ucam-webauth

Release 0.9.2

Aug 13, 2019

Contents

Contents:

CHAPTER 1

Quickstart

1.1 Using the flask decorator

```
import flask
from flask import Flask
from ucam_webauth.raven.flask_glue import AuthDecorator

# Werkzeug deduces the hostname from the 'Host' or
# 'X-Forwarded-Host' headers, so we need a whitelist
class R(flask.Request):
    trusted_hosts = {'your-domain.com', 'www.your-domain.com'}

app = Flask(__name__)
app.request_class = R
app.secret_key = "a secret key"
auth_decorator = AuthDecorator(desc="My website")

@app.route("/some_url")
@auth_decorator
def my_view():
    return "You are " + auth_decorator.principal

if __name__ == '__main__':
    app.run()
```

1.2 Requiring all flask requests be authenticated

```
import flask
from flask import Flask
from ucam_webauth.raven.flask_glue import AuthDecorator

```

(continues on next page)

(continued from previous page)

```
# Werkzeug deduces the hostname from the 'Host' or
# 'X-Forwarded-Host' headers, so we need a whitelist
class R(flask.Request):
    trusted_hosts = {'your-domain.com', 'www.your-domain.com'}

app = Flask(__name__)
app.request_class = R
app.secret_key = "a secret key"
auth_decorator = AuthDecorator()

app.before_request(auth_decorator.before_request)

@app.route("/")
def home():
    return "You are " + auth_decorator.principal

if __name__ == '__main__':
    app.run()
```

1.3 Manual request building and response parsing

To create requests:

```
>>> from ucam_webauth.raven import Request, Response
>>> r = Request(url="http://host/response/path", desc="My website")
>>> print str(r)
https://raven.cam.ac.uk/auth/authenticate.html?url=http%3A%2F%2Fhost%2Fresponse
↪%2Fpath&ver=3&desc=My+website
```

And parse responses:

```
>>> r = Response("3!200!!20130705T150000Z!1373000000-00000-00!"
                 "http%3A%2F%2Fhost%2Fpath!djr61!current!pwd!!"
                 "36000!!2!signature-omitted")
>>> r.success
True
>>> r.principal
"djr61"
>>> r.ptags
set(["current"])
```

1.3.1 Warning

You must check various properties of received responses. See *Checking response values*

1.4 Integrating with existing authentication or session management

```
from ucam_webauth import raven
from datetime import datetime
from flask import Flask, session, flash, url_for, redirect, abort, request
```

(continues on next page)

(continued from previous page)

```
app = Flask(__name__)
app.secret_key = "a secret key"

@app.route("/")
def home():
    return "<a href='{0}'>Log in</a>".format(url_for('login'))

@app.route("/login")
def login():
    u = url_for("response", _external=True)
    r = raven.Request(url=u)
    return redirect(str(r))

@app.route("/response")
def response():
    r = raven.Response(request.args["WLS-Response"])

    # checking url, issue, iact and aauth is very important!
    # Werkzeug deduces the hostname from the 'Host' or
    # 'X-Forwarded-Host' headers, so we need a whitelist
    request.trusted_hosts = {'www.your-domain.com', 'your-domain.com'}
    if r.url != request.base_url:
        print "Bad url"
        abort(400)

    issue_delta = (datetime.utcnow() - r.issue).total_seconds()
    if not -5 < issue_delta < 15:
        print "Bad issue"
        abort(403)

    if r.success:
        # a no-op here, but important if you set iact or aauth
        if not r.check_iact_aauth(None, None):
            print "check_iact_aauth failed"
            abort(403)

        session["user"] = r.principal

        return redirect(url_for("secrets"))
    else:
        return redirect(url_for("home"))

@app.route("/secrets")
def secrets():
    if session.get("user", None) is None:
        abort(401)
    return "You are {0}".format(session["user"])

if __name__ == "__main__":
    app.run(debug=True)
```

1.4.1 Warning

You must check various properties of received responses. See *Checking response values*

1.5 See also

The included simple_demo flask app serves as a far more comprehensive example, including:

- decorator usage
- integration with existing authentication (i.e., user is offered to log in via Raven or some other method)
- full Raven logout
- message flashing

CHAPTER 2

Security

2.1 Checking response values

You *must* check the *url*, *issue*, *auth* and *sso* attributes of the response:

- check that *url* matches the current URL being requested / is what you expect.

