polyswarm-client Documentation

PolySwarm

Nov 30, 2021

CONTENTS

CHAPTER

ONE

API REFERENCE

This page contains auto-generated API reference documentation[1].

1.1 conftest

1.2 ambassador

1.2.1 Submodules

ambassador.__main__

Module Contents

ambassador.__main__.**logger**

ambassador.__main__.**choose_backend**(*backend*)

Resolves amabassador name string to implementation

Parameters **backend** (*str*) – Name of the backend to load, either one of the predefined implementations or the name of a module to load (module:ClassName syntax or default of module:Ambassador)

Returns Ambassador class of the selected implementation

Return type (Class)

Raises **(Exception)** – If backend is not found

ambassador.__main__.**main**(*log*, *client_log*, *polyswarmd_addr*, *keyfile*, *password*, *api_key*, *backend*, *testing*, *insecure_transport*, *allow_key_over_http*, *chains*, *watchdog*, *log_format*, *submission_rate*)

Entrypoint for the ambassador driver

ambassador.eicar

Module Contents

ambassador.eicar.**logger**

ambassador.eicar.**EICAR**

[1] Created with sphinx-autoapi

ambassador.eicar.**NOT_EICAR = not a malicious file**

ambassador.eicar.**ARTIFACTS = [None, None]**

ambassador.eicar.**BOUNTY_TEST_DURATION_BLOCKS**

class ambassador.eicar.**Ambassador**(*client*, *testing=0*, *chains=None*, *watchdog=0*, *submission_rate=30*)

Bases: *polyswarmclient.abstractambassador.AbstractAmbassador*

Ambassador which submits the EICAR test file

ambassador.filesystem

Module Contents

ambassador.filesystem.**logger**

ambassador.filesystem.**ARTIFACT_DIRECTORY**

ambassador.filesystem.**ARTIFACT_BLACKLIST**

ambassador.filesystem.**ARTIFACTS_PER_BOUNTY**

ambassador.filesystem.**BOUNTY_TEST_DURATION_BLOCKS**

class ambassador.filesystem.**Ambassador**(*client*, *testing=0*, *chains=None*, *watchdog=0*, *submission_rate=30*)

Bases: *polyswarmclient.abstractambassador.AbstractAmbassador*

Ambassador which submits artifacts from a directory

1.3 worker

1.3.1 Submodules

worker.__main__

Module Contents

worker.__main__.**logger**

worker.__main__.**choose_backend**(*backend*)
Resolves scanner name string to implementation

Parameters

- **backend** (*str*) – Name of the backend to load, either one of the predefined implementations or the name of a module
- **load** (*to*) –
- **(module** – ClassName syntax or default of module:Scanner)

Returns Scanner class of the selected implementation

Return type (Class)

Raises **(Exception)** – If backend is not found

worker.__main__.**main**(*log*, *client_log*, *redis_addr*, *queue*, *backend*, *tasks*, *download_limit*, *scan_limit*, *api_key*, *testing*, *log_format*, *scan_time_requirement*, *daily_rate_limit*, *hourly_rate_limit*, *minutely_rate_limit*, *secondly_rate_limit*, *allow_key_over_http*)

Entrypoint for the worker driver

worker.base

Module Contents

worker.base.**logger**

worker.base.**REQUEST_TIMEOUT = 5.0**

class worker.base.**RateLimitAggregate**(*redis*, *queue*, *daily_rate_limit*, *hourly_rate_limit*, *minutely_rate_limit*, *secondly_rate_limit*)

Bases: *polyswarmclient.ratelimit.abstractratelimit.AbstractRateLimit*

daily :Optional[RedisRateLimit]

hourly :Optional[RedisRateLimit]

minutely :Optional[RedisRateLimit]

secondly :Optional[RedisRateLimit]

class worker.base.**OptionalSemaphore**(*value=1*, *loop=None*)

