Pyzor Documentation

Release 1.0

Frank Tobin

June 29, 2016

Contents

Contents:

CHAPTER 1

Introduction

Pyzor is a collaborative, networked system to detect and block spam using digests of messages.

Using Pyzor client a short digest is generated that is likely to uniquely identify the email message. This digest is then sent to a Pyzor server to:

- check the number of times it has been reported as spam or whitelisted as not-spam
- report the message as spam
- whitelist the message as not-spam

Since the entire system is released under the GPL, people are free to host their own independent servers. There is, however, a well-maintained and actively used public server available (courtesy of SpamExperts) at:

```
public.pyzor.org:24441
```

1.1 Contribute

- Issue Tracker
- Source Code

1.2 Getting the source

To clone the repository using git simply run:

```
git clone https://github.com/SpamExperts/pyzor
```

Please feel free to fork us and submit your pull requests.

1.3 Running tests

The pyzor tests are split into *unittest* and *functional* tests.

Unitests perform checks against the current source code and **not** the installed version of pyzor. To run all the unittests suite:

```
env PYTHONPATH=. python tests/unit/__init__.py
```

Functional tests perform checks against the installed version of pyzor and **not** the current source code. These are more extensive and generally take longer to run. They also might need special setup. To run the full suite of functional tests:

```
env PYTHONPATH=. python tests/functional/__init__.py
```

There is also a helper script available that sets-up the testing enviroment, also taking into consideration the python version you are currently using:

```
./scripts/run_tests
```

Note: The authentication details for the MySQL functional tests are taken from the test.conf file.

1.4 License

The project is licensed under the GNU GPLv2 license.

CHAPTER 2

Getting Pyzor

2.1 Installing

The recommended and easiest way to install Pyzor is with `pip`:

```
pip install pyzor
```

In order to upgrade your Pyzor version run:

```
pip install --upgrade pyzor
```

2.2 Compatibility

Pyzor is compatible with the following Python versions and implementations:

- Python 2.6
- Python 2.7
- Python 3.2+
- PyPy
- PyPy3

Pyzor will also work on Windows.

2.3 Downloading

If you don't want to or cannot use `pip` to download and install Pyzor. You can do so directly from the source:

```
python setup.py install
```

You can find the latest and older versions of Pyzor on PyPI.

2.4 Dependencies

2.4.1 Pyzor Client

If you plan on only using Pyzor to check message against our public server, then there are no required dependencies.

2.4.2 Pyzor Server

If you want to host your own Pyzor Server then you will need an appropriate back-end engine. Depending on what engine you want to use you will also need to install the required python dependecies. Please see *Engines* for more details.

The Pyzor also support the gevent library. If you want to use this feature then you will need to first install it.

CHAPTER 3

Usage

Contents:

3.1 Pyzor Client

The Pyzor Client is a Python script deployed with the package. It provides a command line interface to the Pyzor Client API:

```
pyzor [options] command
```

You can also use the Python API directly to integrate Pyzor in your solution. For more information see pyzor.client.

3.1.1 Commands

Check

Checks the message read from stdin and prints the number of times it has been reported and the number of time it has been whitelisted. If multiple servers are listed in the configuration file each server is checked:

```
$ pyzor check < spam.eml
public.pyzor.org:24441  (200, 'OK')     134504  4681
```

The exit code will be:

- 1 if the report count is 0 **or** the whitelist count is > 0
- 0 if the report count is > 0 **and** the whitelist count is 0

Note that you can configure this behaviour by changing the report/whitelist thresholds from the configuration file or the command-line options. See *client configuration*.

Info

Prints detailed information about the message. The exit code will always be zero (0) if all servers returned (200, 'OK'):

```
$ pyzor info < spam.eml
public.pyzor.org:24441  (200, 'OK')
        Count: 134538
        Entered: Sat Jan  4 10:01:34 2014
        Updated: Mon Mar 17 12:52:04 2014
```

```
WL-Count: 4681
        WL-Entered: Mon Jan  6 14:32:01 2014
        WL-Updated: Fri Mar 14 16:11:02 2014
```

Report

Reports to the server a digest of each message as spam. Writes to standard output a tuple of (error-code, message) from the server. If multiple servers are listed in the configuration file the message is reported to each one:

```
$ pyzor report < spam.eml
public.pyzor.org:24441      (200, 'OK')
```

Whitelist

Reports to the server a digest of each message as not-spam. Writes to standard output a tuple of (error-code, message) from the server. If multiple servers are listed in the configuration file the message is reported to each one:

```
$ pyzor whitelist < spam.eml
public.pyzor.org:24441      (200, 'OK')
```

Note: This command is not available by default for the anonymous user.

Ping

Merely requests a response from the servers:

```
$ pyzor ping
public.pyzor.org:24441      (200, 'OK')
```

Pong

Can be used to test pyzor, this will always return a large number of reports and 0 whitelist, regardless of the message:

```
$ pyzor pong < ham.eml
public.pyzor.org:24441  (200, 'OK')     9223372036854775807     0
```

Predigest

Prints the message after the predigest phase of the pyzor algorithm:

```
$ pyzor predigest < test.eml
Thisisatest.
```

Digest

Prints the message digest, that will be sent to the server:

```
$ pyzor digest < spam.eml
c3a8e8d987f07843792d2ab1823b04cc3cb87482
```

Genkey

Based upon a secret passphrase gathered from the user and randomly gathered salt, prints to standard output a tuple of "salt,key". Used to put account information into the accounts file.

