Table of Contents

Remote authentication allows you to integrate your organization's authentication system with KBPublisher.

Before you start:

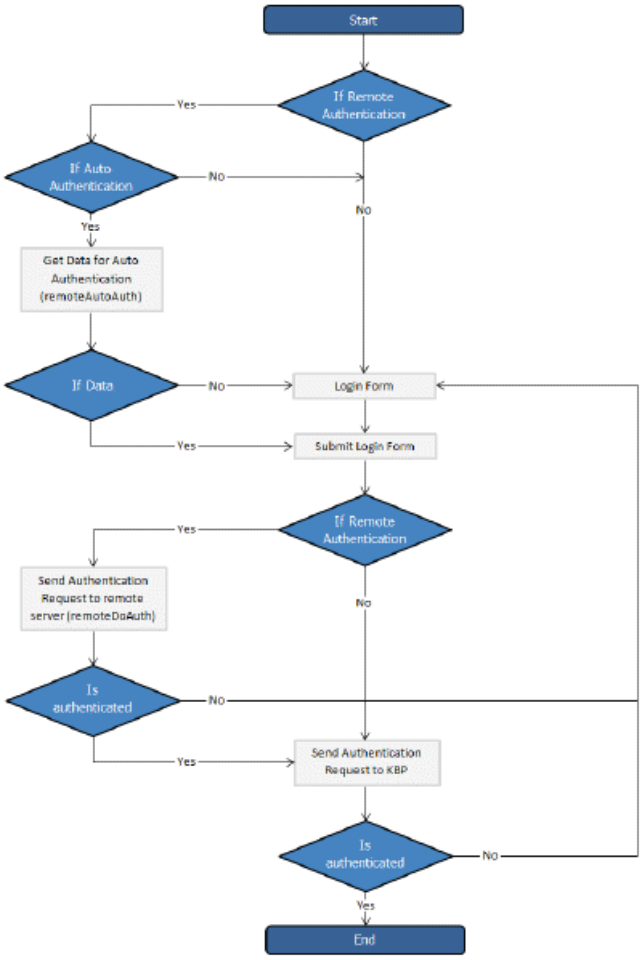
- We assume that you have some experience with PHP and with the system you are connecting to.

Steps to enable Remote Authentication

- Click on **Settings -> Authentication Provider -> Remote**
- Check **Enable Remote Authentication** checkbox (make sure that **$conf['auth_remote']** in the file *admin/config.inc.php* is set to **1**)
- Set required values for constants in file *admin/lib/custom/remote_auth.php*
- Customize the **_remoteDoAuth** function in the file *admin/lib/custom/remote_auth.php* to authenticate the username and password passed to it against your own authentication system
- Rename the function **_remoteDoAuth** to **remoteDoAuth**

Quick summary of the process

- End user goes to site
- Remote Authentication checks for valid user credentials
  - If auto-authentication is set, does this automatically
  - If auto-authentication is not set, user logs in first
- KBPublisher authenticates the user .



Customizing the remoteDoAuth function

In your installation there is a folder *admin/lib/custom*. Within that folder is a file called *remote_auth.php*. This file contains the *_remoteDoAuth* function. Customize this function to do authentication against your internal system by using the username and password provided.

Here is a simple example of the function customized to authenticate against a MySQL database:

```
function remoteDoAuth($username, $password) {

    $user = false;
```

```
    $db = &DBUtil::connect($conf);

    $sql = "SELECT
        id AS 'remote_user_id',
        email, username, first_name, last_name
    FROM your_remote_users_table
    WHERE username = '%s' AND password = '%s'";
    $sql = sprintf($sql, $username, $password);
    $result = $db->Execute($sql) or die(DBUtil::error($sql, true, $db));

    // if found
    if($result->RecordCount() == 1) {
        $user = $result->FetchRow();
        $user['password'] = $password; // here you should provide not md5ing password

        // assign a priv to user (optional)
        // it is fully up to you how to determine who is authenticated and what priv to assign
        // set to off to not rewrite on login
        $user['priv_id'] = 'off';

        // assign a role to user (optional)
        // it is fully up to you how to determine who is authenticated and what role to assign
        // set to off to not rewrite on login
        $user['role_id'] = 1;
    }

    return $user;
}
```

Also see examples in attached files.

