Zorp Reference Guide

BalaBit IT Ltd

Zorp Reference Guide

by BalaBit IT Ltd

Table of Contents

List of Tables

List of Examples

Preface

1. Scope

Reference material about Zorp Firewall Suite.

2. Intended audience

This guide is intended for use by system administrators who are responsible for creating and maintaining network security. You'll need experience in the following fields before reading this guide:

- Unix system administration
- Internet protocols (IP, TCP, UDP)
- Network topology and architecture

3. Summary of contents

- Chapter 1, Files and directories, describes the files and directories used by Zorp.
- Chapter 2, Command line interface, describes all commands and their possible arguments.
- Chapter 3, Messages and logging, describes the message format emitted by Zorp.
- Chapter 4, reference manual for Python functions and classes comprising Zorp.

Chapter 1. Files and directories

This chapter describes the files and directories used by the Zorp Firewall Suite.

1.1. Directories

Zorp is using the standard Unix directory hierarchy for its files, but the exact location of the files is determined at configuration time.

An installation prefix can be chosen at configuration time which is represented by `$prefix` in the Table 1-1 below.

Table 1-1. Directories used by Zorp

Directory	Description
`$prefix/bin`	Binaries and scripts
`$prefix/sbin`	Superuser binaries and scripts
`$prefix/lib/zorp`	Main Zorp library and proxy modules.
`$prefix/share/zorp`	Python modules and other architecture independent data.
`$prefix/share/man`	Manual pages for Zorp commands.
`/etc/zorp`	Configuration data for Zorp. Note that the location of this directory doesn't depend on `$prefix`.

1.2. Files

Table 1-2. Files used by Zorp

File	Description

File	Description
`$prefix/bin/zorp-config`	This script is to be used by third party proxy modules to find out necessary compilation options.
`$prefix/sbin/zorpctl`	This script is used to start, stop and query Zorp instances. It sets up the necessary environment (`LD_LIBRARY_PATH` and `PYTHONPATH`) for Zorp to run and uses the information stored in `/etc/zorp/instances.conf`
`/etc/zorp/instances.conf`	Contains startup parameters for Zorp instances.
`/etc/zorp/policy.py`	Local firewall policy used by zorp instances running on the given host.
`$prefix/share/man/`	Manual pages for zorp(8), zorpctl(8) and instances.conf(5).
`$prefix/share/zorp/pylib/*.py`	Utility classes and proxy wrappers to be used by firewall policies. `PYTHONPATH` must be set to point to this directory.
`$prefix/share/zorp/policy.boot`	Policy boot file used by Zorp at startup.
`$prefix/lib/zorp/libzorp.so`	Zorp core library, it's used by all proxy modules and the main zorp binary itself. `LD_LIBRARY_PATH` must be set so that Zorp can find this library.
`$prefix/lib/zorp/lib*.so`	Each proxy module has a corresponding shared object to be loaded on demand. A proxy module named "http" will be loaded from libhttp.so.0.

1.2.1. `/etc/zorp/instances.conf`

This file contains startup parameters for Zorp instances. Empty lines and those beginning with '#' are ignored. Otherwise the first word of each line is taken as the name of an instance. The instance name must be a valid Python identifier, so it should begin with a letter and continue with letters, numbers or underscore.

After the instance identifier, command line arguments for Zorp can be given. They'll be passed to Zorp when the instance is started. The `--as` command line argument is always passed regardless whether it is listed among parameters or not.

Example 1-1. A sample `instances.conf` file

```
zorp_intra --verbose=3 --log-spec ftp.*:8
zorp_dmz --verbose=5 --log-spec http.*:8
```

Chapter 2. Command line interface

This chapter contains reference documentation for all commands included in Zorp.

2.1. zorp-config

zorp-config is a simple script generated during the configuration phase to help third party modules finding your Zorp installation and compilation settings. It is only used for compiling modules.

2.1.1. Synopsis

```
zorp-config [--prefix[=DIR]] [--exec-prefix[=DIR]] [--
version] [--libs] [--cflags]
```

2.1.2. Arguments

Table 2-1. Command line arguments of zorp-config

Argument	Description
--prefix	
--exec-prefix	
--version	
--libs	
--cflags	

2.2. zorpctl

zorpctl is primarily used to control Zorp instances. It is able to start an instance, to

stop an already running instance, and to query version information of the currently installed Zorp Firewall.

zorpctl is a wrapper script around the main **zorp** binary, first setting the necessary environment variables (for example `LD_LIBRARY_PATH` and `PYTHONPATH`) and doing whatever is required for the currently requested operation.

To start an instance you'll need to issue a **zorpctl start <instancename>** command. Empty instance name refer to all instances, thus a simple **zorpctl start** will start all instances listed in your `instances.conf` file.

An instance is assumed to be running if its pidfile exists. Zorp creates this pidfile on startup, and deletes during exit, so this is a safe assumption. However **zorpctl** will not be able to stop instances without a pidfile.

2.2.1. Synopsis

```
zorpctl <operation> [options]
```

Where operation is one of "start", "stop", "restart" or "version" and options is usually a list of instance names.

2.2.2. Arguments

Table 2-2. Command line arguments of zorpctl

Argument	Description
`start`	Start the given or all instances.
`stop`	Stop the given or all instances.
`restart`	Stop and then start the given or all instances.
`version`	Display version and compilation information of your Zorp installation.

2.3. zorp

zorp is the main Zorp binary. Starting it will start a new Zorp instance with the specified parameters.


Note: prior to starting zorp, you have to correctly set the `LD_LIBRARY_PATH` and `PYTHONPATH` environment variables. **zorpctl** described in Section 2.2 does this for you.

2.3.1. Synopsis

```
zorp [options]
```

Where options can be one of the arguments as specified in the following table.

2.3.2. Arguments

Table 2-3. Command line arguments of zorp

Argument	Short	Description
--as <name>	*-a*	Specifies the instance name of the instance to be run.
--caps <spec>	*-C*	Set the POSIX 1003.1e capability mask of the running process as specified by `spec`. If this option is not specified the default `cap_net_bind_service,cap_net_admin=p` is used. The exact syntax of the spec is described in the `cap_from_text(3)` manual page.
--gid <num>	*-g*	Set the gid of the running process to `num`.
--help	*-h*	Display summary of command line parameters.

Argument	Short	Description
--log-spec <spec>	*-s*	Set verbosity level on a per message category basis. `spec` is a filter specification as descibed in Section 3.2.
--log-tags	*-T*	Prepend the message tag of each message when sent to the log.
--no-caps	*-N*	Zorp sets its own capability set to a default even if *--caps is not specified. This behaviour can be overridden with this parameter.*
--no-syslog	*-l*	Do not send messages to syslog, write them to stdout instead
--policy <file>	*-p*	Use `file` as the file containing the policy to use.
--uid <num>	*-u*	Set the uid of the running process to `num`.
--verbose=<num>	*-v<num>*	Set verbosity level to `num`. This is the default verbosity if the parameter *--log-spec doesn't specify one for a given log tag.*
--version	*-V*	Display version and compilation information.

Chapter 3. Messages and logging

This chapter describes the format of messages generated by Zorp, and the features you can use to control which messages are generated.

Zorp sends all its messages to the standard UNIX syslog subsystem by default, but you can change this using the `--no-syslog` command line option, which tells Zorp to send messages to its standard output. Generally this feature is meant for debugging purposes only, messages should go to syslog.

Zorp has a number of features to aid log analysis and filtering:

- consistent message format
- message tags attached to each message specifying log category
- verbosity levels on a per category basis

3.1. Verbosity level

Each message has a minimum verbosity level which specifies the minimum verbosity level of Zorp, where that given message should be written to the logfiles. This means that if a message has a minimum verbosity level of 4, and Zorp is running at `--verbose=3` that message will not be sent to your syslog. Of course at `--verbose=4` or greater the message will be displayed.

You can change the verbosity level of Zorp via the `--verbose` command line option, or on a message tag basis, using the `--log-spec` parameter.

3.2. Message tags

A message tag is a dot separated list of identifiers, specifying the non-hierarchical category for the given message. These non-hierarchical categories are named hierarchically, each part being more and more specific. Examples for message tags are: `core.policy` or `http.request`.

Message tags can be used for at least two different purposes:

1. write them to the syslog, so later analysis may make use of it
2. filter messages in different categories

The first possibility can be enabled using the `--log-tags` command line switch to Zorp.

In addition to simply writing it to your syslog, you can specify different verbosity levels for different log categories with the `--log-spec` command line switch.

`--log-spec` is a list of comma separated clauses. Each clause specifies the verbosity level for a matching set of categories. A clause is in the form `messagetag:level`, where messagetag is matched left to right, and may contain '*' as a wild-card for each part. Level is the verbosity level for the matching message tag. An example log specification might be:
`--log-spec=ftp.*:10,core.debug:0`

Processing is not stopped on the first match, the last matching clause will be used. If none of them matches, the default verbosity level specified by `--verbose` will be used.

3.3. Session ID

Zorp assigns a session identifier to each session that goes through it. This session ID is unique during a single run.

The session ID is made up from the following parts:

- instance name (as specified with the `--as` command line option).
- service name (as named in the policy file)
- service instance number (sequence number) with a ':' prepended, if the message belongs to a given instance.
- proxy module sending the message, if applicable.

The parts discussed above are separated by a slash (`/`), and the whole session id is enclosed in parentheses.

Example 3-1. Sample session ID

(zorp/intra_http:123/http)

- `zorp` is the instance name,
- `intra_http` is the name of the service
- `:123` is the sequence number of the service instance
- `http` is the module name this message was sent by.

Messages that are not related to a service are using a fake session ID with service name: "nosession".

3.4. Message format

Messages are formatted as follows:

1. information inserted by the syslog subsystem: date, hostname and program.
2. message tag with a colon appended, if enabled by the `--log-tags` command line argument.
3. session

Chapter 4. Python reference

4.1. Module AnyPy

This module defines an interface to the AnyPy proxy as implemented in Zorp. AnyPy is basically a Python proxy which means that the proxy behaviour is defined in Python by the administrator.

4.1.1. Imported modules

- `import` Connector

  Implements classes to establish a connection.

- `import` Proxy

  Module defining classes encapsulating native proxies.

- `import` Service

  Module defining service related classes.

- `import` Zorp

  Module defining global constants, and interface entry points to the Zorp core.

4.1.2. Class AnyPyProxy

This class encapsulates AnyPy, a proxy module calling a Python function to do all of its work. It can be used for defining proxies for protocols not directly supported by Zorp.

4.1.2.1. Attributes

none

4.1.2.2. Constructor __init__

Constructor to initialize an AnyPy instance.

Synopsis

```
__init__ ( self,  session )
```

Description

This constructor initializes a new AnyPy instance based on arguments and calls the inherited constructor.

Arguments

Table 4-1. Arguments for AnyPyProxy.__init__()

self	this instance
session	session we belong to

4.1.2.3. Method proxyThread

Function called by the low level proxy core to perform transferring requests.

Synopsis

```
proxyThread ( self )
```

Description

This function is called by the proxy module to perform transferring requests. It may use the `self.session.client_stream` and `self.session.server_stream` streams to read data from and write data to.

Arguments

Table 4-2. Arguments for AnyPyProxy.proxyThread()

self	this instance

Exceptions

- NotImplementedError

4.2. Module Chainer

This module defines an abstract interface (class `AbstractChainer`) used by proxies to connect to their server endpoint, and some derived classes which fill the abstract class with some real function, most notably TransparentChainer and DirectedChainer.