Local Whitelist

Add a message to the local whitelist file, and therefore ignoring the digest and returning 0 reports for the digest without contacting the pyzor server:

$ pyzor local_whitelist < false_positive.eml

Local UnWhitelist

Remove a message from the local whitelist file:

$ pyzor local_unwhitelist < false_positive.eml

3.1.2 Servers File

This file contains a list of servers that will be contacted by the Pyzor client for every operation. If no servers are specified it defaults to the public server:

```
public.pyzor.org:24441
```

The servers can also be specified as IP addresses, but they must always be followed by the port number.

For example having this in `~/.pyzor/servers`:

```
# This is comment
public.pyzor.org:24441
127.0.0.1:24441
```

Will configure the client to check both the public server and a local one:

```
$ pyzor ping
public.pyzor.org:24441  (200, 'OK')
127.0.0.1:24441 (200, 'OK')
```

3.1.3 Input Style

Pyzor accepts messages in various forms. This can be controlled with the *style* configuration or command line option. Currently support are:

- msg - individual RFC5321 message
- mbox - mbox file of messages
- digests - Pyzor digests, one per line

3.2 Pyzor Server

The Pyzor Server will listen on the specified address and any serve request from Pyzor Clients.

3.2.1 Daemon

Starting

The Pyzor Server can be started as a daemon by using the `--detach` option. This will:

- daemonize the script and detach from tty
- create a pid file
- redirect any output to the specified file

Example:

```
$ pyzord --detach /dev/null --homedir=/home/user/.pyzor/
```

Stopping

To safely stop the Pyzor Server you can use the `TERM` signal to trigger a safe shutdown:

```
$ kill -TERM `cat /home/user/.pyzor/pyzord.pid`
```

Reloading

The reload signal will tell the Pyzor Server to reopen and read the access and passwd files. This is useful when adding new accounts or changing the permissions for an existing account. This is done by sending the `USR1` signal to the process:

```
$ kill -USR1 `cat /home/user/.pyzor/pyzord.pid`
```

3.2.2 Engines

The Pyzor Server supports a number of back-end database engines to store the message digests.

Gdbm

This is the default engine, and the easiest to use and configure. But this it is also highly inefficient and not recommended for servers that see a large number of requests.

To use the the `gdbm` engine simply add to the config file `~/.pyzor/config`:

```
[server]
Engine = gdbm
DigestDB = pyzord.db
```

The database file will be created if it didn't previously exists, and will be located as usual in the specified Pyzor homedir.

For more information about GDBM see http://www.gnu.org.ua/software/gdbm/.

MySQL

This will require the MySQL-python library.

Note: *MySQL-python* does not currently support Python 3

To configure the `MySQL` engine you will need to:

- Create a MySQL database (for e.g. pyzor)
- Create a MySQL table with the following schema:

```
    CREATE TABLE `digests` (
            `digest` char(40) default NULL,
            `r_count` int(11) default NULL,
            `wl_count` int(11) default NULL,
            `r_entered` datetime default NULL,
            `wl_entered` datetime default NULL,
            `r_updated` datetime default NULL,
            `wl_updated` datetime default NULL,
            PRIMARY KEY  (`digest`)
    )
```

- Create a MySQL user
- Grant `ALL PRIVILEGES` to that user on the newly created table

To use the `MySQL` engine add to the configuration file:

```
[server]
Engine = mysql
DigestDB = localhost,user,password,pyzor,digests
```

Redis

This will require the redis library.

To use the `redis` engine simply add to the configuration file:

```
[server]
Engine = redis
DigestDB = localhost,6379,,0
```

Or if a password is required:

```
[server]
Engine = redis
DigestDB = localhost,6379,password,0
```

In the example above the redis database used is 0.

Migrating

If you want to migrate your database from one engine to another there is an utility script installed with pyzor designed to do this. Note that the arguments are the equivalent of the `Engine` and `DigestDB` options. Some usage examples:

- Moving a database from gdbm to redis:

```
    pyzor-migrate --se gdbm --sd testdata/backup.db --de redis --dd localhost,6379,,0
```

- Moving a database from redis to MySQL:

```
    pyzor-migrate --se redis --sd localhost,6379,,0 --de mysql --dd localhost,root,,pyzor,public
```

3.2.3 Access File

This file can be used to restrict or grant access to various server-side operations to accounts. For more information on setting up accounts see *accounts*.

The format is very similar to the popular tcp_wrappers hosts.{allow,deny}:

```
privilege ... : username ... : allow|deny
```

privilege a list of whitespace-separated commands The keyword `all` can be used to to refer to all commands.

username a list of whitespace-separated usernames. The keyword `all` can be used to refer to all users other than the anonymous user. The anonymous user is refereed to as `anonymous`.

allow|deny whether or not the specified user(s) can perform the specified privilege(s) on the line.

The file is processed from top to bottom, with the first match for user/privilege being the value taken. Every file has the following implicit final rule:

```
all : all anonymous : deny
```

If this file is non-existant, the following default is used:

```
check report ping pong info : anonymous : allow
```

3.3 Accounts

Pyzor Accounts can be used to grant or restrict access to the Pyzor Server, by ensuring the client are authenticated.