Tracking logins

You can see how your remote authentication works in logs Logs/Logins

For debugging every last login is logged to a file called *last_remote_login.log* in the KBPublisher cache directory (*APP_CACHE_DIR in admin/config.inc.php*).
For example: */home/username/ kb_cache/last_remote_login.log*

It is possible to use Remote Authentication with your LDAP server.

Before you start:

- We assume that you have some experience with remote authentication, with PHP, and with lightweight directory access protocols (LDAP).

Requirements

LDAP support in PHP is not enabled by default. You will need to enable it. For more details check PHP documentation at http://php.net/ldap.

You may want to use Active Directory/PHP Helper library from http://adldap.sourceforge.net. If you do want to use it, download the library and place it into the *kb_installation_dir/admin/lib/custom* directory.

Here is an simple example of the function customized to authenticate against a LDAP server:

```
function remoteDoAuth($username, $password) {

    require_once 'custom/adLDAP.php';

    $auth = false;
    if(empty($username) || empty($password)) {
        return $auth;
    }


    //create the AD LDAP connection
    $adldap = new adLDAP();

    $user = array();
    $ldap_user = $adldap->user_info($username, array(*));

    // if found, populate $user array
    if($adldap->authenticate($username, $password)){
        $user['first_name'] = $ldap_user[0]['givenname'][0];
        ...
    }

    return $user;
}
```

You can find more examples in *kb_installation_dir/admin/lib/custom* directory.

There are two different types of remote authentication. It is controlled by the **KB_AUTH_TYPE** constant:

1. Adding/refreshing remote user data to KBPublisher and authenticate user.
2. Authentication by existing KBPublisher user.

Adding/refreshing remote user data to KB and authenticate user

KB_AUTH_TYPE = 1

On success, the authentication function ***remoteDoAuth*** should return an associative array with the following keys:

- first_name
- last_name
- email
- username
- password-- as the user types when they login, that is, not encrypted
- remote_user_id -- a unique userID stored in your system
- role_id - (optional)
- priv_id - (optional) privilege for user. If user has a privilege, he will have access to Admin Area

Authentication by existing KBPublisher user

KB_AUTH_TYPE = 2

On success, the authentication function ***remoteDoAuth*** should return.

- the user_id of the user in the KBPublisher USER table (kbp_user)
  OR
- Associative array with keys (user_id, username), for example: array('user_id'=>7, 'username'=>'Test').

There are also other configuration variables

- **KB_AUTH_AREA**
  1 - Enabled for Public area only, remote authentication allowed on Public Area login screen
  2 - Enabled for Public and Admin areas

- **KB_AUTH_LOCAL**
  0 - never try to authenticate by KBPublisher built in authentication
  1 - always try to authenticate by KBPublisher built in authentication first
  2 - will try to authenticate by KBPublisher built in authentication if Remote Authentication failed

- **KB_AUTH_LOCAL_IP**
  Only users with specified IP(s) are allowed to be authenticated by KBPublisher's built in authentication
  it only matters when KB_AUTH_LOCAL = 1 or 2.
  You can set a specific IP or an IP range. Use an "-" to separate IP ranges, and a";" to separate individual IP addresses.
  For example: *127.0.0.1;210.234.12.15;192.168.1.1-192.168.255.255*

- **KB_AUTH_REFRESH_TIME**
  The time, in seconds, to rewrite user data, (3600*24*30 = 30 days), works if KB_AUTH_TYPE = 1
  0 - never. Once the user is created, data in kb table never updated by script
  1 - on every authentication request user data in the knowledgebase will be synchronized with data provided by script.