  Not checking *url* will allow another evil website administrator to replay responses produced by Raven log-ins to her website to yours, thereby impersonating someone else. (Using params as a token (below) doesn't help, since the attacker can obtain a matching *(cookie, params)* pair from you first, and then ask the victim to authenticate with *params* set to that value.)

  Some frameworks, notably Werkzeug, deduce the current hostname from the *Host* or *X-Forwarded-Host* headers (with the latter taking precedence).

  See also:

  werkzeug#609 and issue 5

  This technique may be used to whitelist domains in Flask:

  ```
  class R(flask.Request):
      trusted_hosts = {'www.danielrichman.co.uk'}
  app.request_class = R
  ```

  Alternatively, you could sanitise *Host* and *X-Forwarded-Host* in your web-server.

  If you might have query parameters in your *url*, you need to take care to handle negative respones from the WLS. See *Response URL for "cancels"*.

- check *issue* is within an acceptable range of *now*

  . . . lest someone replay an old response to log in again

- check *auth* and *sso* match *iact* and *aauth*

  see *ucam_webauth.Response.check_iact_aauth()*

  Not checking *iact*/*aauth* will allow those restrictions to be bypassed by crafting a custom request to the WLS.

2.2 Using params as a token

You might like to set a random nonce in the Request's *params*, save a hashed (with secret salt) or signed copy in a cookie, and check that they match in the *Response*.

This is *not* a substitute for any of the checks above, but does make the *WLS-Response* values in your web server access logs useless.

ucam_webauth.flask_glue.AuthDecorator does this.

2.3 Signing keys

The keys used by Raven to sign responses are included with *python-ucam-webauth*. I took care in retrieving them, however you should trust neither me nor the method by which you installed this package. *You should check that the copies of the certificates you have are correct / match the files at the links below* (and audit the code you've just installed, I guess).

- `pubkey2` from https://raven.cam.ac.uk/project/keys/
- `pubkey901` from https://raven.cam.ac.uk/project/keys/demo_server/

CHAPTER 3

Misc

3.1 Response URL for "cancels"

The short story is that when the WLS wants to send a "response" to the WAA, it takes the URL you provided in the request, adds a *WLS-Response* query parameter, and redirects the client to that URL.

Happily, it guarantees that this will be done by appending *(?|&)WLS-Response=...* to the URL (which means that this process is easy to undo, which is a necessary part of *Checking response values*).

However: while in version 3 it preserves any query parameters that were already in the request URL, in version 1 of the protocol it will not: that is, it deletes the query component before appending *?WLS-Response...*. Furthermore, while the current version of the WLS appears to reply with version 3 upon success, if you click "cancel" then it will use version 1, presumably because of reasons.

The WLS does include in its response a copy of some of the request parameters, in particular, the return URL. It is possible to extract this from the response, and after inspecting WLS-Response, perform a redirect to it, recovering the deleted query parameters. The *flask_glue* does exactly this, and so hopefully you should not suffer problems on account of this behaviour.

Note that if you for some reason had the requirement that requests to a certain page need only be Raven authenticated if a certain query parameter is present, then something like this would not work correctly:

```
def my_before_request():
    if "special" in request.args:
        return flask_glue.before_request()
    else:
        return None
```

... since if a user clicks Cancel, the special query parameter would not be set, so the *before_request* function would run, and the response from the WLS would not be handled. Instead, something like this would be necessary:

```
def my_before_request():
    if "special" in request.args or "WLS-Response" in request.args:
        return flask_glue.before_request()
```

(continues on next page)

(continued from previous page)

```
    else:
        return None
```

If you are not using the *flask_glue*, I suggest where possible just avoiding having significant query parameters on the URL that you use to perform Raven authentication, and then simply check that *request.base_url* matches the URL in the signed response. Otherwise, have a look at the implementation of *flask_glue* for inspiration.

CHAPTER 4

python module documentation

4.1 ucam_webauth

The ucam_webauth module implements version 3 of the WAA to WLS protocol.

It is not set up to talk to a specific WAA (i.e., Raven), and subclassing this modules' classes is required to make it functional. In particular, you probably want to use *ucam_webauth.raven*.