To get an account on a server requires coordination between the client user and server admin. Use the following steps:

1. User and admin should agree on a username for the user. Allowed characters for a username are alpha-numerics, the underscore, and dashes. The normative regular expression it must match is `^[-\.\w]+$`. Let us assume they have agreed on *bob*.
2. User generates a key with `pyzor genkey`. Let us say that it generates the salt,key of:

```
    227bfb58efaba7c582d9dcb66ab2063d38df2923,8da9f54058c34e383e997f45d6eb74837139f83b
```

3. Assuming the server is at `127.0.0.1:9999`, the user puts the following entry into `~/.pyzor/accounts`:

```
    127.0.0.1 : 9999 : bob : 227bfb58efaba7c582d9dcb66ab2063d38df2923,8da9f54058c34e383e997f45d6eb74
```

   This tells the Pyzor Client to use the *bob* account for server `127.0.0.1:9999`. It will still use the *anonymous* user for all other servers.

4. The user then sends the key (the part to the right-hand side of the comma) to the admin.
5. The admin adds the key to their `~/.pyzor/pyzord.passwd`:

```
    bob : 8da9f54058c34e383e997f45d6eb74837139f83b
```

6. Assuming the admin wants to give the privilege of whitelisting (in addition to the normal permissions), the admin then adds the appropriate permissions to `~/.pyzor/pyzord.access`:

   ```
       check report ping pong info whitelist : bob : allow
   ```

   For more information see *Access File*.

7. To reload the account and access information send the `USR1` signal to the daemon.

3.4 Procmail

To use Pyzor in a procmail system, consider using the following simple recipe:

```
 :0 Wc
| pyzor check :0 a
pyzor-caught
```

If you prefer, you can merely add a header to message marked with Pyzor, instead of immediately filtering them into a separate folder:

```
 :0 Wc
| pyzor check :0 Waf
| formail -A 'X-Pyzor: spam'
```

3.5 ReadyExec

ReadyExec is a system to eliminate the high startup-cost of executing scripts repeatedly. If you execute Pyzor a lot, you might be interested in installing ReadyExec and using it with Pyzor.

To use Pyzor with ReadyExec, the readyexecd.py server needs to be started as:

```
readyexecd.py socket_file pyzor.client.run
```

`socket_file` can be any (non-existing) filename you wish ReadyExec to use, such as /tmp/pyzor:

```
readyexecd.py /tmp/pyzor pyzor.client.run
```

Individual clients are then executed as:

```
readyexec socket_file options command cmd_options
```

For example:

```
readyexec /tmp/pyzor check
readyexec /tmp/pyzor report
readyexec /tmp/pyzor whitelist --style=mbox
readyexec /tmp/pyzor -d ping
```

CHAPTER 4

Configuration

The format of this file is INI-style (name=value, divided into [sections]). Names are case insensitive. All values which are filenames can have shell-style tildes (~) in them. All values which are relative filenames are interpreted to be relative to the Pyzor homedir. All of these options can be overridden by command-line arguments.

It is recommended to use the provided sample configuration. Simply copy it in pyzor's `homedir`, remove the `.sample` from the name and alter any configurations you prefer.

4.1 client configuration

ServersFile Must contain a newline-separated list of server addresses to report/whitelist/check with. All of these server will be contacted for every operation. See *Servers File*.

AccountsFile File containing information about accounts on servers. See Accounts.

LogFile If this is empty then logging is done to stdout.

LocalWhitelist Specify the local whitelist file name.

Timeout This options specifies the number of seconds that the pyzor client should wait for a response from the server before timing out.

Style Specify the message input style. See *Input Style*.

ReportThreshold If the number of reports exceeds this threshold then the exit code of the pyzor client is 0.

WhitelistThreshold If the number of whitelists exceed this threshold then exit code of the pyzor client is 1.

4.2 server configuration

Port Port to listen on.

ListenAddress Address to listen on.

LogFile File to contain server logs.

SentryDSN If set add a SentryHandler to the log file.

SentryLogLevel Set the log level for the SentryHandler. (default is `WARN`)

UsageLogFile File to contain server usage logs (information about each request).

UsageSentryDSN If set add a SentryHandler to the usage log file.

UsageSentryLogLevel Set the log level for the usage SentryHandler. (default is `WARN`)

PidFile This file contain the pid of the pyzord daemon when used with the *–detach* option.

PasswdFile File containing a list of user account information. See Accounts.

AccessFile File containing information about user privileges. See *Access File*.

Gevent If set to true uses the gevent library.

Engine Then engine type to be used for storage. See *Engines*.

DigestDB The database connection information. Format varies depending on the engine used. See *Engines*.

CleanupAge The maximum age of a record before it gets removed (in seconds). To disable this set to 0.

PreFork The number of workers the pyzor server should start. The server will pre-fork itself and split handling the requests among all workers. This is disabled by default.

Threads If set to true, the pyzor server will use multi-threading to serve requests.

MaxThreads The maximum number of concurrent threads (0 means unlimited).

DBConnections The number of database connections kept opened by the server (0 means a new one for each request).

Note: *DBConnections* only applies to the MySQL engine.

Processes If set to true, the pyzor server will use multi-processing to serve requests.

MaxProcesses The maximum number of concurrent processes (cannot be unlimited).

CHAPTER 5

Whitelisting

Whitelisting messages is disabled by default on the public server (`public.pyzor.org`). However if you want to request a whitelist you can use the web service at:

http://public.pyzor.org/whitelist/

You need to upload the raw message and the corresponding pyzor message digest.