- **KB_AUTH_RESTORE_PASSWORD_LINK**
  Here you may provide a link where your remote users can restore their password.
  Set to false not to display the restore password link at all.
  KBPublisher will determine whether to set your link or the built-in one.

- **KB_AUTH_AUTO** (Using Auto Authentication)
  This variable controls whether or not the user sees a login screen and has to log in to KBPublisher, or whether they are automatically logged in.
  0 - Disabled, user gets login screen
  1 - Enabled, user doesn't see login screen
  2 - Enabled, in debug mode. User doesn't see login screen. It allows not to block "Auto Auth" if authentication failed.
  Use only for debugging and don't forget to change back to 1 or 0 when you have authentication working.

Remote Authentication scenarios

Here are some examples/scenarios for Remote Authentication:

1. On the first authentication request the remote user is added to KBPublisher table of users.
   On the second and subsequent requests he/she is authenticated by KBPublisher's built in authentication.
   KB_AUTH_LOCAL = 1
   KB_AUTH_TYPE = 1
   KB_AUTH_REFRESH_TIME = 1.

2. Always authenticate by Remote Authentication and rewrite user data in the knowledgebase.
   KB_AUTH_LOCAL = 0
   KB_AUTH_TYPE = 1
   KB_AUTH_REFRESH_TIME = 1

3. On the first authentication request the remote user is added to KBPublisher users.
   On the second and subsequent requests the user is authenticated by remote authentication and his/her KBPublisher data is synchronized with data provided by your script, depending on the KB_AUTH_REFRESH_TIME.
   KB_AUTH_LOCAL = 0
   KB_AUTH_TYPE = 1
   KB_AUTH_REFRESH_TIME = 3600*24*30 (30 days).

4. KBPublisher tries to authenticate the user by built-in Authentication first.On failure KBPublisher tries to authenticate the user by Remote Authentication.
   KB_AUTH_LOCAL = 1
   KB_AUTH_TYPE = 2

5. If user IP matches KB_AUTH_LOCAL_IP range, then KBPublisher tries to authenticate the user by built-in Authentication first. If the IP does not match, or built-in authentication fails, KBPublisher tries to authenticate the user by Remote Authentication.
   KB_AUTH_LOCAL = 1
   KB_AUTH_LOCAL_IP = '192.168.1.1-192.168.255.255';
   KB_AUTH_TYPE = 2

Adding KB_AUTH_AUTO = 1 to any of these means the user will not be asked to type in their username and password.
System will get that information automatically. See this article how to set up Auto Authentication.

Auto authentication allows you to automatically authenticate users.

Steps to enable Auto Authentication

- Set the constant **KB_AUTH_AUTO** in the file *admin/lib/custom/remote_auth.php* to **1**
- Customize the **_remoteAutoAuth** function in the file *admin/lib/custom/remote_auth.php* to catch current user data and return it
- Rename the function **_remoteAutoAuth** to **remoteAutoAuth**

Customizing the remoteAutoAuth function

In your installation there is a folder *admin/lib/custom*. Within that folder is a file called *remote_auth.php*. This file contains the *_remoteAutoAuth* function. Customize this function to get the current user's credentials.

On success, the function ***remoteAutoAuth*** should return an associative array with keys (username, password) for the user. For example: array('username'=>'John', 'password'=>'Test').

Here is a simple example of the function customized using HTTP authentication:

```
function remoteAutoAuth() {

   $user = false;

   if(isset($_SERVER['PHP_AUTH_USER']) && isset($_SERVER['PHP_AUTH_PW'])) {
       $user = array();
       $user['username'] = $_SERVER['PHP_AUTH_USER'];
       $user['password'] = $_SERVER['PHP_AUTH_PW'];
   }

   return $user;
}
```

Debugging auto-remote authentication

You can debug auto-authentication by setting **KB_AUTH_AUTO** = 2, which allows you to continually try relogging in so that you can fix any problems without having to start over every time. **Important**: Don't forget to reset this back to 1 (or 0) when you have finished debugging.