HelpOnAuthentication

- ExternalCookie

ExternalCookie

This is some sample code you might find useful when you want to use some

external cookie (made by some other program, not moin) with moin.

See the +++ places for customizing it to your needs.
Copy this

code into your farmconfig.py or wikiconfig.py by pasting it over the current:

class Config(DefaultConfig):

OR

class FarmConfig(DefaultConfig):

from MoinMoin.config.multiconfig import DefaultConfig from MoinMoin.auth import BaseAuth

This is included in case you want to create a log file during testing

import time def writeLog(*args): '''Write an entry in a log file with a timestamp and all of the args.''' s = time.strftime('%Y-%m-%d %H:%M:%S ',time.localtime()) for a in args: s = '%s %s;' % (s,a) log = open('/somelogfile', 'a') # +++ location for log file log.write('\n' + s + '\n') log.close() return

these 2 methods are examples of how to "authenticate" the other application cookie

import MySQLdb def verifySession(sidHash): # +++ to use this, uncomment a procedure call below in ExternalCookie """Return True if sidHash value exists (meaning user is currently logged on), false otherwise.

```
If you are not a MySQL user, find another way to store this information.  ActiveSession is a
two-column table
containing a hashed cookie value (sidHash) plus a date-time stamp (tStamp).
Your other application must add an entry to this table each time a user logs on and delete the
entry when the user logs off.
"""
db = MySQLdb.connect(db='mydb',user='myid',passwd='mypw') #+++ user ID needs read access
c = db.cursor()
q = 'select sidHash from ActiveSession where sidHash="%s"' % sidHash
result = c.execute(q)
c.close()
if result == 1:
    return True
return False
```

def verifySessionPlus(sidHash,timeout=3600*4): # +++ to use this, uncomment a procedure call below in ExternalCookie """Return True if sidHash value exists (meaning user is currently logged on), false otherwise.

```
This version of verifySession deletes entries inactive for more than 4 hours and
updates the tStamp field with each moin transaction.  If performance is
important, find another way to delete inactive sessions.
"""
db = MySQLdb.connect(db='mydb',user='myid',passwd='mypw') # +++ user ID needs write access
c = db.cursor()
q = 'delete from ActiveSession where tStamp<"%s"' % int(time.time() - timeout) # delete
inactive entries
result = c.execute(q)
q = 'update ActiveSession set tStamp=%s where sidHash="%s"' % (int(time.time()),sidHash)
result = c.execute(q)
c.close()
if result == 1:
    return True
```

```
return False
```