4.2.1. Imported modules

- `from` Connector `import` Connect

  Implements classes to establish a connection.

- `from` SockAddr `import` SockAddrInet

Module implementing SockAddr handling functions.

- `from` Stream `import` Stream

  Module exporting an interface to the Zorp.Stream component.

- `from` Zorp `import` *

  Module defining global constants, and interface entry points to the Zorp core.

- `import` socket

4.2.2. Class AbstractChainer

Proxies are organized hierarchically, where the top level proxy is directly connected to server, and the low level proxy is connected to the upper level proxy. The term chaining means establishing the connection between a proxy and its parent. A chainer is a class responsible for chaining.

A chainer is only required for top level proxies, but may present in stacked ones too. The default chaining behaviour when there's no chainer defined (ie. `self.session.chainer` is `None`) is to pick up `self.session.server_fd`, which in turn gets set when the stacked proxy is started.

The task of chainers used for top level proxies is to connect to the target server.

4.2.2.1. Attributes

none

4.2.2.2. Method connectServer

Function to perform chaining to the parent proxy or remote server.

Synopsis

```
connectServer (
        self,
```

```
session,
host,
port,
)
```

Description

This is an interface function to be called to chain up to the parent proxy (or to the remote server)

The parameters `host` and `port` are hints given by the proxy, they may be ignored (like in TransparentChainer) or taken into account (like in InbandChainer).

Arguments

Table 4-3. Arguments for AbstractChainer.connectServer()

self	this instance
session	Session object
host	host to connect to (sockaddr).
port	port to connect to

Exceptions

- NotImplementedError

4.2.2.3. Method establishConnection

Function to actually establish a connection.

Synopsis

```
establishConnection (
        self,
        session,
        local,
        remote,
        )
```

Description

Internal function to establish a connection with the given local and remote addresses. It is used by derived chainer classes after finding out the destination address to connect to. This function performs access control checks.

Arguments

Table 4-4. Arguments for AbstractChainer.establishConnection()

self	this instance
session	Session object
local	bind address
remote	host to connect to

Exceptions

- DACException

Returns

The fd of the connection to the server

4.2.3. Class DirectedChainer

This chainer connects to a predefined server determined at instance creation time, independently of the original destination of the connection request. It can be used to address a nonrouteable host from the firewall.

4.2.3.1. Attributes

Table 4-5. Attributes for class DirectedChainer

local	The local data of the connection
remote	The data of the server
forge_addr	forge client address when connecting to the server

4.2.3.2. Examples

Example 4-1. Sample for DirectedChainer usage

```
Service("inter_HTTP_dmz",
    DirectedChainer(SockAddr(192.168.0.2, 80)),
    HttpProxy)
```

4.2.3.3. Constructor __init__

Constructor to initialize a DirectedChainer instance.

Synopsis

```
__init__ (
        self,
        remote,
        local=None,
        forge_addr=0,
```

```
        )
```

Description

Sets instance attributes based on parameters.

Arguments

Table 4-6. Arguments for DirectedChainer.__init__()

self	this instance
local	Local address of the connection
remote	Address of the server
forge_addr	forge client address when connecting to the server

4.2.3.4. Method connectServer

Function to connect to the predefined remote server.

Synopsis

```
connectServer (
        self,
        session,
        host,
        port,
        )
```

Description

This function establishes connection to the remote server forging the client address if necessary.

Arguments

Table 4-7. Arguments for DirectedChainer.connectServer()

self	this instance
session	The data of the current session
host	not used
port	not used

Returns

The fd given by establishConnection

4.2.4. Class FailoverChainer

Failover chainer tries to connect several hosts in round robin fashion until one of them succeeds, forming a simple failover and load balance solution.

4.2.4.1. Attributes

Table 4-8. Attributes for class FailoverChainer

hosts	The list of addresses to try
local	the address of the local end of the connection
forge_addr	forge client address
current_host	The index of the actual host in the list

4.2.4.2. Constructor __init__

Constructor to initializes an instance of FailoverChainer

Synopsis

```
__init__ (
        self,
        hosts,
        local=None,
        forge_addr=0,
        )
```

Description

Sets attributes based on parameters.

Arguments

Table 4-9. Arguments for FailoverChainer.__init__()

self	this instance
hosts	The list of addresses to try
local	address to bind to (Optional)
forge_addr	forge client address (Optional)

4.2.4.3. Method connectServer

Called by the proxy to connect to the remote server.

Synopsis

```
connectServer (
        self,
```

```
        session,
        host,
        port,
        )
```

Description

Tries to connect to each host round-robin until one of them succeeds.

Arguments

Table 4-10. Arguments for FailoverChainer.connectServer()

self	this instance
session	The data of the current session
host	not used
port	not used

Returns

The fd given by establishConnection

4.2.4.4. Method getHostAddress

Function to returns the actual host to attempt connection to.

Synopsis

```
getHostAddress ( self )
```

Description

Returns the actual `current_host` element of the host list `hosts`.

Arguments

Table 4-11. Arguments for FailoverChainer.getHostAddress()

self	this instance

Returns

address to connect to

4.2.4.5. Method nextHost

Function to skip to the next host to try.

Synopsis

```
nextHost ( self )
```

Description

This function is called to skip to the next host to try in the hosts array. Increments `current_host`, zeroes it if overflowed.

Arguments

Table 4-12. Arguments for FailoverChainer.nextHost()

self	this instance

4.2.5. Class InbandChainer

This chainer can be used for protocols where the target address is determined during the communication with the client. An example would be non-transparent HTTP where the target address is sent in the request.

4.2.5.1. Attributes

Table 4-13. Attributes for class InbandChainer

local	the address of the local end of the connection
forge_addr	use the client address for outgoing local address

4.2.5.2. Constructor __init__

Constructor to initialize an InbandChainer instance.

Synopsis

```
__init__ (
        self,
        local=None,
        forge_addr=0,
        )
```

Description

Creates an instance, sets its attributes based on constructor arguments.

Arguments

Table 4-14. Arguments for InbandChainer.__init__()

self	this instance
local	local address (optional)
forge_addr	forge client address (optional)

4.2.5.3. Method connectServer

Function called by the proxy to connect to the remote end.

Synopsis

```
connectServer (
        self,
        session,
        host,
        port,
        )
```

Description

This function looks up the host given as parameter in the DNS and starts connecting there.

Arguments

Table 4-15. Arguments for InbandChainer.connectServer()

self	this instance
session	Session object

host	Remote hostname as determined by the protocol
port	Remote port as determinted by the protocol

Returns

The fd given by establishConnection

4.2.6. Class TransparentChainer

This chainer connects to the destination originally addressed by the client and is used for connections intercepted by the firewall.

You have to add a REDIRECT rule to your packet filter configuration to force connections going through your firewall to be serviced locally.

Make sure that the Listener port is protected by a DENY rule so that clients cannot connect to it directly, otherwise TransparentProxy may cause an infinite loop.

4.2.6.1. Attributes

Table 4-16. Attributes for class TransparentChainer

local	bind address
forge_addr	use client address as outgoing local address
forced_port	if not 0 force it as destination port

4.2.6.2. Examples

Example 4-2. Sample for TransparentChainer usage

```
Service("http", TransparentChainer(), HttpProxy)
```

4.2.6.3. Constructor __init__

Constructor to initialize a TransparentChainer instance.

Synopsis

```
__init__ (
        self,
        local=None,
        forge_addr=0,
        forced_port=0,
        )
```

Description

Sets various attributes as passed in as parameters.

Arguments

Table 4-17. Arguments for TransparentChainer.__init__()

self	this instance
local	local address, may be none (optional)
forge_addr	if the client address shall be forged (optional)
forced_port	use this port on the remote server (optional)

4.2.6.4. Method connectServer

Function to connect to the remote server.

Synopsis

```
connectServer (
        self,
        session,
        host,
        port,
        )
```

Description

Finds out the original destination of the given connection and connects there.

Arguments

Table 4-18. Arguments for TransparentChainer.connectServer()

self	this instance
session	The data of the current session
host	not used
port	not used

Returns

The fd given by establishConnection

4.3. Module Connector

This module implements a Python wrapper around the ZorpConnect class implemented in C.

4.3.1. Imported modules

- `import` Zorp

  Module defining global constants, and interface entry points to the Zorp core.

4.3.2. Class Connect

This class is a simple wrapper around the ZorpConnect class implemented by the Zorp core, and as such it'll be a placeholder for extensions implemented in Python.

4.3.2.1. Attributes

Table 4-19. Attributes for class Connect

local	Address of local end of the connection
remote	Address where we connect
connect	a Zorp.Connect object

4.3.2.2. Constructor __init__

Initializes a Connect instance.

Synopsis

```
__init__ (
```

```
        self,
        local,
        remote,
        callback=None,
        )
```

Description

Sets the attributes based on constructor parameters, and start connecting in a separate thread if callback is not None.

Arguments

Table 4-20. Arguments for Connect.__init__()

self	this instance
local	Address of local end of the connection
remote	Address where we connect
callback	Callback to be called when the connection is established. If this is None callback is not used, and blockingConnect should be called to really establish a connection.

4.3.2.3. Method blockingConnect

Establish a connection in blocking mode.

Synopsis

```
blockingConnect ( self )
```

Description

Establishes the connection and returns its file descriptor.

Arguments

Table 4-21. Arguments for Connect.blockingConnect()

self	this instance

Returns

the fd

4.3.2.4. Method destroy

Destroy this connector instance.

Synopsis

```
destroy ( self )
```

Description

Stop connecting and destroy the underlying ZorpConnect instance.

Arguments

Table 4-22. Arguments for Connect.destroy()

self	instance

4.4. Module Domain

This module implements the class AbstractDomain and some derived classes, which encapsulate a set of physical addresses.

The class InetDomain implements IPv4 addresses, and IP segments represented in the form A.B.C.D/M, where A.B.C.D is the network address, and M specifies the number of ones in the netmask.

4.4.1. Imported modules

- `from` SockAddr `import` SockAddrInet, SockAddr, SockAddr

  Module implementing SockAddr handling functions.

- `from` socket `import` ntohl, htonl
- `from` string `import` split, atoi

4.4.2. Class AbstractDomain

An address domain encapsulates an address type (AF_INET, AF_INET6 etc) and provides functions to parse and compare these addresses. This functionality is primarily used by Zone classes (see the module `Zone`), and the PacketFilter classes (see the module `IPChains`)

4.4.2.1. Attributes

Table 4-23. Attributes for class AbstractDomain

family	address family this domain uses

4.4.2.2. Constructor __init__

Constructor initializing an AbstractDomain instance.

Synopsis

```
__init__ ( self )
```

Description

This constructor is basically empty and does nothing.

Arguments

Table 4-24. Arguments for AbstractDomain.__init__()

self	this instance

4.4.2.3. Method __cmp__

Function to compare two domains.

Synopsis

```
__cmp__ ( self,  other )
```

Description

This function is a placeholder and is to be overridden by derived classes. It should return -1, 0 and 1 as described in the Python documentation.

Arguments

Table 4-25. Arguments for AbstractDomain.__cmp__()

self	this instance
other	instance to compare to

Exceptions

- NotImplementedError

4.4.3. Class InetDomain

A class representing internet addresses. The inet objects are comparable, comparison means "contains", "equal to" and "contained by". The comparison can raise ValueError for incomparable ip addresses.