The protocol is implemented as defined at https://raven.cam.ac.uk/project/waa2wls-protocol.txt at the time of writing (though that URL may have since been replaced with a newer version). A copy of wawa2wls-protocol.txt is included with python-raven, and more information can be found at https://raven.cam.ac.uk/project/.

WAA A WAA is a "Web Application Agent" (i.e., an application using this module)

WLS The "Web Login Service" (i.e., Raven)

ucam_webauth.**ATYPE_PWD**
ucam_webauth.**STATUS_SUCCESS**
ucam_webauth.**STATUS_CANCELLED**
ucam_webauth.**STATUS_NOATYPES**
ucam_webauth.**STATUS_UNSUPPORTED_VERSION**
ucam_webauth.**STATUS_BAD_REQUEST**
ucam_webauth.**STATUS_INTERACTION_REQUIRED**
ucam_webauth.**STATUS_WAA_NOT_AUTHORISED**
ucam_webauth.**STATUS_AUTHENTICATION_DECLINED**
AuthenticationType and *Status* instances used as constants in requests and responses

They compare equal with their corresponding integers (for status codes) and strings (for atypes).

ucam_webauth.**STATUS_CODES**
A dict mapping status.code (i.e., the integer status code) to the relevant status object

class ucam_webauth.**AuthenticationType**(*name*, *description*)
An Authentication Type

This class exists to create the ucam_webauth.AUTH_PWD constant.

name
the name by which Ucam-webauth knows it

description
a sentence describing it

Note that comparing an *AuthenticationType* object with a `str` (or another *AuthenticationType* object) will compare the *name* attribute only. Further, `str(atype) == atype.name`.

class ucam_webauth.**Status**(*code*, *name*, *description*)
A WLS response Status

code
a (three digit) integer

name
short name for the status

description
description: a sentence describing the status

Note that comparing a *Status* object with an integer (or another *Status* object) will compare the *code* attribute only. Further, *int(status_object) == status_object.code*

class ucam_webauth.**Request**(*url*, *desc=None*, *aauth=None*, *iact=None*, *msg=None*, *params=None*, *fail=None*, *encode_strings=True*)
A Request to the WLS

Parameters

- **url** (`str`) – a fully qualified URL; the user will be returned here (along with the Response as a query parameter) afterwards
- **desc** (`str`) – optional description of the resource/website (encoding - see below)
- **aauth** (`set` of *AuthenticationType* objects) – optional set of permissible authentication types; we require the user to use one of them (if empty, the WLS uses its default set)
- **iact** (`True`, `False` or `None`) – interaction required, forbidden or don't care (respectively)
- **msg** (`str`) – optional message explaining why authentication is required (encoding - see below)
- **params** (`str`) – data, which is returned unaltered in the *Response*
- **fail** (`bool`) – if True, and authentication fails, the WLS must show an error message and not redirect back to the WAA

All parameters are available as attributes as of Request object, once created.

iact

- `True`: the user must re-authenticate
- `False`: no interaction with the user is permitted (the request will only succeed if the user's identity can be returned without interacting at all)
- `None` (default): interacts if required

msg

desc
The 'msg' and 'desc' parameters are restricted to printable ASCII characters (0x20 - 0x7e). The WLS will convert '<' and '>' to '<' and '>' before using either string in HTML, preventing the inclusion of markup. However, it does not touch '&', so HTML character and numeric entities may be used to represent other characters.

If *encode_strings* is `True`, `&` will be escaped to `&`, and non-ascii characters in *msg* and *desc* will be converted to their numeric entities.

Otherwise, it is up to you to encode your strings. An error will be raised if *msg* or *desc* contain non-printable-ASCII characters.

params
The ucam-webauth protocol does not specify any restrictions on the content of params. However, awful things may happen if you put arbitrary binary data in here. The Raven server appears to interpret non-ascii contents as latin-1, turn them into html entities in order to put them in a hidden HTML input element, then turn them back into (hopefully) the same binary data to be returned in the Response. As a result it outright rejects 'params' containing bytes below 0x20, and has the potential to go horribly wrong and land you in encoding hell.

Basically, you probably want to base64 params before giving it to a Request object.

__str__(*self*)
Evaluating `str(request_object)` gives a query string, excluding the `?`

class ucam_webauth.**Response**(*string*)
A Response from the WLS

Constructed by parsing *string*, the 'encoded response string' from the WLS.