class ExternalCookie(BaseAuth): name = 'external_cookie' # +++ The next 2 lines may be useful if you are overriding the username method in your themes. # +++ If commented out, wiki pages will not have login or logout hyperlinks login_inputs = ['username', 'password'] # +++ required to get a login hyperlink in wiki navigation area # logout_possible = True # +++ required to get a logout hyperlink in wiki navigation area

```
def __init__(self, autocreate=False):
    self.autocreate = autocreate
    BaseAuth.__init__(self)

def request(self, request, user_obj, **kw):
    """Return (user-obj,False) if user is authenticated, else return (None,True). """
    # login = kw.get('login') # +++ example does not use this; login is expected in other
application
    # user_obj = kw.get('user_obj')  # +++ example does not use this
    # username = kw.get('name') # +++ example does not use this
    # logout = kw.get('logout') # +++ example does not use this; logout is expected in other
application
    import Cookie
    user = None  # user is not authenticated
    try_next = True  # if True, moin tries the next auth method in auth list

    otherAppCookie = "MoinAuth" # +++ username, email,useralias, session ID separated by #
    try:
        cookie = Cookie.SimpleCookie(request.cookies) #moin 1.9
    except Cookie.CookieError:
        cookie = None # ignore invalid cookies

    if cookie and otherAppCookie in cookie: # having this cookie means user auth has already
been done in other application
        import urllib
        if sys.version_info[:2] > (2, 5):
            cookievalue = cookie[otherAppCookie].value # moin 1.9 & python 2.6
        else:
            cookievalue = cookie[otherAppCookie] # moin 1.9 & python 2.5
        # writeLog('cookievalue',cookievalue)
        # +++ decode and parse the cookie value - edit this to fit your needs.
        #~ cookievalue = urllib.unquote(cookievalue) # cookie value is urlencoded, decode it
moin 1.9
        #~ cookievalue = cookievalue.decode('iso-8859-1') # decode cookie charset to unicode
moin 1.9
        cookievalue = cookievalue[1: -1] # strip leading and ending quotes moin 1.9
        # writeLog('cookievalue',cookievalue)
        cookievalues = cookievalue.split('#') # cookie has format
loginname#email#aliasname#sessionid

        email = aliasname = sessionid = ''
        try:  # extract fields from other app cookie
            auth_username = cookievalues[0] # the wiki username
            email = cookievalues[1] # email is required for user to change and save
userpreferences
            aliasname = cookievalues[2] # aliasname is useful only if auth_username is not
wiki-like
            sessionid = cookievalues[3] # optional other app session ID  -- a unique timestamp
or random number
```

```
        except IndexError:
            pass  # do not need aliasname or sessionid unless you uncomment lines below
        # writeLog('auth_username',auth_username)
        # writeLog('email',email)
        # writeLog('aliasname',aliasname)
        # writeLog('sessionid',sessionid)

        # +++ any hacker can create a cookie, uncomment these lines to verify the cookie was
created by the other application
        if auth_username:
            import hashlib  # +++ python 2.5; see http://code.krypto.org/python/hashlib for
2.3, 2.4 download
            sidHash = hashlib.md5(cookievalue).hexdigest() # moin 1.9
            # writeLog('sidHash',sidHash)
            sidOK = verifySession(sidHash) # verify user is currently logged on to the other
application +++ or use verifySessionPlus
            if not sidOK:
                auth_username = None
        # writeLog('auth_username after verifySession',auth_username)

        if auth_username:
            # we have an authorized user, create the moin user object
            from MoinMoin.user import User
            # giving auth_username to User constructor means that authentication has already
been done.
            user = User(request, name=auth_username, auth_username=auth_username,
auth_method=self.name)
            changed = False
            if email != user.email: # was the email addr externally updated?
                user.email = email ;
                changed = True # yes -> update user profile
            # if aliasname != user.aliasname: # +++ was the aliasname externally updated?
                # user.aliasname = aliasname ;
                # changed = True # yes -> update user profile

            if user:
                user.create_or_update(changed)
            if user and user.valid:
                try_next = False # have valid user; stop processing auth method list
    # writeLog(str(user), try_next)
    return user, try_next
```

from MoinMoin.config import multiconfig, url_prefix_static

class Config(multiconfig.DefaultConfig):

class FarmConfig(multiconfig.Defa

ultConfig):

```
auth = [ExternalCookie(autocreate=True)] # only way to login is external application #
autocreate parameter new in 1.8.0

# +++ these are suggested changes to the user preferences form if the  external_cookie  is the
only auth method
user_form_disable = ['name', 'aliasname', 'email',] # don't let the user change these, but
show them:
user_form_remove = ['password', 'password2', 'css_url', 'logout', 'create',
'account_sendmail','jid'] # remove completely:
#~ user_autocreate = True # +++ Moin will autocreate new user ID if none exists
cookie_lifetime = (12, 12)   # (anonymous,logged-in) default (0,12)  page trails for anonymous
users moin 1.9
```

++++++++++++++++++

end of external_cookie example, your custom configurations continue below

}}}

== Initial Testing ==

If you use the Firefox browser and have the Web Developer addon extension installed, initial testing will be fast and painless. Log on to your other application and click on Tools... Web Developer... Cookies... View Cookie Information. You will probably find at least one cookie that looks like a session identifier created by your application at login, usually these will have a value with a large random number and perhaps a time stamp.

Next, click on Tools... Web Developer... Cookies... Add Cookie. Give the cookie a name of ''Moin``Auth'' (case sensitive). Set the value to `yourname#youremail@yourprovider.com` with no

spaces, and be sure to use the '''#''' character as a separator. If your other application and the wiki run on the same host, set the host to the same value used by your other application (if not, omit the host from the domain name to share cookies across hosts). Set the path value to "/" without the quotes. Click the ''Session Cookie'' check box and click save.

Open a second browser window and load your starting wiki page. The wiki should recognize you as a logged in user. Do a quick test to verify that you can edit and save a wiki page and verify that you can change your User Preferences.

== Changing the Other Application ==

The example external_cookie method expects the cookie value to contain several pieces of information separated with the '''#''' character:

- the wiki user id -- always required
- the user's email address -- needed if the user is to be able to update and save MoinMoin User Preferences
- user alias -- usually not required, used to override user IDs that are not wiki-like user names
- unique session ID -- a big random number or timestamp and random number

Your application must store the Moin``Auth cookie with a path set to '/'. Setting the path to '/' is normally not recommended because it presents a small security risk. It allows an application to read the cookies created by a different application. In this case, that is exactly our intent -- MoinMoin must be able to read the cookies set by the other application.