4.4.3.1. Attributes

Table 4-26. Attributes for class InetDomain

mask_bits	number of bits in the netmask
mask	netmask in network byte order
ip	network addresss in network byte order

4.4.3.2. Constructor __init__

Initializes an InetDomain instance

Synopsis

```
__init__ ( self,  addr )
```

Description

Parses the argument addr and fills in attributes accordingly.

Arguments

Table 4-27. Arguments for InetDomain.__init__()

addr	the string representation of an address, or address range

4.4.3.3. Method __cmp__

Compare this instance to another.

Synopsis

```
__cmp__ ( self,  other )
```

Description

Compare this instance to another InetDomain instance or to a SockAddrInet instance using set inclusion on addresses. An address is less than another, if it’s fully contained by other.

Arguments

Table 4-28. Arguments for InetDomain.__cmp__()

self	this instance
other	the other InetDomain object to compare to

Exceptions

- ValueError

Returns

-1, 0, or 1

4.4.3.4. Method __str__

Returns the string representation of this instance.

Synopsis

```
__str__ ( self )
```

Description

Returns the string representation of this instance in the form address/mask.

Arguments

Table 4-29. Arguments for InetDomain.__str__()

self	this instance

Returns

string

4.4.3.5. Method broadcast

Calculate the broadcast address of this domain

Synopsis

```
broadcast ( self )
```

Description

Return the broadcast address of this domain calculated based on attributes.

Arguments

Table 4-30. Arguments for InetDomain.broadcast()

self	this instance

Returns

the broadcast address in network byte order

4.4.3.6. Method netaddr

Calculate the network address of this domain.

Synopsis

```
netaddr ( self )
```

Description

Return the network address of this domain.

Arguments

Table 4-31. Arguments for InetDomain.netaddr()

self	this instance

Returns

ip address in network byte order

4.4.3.7. Method netmask

Calculate netmask of this domain.

Synopsis

```
netmask ( self )
```

Description

Calculates and returns the netmask of this domain as an integer in network byte order.

Arguments

Table 4-32. Arguments for InetDomain.netmask()

self	this instance

Returns

the network mask as ip in network byte order

4.5. Module Finger

4.5.1. Imported modules

- `from` Proxy `import` Proxy

  Module defining classes encapsulating native proxies.

- `from` Zorp `import` *

  Module defining global constants, and interface entry points to the Zorp core.

4.5.2. Class FingerProxy

This proxy implements the finger protocol as specified in rfc1288.

4.5.2.1. Usage

You can limit username length and response length by setting various attributes. Finger proxy also has the capability of limiting the number of hosts in a request, like:

finger bazsi@balabit@tudor

which normally results in fingering bazsi@tudor performed by the host balabit. By default this proxy strips off everything after and including the first `@` character, you can change this behaviour by setting max_hop_count to a nonzero value.

4.5.2.2. Attributes

Table 4-33. Attributes for class FingerProxy

max_hop_count	the maximum number of `@ characters in the request`
max_username_length	the maximum length of the username part in the request
max_line_length	the maximum number of characters in a single line
strict_username_check	if enabled usernames are checked strictly [a-zA-Z0-9_]

4.5.2.3. Constructor __init__

Initialize a FingerProxy instance.

Synopsis

```
__init__ ( self,  session )
```

Description

Create and set up a FingerProxy instance.

Arguments

Table 4-34. Arguments for FingerProxy.__init__()

self	this instance
session	session this instance belongs to

4.6. Module Ftp

This module defines the Ftp proxy interface as implemented by the Ftp proxy module.

4.6.1. Imported modules

- `from` Plug `import` PlugProxy
- `from` Proxy `import` Proxy, Proxy

  Module defining classes encapsulating native proxies.
- `from` Session `import` StackedSession

  Module defining session related classes and functions.
- `from` SockAddr `import` SockAddrInet, SockAddrInetRange

  Module implementing SockAddr handling functions.
- `from` Stream `import` Stream

  Module exporting an interface to the Zorp.Stream component.

- `from` Zorp `import *`

  Module defining global constants, and interface entry points to the Zorp core.

4.6.2. Class FtpDataProxy

This class ensures one way data connections by setting the appropriate copy_to_* variable in Plug.

4.6.2.1. Method config

Synopsis

```
config ( self )
```

Description

4.6.2.2. Method shutDown

Synopsis

```
shutDown ( self )
```

Description

4.6.3. Class FtpProxy

By default deny everything, but data connection establishment. You should dervice your customized Ftp proxy classes in order to do something useful.

4.6.3.1. Usage

This section contains basic usage patterns, you as an administrator may need to do.

Default processing of commands

All requests are denied in the low level proxy implementation by default. To enable different commands and answers you'll need to extend the low-level proxy in Python. To see how this is done, see the next two sections.

Setting policy for commands

Changing the default behaviour of commands can be done using the hash named "command". This hash is indexed by the command name (e.g: USER or PWD), and each item in this hash is an action tuple, defining proxy behaviour for the given command.

The first item in this tuple is an integer value, determining the action to take, and also the interpretation of the remaining items in the tuple.

Possible values for the first item:

Example 4-3. Sample for anonymous only sessions in FTP

```
class AnonFtp(FtpProxy):

  def config(self):
    self.fw_server_data.ip_s="192.168.0.1"
    self.fw_client_data.ip_s="10.0.0.1"
    self.commands["USER"] = (Ftp.FTP_CMD_POLICY, self.pUser)
    self.commands["*"] = (Ftp.FTP_CMD_ACCEPT)

  def pUser(self,command):
```

```
if self.request_parameter == "ftp" or self.request_parameter == "anonymous":
    return Z_ACCEPT
return Z_DENY
```

Changing answer code or string

Like with commands, we have a hash for answers as well. This hash indexed by command concatenated with answer, and contains an action tuple in each item. A special wildcard character `*` matches every command. Another feature is that you don't need to write the full answer code, it's possible to write only the first, or the first two digits.

For example, all three index below is valid:

Table 4-35. Untitled

PWD200	Match exactly the answer 200 coming in reply to a PWD command.
PWD2	Match every answer starting with `2 in reply to a PWD command.`
*20	Match every answer between 200 and 209 in reply to any command.

All answers are "denied" default. It's changed to "500 Error parsing answer", because dropping an answer is not permitted by RFC959. You may enable all answers with "*".

The precedence in processing hashtable entries is the following:

1. Exact match. (PWD200)

2. Exact command match, partial answer matches (PWD20, PWD2, PWD)

3. Wildcard command, with answer codes repeated as above. (*200, *20, *2, *)

Possible values for the first item in the action tuple:

Table 4-36. Untitled

FTP_ANS_ACCEPT	allow answer without modification.
FTP_ANS_CHANGE	change answer code and message. (default) Second parameter contains a string, which will be sent back to client.
FTP_ANS_POLICY	call a Python function to decide what to do. this value uses an additional parameter, which must be a callable Python function. The function takes three parameters: self, command, answer
FTP_ANS_ABORT	deny request, and drop conection.

4.6.3.2. Attributes

Table 4-37. Attributes for class FtpProxy

data_connect	Object for the data connection
data_proxy	proxy to use for data connections
data_port_min	ports should be allocated above this variable
data_port_max	ports should be allocated below this variable
data_mode	FTP_DATA_KEEP, FTP_DATA_PASSIVE or FTP_DATA_ACTIVE can be used to force a connection type in server side. (default: FTP_DATA_KEEP)
transparent_mode	(logical) TRUE for transparent proxy, FALSE otherwise (default: TRUE)
fw_server_data	(ZorpSocket) Firewall address in server side. By default it contains the IP address of the interface `towards the server.`

fw_client_data	(ZorpSocket) Firewall address in client side. By default it contains the IP address of the interface `towards the client.`
permit_unknown_command	(boolean) Enable commands not understood (and not handled) by proxy. (Default: FALSE)
permit_empty_command	(boolean) Enable lines without commands. (Default: FALSE)
max_line_length	(integer) Maximum line lenght, what a proxy may transfer. (Default: 255)
max_username_length	(integer) Maximum length of usernames. (Default: 32)
max_password_length	(integer) Maximum length of passwords. (Default: 64)
commands	(hash) normative policy hash, directing the proxy to do something with requests, without the need to call Python. indexed by the command (e.g. "USER", "PWD" etc) (default: empty)
answers	(hash) normative policy hash directing the proxy to do something with answers. Indexed by the command, and the answer (e.g. "USER331", "PWD200" etc) (default: empty)
request_command	(string) When a command goes up to policy level, you may change it with this. FIXME: Not yet used.
request_parameter	(string) When a command goes up to policy level, this variable contains command parameters.
answer_code	(string) When an answer goes up to policy level, this variable contains answer code.

answer_parameter	(string) When an answer go up to policy level, this variable contain answer parameters.
restrict_client_connect	(boolean) Restricting data connection in client side. Accept data connection only if it’s coming from IP number equal with command channel endpoint. (Default: TRUE)
restrict_server_connect	(boolean) Restricting data connection in server side. Accept data connection only if it’s coming from IP number equal with command channel endpoint. (Default: FALSE)

4.6.3.3. Private attributes

Table 4-38. Untitled

state	connection state 0 is initial, 1 one of the connections were established, 2 both connections established ready to start data proxy.
modes	array indexed by side, `L for Listen or C for connect`
listens	listen objects created by us, array indexed by side
connects	connect objects created by us, array indexed by side
sides	determines the order in which sides must be processed (first listen/connect on sides[0] then on sides[1])

fds	array indexed by side, containing the side specific fd
localsesa	local socket adress in server side. Used when data connections port is in a range.
localclsa	local socket adress in client side. Used when data connecion port is in range.

4.6.3.4. Method __destroy__

Deinitializes an FtpProxy class

Synopsis

```
__destroy__ ( self )
```

Description

Calls resetState() to destroy bound listeners and connectors. It destroys running data connection.

Arguments

Table 4-39. Arguments for FtpProxy.__destroy__()

self	this instance

4.6.3.5. Constructor __init__

Initialize an FtpProxy instance

Synopsis

```
__init__ ( self,  session )
```

Description

This constructor initializes an FtpProxy instance by calling the inherited __init__ constructor with appropriate parameters, and setting up local attributes based on arguments.

Arguments

Table 4-40. Arguments for FtpProxy.__init__()

self	this instance
session	The session object

4.6.3.6. Method acceptedCallback

Callback called when a connection was accepted on either side.

Synopsis

```
acceptedCallback (
        self,
        client,
        fd,
        )
```

Description

This callback is called if any of the sides was in `L` mode, and a connection was accepted on the listening socket.

Arguments

Table 4-41. Arguments for FtpProxy.acceptedCallback()

self	this instance
client	the client address which has connected
fd	the file descriptor for the connection

Exceptions

- InternalError

Notes

stores the parameters, increases the state and continues with the state machine

4.6.3.7. Method connectedCallback

Callback called when a connection was established.

Synopsis

```
connectedCallback ( self,  fd )
```

Description

This function is similar to [acceptedCallback], but used if we were in `C` (connect) mode.