The Response class has the following attributes, which must be set by subclassing it (see *raven.Response*):

old_version_ptags
A `set` of `str` objects

The ptags attribute is set to this value if the version of the response is less than 3

keys
A dict mapping key identifiers (*kid*) to a RSA public key (which must be an object with a `verify(digest, signature)` method that returns a `bool`)

A Response object has the following attributes:

Always present

ver
response protocol version (`int`)

status
response status (*Status* constant)

msg
a text message describing the status of the authentication request, suitable for display to the end-user (`str`)

issue
response creation time (`datetime`, timezone naive - the values are UTC)

id
an "identifier" for the response. (`int`)

The tuple (issue, id) is guaranteed to be unique

url
the value of *url* supplied in the request, or equivalently, the URL to which this response was delivered (`str`)

success
shorthand for `status == STATUS_SUCCESS` (`bool`)

params
a copy of *params* from the request (`str`)

signed
whether the signature was present and has been verified (`bool`)

Note that a present but invalid signature will produce an exception when parsed.

Present if authentication was successful, otherwise ‘‘None‘‘:

principal:
the authenticated identity of the user (`str`)

ptags
attributes or properties of the principal (`frozenset` of `str` objects)

auth
method of authentication used (*AuthenticationType* constant, or `None`)

If authentication was not established by interaction (i.e., the client was already authenticated) then *auth* is `None`

sso
previous successful authentication types used (`frozenset` of *AuthenticationType* constants)

sso will not be the empty set if auth is `None`

Optional if authentication was successful, otherwise ‘‘None‘‘:

life
remaining life of the user's WLS session (`int`, in seconds)

Required if signed is True:

kid
identifies the RSA key used to sign the request (`str`)

check_iact_aauth(*iact*, *aauth*)
Check that the WLS honoured *iact*, *aauth*

This method checks that *self.auth*, *self.sso* are consistent with the *iact* and *aauth*, which should be the same as the values used to construct the *Request*.

4.1.1 flask_glue

This module provides glue to make using python-raven with Flask easy

class ucam_webauth.flask_glue.**AuthDecorator**(*desc=None*, *aauth=None*, *iact=None*, *msg=None*, *max_life=7200*, *use_wls_life=False*, *inactive_timeout=None*, *issue_bounds=(15, 5)*, *require_principal=None*, *require_ptags=frozenset([u'current'])*, *can_trust_request_host=False*)
An instance of this class decorates views to add authentication.

To use it, you'll need to subclass it and set response_class, request_class and logout_url (see `raven.flask_glue.AuthDecorator`). Then:

```
auth_decorator = AuthDecorator() # settings, e.g., desc="..." go here

@app.route("/some_url")
@auth_decorator
def my_view():
    return "You are " + auth_decorator.principal
```

Or to require users be authenticated for all views:

```
app.before_request(auth_decorator.before_request)
```

Note that since it uses flask.session, you'll need to set `app.secret_key`.

We need to be able to reliably determine the hostname of the current website. This is retrieved from `flask.Request.url`. By default, Werkzeug will respect the value of a `X-Forwarded-Host` header, which means that a man-in-the-middle can have someone authenticate to *their* website, and forward the response from the WLS on to you. You must either set `flask.Request.trusted_hosts`, for example like so:

```
class R(flask.Request):
    trusted_hosts = {'www.danielrichman.co.uk'}
app.request_class = R
```

... or sanitise both the *Host* header *and* the *X-Forwarded-Host* header in your web-server. If you choose the second option, set *can_trust_request_host*.

This tries to emulate the feel of applying mod_ucam_webauth to a file.

The decorator wraps the view in a function that calls *before_request()* first, calling the original view function if it does not return a redirect or abort.

You may wish to catch the 401 and 403 aborts with `app.errorhandler`.

The *principal*, their *ptags*, the *issue* and *life* from the WLS are available as attributes of the *AuthDecorator* object (magic properties that retrieve the current values from `flask.session`). Further, the attributes *expires* and *expires_all* give information on when the ucam_webauth session will expire.

For the *desc*, *aauth*, *iact*, *msg* parameters, see *ucam_webauth.Request*.

Note that the *max_life*, *use_wls_life* and *inactive_timeout* parameters deal with the ucam_webauth session *only*; they only affect `flask.session["_ucam_webauth"]`. Flask's session expiry, cookie lifetimes, etc. are independent.