Bases: asyncio.Semaphore

given_value :int

release(*self*)

class worker.base.**Worker**(*redis_addr*, *queue*, *task_count=0*, *download_limit=0*, *scan_limit=0*, *api_key=None*, *testing=0*, *scanner=None*, *scan_time_requirement=0*, *daily_rate_limit=None*, *hourly_rate_limit=None*, *minutely_rate_limit=None*, *secondly_rate_limit=None*, *allow_key_over_http=False*)

run(*self*)

start(*self*, *loop: asyncio.AbstractEventLoop*)

setup_synchronization(*self*, *loop: asyncio.AbstractEventLoop*)

setup_graceful_shutdown(*self*, *loop: asyncio.AbstractEventLoop*)

handle_signal(*self*)

get_remaining_time(*self*, *job: JobRequest*)

rate_limit_respond(*self*, *job: JobRequest*)

is_key_secure(*self*, *job: JobRequest*)

worker.exceptions

Module Contents

class worker.exceptions.**ExpiredException**

Bases: *polyswarmclient.exceptions.PolyswarmClientException*

Worker skipped scanning some artifact due to the bounty expiring before getting scanned. Seen when the worker was down for a period of time, or when there aren't enough workers to keep up with load.

class worker.exceptions.**EmptyJobsQueueException**
Bases: *polyswarmclient.exceptions.PolyswarmClientException*

Worker Queue is empty

1.3.2 Package Contents

class worker.**Worker**(*redis_addr*, *queue*, *task_count=0*, *download_limit=0*, *scan_limit=0*, *api_key=None*, *testing=0*, *scanner=None*, *scan_time_requirement=0*, *daily_rate_limit=None*, *hourly_rate_limit=None*, *minutely_rate_limit=None*, *secondly_rate_limit=None*, *allow_key_over_http=False*)

run(*self*)

start(*self*, *loop: asyncio.AbstractEventLoop*)

setup_synchronization(*self*, *loop: asyncio.AbstractEventLoop*)

setup_graceful_shutdown(*self*, *loop: asyncio.AbstractEventLoop*)

handle_signal(*self*)

get_remaining_time(*self*, *job: JobRequest*)

rate_limit_respond(*self*, *job: JobRequest*)

is_key_secure(*self*, *job: JobRequest*)

1.4 balancemanager

1.4.1 Submodules

balancemanager.__main__

Module Contents

balancemanager.__main__.**logger**

balancemanager.__main__.**validate_optional_transfer_amount**(*ctx*, *param*, *value*)

balancemanager.__main__.**validate_transfer_amount**(*ctx*, *param*, *value*)

balancemanager.__main__.**polyswarm_client**(*func*)

balancemanager.__main__.**cli**(*log*, *client_log*, *log_format*)
Entrypoint for the balance manager driver

balancemanager.__main__.**deposit**(*polyswarmd_addr*, *keyfile*, *password*, *api_key*, *testing*, *insecure_transport*, *allow_key_over_http*, *denomination*, *all*, *amount*)
Deposit NCT into a sidechain

balancemanager.__main__.**withdraw**(*polyswarmd_addr*, *keyfile*, *password*, *api_key*, *testing*, *insecure_transport*, *allow_key_over_http*, *denomination*, *all*, *amount*)
Withdraw NCT from a sidechain

balancemanager.__main__.**deposit_stake**(*polyswarmd_addr*, *keyfile*, *password*, *api_key*, *testing*, *insecure_transport*, *allow_key_over_http*, *denomination*, *all*, *chain*, *amount*)
Deposit NCT into the ArbiterStaking contract

balancemanager.__main__.**withdraw_stake**(*polyswarmd_addr*, *keyfile*, *password*, *api_key*, *testing*, *insecure_transport*, *allow_key_over_http*, *denomination*, *all*, *chain*, *amount*)
Withdraw NCT from the ArbiterStaking contract

balancemanager.__main__.**maintain**(*polyswarmd_addr*, *keyfile*, *password*, *api_key*, *testing*, *insecure_transport*, *allow_key_over_http*, *denomination*, *maximum*, *withdraw_target*, *confirmations*, *minimum*, *refill_amount*)
Maintain min/max NCT balance in sidechain

balancemanager.__main__.**view_balance**(*polyswarmd_addr*, *keyfile*, *password*, *api_key*, *testing*, *insecure_transport*, *allow_key_over_http*, *denomination*, *chain*)

balancemanager.__main__.**view_stake**(*polyswarmd_addr*, *keyfile*, *password*, *api_key*, *testing*, *insecure_transport*, *allow_key_over_http*, *denomination*, *chain*)