.. [acceptedCallback] FtpDataConnect.acceptedCallback

Arguments

Table 4-42. Arguments for FtpProxy.connectedCallback()

self	this instance
fd	the file descriptor for the connection

Exceptions

- InternalError

Notes

stores the parameter, increases the state and continues with the state machine

4.6.3.8. Method loadAnswers

This function can be called by derived classes to initialize internal hashtables.

Synopsis

```
loadAnswers ( self )
```

Description

This function fills in the self.answers hash so that commonly used request/answer combinations are enabled.

Arguments

Table 4-43. Arguments for FtpProxy.loadAnswers()

self	this instance

4.6.3.9. Method prepareData

Prepares the data channel on `side` side using `mode` mode.

Synopsis

```
prepareData (
        self,
        side,
        mode,
        )
```

Description

This function prepares the given side of the data connection. This means either to listen for connections (like passive mode on the client side) or to start establishing a connection (passive mode on the server side).

Arguments

Table 4-44. Arguments for FtpProxy.prepareData()

self	this instance
side	FTP_SIDE_CLIENT or FTP_SIDE_SERVER representing either the client or the server
mode	`L or C for Listen or Connect`

4.6.3.10. Method resetData

Reset data connection state.

Synopsis

```
resetData ( self )
```

Description

Sets the initial state of everything, and destroy pending listeners.

Arguments

Table 4-45. Arguments for FtpProxy.resetData()

self	this instance

4.6.3.11. Method startData

Start the data connection procedure as prepared by earlier prepare calls.

Synopsis

```
startData (
        self,
        side1,
        side2,
        way,
        )
```

Description

Previous calls to prepare determines how the data connection should be built. This function starts the state machine whose final state is to start a proxy on the established data connection.

Arguments

Table 4-46. Arguments for FtpProxy.startData()

side1	connect/listen on this side first
side2	connect/listen on this side after the first one succeeded
way	direction

Exceptions

- InternalError

4.6.3.12. Method stepData

Do the processing as required by the current state.

Synopsis

```
stepData ( self )
```

Description

This function is called each time after processing the current state finishes. This means that if the data connection on the server side is established we are ready to step one further and start accepting connections on the client side.

Arguments

Table 4-47. Arguments for FtpProxy.stepData()

self	this instance

Exceptions

- InternalError

Notes

If state == 2, stacks in the proxy and starts it with a session stacked out from self.session

In state 0 and 1 it either listens or connects, based on the type field of parms[state]

4.6.3.13. Method stopDataConnection

Callback called by the data proxy to indicate the end of the data connection.

Synopsis

```
stopDataConnection ( self )
```

Description

This callback is registered with the data proxy ([FtpDataProxy], see below), and is called when the data connection was terminated. It’s task is to signal this condition to the Ftp proxy core.

.. [FtpDataProxy] FtpDataProxy

Arguments

Table 4-48. Arguments for FtpProxy.stopDataConnection()

self	this instance

4.6.4. Class FtpProxyAllow

We defined two example classes derived from the default FtpProxy class, enabling some additional commands/answers from the FTP protocol. FtpProxyAllow as the name suggests allows any FTP command to pass through the proxy, and allows any answer in reply to them.

4.6.4.1. Method config

Default config for FtpProxyAllow.

Synopsis

```
config ( self )
```

Description

Enables all commands by setting permit_unknown_commands to TRUE and adding two wildcard entries to the commands and answers hash.

Arguments

Table 4-49. Arguments for FtpProxyAllow.config()

self	this instance

4.6.5. Class FtpProxyMinimal

This proxy enables the minimal required protocol elements required by daily usage.

4.6.5.1. Method config

Configuration for FtpProxyMinimal

Synopsis

```
config ( self )
```

Description

It enables a minimal set of commands for a working FTP proxy, and sets permit_unknown_commands to FALSE.

Arguments

Table 4-50. Arguments for FtpProxyMinimal.config()

self	this instance

4.6.5.2. Method test

Test function to log answers not enabled in config

Synopsis

```
test (
        self,
        command,
        answer,
        )
```

Description

This function is used to gather still useful answers not enabled in FtpProxyMinimal by default. It accepts any answers but sends a log message about it.

Arguments

Table 4-51. Arguments for FtpProxyMinimal.test()

self	this instance
command	command
answer	answer

4.7. Module Http

This module defines the interface to the Http proxy as implemented by the Http module.

4.7.1. Imported modules

- `from` Plug `import` PlugProxy
- `from` Proxy `import` Proxy

  Module defining classes encapsulating native proxies.
- `from` Session `import` StackedSession

  Module defining session related classes and functions.
- `from` Zorp `import` *

Module defining global constants, and interface entry points to the Zorp core.

4.7.2. Class HttpProxy

HttpProxy is a wrapper class for the built in Http proxy implemented in Zorp. It features both transparent and non-transparent modes of operation, advanced filtering and more.

4.7.2.1. Usage

This section contains basic usage patterns, you as an administrator may need to do.

Setting policy for requests

Changing the default behaviour of requests can be done using the hash named "request". This hash is indexed by the method name (e.g: GET or POST), and each item in this hash is a tuple, whose first item is an integer value, determining the action to be done with the request, and also the interpretation of the remaining items in the tuple.

All requests are denied by default in the low level proxy implementation. The most common methods (GET, POST and HEAD) are enabled from Python code.

Possible values for the first item:

Example 4-4. Sample for URL filtering in HTTP proxy

```
class DmzHTTP(HttpProxy):

  def config(self):
    HttpProxy.config(self)
    self.request["GET"] = (HTTP_POLICY, self.filterURL)

  def self.filterURL(self, method, url, version):
    if (url == "http://www.balabit.hu")
      return Z_ACCEPT
    return Z_DENY
```

Changing headers in requests or responses

Both request headers and response headers can be modified during transit. New header lines can be inserted, entries can be modified or deleted. To change headers in the request use the request_headers hash, for response headers you need the response_headers hash.

Similarly to the request hash, these hashes contain a variable-length tuple, where the first item determines the interpretation of the remaining items. The hash index is the name of the header to be modified.

Headers are not touched by default, except the "Host:", "Connection:" and "Proxy-Connection" headers. However the way these are modified can be changed here.

Possible values for the first item:

Example 4-5. Sample for header filtering in HTTP

```
class MyHttp(HttpProxy):

  def config(self):
    HttpProxy.config(self)
    self.request_headers["User-
Agent"] = (HTTP_CHANGE_VALUE, "Lynx 2.4.1")
  self.request_headers["Cookie"] = (HTTP_POLICY, self.processCookies)
    self.response_headers["Set-Cookie"] = (HTTP_DROP,)

  def processCookies(self, name, value):
    log("http.message", 7, "cookie: value=%s" % value,)
    # you could change the current header in self.current_header_name
    # or self.current_header_value, the current request url
    # in self.request_url
    return Z_DROP
```

Redirecting urls

Example 4-6. Sample for URL redirection in HTTP

```
class MyHttp(HttpProxy):

  def config(self):
    HttpProxy.config(self)
    self.request["GET"] = (HTTP_POLICY, self.filterURL)

  def filterURL(self, method, url, version):
    self.request_url = "http://www.balabit.hu/"
```

Using parent proxies

Example 4-7. Sample for using parent proxies in HTTP

```
class MyHttp(HttpProxy):

  def config(self):
    HttpProxy.config(self)
    self.parent_proxy = "proxy.balabit.hu"
    self.parent_proxy_port = 3128
```

4.7.2.2. Attributes

Table 4-52. Attributes for class HttpProxy

transparent_mode	(logical) TRUE for transparent proxy, FALSE otherwise (default: TRUE)
transparent_server_requests	(logical) allow server requests in transparent mode (default: TRUE)

transparent_proxy_requests	(logical) allow proxy requests in transparent mode (default: FALSE)
connection_mode	HTTP_CONNECTION_CLOSE or HTTP_CONNECTION_KEEPALIVE can be used to forcibly close a keepalive connection.
parent_proxy	(string) address or hostname of the parent proxy to connect to. You have to use DirectedChainer or InbandChainer for this option to take effect.
parent_proxy_port	(integer) the port of the parent proxy to connect to. (default: 3128)
default_port	(integer) if the port number is not specified in the URL use this port. (default: 80)
rewrite_host_header	(logical) rewrite Host header when redirecting an url (default: TRUE)
max_line_length	(integer) maximum length of non-transfer mode lines (default: 4096)
max_header_lines	(integer) maximum number of header lines in requests or responses (default: 50)
max_keepalive_requests	(integer) maximum number of requests in a single session
timeout	(integer) I/O timeout in milliseconds (default: 30000)
request	(hash) normative policy hash, directing the proxy to do something with requests, without the need to call Python. indexed by the method (e.g. "GET", "PUT" etc) (default: empty) See below for more information.

request_headers	(hash) normative policy hash, directing the proxy to do something with request headers (drop, insert, rewrite etc) It is indexed by the header name (e.g. "Set-cookie") (default: empty) See below for more information.
response	(hash) normative policy hash directing the proxy to do something with responses. FIXME: not yet used
response_headers	(hash) similar to request_headers for response headers.
request_url	(string) request url string, can be changed to redirect the current request.
current_header_name	(string) defined during header processing functions, and can be changed to actually change a header in the request or response.
current_header_value	(string) similar to current_header_name but contains the header value
error_response	(integer) if the request is denied use this HTTP response code (default: 500)
error_info	(string) a string included in error message.

4.7.2.3. Constructor __init__

Initializes a HttpProxy instance.

Synopsis

```
__init__ ( self,  session )
```

Description

Creates and initializes a HttpProxy instance.

Arguments

Table 4-53. Arguments for HttpProxy.__init__()

self	this instance
session	the session this instance participates in

4.7.2.4. Method config

Default config event handler.

Synopsis

```
config ( self )
```

Description

Enables the most common HTTP methods so we have a useful default configuration.

Arguments

Table 4-54. Arguments for HttpProxy.config()

self	this instance

4.7.2.5. Method connectMethod

Synopsis

```
connectMethod ( self )
```

Description

4.7.3. Class HttpProxyNonTransparent

This class encapsulates a non-transparent HTTP proxy using the features provided by HttpProxy.

4.7.3.1. Method config

Config event handler

Synopsis

```
config ( self )
```

Description

Sets self.transparent_mode to FALSE to indicate non-transparent mode.

Arguments

Table 4-55. Arguments for HttpProxyNonTransparent.config()

self	this instance

4.8. Module IPChains

IPChains is used by Linux kernels version 2.2.x, and this code can be used to append, delete or change packet filtering rules from within Zorp.

 Note that this code doesn't call the user space program `ipchains`, it calls system calls directly.

4.8.1. Imported modules

- `from` Domain `import` InetDomain

  Module implementing address domains.

- `import` Zorp

  Module defining global constants, and interface entry points to the Zorp core.

4.8.2. Functions

4.8.2.1. Function calcIpMask

Function calculating the IP, netmask values of dom.

Synopsis

```
calcIpMask ( dom )
```

Description

This function returns a tuple containing an IP and a netmask value in network byte order extracted from the parameter `dom`.

Arguments

Table 4-56. Arguments for .calcIpMask()

dom	an InetDomain or SockAddrInet instance, the exact type is determined at runtime.

Exceptions

- ValueError

Returns

a tuple of (ip, netmask)

4.8.3. Class IPChains

This class is an interface to the kernel packet filter named `ipchains`.

4.8.3.1. Attributes

Table 4-57. Attributes for class IPChains

rules	an array of rules added by this object, these rules are implicitly cleared when this object is deleted
ipchains	an instance of the Zorp.IPChains class implemented by the Zorp core.