Parameters

- **max_life** (`int` (seconds) or `None`) – upper bound on how long a successful authentication can last before it expires and the user must reauthenticate
- **use_wls_life** (`bool`) – should we lower the life of the session to the life reported by the WLS, if it is less than *max_life*?
- **inactive_timeout** (`int` (seconds) or `None`) – expire the session if no request is processed via this decorator in *inactive_timeout* seconds
- **issue_bounds** (`tuple`: (`int`, `int`) (seconds)) – a tuple, (lower, upper) - how close the *issue* (datetime that the WLS says the authentication happened at) must be

to *now* (i.e., require `now - lower < issue < now + upper`; this is a combination of two settings found in mod_ucam_webauth: *clock skew* and *response timeout*, `issue_bounds=(clock_skew + response_timeout, clock_skew)` is equivalent)

- **require_principal** (`set` of `str`, or `None`) – require the principal to be in the set
- **require_ptags** (`set` of `str`, or `None`) – require the ptags to contain *any* string in *require_ptags* (i.e., non empty intersection)
- **can_trust_request_host** (`bool`) – Can we trust the hostname in `request.url`? (see *Checking response values*)

More complex customisation is possible:

- override *check_authorised()* to do more complex checking than *require_principal*, *require_ptags* (note that this replaces checking *require_principal*, *require_ptags*)
- override *session_new()*

  The *AuthDecorator* only touches `flask.session["_ucam_webauth"]`. If you've saved other (important) things to the session object, you may want to clear them out when the state changes.

  You can do this by subclassing and overriding session_new. It is called whenever a response is received from the WLS, except if the response is a successful re-authentication after session expiry, with the same *principal* and *ptags* as before.

To log the user out, call *logout()*, which will clear the session state. Further, *logout()* returns a `flask.redirect()` to the Raven logout page. Be aware that the default flask session handlers are susceptible to replay attacks.

POST requests: Since it will redirect to the WLS and back, the auth decorator will discard any POST data in the process. You may wish to either work around this (by subclassing and saving it somewhere before redirecting) or ensure that when it returns (with a GET request) to the URL, a sensible page is displayed (the form, or an error message).

__call__(*view_function*)
Wraps *view_function* with the auth decorator

(*AuthDecorator* objects are callable so that they can be used as function decorators.)

Calling it returns a 'wrapper' view function that calls `request()` first.

principal
The current principal, or `None`

ptags
The current ptags, or `None`

issue
When the last WLS response was issued

issue is converted to a unix timestamp (`int`), rather than the `datetime` object used by *ucam_webauth.Response*. (*issue* is `None` if there is no current session.)

life
life of the last WLS response (`int` seconds), or `None`

last
Time (`int` unix timestamp) of the last decorated request

expires
When (`int` unix timestamp) the current auth. will expire

expires_all
A list of all things that could cause the current auth. to expire

A list of (`str`, `int` unix timestamp) tuples; (*reason*, *when*).

reason will be one of "config max life", "wls life" or "inactive".

logout()
Clear the auth., and return a redirect to the WLS' logout page

before_request()
The "main" method

- checks if there is a response from the WLS
  - checks if the current URL matches that which the WLS said it redirected to (avoid an evil admin of another site replaying successful authentications)
  - checks if `flask.session` is empty - if so, then we deduce that the user has cookies disabled, and must abort immediately with 403 Forbidden, or we will start a redirect loop
  - checks if *params* matches the token we set (and saved in `flask.session`) when redirecting to Raven
  - checks if the authentication method used is permitted by *aauth* and user-interaction respected *iact* - if not, abort with 400 Bad Request
  - updates the state with the response: updating the *principal*, *ptags* and *issue* information if it was a success, or clearing them (but setting a flag - see below: 401 Authentication Required will be thrown after redirect) if it was a failure
  - returns a redirect that removes `WLS-Response` from `request.args`
- checks if the "response was an authentication failure" flag is set in `flask.session` - if so, clears the flag and aborts with 401 Authentication Required
- checks to see if we are authenticated (and the session hasn't expired)
  - if not, returns a redirect that will sends the user to the WLS to authenticate
- checks to see if the *principal* / *ptags* are permitted
  - if not, aborts with a 403 Forbidden
- updates the 'last used' time in the state (to implement *inactive_timeout*)

Returns `None`, if the request should proceed to the actual view function.

check_authorised(*principal*, *ptags*)
Check if an authenticated user is authorised.