As you have already demonstrated to yourself above, any user with the skill to install the Firefox Web Developer addon could easily create a cookie and hack his way into the wiki using someone else's ID.
To prevent this from happening, MoinMoin will need to validate the cookie value against an entry in a secure shared storage area created by the other application.

We want to avoid creating a potential new security issue by building a cross-reference of user ID to session ID or even a list of session IDs that might be of use to a hacker. A better way is for your application to hash (not encrypt) the Moin``Auth cookie value and store the result in a secure shared storage area. Add a timestamp to each entry so obsolete entries can be removed later.

There is commented out code in the example wikiconfig.py to perform a hash of the cookie contents and validate the result against a MySQL table. In addition, there is alternative commented out code that will delete entries from the My``SQL table that are more than 4 hours old, validate the hashed cookie value against the table and update the timestamp. A better method is to modify the other application to remove obsolete entries from the table, perhaps each time a user logs in.

As you begin to modify your application almost all of your testing can be done with the Firefox Web Developer extension. Before you login to the other application there should be no Moin`Auth cookie, after login there should be a Moin`Auth cookie, after logout there should be no cookie. In addition, the shared storage area should be updated with a new hashed cookie value after login

and cleared after logout. Accessing a wiki page after you are logged in will cause moin to create a MOIN_SESSION cookie.

== Modifying themes ==

The next step is to modify the login and logout links in the navigation area of wiki pages. If you limit your users to one theme, modify the code below to point to your other application's login and logoff pages. If you allow users to choose from among several themes, it may be easiest to modify the /Moin``Moin/theme/**init**.py script directly.

If you always log on to your other application before accessing a wiki page, this modification and the one following are not required. If you comment out the `login_inputs` and `logout_possible` variables near the top of the External``Cookie class, then your wiki pages will not contain login and logout hyperlinks.

The code below is for moin 1.9.

{{{#!python def username(self, d): """ Assemble the username / userprefs link

```
    @param d: parameter dictionary
    @rtype: unicode
    @return: username html
    """
    request = self.request
    _ = request.getText

    userlinks = []
    # Add username/homepage link for registered users. We don't care
    # if it exists, the user can create it.
    if request.user.valid and request.user.name:
        interwiki = wikiutil.getInterwikiHomePage(request)
        name = request.user.name
        aliasname = request.user.aliasname
        if not aliasname:
            aliasname = name
        title = "%s @ %s" % (aliasname, interwiki[0])
        # link to (interwiki) user homepage
        homelink = (request.formatter.interwikilink(1, title=title, id="userhome",
generated=True, *interwiki) +
                    request.formatter.text(name) +
                    request.formatter.interwikilink(0, title=title, id="userhome",
*interwiki))
        userlinks.append(homelink)
        # link to userprefs action
        if 'userprefs' not in self.request.cfg.actions_excluded:
            userlinks.append(d['page'].link_to(request, text=_('Settings'),
                                       querystr={'action': 'userprefs'}, id='userprefs',
rel='nofollow'))

    if request.user.valid:
        if request.user.auth_method in request.cfg.auth_can_logout:
            userlinks.append('<a href="https://wiki.raz-ev.org/myapp/Logout">%s</a>' %
_('Logout', formatted=False)) # +++ other app logout page
    else:
```

```
        query = {'action': 'login'}
        # special direct-login link if the auth methods want no input
        if request.cfg.auth_login_inputs == ['special_no_input']:
            query['login'] = '1'
        if request.cfg.auth_have_login:
            userlinks.append('<a  href="https://wiki.raz-ev.org/myapp/Login">%s</a>' %
 _("Login", formatted=False)) # +++ other app login page

    userlinks = [u'<li>%s</li>' % link for link in userlinks]
    html = u'<ul id="username">%s</ul>' % ''.join(userlinks)
    return html
```

}}}

== Modifying the Application Login Page ==

As a final touch, you may want to consider modifying your login page for wiki users who decide to login from a wiki page. This depends upon your choice of web servers, but most will pass something similar to an HTTP_REFERER field which will contain the prior URL.

Check this value to determine if the referrer page was a wiki page, if so save the URL and at the end of a successful logon either redirect the current page back the referring wiki page or open a new window with the URL of the referring page.