4.8.3.2. Constructor __init__

Constructor to initialize an IPChains instance

Synopsis

```
__init__ ( self )
```

Description

This constructor initializes an IPChains instance by setting default values for attributes, and creating a Zorp.IPChains instance.

Arguments

Table 4-58. Arguments for IPChains.__init__()

self	this instance

4.8.3.3. Destructor __del__

Destructor to delete rules added by this object.

Synopsis

```
__del__ ( self )
```

Description

This destructor is called when the IPChains instance is freed. It automatically removes all rules we didn’t explicitly remove.

Arguments

Table 4-59. Arguments for IPChains.__del__()

self	this instance

4.8.3.4. Method addACCEPT

Method specialized to add an `ACCEPT` rule.

Synopsis

```
addACCEPT (
        self,
        chain,
        sdom,
        sports,
        ddom,
        dports,
        iface="",
        mark=0,
        proto=0,
        flags=0,
        inv_flags=0,
        redirport=0,
        tosand=0xff,
        tosxor=0x00,
        )
```

Description

This is a wrapper around addRule() to add an `ACCEPT` rule to one of the chains.

Arguments

Table 4-60. Arguments for IPChains.addACCEPT()

self	this instance
others	rule details, see the documentation for `addREDIRECT or the ipchains(8) manual page.`

4.8.3.5. Method addDENY

Method specialized to add a `DENY` rule.

Synopsis

```
addDENY (
        self,
        chain,
        sdom,
        sports,
        ddom,
        dports,
        iface="",
        mark=0,
        proto=0,
        flags=0,
        inv_flags=0,
        redirport=0,
        tosand=0xff,
        tosxor=0x00,
        )
```

Description

This is a wrapper around addRule() to add an `DENY` rule to one of the chains.

Arguments

Table 4-61. Arguments for IPChains.addDENY()

self	this instance
others	rule details, see the documentation for `addREDIRECT or the ipchains(8) manual page.`

4.8.3.6. Method addREDIRECT

Method specialized to add a `REDIRECT` rule

Synopsis

```
addREDIRECT (
        self,
        chain,
        sdom,
        sports,
        ddom,
        dports,
        redirport,
        iface="",
        mark=0,
        proto=0,
        flags=0,
        inv_flags=0,
        tosand=0xff,
        tosxor=0x00,
        )
```

Description

This is a wrapper around the addRule() function making it easier to add `REDIRECT` rules. It returns the full rule added, so that later invocations of delRule() can delete it. For more information on different arguments check out the ipchains(8) manual page.

Arguments

Table 4-62. Arguments for IPChains.addREDIRECT()

self	this instance
chain	chain to append this rule to
sdom	source address range (specified as an InetDomain or SockAddrInet instance)
sports	a tuple of two ports specifying matching port range
ddom	destination address range
dports	destination port range
redirport	redirect connections to this local port
iface	interface specified as a string (default none)
mark	mark packets with this fwmark, (default 0)
proto	match packets with this protocoll (default 0)
flags	a combination of IP_FW_F_* flags above (default 0)
inv_flags	a combination of IP_FW_INV_F_* flags above (default 0)
tosand	mask to type of service (default 0xff)
tosxor	xor to type of service (default 0x00)

4.8.3.7. Method addREJECT

Method specialized to add a `REJECT` rule.

Synopsis

```
addREJECT (
        self,
        chain,
        sdom,
        sports,
        ddom,
        dports,
        iface="",
        mark=0,
        proto=0,
        flags=0,
        inv_flags=0,
        redirport=0,
        tosand=0xff,
        tosxor=0x00,
        )
```

Description

This is a wrapper around addRule() to add an `REJECT` rule to one of the chains.

Arguments

Table 4-63. Arguments for IPChains.addREJECT()

self	this instance
others	rule details, see the documentation for `addREDIRECT or the ipchains(8) manual page.`

4.8.3.8. Method addRETURN

Method specialized to add a `RETURN` rule.

Synopsis

```
addRETURN (
        self,
        chain,
        sdom,
        sports,
        ddom,
        dports,
        iface="",
        mark=0,
        proto=0,
        flags=0,
        inv_flags=0,
        redirport=0,
        tosand=0xff,
        tosxor=0x00,
        )
```

Description

This is a wrapper around addRule() to add an `RETURN` rule to one of the chains.

Arguments

Table 4-64. Arguments for IPChains.addRETURN()

self	this instance
others	rule details, see the documentation for `addREDIRECT or the ipchains(8) manual page.`

4.8.3.9. Method addRule

Method to actually add a rule to a chain.

Synopsis

```
addRule ( self,  rule )
```

Description

This is a general function called by more specific methods.

Arguments

Table 4-65. Arguments for IPChains.addRule()

self	this instance
rule	a tuple describing the rule to add in the syntax (target, (src, smask), (dst, dmask), mark, proto, flags, inv_flags, (sportmin, sportmax), (dportmin, dportmax), redirport, iface, tosand, tosxor)

4.8.3.10. Method delRule

Method to delete a rule.

Synopsis

```
delRule ( self,  rule )
```

Description

This method deletes a rule specified in its rule argument.

Arguments

Table 4-66. Arguments for IPChains.delRule()

self	this instance
rule	rule to delete

4.9. Module Listener

This module defines the Listener class, the main entry point for client requests. Listener binds to a given address, waits for connections, and for each connection spawns a new session to handle that connection.

4.9.1. Imported modules

- `from` Service `import` services

  Module defining service related classes.

- `from` Session `import` MasterSession

  Module defining session related classes and functions.

- `from` Zorp `import` *

  Module defining global constants, and interface entry points to the Zorp core.

- `from` traceback `import` print_exc

4.9.2. Class Listener

This is the starting point of Zorp services. It listens on the given port, and when a connection is accepted it starts a session and the given service.

4.9.2.1. Attributes

Table 4-67. Attributes for class Listener

listen	A Zorp.Listen instance
service	the service to be started
bindto	bind address
local	local address where the listener is bound

4.9.2.2. Constructor __init__

Constructor to initialize a Listen instance

Synopsis

```
__init__ (
        self,
        bindto,
        service,
        transparent=FALSE,
        )
```

Description

Creates the instance, sets the initial attributes, and starts the listener

Arguments

Table 4-68. Arguments for Listener.__init__()

self	this instance
bindto	the address to bind to
service	the service name to start
transparent	TRUE if this is a listener of a transparent service, specifying this is not mandatory but performs additional checks

Exceptions

- ServiceException

4.9.2.3. Destructor __del__

Synopsis

```
__del__ ( self )
```

Description

4.9.2.4. Method accepted

Callback to inform the python layer about incoming connections.

Synopsis

```
accepted (
        self,
        client,
        fd,
        )
```

Description

This callback is called by the core when a connection is accepted. Its primary function is to check access control (whether the client is permitted to connect to this port), and to spawn a new session to handle the connection.

Exceptions raised due to policy violations are handled here.

Arguments

Table 4-69. Arguments for Listener.accepted()

self	this instance
client	the address of the client
fd	the fd of the connection to the client

Exceptions

- DACException

Returns

TRUE if the connection is accepted

4.9.2.5. Method destroy

Stops the listener on the given port

Synopsis

```
destroy ( self )
```

Description

Calls the destroy method of the low-level object

Arguments

Table 4-70. Arguments for Listener.destroy()

self	this instance

4.9.2.6. Method getService

Returns the service associated with the listener

Synopsis

```
getService ( self,  session )
```

Description

Returns the service to start.

Arguments

Table 4-71. Arguments for Listener.getService()

self	this instance
session	session reference

4.9.3. Class ZoneListener

This class is similar to a simple Listener, but instead of starting a fixed service, it chooses one based on the client zone.

It takes a mapping of services indexed by a zone name, with an exception of the * service, which matches anything.

4.9.3.1. Attributes

Table 4-72. Attributes for class ZoneListener

services	services mapping indexed by zone name

4.9.3.2. Constructor __init__

Initialize a ZoneListener instance.

Synopsis

```
__init__ (
        self,
        bindto,
        services,
        )
```

Description

Initialize a ZoneListener instance and set initial attribute values based on arguments.

Arguments

Table 4-73. Arguments for ZoneListener.__init__()

self	this instance
bindto	bind to this address
services	a mapping between zone names and services

4.9.3.3. Method getService

Virtual function which returns the service to be ran

Synopsis

```
getService ( self,  session )
```

Description

Called by our base class to find out the service to be used for the current session.

Arguments

Table 4-74. Arguments for ZoneListener.getService()

self	this instance
session	session we are starting

4.10. Module Plug

4.10.1. Imported modules

- `import` Connector

  Implements classes to establish a connection.

- `import` Proxy

  Module defining classes encapsulating native proxies.

- `import` Service

  Module defining service related classes.

- `import` Session

  Module defining session related classes and functions.

- `import` Stream

  Module exporting an interface to the Zorp.Stream component.

- `from` Zorp `import *`

  Module defining global constants, and interface entry points to the Zorp core.

4.10.2. Class PlugProxy

Implements a general plug proxy, with optionally disabling data transfer in either direction.

By default plug copies all data in both directions. To change this set the `copy_to_client` or `copy_to_server` attributes to FALSE.

 Copying out of band data is not supported.

4.10.2.1. Attributes

Table 4-75. Attributes for class PlugProxy

copy_to_server	Copy data in client->server direction
copy_to_client	Copy data in server->client direction
bandwidth_to_client	Readonly variable containing the utilized bandwidth in server->client direction.
bandwidth_to_server	Readonly variable containing the utilized bandwidth in client->server direction.
packet_stats_interval	The number of milliseconds between two successive packetStats() events. By default: 0. NOTE: this is currently implemented as the number of passing packages, not as milliseconds.
stack_proxy	the proxy class to stack into the connection

4.10.2.2. Notes

packetStats() event is called whenever the time interval elapses (or as currently implemented the given number of packets were transmitted). this event receives packet statistics as parameters.

It’s useful for terminating connections with excessive bandwidth requirements (for instance to limit the impact of the covert channel opened when using plug instead of a protocol specific proxy)

packetStats is defined as:

def packetStats(self, client_bytes, client_pkts, server_bytes, server_pkts)

4.10.2.3. Constructor __init__

Initialize a PlugProxy instance.

Synopsis

```
__init__ ( self,  session )
```

Description

Create and set up a PlugProxy instance.

Arguments

Table 4-76. Arguments for PlugProxy.__init__()

self	this instance
session	session this instance belongs to

4.10.2.4. Method requestStack

Query if something is to be stacked into this plug.

Synopsis

```
requestStack ( self )
```

Description

Callback called by the underlying C proxy to query if something is to be stacked.

Arguments

Table 4-77. Arguments for PlugProxy.requestStack()

self	this instance

Returns

the class of the proxy to stack in

4.10.2.5. Method stackProxy

Stack a proxy instance within this plug.

Synopsis

```
stackProxy (
        self,
        client_fd,
        server_fd,
        )
```

Description

Callback called by the underlying C proxy to actually stack in something.

Arguments

Table 4-78. Arguments for PlugProxy.stackProxy()

self	this instance
client_fd	client fd
server_fd	server fd

4.11. Module Pop3

This module defines a wrapper around the POP3 proxy implemented in Zorp.