CHAPTER 6

Changelog

6.1 Pyzor 1.0.0

New features:

- New pyzor commands `local_[un]whitelist` are available for managing a local whitelist on the client side. (#10)
- New `PreFork` option for the pyzor server. This allows creating multiple workers for handling pyzor requests. (#26)

Perfomance enhancements:

- Improve usage of the Redis engine by using Hashes instead of string for storing digests. The migration tool can be used to update you current database. (#29)

Others:

- PyPy3 compatibility verified and introduced into the Travis-CI system. (#24)
- Unification of the storage engines types. (#30)
- Improved check on the public whitelisting request service to skip sending requests to whitelist message that have not been reported to the public database or have been already whitelisted. (#27)

6.2 Pyzor 0.9.0

Bug fixes:

- Fix gdbm decoding issue. (#20)
- Fix inconsistencies accounts and addresses. (#22)

New features:

- Strip content inside `<style>` and <*script*> tags during HTML normalization. (#19)
- Improvements in Pyzor client error codes. (#17)
- Add support for logging to Sentry (#7)

Perfomance enhancements:

- Do report and whitelist in a single step for MySQL Server Engine. (#5)

Others:

- You can now requests whitelisting a message by using a simple form available at: http://public.pyzor.org/whitelist/

6.3 Pyzor 0.8.0

Bug fixes:

- Fix unicode decoding issues. (#1)

New features:

- A new option for the pyzor server to set-up digest forwarding.
- A new script `pyzor-migrate` is now available. The script allows migrating your digest database from one engine to another. (#2)

Perfomance enhancements:

- Use multiple threads when connecting to multiple servers in the pyzor client script. (#5)
- A new `BatchClient` is available in pyzor client API. The client now send reports in batches to the pyzor server. (#13)

Others:

- Small adjustments to the pyzor scripts to add Windows compatibility.
- Automatically build documentation.
- Continuous integration on Travis-CI.
- Test coverage on coveralls.

6.4 Pyzor 0.7.0

Bug fixes:

- Fix decoding bug when messages are badly formed
- Pyzor now correctly creates the specified homedir, not the user's one

New features:

- Logging is now disabled by default
- Automatically run 2to3 during installation (if required)

New pyzord features:

- Added ability to disable expiry
- New redis engine support has been added
- New option to enable gevent
- Added the ability to reload accounts and access files using USR1 signal
- Added the ability to safely stop the daemon with TERM signal
- Split the usage-log and normal log in two separate files
- Pyzord daemon can now daemonize and detach itself

6.5 Pyzor 0.6.0

- pyzor and pyzord will now work with Python3.3 (if the the 2to3-3.3 is previously ran)
- pyzord and pyzor now supports IPv6
- Improved handling of multi-threading (signals where again removed) for the mysql engine
- Introduced multi-processing capabilities
- Improved HTML parsing
- Introduced self document sample configurations
- Introduced ability to set custom report/whitelist thresholds for the pyzor client
- Greatly improved tests coverage

6.6 Pyzor 0.5.0

Note that the majority of changes in this release were contributed back from the Debian pyzor package.

- Man pages for pyzor and pyzord.
- Changing back to signals for database locking, rather than threads. It is likely that signals will be removed again in the future, but the existing threading changes caused problems.
- Basic checks on the results of “discover”.
- Extended mbox support throughout the library.
- Better handling on unknown encodings.
- Added a –log option to log to a file.
- Better handling of command-line options.
- Improved error handling.

CHAPTER 7

About

7.1 History

Pyzor initially started out to be merely a Python implementation of Razor, but due to the protocol and the fact that Razor's server is not Open Source or software libre, Frank Tobin decided to implement Pyzor with a new protocol and release the entire system as Open Source and software libre.

7.2 Protocol

The central premise of Pyzor is that it converts an email message to a short digest that uniquely identifies the message. Simply hashing the entire message is an ineffective method of generating a digest, because message headers will differ when the content does not, and because spammers will often try to make a message unique by injecting random/unrelated text into their messages.

To generate a digest, the 2.0 version of the Pyzor protocol:

- Discards all message headers.
- If the message is greater than 4 lines in length:
- Discards the first 20% of the message.
- Uses the next 3 lines.
- Discards the next 40% of the message.
- Uses the next 3 lines.
- Discards the remainder of the message.
- Removes any 'words' (sequences of characters separated by whitespace) that are 10 or more characters long.
- Removes anything that looks like an email address (X@Y).
- Removes anything that looks like a URL.
- Removes anything that looks like HTML tags.
- Removes any whitespace.
- Discards any lines that are fewer than 8 characters in length.

This is intended as an easy-to-understand explanation, rather than a technical one.

CHAPTER 8

Reference

8.1 pyzor.engines

8.1.1 pyzor.engines.common

Common library shared by different engines.

class `pyzor.engines.common.`**`DBHandle`**(*single_threaded*, *multi_threaded*, *multi_processing*, *prefork*)

Bases: `tuple`

`multi_processing`
Alias for field number 2

`multi_threaded`
Alias for field number 1

`prefork`
Alias for field number 3

`single_threaded`
Alias for field number 0

exception `pyzor.engines.common.`**`DatabaseError`**

Bases: `exceptions.Exception`

class `pyzor.engines.common.`**`Record`**(*r_count=0*, *wl_count=0*, *r_entered=None*, *r_updated=None*, *wl_entered=None*, *wl_updated=None*)

Bases: `object`

Prefix conventions used in this class: r = report (spam) wl = whitelist

`r_increment`()

`r_update`()

`wl_increment`()

`wl_update`()

class `pyzor.engines.common.`**`BaseEngine`**

Bases: `object`

Base class for Pyzor engines.