1.4.2 Package Contents

balancemanager.**logger**

balancemanager.**RELAY_LEEWAY = 5**

balancemanager.**convert**(*client*, *denomination*, *amount*)
Convert the amount from it's original precision to 18 decimals

balancemanager.**convert_from**(*client*, *denomination*, *amount*)
Convert the amount from 18 decimals to the dedsired precision

class balancemanager.**BalanceManager**(*client*, *denomination*, *transfer_all*, *amount*, *testing=0*, *chains=None*)

Bases: object

Balance manager is used for single transfer events in either direction. Create a client, choose a chain and amount then run it.

run(*self*)
Starts the client on whichever chain this uses.

run_oneshot(*self*)
Runs run_task once

class balancemanager.**Deposit**(*client*, *denomination*, *transfer_all*, *amount*, *testing=0*)
Bases: *balancemanager.BalanceManager*

Deposit only version of Balance Manager

class balancemanager.**Withdraw**(*client*, *denomination*, *transfer_all*, *amount*, *testing=0*)
Bases: *balancemanager.BalanceManager*

Withdraw only version of Balance Manager

class balancemanager.**DepositStake**(*client*, *denomination*, *transfer_all*, *amount*, *testing=0*, *chain='side'*)
Bases: *balancemanager.BalanceManager*

Deposit only version of Balance Manager

class balancemanager.**WithdrawStake**(*client*, *denomination*, *transfer_all*, *amount*, *testing=0*, *chain='side'*)
Bases: *balancemanager.BalanceManager*

Withdraw only version of Balance Manager

class balancemanager.**Maintainer**(*client*, *denomination*, *confirmations*, *minimum*, *refill_amount*, *maximum*, *withdraw_target*, *testing=0*)
Bases: object

This class maintains a balance on the sidechain. It requires a base setup of a minimum balance. Optionally, it can take a maximum balance, so that earnings can automatically be transferred back to the homechain.

run(*self*)
Starts the client. Have to run with both chains, or lots of nonce errors

class balancemanager.**ViewBalance**(*client*, *denomination*, *chain*)
Bases: object

ViewBalance retrieves the NCT balance from a chain Create a client, choose a chain and amount then run it.

run(*self*)
Starts the client on whichever chain this uses.

run_oneshot(*self*)
Runs run_task once

class balancemanager.**ViewStake**
Bases: *balancemanager.ViewBalance*

1.5 arbiter

1.5.1 Subpackages

arbiter.verbatimdb

Submodules

arbiter.verbatimdb.__main__

Module Contents

arbiter.verbatimdb.__main__.**main**(*malicious*, *benign*, *output*, *log_format*)

arbiter.verbatimdb.db

Module Contents

arbiter.verbatimdb.db.**generate_db**(*db_file*, *malicious_dir*, *benign_dir*)

arbiter.verbatimdb.db.**insert**(*cursor*, *path*, *result*)

1.5.2 Submodules

arbiter.__main__

Module Contents

arbiter.__main__.**logger**

arbiter.__main__.**choose_backend**(*backend*)
Resolves arbiter name string to implementation

Parameters **backend** (*str*) – Name of the backend to load, either one of the predefined implementations or the name of a module to load (module:ClassName syntax or default of module:Arbiter)

Returns Arbiter class of the selected implementation

Return type (Class)

Raises **(Exception)** – If backend is not found

arbiter.__main__.**main**(*log*, *client_log*, *polyswarmd_addr*, *keyfile*, *password*, *api_key*, *backend*, *testing*, *insecure_transport*, *allow_key_over_http*, *chains*, *log_format*, *artifact_type*)
Entrypoint for the arbiter driver

arbiter.clamav

Module Contents

arbiter.clamav.**logger**

class arbiter.clamav.**Arbiter**(*client*, *testing=0*, *scanner=None*, *chains=None*, *artifact_types=None*)

Bases: *polyswarmclient.abstractarbiter.AbstractArbiter*

Arbiter which scans samples through clamd.