4.11.1. Imported modules

- `from` Proxy `import` Proxy

  Module defining classes encapsulating native proxies.

- `from` Zorp `import` *

  Module defining global constants, and interface entry points to the Zorp core.

4.11.2. Class Pop3Proxy

This proxy implements the pop3 protocol as specified in rfc1939

4.11.2.1. Usage

Default processing of commands

By default accept all recommended commands wroten in rfc1939. And the following optional commands: USER, PASS, AUTH. It’s knowing about all commands in rfc1939 and the AUTH command. If you want to enable this commands too, you may enable this by hand, or use Pop3ProxyFull

Setting policy for commands

Changing the default behaviour of commands can be done using the hash named "command". This hash is indexed by the command name (e.g: USER or AUTH), and each item in this hash is an action tuple, defining proxy behaviour for the given command.

The first item in this tuple is an integer value, determining the action to take, and also the interpretation of the remaining items in the tuple.

Possible values for the first item

Example 4-8. Sample for converting simple USER/PASS authentication to APOP in POP3

```
class UToAPop3(Pop3Proxy):
  def config(self)
    Pop3Proxy.config(self)
self.commands["USER"] = (Pop3.POP3_CMD_POLICY,self.DropUSER)
    self.comamnds["PASS"] = (Pop3.POP3_CMD_POLICY,self.UToA)

  def DropUSER(self,command)
    self.response = "+OK"
    self.response_param = "User ok Send Password"
    return Z_DENY

  def UToA(self,command)
    # We got username in self->username,
    # password in self->command_param,
    # and the server timestamp in self->timestamp
    # therefore we can calculate the digest
    # NOTE: This is a sample only, calcdigest must be
    # implemented separately
    digest = calcdigest(self->timestamp+self->command_param)
    self->command = "APOP"
    self->command_param = name + " " + digest
    return Z_ACCEPT
```

4.11.2.2. Attributes

Table 4-79. Attributes for class Pop3Proxy

timeout	(int) timeout in milisec. If no packet in this intervall, connections will be dropped. (C:RW;P:R, Default: 60 sec)
username	(string) Used username in connection. (C:-;P:R, Default: -)
commands	(hash) normative policy hash, directing the proxy to do something with requests, without the need to call Python. indexed by the command (e.g. "USER", "UIDL" etc) (C:-;P:RW, default: empty)
command	(string) When a command goes up to policy level, you may change it with this.
command_param	(string) When a command goes up to policy level, you may change it with this.
response	(string) When a command or response goes up to policy level, you may change it with this.
response_param	(string) When a command or response goes up to policy level, you may change it with this.
timestamp	(string) If pop3 server implements APOP command, it’s send a timestamp with greeting, and it’s used with APOP Pop3 Proxy save timestamp in this string
answer_multiline	(boolean) It’s must set in policy, when a command answer is multiline.
permit_empty_command	(boolean) Enable lines without commands. (Default: FALSE)

4.11.2.3. Constructor __init__

Initialize a Pop3Proxy instance.

Synopsis

```
__init__ ( self,  session )
```

Description

Create and set up a Pop3Proxy instance.

Arguments

Table 4-80. Arguments for Pop3Proxy.__init__()

self	this instance
session	session this instance belongs to

4.11.2.4. Method config

Default Pop3 config

Synopsis

```
config ( self )
```

Description

Fill commands hash with most common values. (It’s work for all pop3 client know by me)

4.12. Module Proxy

This module defines the class `Proxy`, encapsulating the ZorpProxy component implemented by the Zorp core. Each protocol specific proxy is derived from this class.

4.12.1. Imported modules

- `from` Stream `import` Stream

  Module exporting an interface to the Zorp.Stream component.

- `from` Zorp `import` *

  Module defining global constants, and interface entry points to the Zorp core.

- `import` new
- `from` traceback `import` print_exc

4.12.2. Functions

4.12.2.1. Function proxyLog

Log a proxy message.

Synopsis

```
proxyLog (
        self,
        type,
        level,
        msg,
        )
```

Description

Log messages about proxies. Automatically puts session_id into each message.

Arguments

Table 4-81. Arguments for .proxyLog()

self	should be a Proxy derivative
type	log class
level	message verbosity level
msg	msg to log

4.12.3. Class DatagramProxy

This class is similar to [Proxy] with the exception that it’s used for datagram based protocols.

.. [Proxy] Proxy

4.12.3.1. Attributes

Table 4-82. Attributes for class DatagramProxy

self	this instance
session	the session we belong to

4.12.3.2. Constructor __init__

Initializes a DatagramProxy instance.

Synopsis

```
__init__ (
        self,
        name,
        session,
        )
```

Description

Sets attributes based on arguments.

Arguments

Table 4-83. Arguments for DatagramProxy.__init__()

self	this instance
name	The type of the proxy
session	The descriptor of the connection

4.12.4. Class Proxy

This class is inherited from an ExtensionClass implemented by the Zorp core. It encapsulates a protocol specific proxy. As an instance is created, the required module is loaded (identified by the name attribute), and a new thread is started to handle the connection.

4.12.4.1. Attributes

Table 4-84. Attributes for class Proxy

session	descriptor of the connection
name	the native proxy module used for this session

4.12.4.2. Method __destroy__

Synopsis

```
__destroy__ ( self )
```

Description

4.12.4.3. Constructor __init__

Initializes a Proxy instance.

Synopsis

```
__init__ (
        self,
        name,
        session,
        )
```

Description

Initializes a Proxy instance and sets attributes.

Arguments

Table 4-85. Arguments for Proxy.__init__()

name	The type of the proxy
session	reference to the session

4.12.4.4. Destructor __del__

Synopsis

```
__del__ ( self )
```

Description

4.12.4.5. Method addPolicy

Adds a policy to the proxy.

Synopsis

```
addPolicy ( self,  klass )
```

Description

This function extends this class with another parent class at runtime. It can be used to add methods to this event handler class based on some runtime variable.

Arguments

Table 4-86. Arguments for Proxy.addPolicy()

self	this instance
klass	the class implementing the policy

4.12.4.6. Method connectServer

Callback method called when a connection established

Synopsis

```
connectServer (
        self,
        host,
        port,
        )
```

Description

If the chainer is defined, it is called with the given host and port parameters to connect to the remote server.

If there is no chainer defined, it tries to return the server-side fd from the session. If there's no server_fd it raises an error.

Arguments

Table 4-87. Arguments for Proxy.connectServer()

self	this instance
host	The address of the server

port	The port of the server

Exceptions

- InternalError

Returns

The descriptor of the server stream

4.13. Module Pssl

4.13.1. Imported modules

- `import` Connector

  Implements classes to establish a connection.

- `import` Proxy

  Module defining classes encapsulating native proxies.

- `import` Service

  Module defining service related classes.

- `import` Session

  Module defining session related classes and functions.

- `import` Stream

  Module exporting an interface to the Zorp.Stream component.

- `from` Zorp `import *`

  Module defining global constants, and interface entry points to the Zorp core.

4.13.2. Class PsslProxy

A plug proxy which implements SSL on either sides.

Attributes:

Table 4-88. Untitled

stack_proxy	the proxy to stack into Pssl
copy_to_client	Copy data in server->client direction
copy_to_server	Copy data in client->server direction
bandwidth_to_client	Readonly variable containing the utilized bandwidth in server->client direction.
bandwidth_to_server	Readonly variable containing the utilized bandwidth in client->server direction.
packet_stats_interval	The number of milliseconds between two successive packetStats() events. By default: 0. NOTE: this is currently implemented as the number of passing packages, not as milliseconds.
client_need_ssl	Use SSL on the client side of the proxy. This requires setting client_key and client_cert
client_key	Client side authentication private key
client_cert	Client side authentication certificate

server_need_ssl	Use SSL on the server side of the proxy. Optionally you can set the server_key and server_cert attributes if you want to perform authentication.
server_key	Server side authentication private key
server_cert	Server side authentication certificate
CADirecory	Directory containing acceptable CA-s in PEM format.
CRLDirectory	Direcory containint Certificate Revokation Lists.
verify_type	Type of client (or server) verify. It's may be SSL_VERIFY_NONE, for no verification, SSL_VERIFY_OPTIONAL, for optional verification, SSL_VERIFY_REQUIRED, if Certificate is required, but may not signed with any CA, and SSL_VERIFY_REQUIRED_WITH_CA, if client (or server) must have a valid certificate signed with valid CA.
verify_depth	How deep a CA accepted.

4.13.2.1. Constructor __init__

Initializes a PsslProxy instance.

Synopsis

```
__init__ ( self,  session )
```

Description

Sets attributes based on arguments.

Arguments

Table 4-89. Arguments for PsslProxy.__init__()

self	this instance
session	the reference of the owning session

4.13.2.2. Method requestStack

Query whether to stack anything to Pssl.

Synopsis

```
requestStack ( self )
```

Description

Callback called by the underlying C proxy to query if something is to be stacked.

Arguments

Table 4-90. Arguments for PsslProxy.requestStack()

self	this instance

4.13.2.3. Method stackProxy

Actually do the stacking.

Synopsis

```
stackProxy (
        self,
        client_fd,
        server_fd,
        )
```

Description

Callback called by the underlying C proxy to actually stack in something.

Arguments

Table 4-91. Arguments for PsslProxy.stackProxy()

self	this instance
client_fd	upstream client side fd
server_fd	upstream server side fd

4.14. Module Receiver

This module defines the class `Receive`, the main entry point for UDP based protocols.

4.14.1. Imported modules

- `import` Service

  Module defining service related classes.

- `import` Session

  Module defining session related classes and functions.

- `import` Zorp

  Module defining global constants, and interface entry points to the Zorp core.

- `from` Zorp `import *`

  Module defining global constants, and interface entry points to the Zorp core.

4.14.2. Class Receive

Wrapper for the ZorpReceiver class implemented in C.

4.14.2.1. Attributes

Table 4-92. Attributes for class Receive

listen	the low-level receiver object
service	the service to start
session	the session data

4.14.2.2. Constructor __init__

Initializes a Receive instance.

Synopsis

```
__init__ (
        self,
        bindto,
        service,
        )
```

Description

Initialize a Receive instance.

Arguments

Table 4-93. Arguments for Receive.__init__()

bindto	the address to bind to
service	the service to start for incoming datagrams

Exceptions

- ServiceException

4.14.2.3. Method destroy

Destroys the receiver

Synopsis

```
destroy ( self )
```

Description

This function calls the destroy method of the low-level object.

Arguments

Table 4-94. Arguments for Receive.destroy()

self	this instance

4.15. Module Service

This module defines classes encapsulating service descriptions. A service defines how an established client request should be handled. When a connection is accepted (usually by a listener, see the Listener module), the service bound to the given Listener is requested the create an instance of itself (using the startInstance() method, see below), which will handle the connection and takes care about proxying traffic between the client and the server.

Services are uniquely identified by their name, this name is then used as a reference to bind services to listeners. This name resolution is done through the mapping named "services" below. As a new service instance is created, it’s name is registered in this hash.

The abstract interface a Service object is required to support is defined in AbstractService, customized service classes can be derived from this base class.

4.15.1. Imported modules

- `from` Session `import` StackedSession

  Module defining session related classes and functions.

- `from` Stream `import` Stream

  Module exporting an interface to the Zorp.Stream component.

- `from` Zorp `import` ServiceException, LimitException

  Module defining global constants, and interface entry points to the Zorp core.