`absolute_source` = **True**

classmethod `get_prefork_connections` (*fn*, *mode*, *max_age=None*)
Yields an unlimited number of partial functions that return a new engine instance, suitable for using toghether with the Pre-Fork server.

`handles_one_step` = False

`items` ()
Return a list of (key, record).

`iteritems` ()
Iterate over pairs of (key, record).

`report` (*keys*)
Report the corresponding key as spam, incrementing the report count.

Engines that implement don't implement this method should have handles_one_step set to False.

`whitelist` (*keys*)
Report the corresponding key as ham, incrementing the whitelist count.

Engines that implement don't implement this method should have handles_one_step set to False.

8.1.2 pyzor.engines.gdbm

Gdbm database engine.

class `pyzor.engines.gdbm_.`**`GdbmDBHandle`** (*fn*, *mode*, *max_age=None*)
Bases: *`pyzor.engines.common.BaseEngine`*

`absolute_source` = True

`apply_method` (*method*, *varargs=()*, *kwargs=None*)

classmethod `decode_record` (*s*)

static `decode_record_0` (*s*)

classmethod `decode_record_1` (*s*)

classmethod `encode_record` (*value*)

`fields` = ('r_count', 'r_entered', 'r_updated', 'wl_count', 'wl_entered', 'wl_updated')

`handles_one_step` = False

`items` ()

`iteritems` ()

`log` = <logging.Logger object>

`reorganize_period` = 86400

`start_reorganizing` ()

`start_syncing` ()

`sync_period` = 60

`this_version` = '1'

class `pyzor.engines.gdbm_.`**`ThreadedGdbmDBHandle`** (*fn*, *mode*, *max_age=None*, *bound=None*)
Bases: *`pyzor.engines.gdbm_.GdbmDBHandle`*

Like GdbmDBHandle, but handles multi-threaded access.

`apply_method` (*method*, *varargs=()*, *kwargs=None*)

8.1.3 pyzor.engines.mysql

MySQLdb database engine.

class `pyzor.engines.mysql.`**`MySQLDBHandle`**(*fn*, *mode*, *max_age=None*)

Bases: *`pyzor.engines.common.BaseEngine`*

`absolute_source` = **False**

classmethod **`get_prefork_connections`**(*fn*, *mode*, *max_age=None*)

Yields a number of database connections suitable for a Pyzor pre-fork server.

`handles_one_step` = **True**

`items`()

`iteritems`()

`log` = **<logging.Logger object>**

`reconnect`()

`reconnect_period` = **60**

`reorganize_period` = **86400**

`report`(*keys*)

`start_reorganizing`()

`whitelist`(*keys*)

class `pyzor.engines.mysql.`**`ProcessMySQLDBHandle`**(*fn*, *mode*, *max_age=None*)

Bases: *`pyzor.engines.mysql.MySQLDBHandle`*

`reconnect`()

class `pyzor.engines.mysql.`**`ThreadedMySQLDBHandle`**(*fn*, *mode*, *max_age=None*, *bound=None*)

Bases: *`pyzor.engines.mysql.MySQLDBHandle`*

`reconnect`()

8.1.4 pyzor.engines.redis

Redis database engine.

class `pyzor.engines.redis_.`**`RedisDBHandle`**(*fn*, *mode*, *max_age=None*)

Bases: *`pyzor.engines.common.BaseEngine`*

`absolute_source` = **False**

classmethod **`get_prefork_connections`**(*fn*, *mode*, *max_age=None*)

Yields a number of database connections suitable for a Pyzor pre-fork server.

`handles_one_step` = **True**

`items`()

`iteritems`()

`log` = **<logging.Logger object>**

`report`(**args*, ***kwargs*)

`whitelist`(**args*, ***kwargs*)

class `pyzor.engines.redis_.`**`ThreadedRedisDBHandle`**(*fn*, *mode*, *max_age=None*, *bound=None*)

Bases: *`pyzor.engines.redis_.RedisDBHandle`*

`pyzor.engines.redis_.`**`decode_date`**(*stamp*)

Return a datetime object from a Unix Timestamp.

`pyzor.engines.redis_.`**`encode_date`**(*date*)

Convert the date to Unix Timestamp

`pyzor.engines.redis_.`**`safe_call`**(*f*)

Decorator that wraps a method for handling database operations.

Database backends for pyzord.

The database class must expose a dictionary-like interface, allowing access via __getitem__, __setitem__, and __delitem__. The key will be a forty character string, and the value should be an instance of the Record class.

If the database backend cannot store the Record objects natively, then it must transparently take care of translating to/from Record objects in __setitem__ and __getitem__.

The database class should take care of expiring old values at the appropriate interval.

8.2 pyzor.hacks

8.2.1 pyzor.hacks.py26

Hacks for Python 2.6

`pyzor.hacks.py26.`**`hack_all`**(*email=True*, *select=True*)

Apply all Python 2.6 patches.

`pyzor.hacks.py26.`**`hack_email`**()

The python2.6 version of email.message_from_string, doesn't work with unicode strings. And in python3 it will only work with a decoded.

So switch to using only message_from_bytes.