Re-uses the scanner from the clamav microengine

Parameters

- **client** (*Client*) – Client to use
- **testing** (*int*) – How many test bounties to respond to
- **chains** (*set[str]*) – Chain(s) to operate on

arbiter.eicar

Module Contents

arbiter.eicar.**logger**

arbiter.eicar.**EICAR**

class arbiter.eicar.**Arbiter**(*client*, *testing=0*, *scanner=None*, *chains=None*, *artifact_types=None*)

Bases: *polyswarmclient.abstractarbiter.AbstractArbiter*

Arbiter which matches hashes to a database of known samples

arbiter.producer

Module Contents

arbiter.producer.**logger**

arbiter.producer.**REDIS_ADDR**

arbiter.producer.**QUEUE**

arbiter.producer.**RATE_LIMIT**

arbiter.producer.**TIME_TO_POST_VOTE = 6**

class arbiter.producer.**Arbiter**(*client*, *testing=0*, *scanner=None*, *chains=None*, *artifact_types=None*)

Bases: *polyswarmclient.abstractarbiter.AbstractArbiter*

arbiter.verbatim

Module Contents

arbiter.verbatim.**logger**

arbiter.verbatim.**ARTIFACT_DIRECTORY**

arbiter.verbatim.**EICAR**

class arbiter.verbatim.**Arbiter**(*client*, *testing=0*, *scanner=None*, *chains=None*, *artifact_types=None*)

Bases: *polyswarmclient.abstractarbiter.AbstractArbiter*

Arbiter which matches hashes to a database of known samples

1.6 polyswarmclient

1.6.1 Subpackages

polyswarmclient.ethereum

Subpackages

polyswarmclient.ethereum.bountiesclient

Submodules

polyswarmclient.ethereum.bountiesclient.base

Module Contents

polyswarmclient.ethereum.bountiesclient.base.**logger**

class polyswarmclient.ethereum.bountiesclient.base.**BountiesClient**(*client*)
Bases: object

polyswarmclient.ethereum.bountiesclient.transaction

Module Contents

class polyswarmclient.ethereum.bountiesclient.transaction.**PostBountyTransaction**(*client*, *artifact_type*, *amount*, *bounty_fee*, *artifact_uri*, *num_artifacts*, *duration*, *bloom_*, *metadata*)

Bases: *polyswarmclient.ethereum.transaction.EthereumTransaction*

get_path(*self*)

get_body(*self*)

has_required_event(*self*, *transaction_events*)

class polyswarmclient.ethereum.bountiesclient.transaction.**PostAssertionTransaction**(*client*, *bounty_guid*, *bid*, *assertion_fee*, *mask*, *commitment*)

Bases: *polyswarmclient.ethereum.transaction.EthereumTransaction*

get_body(*self*)

get_path(*self*)

has_required_event(*self*, *transaction_events*)

class polyswarmclient.ethereum.bountiesclient.transaction.**RevealAssertionTransaction**(*client*, *bounty_index*, *nonce*, *verdicts*, *metadata*)

Bases: *polyswarmclient.ethereum.transaction.EthereumTransaction*

get_path(*self*)

get_body(*self*)

has_required_event(*self*, *transaction_events*)

class polyswarmclient.ethereum.bountiesclient.transaction.**PostVoteTransaction**(*client*, *bounty_guid*, *votes*, *valid_bloom*)

Bases: *polyswarmclient.ethereum.transaction.EthereumTransaction*

get_path(*self*)

get_body(*self*)

has_required_event(*self*, *transaction_events*)

class polyswarmclient.ethereum.bountiesclient.transaction.**SettleBountyTransaction**(*client*, *bounty_guid*

Bases: *polyswarmclient.ethereum.transaction.EthereumTransaction*

get_path(*self*)

get_body(*self*)

has_required_event(*self*, *transaction_events*)

polyswarmclient.ethereum.transaction

Submodules

polyswarmclient.ethereum.transaction.base

Module Contents

polyswarmclient.ethereum.transaction.base.**logger**

polyswarmclient.ethereum.transaction.base.**LOG_MSG_ENGINE_TOO_SLOW = PLEASE REVIEW YOUR SCA**

Bounty inactive errors indicate that the microengine received the bounty, but was unable to respond to the bounty within the time window. Such errors are considered fatal during testing so you can easily identify them. If your engine is unable to respond within the time window on the live PolySwarm network, you risk losing the bid amount of the bounty at hand. We strongly encourage you to review your artifact scan process to identify areas where engine speed can be improved.

class polyswarmclient.ethereum.transaction.base.**EthereumTransaction**(*client*, *verifiers*)

Used to verify and post groups of transactions that make up a specific action.