4.15.2. Class AbstractService

This is an abstract class defining the interface Zorp uses, and as such it doesn't really do anything other than raising NotImplementedError exceptions in its methods. You should either derive a descendant from AbstractService, or use `Service` instead.

4.15.2.1. Attributes

Table 4-95. Attributes for class AbstractService

name	The name of the service
instance_id	the session serviced by the service

4.15.2.2. Constructor __init__

Initialize an AbstractService or a derived class instance.

Synopsis

```
__init__ ( self,  name )
```

Description

Sets attributes based on arguments, and registers this Service to the "services" hash so that Listeners may resolve service names to service instances.

Arguments

Table 4-96. Arguments for AbstractService.__init__()

self	this instance
name	The name of the service

Exceptions

- ServiceException

4.15.2.3. Method __str__

Function to represent this object as a string

Synopsis

```
__str__ ( self )
```

Description

This function is called by the Python core when this object is used as-, or casted to a string. It simply returns the service name.

Arguments

Table 4-97. Arguments for AbstractService.__str__()

self	this instance

4.15.2.4. Method startInstance

Function to start an instance of this service.

Synopsis

```
startInstance ( self,  session )
```

Description

Abstract method to be implemented in derived classes. Should start an instance of the given service. A service instance takes care of the client connection, connects to the server and supervises the traffic going in either direction.

Tasks of a service instance are implemented by classes derived from `Proxy`.

This method unconditionally raises a NotImplementedError exception to indicate that it must be overridden by descendant classes like `Service`.

Arguments

Table 4-98. Arguments for AbstractService.startInstance()

self	this instance
session	start service within this session

Exceptions

- NotImplementedError

4.15.2.5. Method stopInstance

Function called when an instance of this service is ended

Synopsis

```
stopInstance ( self,  session )
```

Description

This function is called by Session.__del__ and indicates that a given session (instance) of this service is ended.

Arguments

Table 4-99. Arguments for AbstractService.stopInstance()

self	this instance
session	ending session

Exceptions

- NotImplementedError

4.15.3. Class Service

This service class uses the standard Zorp provided Proxy class to implement the tasks required by a Service instance.

4.15.3.1. Attributes

Table 4-100. Attributes for class Service

chainer	The chainer class used for the service
proxy_class	The proxy class used for the service
auth	Authentication handling class (None if no authentication)

4.15.3.2. Constructor __init__

Initializes a Service instance.

Synopsis

```
__init__ (
        self,
        name,
        chainer,
        proxy_class,
        auth=None,
        max_instances=0,
        )
```

Description

This function takes its arguments, and initializes appropriate attributes in self.

Arguments

Table 4-101. Arguments for Service.__init__()

self	this instance
name	Name of this service (used in access control)
chainer	The chainer class used for the service

proxy_class-- The proxy class used for the service

Table 4-102. Arguments for Service.__init__()

auth	Authentication handling class (None if no authentication)

4.15.3.3. Method startInstance

Start a service instance.

Synopsis

```
startInstance ( self,  session )
```

Description

Called by the Listener to create an instance of this service.

Arguments

Table 4-103. Arguments for Service.startInstance()

self	this instance
session	The session object

Exceptions

- LimitException

4.15.3.4. Method stopInstance

Synopsis

```
stopInstance ( self,  session )
```

Description

4.16. Module Session

This module defines the abstract session interface in a class named `AbstractSession`, and two descendants `MasterSession` and `StackedSession`.

Sessions are hierarchically stacked into each other just like proxies. Each session has an owner session (except for the master session which is on the top), and variables are "inherited" from owner sessions. (implemented using a simple getattr wrapper) This way stacked sessions can inherit data from encapsulating proxies. (an HTTP proxy may define an URL and a mime-type, and stacked CVP module may inspect those values)

4.16.1. Imported modules

- `from` Zone `import` root_zone
- `import` Zorp

  Module defining global constants, and interface entry points to the Zorp core.
- `from` Zorp `import` *

Module defining global constants, and interface entry points to the Zorp core.

- `import` os

4.16.2. Class AbstractSession

Both MasterSession and StackedSession are derived from this class.

4.16.2.1. Attributes

none

4.16.2.2. Method destroy

Destroys the session.

Synopsis

```
destroy ( self )
```

Description

We close filedescriptors here, in case no proxy module could be started (because of policy violations, or because the module cannot be found).

Arguments

Table 4-104. Arguments for AbstractSession.destroy()

self	this instance

4.16.3. Class MasterSession

Master session class.

4.16.3.1. Attributes

Table 4-105. Attributes for class MasterSession

client_fd	client fd
client_stream	client stream
client_address	SockAddr instance containing client address
client_local	local address (on the firewall)
client_zone	zone of the client
server_fd	server fd
server_stream	server stream
server_address	SockAddr instance containing server address
server_local	local address (on the firewall)
server_zone	zone of the server
service	service instance this session runs
session_id	unique identifier for this session in the format: "(firewall/service:instance id/proxy)"
instance_id	the instance identifier of the service (sequence number)
started	indicates that the instance has been started
auth	authentication method

4.16.3.2. Constructor __init__

Initializes a MasterSession instance.

Synopsis

```
__init__ ( self )
```

Description

This constructor initializes a new MasterSession instance based on its arguments.

Arguments

Table 4-106. Arguments for MasterSession.__init__()

self	this instance

4.16.3.3. Destructor __del__

Function called when the master session is freed.

Synopsis

```
__del__ ( self )
```

Description

This function is called when the master session is freed, thus the session ended. We inform our spawner service about this event.

4.16.3.4. Method isClientPermitted

Function to actually check access control.

Synopsis

```
isClientPermitted ( self )
```

Description

This function is called when a connection is established to perform access control checks whether the client is permitted to use the requested service. Its return value specifies the result of the check.

Arguments

Table 4-107. Arguments for MasterSession.isClientPermitted()

self	this instance

Returns

Z_ACCEPT for success, and Z_REJECT for failure

4.16.3.5. Method isServerPermitted

Synopsis

```
isServerPermitted ( self )
```

Description

4.16.3.6. Method setClient

Set client address and perform access control.

Synopsis

```
setClient (
        self,
        fd,
        addr,
        )
```

Description

Sets the client address of the given session, and performs access control checks.

Arguments

Table 4-108. Arguments for MasterSession.setClient()

self	this instance
fd	fd of the client
addr	sockaddr of the client

4.16.3.7. Method setServer

Set the server address and perform access control checks.

Synopsis

```
setServer ( self,  addr )
```

Description

Stores the server address of the given connection, looks up server zone and performs access control and raises an exception upon failure.

Arguments

Table 4-109. Arguments for MasterSession.setServer()

self	this instance
addr	Server address

4.16.3.8. Method setService

Sets the service belonging to this session.

Synopsis

```
setService ( self,  service )
```

Description

Stores the service reference, and recalculates the session_id

Arguments

Table 4-110. Arguments for MasterSession.setService()

self	this instance
service	Service instance

4.16.3.9. Method setServiceInstance

Set service instance number and recalculate session id.

Synopsis

```
setServiceInstance ( self,  instance_id )
```

Description

Sets service instance number, and makes up a unique identifier for this session.

Arguments

Table 4-111. Arguments for MasterSession.setServiceInstance()

self	this instance
instance_id	unique identifier of the service instance

4.16.4. Class StackedSession

A StackedSession is a subsession, inheriting attributes from its parent.

4.16.4.1. Attributes

Table 4-112. Attributes for class StackedSession

owner	Parent session
chainer	Chainer used to chain up to parent. If none simply server_fd is used.

4.16.4.2. Constructor __init__

Initializes a StackedSession instance.

Synopsis

```
__init__ (
        self,
        owner,
        chainer=None,
        )
```

Description

This constructor initializes a new StackedSession instance based on parameters.

Arguments

Table 4-113. Arguments for StackedSession.__init__()

self	this instance
owner	Parent session
chainer	Chainer used to chain up to parent.

4.16.4.3. Method __getattr__

Perform attribute inheritance

Synopsis

```
__getattr__ ( self,  name )
```

Description

Wrapper to return variables from parent session, if not overriden in this instance.

Arguments

Table 4-114. Arguments for StackedSession.__getattr__()

self	this instance
name	Name of the attribute to get.

Returns

The value of the given attribute.

4.16.4.4. Method setProxy

Set the proxy name used in this subsession.

Synopsis

```
setProxy ( self,  proxy )
```

Description

Stores a reference to the proxy class, and modifies the session_id to include the proxy name.

Arguments

Table 4-115. Arguments for StackedSession.setProxy()

self	this instance
proxy	Proxy class, derived from Proxy

4.17. Module SockAddr

This module implements inet_ntoa and inet_aton, and provides an interface to SockAddr services provided by the Zorp core.

4.17.1. Imported modules

- `from` socket `import` htonl, ntohl
- `from` string `import` split, atoi

4.17.2. Functions

4.17.2.1. Function inet_aton

Converts an internet address to a 32 bit integer

Synopsis

```
inet_aton ( ip )
```

Description

splits on the dot, atoi the parts, and bitshift the whole thing into one

Arguments

Table 4-116. Arguments for .inet_aton()

ip	A dotted-quad string

Returns

unsigned long in network byte order

4.17.2.2. Function inet_ntoa

Converts a 32 bit integer into IP number's string representation

Synopsis

```
inet_ntoa ( ip )
```

Description

Masks the necessary bytes out and formats them into a string.

Arguments

Table 4-117. Arguments for .inet_ntoa()

ip	The ip number in 32 bit integer (network byte order)

Returns

string representation of IP

4.17.3. Class SockAddrInet

This class encapsulates an IPv4 address:port pair, similar to the struct sockaddr_in in C. It is implemented and exported by the Zorp core.

4.17.3.1. Attributes

Table 4-118. Attributes for class SockAddrInet

ip	ip address (network byte order)
ip_s	ip address in string representation
port	port number (network byte order)

4.17.4. Class SockAddrInetRange

Specialized SockAddrInet class which allocates a new port within the given range of ports when a listener bounds to it.

4.17.4.1. Attributes

Table 4-119. Attributes for class SockAddrInetRange

ip	ip address (network byte order)
ip_s	ip address in string representation
port	port number (network byte order)

4.18. Module Stream

This defines the Stream class, encapsulating a file descriptor and related functions.

4.18.1. Class Stream

This class encapsulates a data tunnel, represented by a UNIX file descriptor.

4.18.1.1. Attributes

none

4.18.1.2. Constructor __init__

Constructor initializing a stream.

Synopsis

```
__init__ (
        self,
        fd,
        name,
        )
```

Description

Initializes a Stream instance setting its attributes according to arguments.

Arguments

Table 4-120. Arguments for Stream.__init__()

self	this instance
fd	the fd to encapsulate
name	name to use in logs

4.18.1.3. Method read

Method reading up to count bytes from the stream.

Synopsis

```
read ( self,  count )
```

Description

This method reads up to count bytes from the stream and returns it as a string.

Arguments

Table 4-121. Arguments for Stream.read()

self	this instance
count	maximum number of bytes to read

4.18.1.4. Method write

Method writing a buffer to the stream.

Synopsis

```
write ( self,  buf )
```

Description

This method writes the contents of the given buffer to the stream.

Arguments

Table 4-122. Arguments for Stream.write()

self	this instance
buf	buffer to write

4.19. Module Zone

4.19.1. Imported modules

- `from` Domain `import` InetDomain

  Module implementing address domains.