`pyzor.hacks.py26.`**`hack_select`**()

The python2.6 version of SocketServer does not handle interrupt calls from signals. Patch the select call if necessary.

Various hack to make pyzor compatible with different Python versions.

8.3 pyzor.account

A collection of utilities that facilitate working with Pyzor accounts.

Note that accounts are not necessary (on the client or server), as an "anonymous" account always exists.

class `pyzor.account.`**`Account`**(*username*, *salt*, *key*)

Bases: `object`

`pyzor.account.`**`hash_key`**(*key*, *user*, *hash_=<built-in function openssl_sha1>*)

Returns the hash key for this username and password.

lower(H(U + ':' + lower(K))) K is key (hex string) U is username H is the hash function (currently SHA1)

`pyzor.account.`**`key_from_hexstr`**(*s*) `pyzor.account.`**`sign_msg`**(*hashed_key*, *timestamp*, *msg*, *hash_=<built-in function openssl_sha1>*)

Converts the key, timestamp (epoch seconds), and msg into a digest.

lower(H(H(M) + ':' T + ':' + K)) M is message T is integer epoch timestamp K is hashed_key H is the hash function (currently SHA1)

`pyzor.account.`**`verify_signature`**(*msg*, *user_key*)

Verify that the provided message is correctly signed.

The message must have "User", "Time", and "Sig" headers.

If the signature is valid, then the function returns normally. If the signature is not valid, then a pyzor.SignatureError() exception is raised.

8.4 pyzor.client

Networked spam-signature detection client.

```
>>> import pyzor
>>> import pyzor.client
>>> import pyzor.digest
>>> import pyzor.config
```

To load the accounts file:

```
>>> accounts = pyzor.config.load_accounts(filename)
```

To create a client (to then issue commands):

```
>>> client = pyzor.client.Client(accounts)
```

To create a client, using the anonymous user:

```
>>> client = pyzor.client.Client()
```

To get a digest (of an email.message.Message object, or similar):

```
>>> digest = pyzor.digest.get_digest(msg)
```

To query a server (where address is a (host, port) pair):

```
>>> client.ping(address)
>>> client.info(digest, address)
>>> client.report(digest, address)
>>> client.whitelist(digest, address)
>>> client.check(digest, address)
```

To query the default server (public.pyzor.org):

```
>>> client.ping()
>>> client.info(digest)
>>> client.report(digest)
>>> client.whitelist(digest)
>>> client.check(digest)
```

Response will contain, depending on the type of request, some of the following keys (e.g. client.ping()['Code']):

All responses will have: - 'Diag' 'OK' or error message - 'Code' '200' if OK - 'PV' Protocol Version - 'Thread'

info and *check* responses will also contain: - '[WL-]Count' Whitelist/Blacklist count *info* responses will also have: - '[WL-]Entered' timestamp when message was first whitelisted/blacklisted - '[WL-]Updated' timestamp when message was last whitelisted/blacklisted

class `pyzor.client.`**`BatchClient`**(*accounts=None*, *timeout=None*, *spec=None*, *batch_size=10*)

Bases: `pyzor.client.Client`

Like the normal Client but with support for batching reports.

`flush`()

Deleting any saved digest reports.

`force`()

Force send any remaining reports.

`report`(*digest*, *address=('public.pyzor.org'*, *24441)*)

`whitelist`(*digest*, *address=('public.pyzor.org'*, *24441)*)

class `pyzor.client.`**`CheckClientRunner`**(*routine*, *r_count=0*, *wl_count=0*)

Bases: `pyzor.client.ClientRunner`

`handle_response`(*response*, *message*)

class `pyzor.client.`**`Client`**(*accounts=None*, *timeout=None*, *spec=None*)

Bases: `object`

`check`(*digest*, *address=('public.pyzor.org'*, *24441)*)

`info`(*digest*, *address=('public.pyzor.org'*, *24441)*)

`max_packet_size` = 8192

`ping`(*address=('public.pyzor.org'*, *24441)*)

`pong`(*digest*, *address=('public.pyzor.org'*, *24441)*)

`read_response`(*sock*, *expected_id*)

`report`(*digest*, *address=('public.pyzor.org'*, *24441)*)

`send`(*msg*, *address=('public.pyzor.org'*, *24441)*)

`timeout` = 5

`whitelist`(*digest*, *address=('public.pyzor.org'*, *24441)*)

class `pyzor.client.`**`ClientRunner`**(*routine*)

Bases: `object`

`handle_response`(*response*, *message*)

mesaage is a string we've built up so far

`run`(*server*, *args*, *kwargs=None*)

class `pyzor.client.`**`InfoClientRunner`**(*routine*)

Bases: `pyzor.client.ClientRunner`

`handle_response`(*response*, *message*)

8.5 pyzor.config

Functions that handle parsing pyzor configuration files.

`pyzor.config.`**`expand_homefiles`**(*homefiles*, *category*, *homedir*, *config*)

Set the full file path for these configuration files.

`pyzor.config.`**`load_access_file`**(*access_fn*, *accounts*)

Load the ACL from the specified file, if it exists, and return an ACL dictionary, where each key is a username and each value is a set of allowed permissions (if the permission is not in the set, then it is not allowed).

'accounts' is a dictionary of accounts that exist on the server - only the keys are used, which must be the usernames (these are the users that are granted permission when the 'all' keyword is used, as described below).