For instance, when approving some funds to move, and calling a contract function that will consumer them.

__sign_transactions(*self*, *transactions*)

Sign a set of transactions

Parameters **transactions** (*List[Transaction]*) – The transactions to sign

Returns The signed transactions

Return type List[Transaction]

has_required_event(*self*, *transaction_events*)

Checks for existence of events in transaction logs, ensuring successful completion

Returns True if the required event was in the list, false otherwise

get_path(*self*)

Get the path of the route to build this transaction

Returns Polyswarmd path to get the transaction data

Return type str

get_body(*self*)

Build the payload to send to polyswarmd :returns: Dict payload

verify(*self*, *transactions*)

Check the given transactions against known expectations

Parameters **transactions** (*list*) –

Returns True if transactions match expectations. False otherwise

Return type (bool)

polyswarmclient.ethereum.transaction.noncemanager

Module Contents

polyswarmclient.ethereum.transaction.noncemanager.**logger**

class polyswarmclient.ethereum.transaction.noncemanager.**NonceManager**(*client*, *chain*)

Manages the nonce for some Ethereum chain

static find_gaps(*nonces*)

Finds any gaps between base nonce and the last nonce in the given nonces list.

Parameters **nonces** (*list[int]*) – list of nonces being checked

Returns (list[int]): Any missing nonces between base_nonce and the last given nonce

Package Contents

class polyswarmclient.ethereum.transaction.**EthereumTransaction**(*client*, *verifiers*)

Used to verify and post groups of transactions that make up a specific action.

For instance, when approving some funds to move, and calling a contract function that will consumer them.

__sign_transactions(*self*, *transactions*)
Sign a set of transactions

Parameters **transactions** (*List[Transaction]*) – The transactions to sign

Returns The signed transactions

Return type List[Transaction]

has_required_event(*self*, *transaction_events*)
Checks for existence of events in transaction logs, ensuring successful completion

Returns True if the required event was in the list, false otherwise

get_path(*self*)
Get the path of the route to build this transaction

Returns Polyswarmd path to get the transaction data

Return type str

get_body(*self*)
Build the payload to send to polyswarmd :returns: Dict payload

verify(*self*, *transactions*)
Check the given transactions against known expectations

Parameters **transactions** (*list*) –

Returns True if transactions match expectations. False otherwise

Return type (bool)

class polyswarmclient.ethereum.transaction.**NonceManager**(*client*, *chain*)
Manages the nonce for some Ethereum chain

static find_gaps(*nonces*)
Finds any gaps between base nonce and the last nonce in the given nonces list.

Parameters **nonces** (*list[int]*) – list of nonces being checked

Returns (list[int]): Any missing nonces between base_nonce and the last given nonce

Submodules

polyswarmclient.ethereum.balanceclient

Module Contents

polyswarmclient.ethereum.balanceclient.**logger**

polyswarmclient.ethereum.balanceclient.**MAX_TRIES**

class polyswarmclient.ethereum.balanceclient.**BalanceClient**(*client*)
Bases: object

polyswarmclient.ethereum.bloom

Module Contents

polyswarmclient.ethereum.bloom.**FILTER_BITS**

polyswarmclient.ethereum.bloom.**HASH_FUNCS = 8**

polyswarmclient.ethereum.bloom.**logger**

class polyswarmclient.ethereum.bloom.**BloomFilter**(*value=0*)

Bases: numbers.Number

value

__int__(*self*)

add(*self*, *value*)

Add a single byte value to the Bloom filter.

Parameters **value** (*bytes*) – Byte encoded value to add to Bloom filter.

extend(*self*, *iterable*)

Add an iterable of byte values to the bloom filter.

Parameters **iterable** (*Iterable[bytes]*) – Iterable of byte values.

classmethod from_iterable(*cls*, *iterable*)

Instantiate a bloom filter from a given iterable.

Parameters **iterable** (*Iterable[bytes]*) – Iterable of byte values.

Returns Instantiated BloomFilter.