- `from` Zorp `import` *

  Module defining global constants, and interface entry points to the Zorp core.

- `from` traceback `import` print_exc
- `import` types

4.19.2. Class InetZone

This is a simple Zone class using InetDomain as its address type.

4.19.2.1. Constructor __init__

Initializes an InetZone instance

Synopsis

```
__init__ (
        self,
        name,
        addr,
        inbound_services=None,
        outbound_services=None,
        admin_parent=None,
        umbrella=0,
        )
```

Description

Initializes an InetZone object instance, and sets its attributes based on arguments.

Arguments

Table 4-123. Arguments for InetZone.__init__()

self	this instance
name	name of this zone
addr	a string representing an address range, interpreted by the domain class (last argument), **or a list of strings representing multiple address ranges.**
inbound_services	set of permitted inbound services as described by RootZone
outbound_services	set of permitted outbound services as described by RootZone
admin_parent	name of the administrative parent
umbrella	TRUE if this zone is an umbrella zone

4.19.3. Class RootZone

A zone is the basis of access control in Zorp. It encapsulates an address range (for example an IPv4 subnet). Parsing and representing an address is done by the Domain class and derivates.

Zones are organized into two different hierarchies. The first is based on the address it encapsulates. In this case a zone is a child of some other zone, if the parent is containing it. (For example 0.0.0.0/0 contains 192.168.0.0/24) This hierarchy is used when searching the containing zone of a given network address.

The other hierarchy is based on administrative decisions. Administrative children inherit security attributes (permitted set of services etc.) from their parents.

The RootZone class serves two purposes:

1. it is the base class for all Zone-like classes, implementing interfaces for access control, and hierarchy travelsal.

2. an instance of this class is an address domain independent root of the Zone address hierarchy and delegates searches to the appropriate Zone.

4.19.3.1. Attributes

Table 4-124. Attributes for class RootZone

name	a unique name of this zone
inbound_services	mapping indexed by service name, containing an item for each permitted inbound service
outbound_services	similar to inbound_services, but used for outbound services.
admin_parent	parent of this zone in the administrative hierarchy
umbrella	true for umbrella zones (ie. zones that do not inherit security attributes from their administrative parents)

4.19.3.2. Constructor __init__

Initialize a RootZone instance.

Synopsis

```
__init__ (
        self,
        name,
        inbound_services=None,
        outbound_services=None,
        admin_parent=None,
        umbrella=0,
        )
```

Description

This function is primarily called by derived classes to initialize basic Zone data structures.

Arguments

Table 4-125. Arguments for RootZone.__init__()

self	this instance
name	name of this zone
inbound_services	an array of allowed inbound service names
outbound_services	an array of allowed outbound service names
admin_parent	administrative parent name
umbrella	this is an umbrella zone
domain	address domain

Exceptions

- ZoneException

4.19.3.3. Method __str__

Overridden operator to return the textual representation of self.

Synopsis

```
__str__ ( self )
```

Description

Called by the Python core to format the object contents when it is written.

Arguments

Table 4-126. Arguments for RootZone.__str__()

self	this instance

4.19.3.4. Method addAddrChild

Add a address-based child to this zone.

Synopsis

```
addAddrChild ( self,  child )
```

Description

This function adds a children to the set of child zones.

Arguments

Table 4-127. Arguments for RootZone.addAddrChild()

self	this instance
child	child to add

4.19.3.5. Method addAdminChild

Add an administrative child

Synopsis

```
addAdminChild ( self,  child )
```

Description

This function adds `child` to the set of administrative children.

Arguments

Table 4-128. Arguments for RootZone.addAdminChild()

self	this instance
child	child zone add

4.19.3.6. Method delAddrChild

Delete an address child.

Synopsis

```
delAddrChild ( self,  child )
```

Description

This function removes an item from the set of address children.

Arguments

Table 4-129. Arguments for RootZone.delAddrChild()

self	this instance
child	child to remove

4.19.3.7. Method findDomain

Find the root Zone of the given address domain.

Synopsis

```
findDomain ( self,  domain )
```

Description

Finds the first child in self which uses

domain

as address domain.

Arguments

Table 4-130. Arguments for RootZone.findDomain()

self	this instance
domain	class implementing address range specifics (for example InetDomain for IPv4)

Exceptions

- ZoneException

4.19.3.8. Method findZone

Find the most specific Zone for address.

Synopsis

```
findZone ( self,  address )
```

Description

This function searches the address hierarchy for the most specific Zone containing `address`.

Arguments

Table 4-131. Arguments for RootZone.findZone()

self	this instance

address	address we are trying to find (should be derived from SockAddr)

Exceptions

- ZoneException

4.19.3.9. Method isInboundServicePermitted

Inbound access control check.

Synopsis

```
isInboundServicePermitted ( self,  session )
```

Description

This function is called when a session is connecting to a server to check whether it is permitted.

Arguments

Table 4-132. Arguments for RootZone.isInboundServicePermitted()

self	this instance
session	session that should be checked

Returns

Z_ACCEPT if the service is permitted, Z_REJECT otherwise

4.19.3.10. Method isOutboundServicePermitted

Outbound access control check.

Synopsis

```
isOutboundServicePermitted ( self,  session )
```

Description

This function is called when an incoming connection is detected to check whether it is allowable.

Arguments

Table 4-133. Arguments for RootZone.isOutboundServicePermitted()

self	this instance
session	session that should be checked

Returns

Z_ACCEPT if the service is permitted, Z_REJECT otherwise

4.19.3.11. Method iterAddrChildren

Iterate over the set of address children

Synopsis

```
iterAddrChildren (
        self,
        fn,
        parm=None,
        )
```

Description

This function iterates over the set of address children and calls `fn` for each item.

Arguments

Table 4-134. Arguments for RootZone.iterAddrChildren()

self	this instance
fn	function to call
parm	opaque argument passed to fn

4.19.3.12. Method iterAdminChildren

Iterate over the set of administrative children.

Synopsis

```
iterAdminChildren (
        self,
        fn,
        parm=None,
        )
```

Description

This function iterates over the set of administrative children calling the function `fn` for each item, with parameters `parm`, `self` and the item.

The callback fn may delete items from admin_children, this function uses a local copy of that array.

Arguments

Table 4-135. Arguments for RootZone.iterAdminChildren()

self	this instance
fn	function to call
parm	opaque object passed to fn

4.19.3.13. Method setAddrParent

Set address parent of this zone.

Synopsis

```
setAddrParent ( self,  parent )
```

Description

This function sets the address parent of this Zone.

Arguments

Table 4-136. Arguments for RootZone.setAddrParent()

self	this instance

parent	parent Zone

4.19.3.14. Method setAdminParent

Set administrative parent of this zone.

Synopsis

```
setAdminParent ( self,  parent )
```

Description

This function sets the administrative parent of this Zone.

Arguments

Table 4-137. Arguments for RootZone.setAdminParent()

self	this instance
parent	parent Zone

4.19.4. Class Zone

This class differs from RootZone in that it uses a real address domain (for IPv4 InetDomain is used), unlike RootZone which is a general wrapper for all address types (IPv4, IPv6, SPX etc.)

4.19.4.1. Constructor __init__

Initializes a Zone instance.

Synopsis

```
__init__ (
        self,
        name,
        addr,
        inbound_services=None,
        outbound_services=None,
        admin_parent=None,
        umbrella=0,
        domain=None,
        )
```

Description

This class initializes a Zone instance by calling the inherited constructor, and setting local attributes.

Arguments

Table 4-138. Arguments for Zone.__init__()

self	this instance
name	name of this zone
addr	a string representing an address range interpreted by the domain class (last argument), **or a list of strings representing multiple address ranges.**
inbound_services	set of permitted inbound services as described by RootZone
outbound_services	set of permitted outbound services as described by RootZone
admin_parent	name of the administrative parent

umbrella	TRUE if this zone is an umbrella zone
domain	address domain class parsing `addr` and performing address comparisons for IPv4 addresses it should be `InetDomain`

Exceptions

- ValueError

Notes

If `addr` is a list of addresses (like ['192.168.1.1', '192.168.1.5']), several subzones are automatically created with administrative parent set to self. This way you can define members with additional privilege easily.

4.19.4.2. Method __str__

Format the Zone as string

Synopsis

```
__str__ ( self )
```

Description

This function is called by the Python core when this object is used as string.

4.19.4.3. Method findZone

Find the most specific containing Zone of `addr`

Synopsis

```
findZone ( self,  addr )
```

Description

This function returns the most specific Zone containing `addr`

Arguments

Table 4-139. Arguments for Zone.findZone()

self	this instance
addr	address to look up

Exceptions

- ZoneException

4.19.4.4. Method setAddrRelatives

Helper function to place this zone into the address hierarchy.

Synopsis

```
setAddrRelatives ( self )
```

Description

This function is called by the Zone constructor to place this zone into the address hierarchy.

Arguments

Table 4-140. Arguments for Zone.setAddrRelatives()

self	this instance

Exceptions

- NotImplementedError

4.20. Module Zorp

This module defines some global constants used by other Zorp components like `TRUE` and `FALSE`, and interface entry points to the Zorp core.

4.20.1. Functions

4.20.1.1. Function debug

Logs a debug message.

Synopsis

```
debug ( level,  msg )
```

Description

Sends the given message to the low level Zorp provided log function.

 this function is obsolete and should not be used.

Arguments

Table 4-141. Arguments for .debug()

level	log level
msg	message

4.20.1.2. Function error

Logs an error message

Synopsis

```
error ( level,  msg )
```

Description

Sends the given message to the low level Zorp provided log function.

 This function is obsolete and should not be used.

Arguments

Table 4-142. Arguments for .error()

level	log level
msg	message

4.20.1.3. Function init

Default init() function provided by Zorp

Synopsis

```
init ( name )
```

Description

This function is a default init() calling the init function identified by name. This way several Zorp instances can easily use the same policy file.

Arguments

Table 4-143. Arguments for .init()

name	name of this instance

4.20.1.4. Function log

Entry point of Zorp logging subsystem.

Synopsis

```
log (
        logclass,
        verbosity,
        msg,
        )
```

Description

This function is implemented in C, and can be used to inject messages into Zorp logging subsystem.

Arguments

Table 4-144. Arguments for .log()

logclass	hierarchical log class as described in zorp(8)
verbosity	verbosity of this message
msg	message

4.20.1.5. Function message

Logs an informational message

Synopsis

```
message ( level,  msg )
```

Description

Sends the given message to the low level Zorp provided log function.

 This function is obsolete and should not be used.

Arguments

Table 4-145. Arguments for .message()

level	log level
msg	message

4.20.2. Class ZorpProxy

Builtin class for proxies implemented as an ExtensionClass in C

This class should not be used directly, derive classes from Proxy.Proxy.

4.20.2.1. Constructor __init__

Initialize a low level proxy instance.

Synopsis

```
__init__ (
        self,
        name,
        session_id,
        client_stream,
        )
```

Description

This function is implemented in C, and is responsible of loading the needed proxy module from a shared object if it has not yet been loaded, and starting a new instance with parameters passed here.

Arguments

Table 4-146. Arguments for ZorpProxy.__init__()

self	this instance
name	proxy module name (`http, plug etc.`)
session_id	session id to use in log messages
client_stream	client stream

4.21. Module __init__