Each line of the file should be in the following format: operation : user : allow|deny

where 'operation' is a space-separated list of pyzor commands or the keyword 'all' (meaning all commands), 'username' is a space-separated list of usernames or the keyword 'all' (meaning all users) - the anonymous user is called "anonymous", and "allow|deny" indicates whether or not the specified user(s) may execute the specified operations.

The file is processed from top to bottom, with the final match for user/operation being the value taken. Every file has the following implicit final rule:

> all : all : deny

If the file does not exist, then the following default is used: check report ping info : anonymous : allow

`pyzor.config.`**`load_accounts`**(*filepath*)

Layout of file is: host : port : username : salt,key

`pyzor.config.`**`load_local_whitelist`**(*filepath*)

Load the local digest skip file.

`pyzor.config.`**`load_passwd_file`**(*passwd_fn*)

Load the accounts from the specified file.

Each line of the file should be in the format: username : key

If the file does not exist, then an empty dictionary is returned; otherwise, a dictionary of (username, key) items is returned.

`pyzor.config.`**`load_servers`**(*filepath*)

Load the servers file.

`pyzor.config.`**`setup_logging`**(*log_name*, *filepath*, *debug*, *sentry_dsn=None*, *sentry_lvl='WARN'*)

Setup logging according to the specified options. Return the Logger object.

8.6 pyzor.digest

Handle digesting the messages.

class `pyzor.digest.`**`DataDigester`**(*msg*, *spec=None*)

Bases: `object`

The major workhouse class.

`atomic_num_lines` **= 4**

`digest`

classmethod **`digest_payloads`**(*msg*)

`email_ptrn` **= <_sre.SRE_Pattern object>**

`handle_atomic`(*lines*)

We digest everything.

`handle_line`(*line*)

handle_pieced (*lines*, *spec*)
Digest stuff according to the spec.

longstr_ptrn = <_sre.SRE_Pattern object>

min_line_length = 8

classmethod normalize (*s*)

static normalize_html_part (*s*)

classmethod should_handle_line (*s*)

unwanted_txt_repl = ‘’

url_ptrn = <_sre.SRE_Pattern object>

value

ws_ptrn = <_sre.SRE_Pattern object>

class pyzor.digest.**HTMLStripper** (*collector*)
Bases: HTMLParser.HTMLParser

Strip all tags from the HTML.

handle_data (*data*)
Keep track of the data.

handle_endtag (*tag*)

handle_starttag (*tag*, *attrs*)

class pyzor.digest.**PrintingDataDigester** (*msg*, *spec=None*)
Bases: *pyzor.digest.DataDigester*

Extends DataDigester: prints out what we’re digesting.

handle_line (*line*)

8.7 pyzor.forwarder

Manage the forwarder process.

class pyzor.forwarder.**Forwarder** (*forwarding_client*, *remote_servers*, *max_queue_size=10000*)
Bases: object

Forwards digest to remote pyzor servers

queue_forward_request (*digest*, *whitelist=False*)
If forwarding is enabled, insert a digest into the forwarding queue if whitelist is True, the digest will be forwarded as whitelist request if the queue is full, the digest is dropped

start_forwarding ()
start the forwarding thread

stop_forwarding ()
disable forwarding and tell the forwarding thread to end itself

8.8 pyzor.message

This modules contains the various messages used in the pyzor client server communication.

class `pyzor.message.`**`CheckRequest`**(*digest=None*)

Bases: *`pyzor.message.SimpleDigestBasedRequest`*

op = 'check'

class `pyzor.message.`**`ClientSideRequest`**

Bases: *`pyzor.message.Request`*

op = None

`setup`()

class `pyzor.message.`**`InfoRequest`**(*digest=None*)

Bases: *`pyzor.message.SimpleDigestBasedRequest`*

op = 'info'

class `pyzor.message.`**`Message`**

Bases: `email.message.Message`

`ensure_complete`()

`init_for_sending`()

`setup`()

class `pyzor.message.`**`PingRequest`**

Bases: *`pyzor.message.ClientSideRequest`*

op = 'ping'

class `pyzor.message.`**`PongRequest`**(*digest=None*)

Bases: *`pyzor.message.SimpleDigestBasedRequest`*

op = 'pong'

class `pyzor.message.`**`ReportRequest`**(*digest=None*, *spec=None*)

Bases: *`pyzor.message.SimpleDigestSpecBasedRequest`*

op = 'report'

class `pyzor.message.`**`Request`**

Bases: *`pyzor.message.ThreadedMessage`*

This is the class that should be used to read in Requests of any type. Subclasses are responsible for setting 'Op' if they are generating a message,

`ensure_complete`()

`get_op`()

class `pyzor.message.`**`Response`**

Bases: *`pyzor.message.ThreadedMessage`*

`ensure_complete`()

`get_code`()

`get_diag`()

`head_tuple`()

`is_ok`()

ok_code = 200

class pyzor.message.**SimpleDigestBasedRequest**(*digest=None*)
Bases: *pyzor.message.ClientSideRequest*

add_digest(*digest*)

class pyzor.message.**SimpleDigestSpecBasedRequest**(*digest=None*, *spec=None*)
Bases: *pyzor.message.SimpleDigestBasedRequest*

class pyzor.message.**ThreadId**
Bases: int

error_value = 0

full_range = (0, 65536)

classmethod generate()

in_ok_range()

ok_range = (1024, 65536)

class pyzor.message.**ThreadedMessage**
Bases: *pyzor.message.Message*

ensure_complete()

get_protocol_version()

get_thread()

init_for_sending()

set_thread(*i*)

class pyzor.message.**WhitelistRequest**(*digest=None*, *spec=None*)
Bases: *pyzor.message.SimpleDigestSpecBasedRequest*

op = 'whitelist'

8.9 pyzor.server

Networked spam-signature detection server.