The default implementation requires the principal to be in the whitelist `require_principal` (if it is not `None`, in which case any principal is allowed) and the intersection of `require_ptags` and *ptags* to be non-empty (unless `require_ptags` is `None`, in which case any ptags (or no *ptags* at all) is permitted).

Note that the default value of `require_ptags` in `raven.flask_glue.AuthDecorator` is `{"current"}`.

session_new()
Called when a new user authenticates

More specifically, when *`principal`* or *`ptags`* changes.

4.2 ucam_webauth.raven

Raven

The Raven module subclasses *ucam_webauth.Request* and *ucam_webauth.Response* in order to use the Raven URLs and the Raven response settings (default ptags and signing keys).

ucam_webauth.raven.**PUBKEY2**
The key used to verify responses, from https://raven.cam.ac.uk/project/keys/

ucam_webauth.raven.**RAVEN_AUTH**
The WLS' authentication start page: `RAVEN_AUTH.format(quoted_query_string)` will produce a request

ucam_webauth.raven.**RAVEN_LOGOUT**
The WLS' logout page: redirecting to this URL will log the user out of Raven completely.

class ucam_webauth.raven.**Request**(*url*, *desc=None*, *aauth=None*, *iact=None*, *msg=None*, *params=None*, *fail=None*, *encode_strings=True*)
ucam_webauth.Request, configured for live Raven

Refer to *ucam_webauth* for documentation.

__str__()
Returns a full URL: the raven authentication url, with the query string set to contain the request data

class ucam_webauth.raven.**Response**(*string*)
ucam_webauth.Response, configured for live Raven

Refer to *ucam_webauth* for documentation.

keys
A single key; *kid* '2' maps to *PUBKEY2*.

old_version_ptags = frozenset([u'current'])

4.2.1 flask_glue

class ucam_webauth.raven.flask_glue.**AuthDecorator**(*desc=None*, *aauth=None*, *iact=None*, *msg=None*, *max_life=7200*, *use_wls_life=False*, *inactive_timeout=None*, *issue_bounds=(15, 5)*, *require_principal=None*, *require_ptags=frozenset([u'current'])*, *can_trust_request_host=False*)
ucam_webauth.flask_glue.AuthDecorator, configured for live Raven

Refer to *ucam_webauth.flask_glue* for documentation.

request_class
alias of *ucam_webauth.raven.Request*

response_class
alias of *ucam_webauth.raven.Response*

logout_url = u'https://raven.cam.ac.uk/auth/logout.html'

4.2.2 demoserver

Raven Demo Server

Provides Request and Response subclasses (as in the raven module), except these use the settings of the Raven Demo Server, http://raven.cam.ac.uk/project/test-demo/

ucam_webauth.raven.demoserver.**PUBKEY901**
The key used to verify responses, from https://raven.cam.ac.uk/project/keys/demo_server/

ucam_webauth.raven.demoserver.**RAVEN_DEMO_AUTH**
The WLS' authentication start page: `RAVEN_DEMO_AUTH.format(quoted_query_string)` will produce a request

ucam_webauth.raven.demoserver.**RAVEN_DEMO_LOGOUT**
The WLS' logout page: redirecting to this URL will log the user out of Raven completely.

class ucam_webauth.raven.demoserver.**Request**(*url*, *desc=None*, *aauth=None*, *iact=None*, *msg=None*, *params=None*, *fail=None*, *encode_strings=True*)
ucam_webauth.Request, configured for the Raven demo server

Refer to *ucam_webauth* for documentation.

__str__()
Returns a full URL: the raven demoserver authentication url, with the query string set to contain the request data

class ucam_webauth.raven.demoserver.**Response**(*string*)
ucam_webauth.Response, configured for the Raven demo server

Refer to *ucam_webauth* for documentation.

keys
A single key; *kid* '901' maps to *PUBKEY901*.

old_version_ptags = frozenset([u'current'])

CHAPTER 5

Links

- source on github
- documentation
- pypi page
- Raven documentation
- WAA2WLS protocol

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

u

Index

Symbols

A

B

C

D

E

I

K

L

M

N

O

P

R

S

U

V

W