Return type *BloomFilter*

__contains__(*self*, *value*)

__index__(*self*)

_combine(*self*, *other*)

__or__(*self*, *other*)

__add__(*self*, *other*)

_icombine(*self*, *other*)

__ior__(*self*, *other*)

__iadd__(*self*, *other*)

static get_bloom_bits(*value*)

Bloom filter helper function. Get the Bloom bits of a given value.

Parameters **value** (*bytes*) – Value to be encoded into the Bloom filter.

static get_chunks_for_bloom(*value_hash*)

Bloom filter helper function. Turn a value hash into a series of chunks.

Parameters **value_hash** (*bytes*) – Hash of to be encoded into the Bloom filter.

Yields *chunk (bytes)* – Chunks of the value hash.

static chunk_to_bloom_bits(*chunk*)

Bloom filter helper function. Turn a chunk into a series of actual bytes.

Parameters **chunk** (*bytes*) – Byte encoded chunk.

polyswarmclient.ethereum.offersclient

Module Contents

polyswarmclient.ethereum.offersclient.**logger**

class polyswarmclient.ethereum.offersclient.**OffersClient**(*client*)
Bases: object

OffersClient to handle offers. Presently stores a given client and parameters.

polyswarmclient.ethereum.relayclient

Module Contents

polyswarmclient.ethereum.relayclient.**logger**

class polyswarmclient.ethereum.relayclient.**RelayDepositTransaction**(*client*, *amount*)
Bases: *polyswarmclient.ethereum.transaction.EthereumTransaction*

get_path(*self*)

get_body(*self*)

has_required_event(*self*, *transaction_events*)

class polyswarmclient.ethereum.relayclient.**RelayWithdrawTransaction**(*client*, *amount*)
Bases: *polyswarmclient.ethereum.transaction.EthereumTransaction*

get_path(*self*)

get_body(*self*)

has_required_event(*self*, *transaction_events*)

class polyswarmclient.ethereum.relayclient.**RelayClient**(*client*)
Bases: object

polyswarmclient.ethereum.stakingclient

Module Contents

polyswarmclient.ethereum.stakingclient.**logger**

class polyswarmclient.ethereum.stakingclient.**StakeDepositTransaction**(*client*, *amount*)
Bases: *polyswarmclient.ethereum.transaction.EthereumTransaction*

get_path(*self*)

get_body(*self*)

has_required_event(*self*, *transaction_events*)

class polyswarmclient.ethereum.stakingclient.**StakeWithdrawTransaction**(*client*, *amount*)

Bases: *polyswarmclient.ethereum.transaction.EthereumTransaction*

get_path(*self*)

get_body(*self*)

has_required_event(*self*, *transaction_events*)

class polyswarmclient.ethereum.stakingclient.**StakingClient**(*client*)

Bases: object

polyswarmclient.ethereum.verifiers

Module Contents

polyswarmclient.ethereum.verifiers.**logger**

polyswarmclient.ethereum.verifiers.**UNKNOWN_PARAMETER = XXX**

class polyswarmclient.ethereum.verifiers.**DecodedTransaction**(*to*, *value*, *data*, *abi*, *signature*, *parameters*)

This is a decoded representation of the transaction object returned by polyswarmd.

classmethod from_transaction(*cls*, *transaction*, *abi*)

Parse a transaction from data returned from polyswarmd.

Parameters

- **transaction** (*dict*) – Transaction to be simplified
- **abi** (*str, list[str]*) – ABI of the expected function call

Returns If valid, returns a SimplifiedTransaction

Return type *DecodedTransaction*

Raises **ValueError** – If invalid transaction is provided

__repr__(*self*)

class polyswarmclient.ethereum.verifiers.**AbstractTransactionVerifier**(*parameters*)

Verifier is used to verify the details of a single transaction.

ABI = ['', []]

verify(*self*, *transaction*)

Called when a list of transactions were returned from polyswarmd. This function will verify the transactions, and determines if the transactions are expected.