The server receives the request in the form of a RFC5321 message, and responds with another RFC5321 message. Neither of these messages has a body - all of the data is encapsulated in the headers.

The response headers will always include a "Code" header, which is a HTTP-style response code, and a "Diag" header, which is a human-readable message explaining the response code (typically this will be "OK").

Both the request and response headers always include a "PV" header, which indicates the protocol version that is being used (in a major.minor format). Both the requestion and response headers also always include a "Thread", which uniquely identifies the request (this is a requirement of using UDP). Responses to requests may arrive in any order, but the "Thread" header of a response will always match the "Thread" header of the appropriate request.

Authenticated requests must also have "User", "Time" (timestamp), and "Sig" (signature) headers.

class pyzor.server.**BoundedThreadingServer**(*address*, *database*, *passwd_fn*, *access_fn*, *max_threads*, *forwarding_server=None*)
Bases: *pyzor.server.ThreadingServer*

Same as ThreadingServer but this also accepts a limited number of concurrent threads.

process_request (*request*, *client_address*)

process_request_thread (*request*, *client_address*)

class pyzor.server.**PreForkServer** (*address*, *database*, *passwd_fn*, *access_fn*, *prefork=4*)

Bases: *pyzor.server.Server*

The same as Server, but prefork itself when starting the self, by forking a number of child-processes.

The parent process will then wait for all his child process to complete.

load_config ()

If this is the parent process send the USR1 signal to all children, else call the super method.

serve_forever (*poll_interval=0.5*)

Fork the current process and wait for all children to finish.

shutdown ()

If this is the parent process send the TERM signal to all children, else call the super method.

class pyzor.server.**ProcessServer** (*address*, *database*, *passwd_fn*, *access_fn*, *max_children=40*, *forwarding_server=None*)

Bases: SocketServer.ForkingMixIn, *pyzor.server.Server*

A multi-processing version of the pyzord server. Each connection is served in a new process. This may not be suitable for all database types.

class pyzor.server.**RequestHandler** (**args*, ***kwargs*)

Bases: SocketServer.DatagramRequestHandler

Handle a single pyzord request.

dispatches = {'info': <function handle_info at 0x7f9144bbe230>, 'whitelist': <function handle_whitelist at 0x7f9144bb

handle ()

Handle a pyzord operation, cleanly handling any errors.

handle_check (*digests*)

Handle the 'check' command.

This command returns the spam/ham counts for the specified digest.

handle_error (*code*, *message*)

Create an appropriate response for an error.

handle_info (*digests*)

Handle the 'info' command.

This command returns diagnostic data about a digest (timestamps for when the digest was first/last seen as spam/ham, and spam/ham counts).

handle_pong (*digests*)

Handle the 'pong' command.

This command returns maxint for report counts and 0 whitelist.

handle_report (*digests*)

Handle the 'report' command in a single step.

This command increases the spam count for the specified digests.

handle_whitelist (*digests*)

Handle the 'whitelist' command in a single step.

This command increases the ham count for the specified digests.

class `pyzor.server.`**`Server`**(*address*, *database*, *passwd_fn*, *access_fn*, *forwarder=None*)
Bases: `SocketServer.UDPServer`

The pyzord server. Handles incoming UDP connections in a single thread and single process.

`handle_error`(*request*, *client_address*)

`load_config`()
Reads the configuration files and loads the accounts and ACLs.

`max_packet_size` **= 8192**

`reload_handler`(**args*, ***kwargs*)
Handler for the SIGUSR1 signal. This should be used to reload the configuration files.

`shutdown_handler`(**args*, ***kwargs*)
Handler for the SIGTERM signal. This should be used to kill the daemon and ensure proper clean-up.

`time_diff_allowance` **= 180**

class `pyzor.server.`**`ThreadingServer`**(*address*, *database*, *passwd_fn*, *access_fn*, *forwarder=None*)
Bases: `SocketServer.ThreadingMixIn`, *`pyzor.server.Server`*

A threaded version of the pyzord server. Each connection is served in a new thread. This may not be suitable for all database types.

Networked spam-signature detection.

exception `pyzor.`**`AuthorizationError`**
Bases: *`pyzor.CommError`*

The signature was valid, but the user is not permitted to do the requested action.

exception `pyzor.`**`CommError`**
Bases: `exceptions.Exception`

Something in general went wrong with the transaction.

`code` **= 400**

exception `pyzor.`**`IncompleteMessageError`**
Bases: *`pyzor.ProtocolError`*

A complete requested was not received.

exception `pyzor.`**`ProtocolError`**
Bases: *`pyzor.CommError`*

Something is wrong with talking the protocol.

`code` **= 400**

exception `pyzor.`**`SignatureError`**
Bases: *`pyzor.CommError`*

Unknown user, signature on msg invalid, or not within allowed time range.

exception `pyzor.`**`TimeoutError`**
Bases: *`pyzor.CommError`*

The connection timed out.

`code` **= 504**

exception `pyzor.`**`UnsupportedVersionError`**
Bases: *`pyzor.ProtocolError`*

Client is using an unsupported protocol version.

Index