHA1(encoding.GetBytes(preauthkey));


 //get timestamp
 DateTime d1 = new DateTime(1970, 1, 1);
 DateTime d2 = DateTime.Now;
 TimeSpan ts = new TimeSpan(d2.Ticks - d1.Ticks);

 //correct .net timezone
 TimeSpan utcOffset  = TimeZone.CurrentTimeZone.GetUtcOffset( d2 ) ;
 Int64 tsint = (long) ts.TotalMilliseconds - (long) utcOffset.TotalMillisecon
 timestamp = tsint.ToString();

 //have everything we need to make our preauth string
 //we need using  SortedDictionary to reorder params by alphabet
SortedDictionary<String, String> parms = new SortedDictionary<String, String>
parms.Add("account", account);
parms.Add("by", name);
parms.Add("expires", 0);
parms.Add("timestamp", timestamp);
StringBuilder sb = new StringBuilder();
foreach (var parm in parms)
{
  if (sb.Length != 0)
{
  sb.Append("|");
}
  sb.Append(parm.Value);
}
 //preauthvalue = account + "|name|0|" + timestamp;
 preauthvalue = sb.toString();

 //encode our preauth string
 byte[] preauthvaluebytes = encoding.GetBytes(preauthvalue);
 byte[] hashmessage = hmacsha1.ComputeHash(preauthvaluebytes);
 StringBuilder sb = new StringBuilder(hashmessage.Length * 2);
 foreach (byte b in hashmessage)
```

```
                        {
                                sb.AppendFormat("{0:x2}", b);
                        }
                        preauthencoded = sb.ToString();

                        //send the user over to zimbra.  hope all is well.
                        Response.Redirect(preauthurl + "?account=" + account + "&by=name&timestamp="
                }

                #region Web Form Designer generated code
                override protected void OnInit(EventArgs e)
                {
                        //
                        // CODEGEN: This call is required by the ASP.NET Web Form Designer.
                        //
                        InitializeComponent();
                        base.OnInit(e);
                }

                /// <summary>
                /// Required method for Designer support - do not modify
                /// the contents of this method with the code editor.
                /// </summary>
                private void InitializeComponent()
                {
                        this.Load += new System.EventHandler(this.Page_Load);
                }
                #endregion
        }
}
```

--User:djlarsu 10:43, 19 May 2009 (EDT)

PHP Code For Redirecting to Zimbra Preauth

The following PHP script takes user name and domain name as GET parameters and redirects the browser to ZCS preauth. In order to use this code, you need to put your own values for $WEB_MAIL_PREAUTH_URL and $PREAUTH_KEY

```
<?php
    /**
    * Globals. Can be stored in external config.inc.php or retreived from a DB.
    */
    $PREAUTH_KEY="0f6f5bbf7f3ee4e99e2d24a7091e262db37eb9542bc921b2ae4434fcb6338284";
    $WEB_MAIL_PREAUTH_URL="http://zimbra.server.com/service/preauth";
```

```
    /**
    * User's email address and domain. In this example obtained from a GET query parameter.
    * i.e. preauthExample.php?email=user@domain.com&domain=domain.com
    * You could also parse the email instead of passing domain as a separate parameter
    */
    $user = $_GET["user"];
    $domain=$_GET["domain"];

    $email = "{$user}@{$domain}";

    if(empty($PREAUTH_KEY)) {
        die("Need preauth key for domain ".$domain);
    }

    /**
    * Create preauth token and preauth URL
    */
    $timestamp=time()*1000;
    $preauthToken=hash_hmac("sha1",$email."|name|0|".$timestamp,$PREAUTH_KEY);
    $preauthURL = $WEB_MAIL_PREAUTH_URL."?account=".$email."&by=name&timestamp=".$timestamp."&expire

    /**
     * Redirect to Zimbra preauth URL
     */
    header("Location: $preauthURL");
?>
```

--User:Greg 2:37, 5 November 2008 (MST)

Sample Ruby code for computing the preauth value

```
require 'openssl'
include OpenSSL
include Digest

def compute_preauth(name, time_st, authkey)

       plaintext="#{name}|name|0|#{time_st}"
       key=authkey

       hmacd=HMAC.new(key, SHA1.new)
       hmacd.update(plaintext)
       return hmacd.to_s
end
```

User: Rahul Chaudhari 10:52, 29 October 2009 (IST)

Django/Python Code For Redirecting to Zimbra Preauth

The following Python code uses Django's built in user management to perform preauth. Can be used as a reference for non-Django Python code as well. In order to use this code, you need to put your own values for preauth_key and preauth_url.

```
def webmail_redirect(request):
    from time import time
    import hmac, hashlib

    preauth_key = "0f6f5bbf7f3ee4e99e2d24a7091e262db37eb9542bc921b2ae4434fcb6338284"
    preauth_url = "http://zimbra.server.com/service/preauth"

    timestamp = int(time()*1000)

    try:
        #If they're not logged in, an exception will be thrown.
        acct = request.user.email

        pak = hmac.new(preauth_key, '%s|name|0|%s'%(acct, timestamp), hashlib.sha1).hexdigest()
        return HttpResponseRedirect("%s?account=%s&expires=0&timestamp=%s&preauth=%s"%(preauth_url,
    except:
        pass

    return HttpResponseRedirect("/not_logged_in/")
```

Perl Code from LemonLDAP::NG

This code is used in LemonLDAP::NG ([1]), an open source WebSSO product. An URL is catched, ant then Zimbra URL is built. The integration with LemonLDAP::NG is explained here: [2].

```
use Digest::HMAC_SHA1 qw(hmac_sha1 hmac_sha1_hex);

## @method private string buildZimbraPreAuthUrl(string key, string url, string account, string by)
# Build Zimbra PreAuth URL
# @param key PreAuthKey
# @param url URL
# @param account User account
# @param by Account type
# @return Zimbra PreAuth URL
sub buildZimbraPreAuthUrl {
    my ( $key, $url, $account, $by ) = splice @_;
```

```
    # Expiration time
    my $expires = 0;

    # Timestamp
    my $timestamp = time() * 1000;

    # Compute preauth value
    my $computed_value =
      hmac_sha1_hex( "$account|$by|$expires|$timestamp", $key );

    # Build PreAuth URL
    my $zimbra_url;
    $zimbra_url .= $url;
    $zimbra_url .= '?account=' . $account;
    $zimbra_url .= '&by=' . $by;
    $zimbra_url .= '&timestamp=' . $timestamp;
    $zimbra_url .= '&expires=' . $expires;
    $zimbra_url .= '&preauth=' . $computed_value;

    return $zimbra_url;
}
```

--Coudot 08:11, 7 May 2010 (UTC)

Cas And PreAuth

Please see:

- Cassifying_Zimbra_5
- CASifying_Zimbra_6.0
- Ajcody-External-Authentication#JA-SIG_Central_Authentication_Service_Or_CAS
- RapidSSL Certificate

Trouble Shooting

Doesn't Work After Upgrade Of ZCS

Please check the data/time on your servers. If they are off by more than 5 minutes preauth will fail.

--Niosop 03:45, 6 January 2009 (UTC)

Want to get involved?

You can contribute in the Community, Wiki, Code, or development of Zimlets.
Find out more. »

Looking for a Video?

Visit our YouTube channel to get the latest webinars, technology news, product overviews, and so much more.
Go to the YouTube channel »