Parameters **transaction** – Transaction representation returned from polyswarmd

Returns True if valid and expected

__repr__(*self*)

class polyswarmclient.ethereum.verifiers.**NctApproveVerifier**(*amount*)

Bases: *polyswarmclient.ethereum.verifiers.AbstractTransactionVerifier*

ABI = ['approve', ['address', 'uint256']]

verify(*self*, *transaction*)

class polyswarmclient.ethereum.verifiers.**NctTransferVerifier**(*amount*)
Bases: *polyswarmclient.ethereum.verifiers.AbstractTransactionVerifier*

ABI = ['transfer', ['address', 'uint256']]

verify(*self*, *transaction*)

class polyswarmclient.ethereum.verifiers.**PostBountyVerifier**(*artifact_type*, *amount*, *artifact_uri*, *num_artifacts*, *duration*, *bloom*, *metadata*)
Bases: *polyswarmclient.ethereum.verifiers.AbstractTransactionVerifier*

ABI = ['postBounty', ['uint128', 'uint256', 'uint256', 'string', 'uint256', 'uint256',

verify(*self*, *transaction*)

class polyswarmclient.ethereum.verifiers.**PostAssertionVerifier**(*bounty_guid*, *bid*, *mask*, *commitment*)
Bases: *polyswarmclient.ethereum.verifiers.AbstractTransactionVerifier*

ABI = ['postAssertion', ['uint128', 'uint256[]', 'uint256', 'uint256']]

verify(*self*, *transaction*)

class polyswarmclient.ethereum.verifiers.**RevealAssertionVerifier**(*bounty_guid*, *index*, *nonce*, *verdicts*, *metadata*)
Bases: *polyswarmclient.ethereum.verifiers.AbstractTransactionVerifier*

ABI = ['revealAssertion', ['uint128', 'uint256', 'uint256', 'uint256', 'string']]

verify(*self*, *transaction*)

class polyswarmclient.ethereum.verifiers.**PostVoteVerifier**(*bounty_guid*, *votes*, *valid_bloom*)
Bases: *polyswarmclient.ethereum.verifiers.AbstractTransactionVerifier*

ABI = ['voteOnBounty', ['uint128', 'uint256', 'bool']]

verify(*self*, *transaction*)

class polyswarmclient.ethereum.verifiers.**SettleBountyVerifier**(*bounty_guid*)
Bases: *polyswarmclient.ethereum.verifiers.AbstractTransactionVerifier*

ABI = ['settleBounty', ['uint128']]

verify(*self*, *transaction*)

class polyswarmclient.ethereum.verifiers.**StakingDepositVerifier**(*amount*)
Bases: *polyswarmclient.ethereum.verifiers.AbstractTransactionVerifier*

ABI = ['deposit', ['uint256']]

verify(*self*, *transaction*)

class polyswarmclient.ethereum.verifiers.**StakingWithdrawVerifier**(*amount*)
Bases: *polyswarmclient.ethereum.verifiers.AbstractTransactionVerifier*

ABI = ['withdraw', ['uint256']]

verify(*self*, *transaction*)

Package Contents

class polyswarmclient.ethereum.**BalanceClient**(*client*)
Bases: object

class polyswarmclient.ethereum.**BountiesClient**(*client*)
Bases: object

class polyswarmclient.ethereum.**OffersClient**(*client*)
Bases: object

OffersClient to handle offers. Presently stores a given client and parameters.

class polyswarmclient.ethereum.**RelayClient**(*client*)
Bases: object

class polyswarmclient.ethereum.**StakingClient**(*client*)
Bases: object

polyswarmclient.fast

Submodules

polyswarmclient.fast.balanceclient

Module Contents

polyswarmclient.fast.balanceclient.**logger**

polyswarmclient.fast.balanceclient.**MAX_TRIES**

polyswarmclient.fast.balanceclient.**TTL**

polyswarmclient.fast.balanceclient.**MAX_SIZE = 20**

class polyswarmclient.fast.balanceclient.**BalanceClient**(*client*)
Bases: object

polyswarmclient.fast.bountiesclient

Module Contents

polyswarmclient.fast.bountiesclient.**logger**

polyswarmclient.fast.bountiesclient.**DEFAULT_METADATA = {"malware_family": ""}**

class polyswarmclient.fast.bountiesclient.**PostBountyTransactionRequest**(*client*, *guid*, *reward*, *artifact_uri*, *artifact_type*, *duration*, *metadata*)

Bases: *polyswarmclient.fast.transaction.PolySwarmTransactionRequest*

path

class polyswarmclient.fast.bountiesclient.**PostAssertionTransactionRequest**(*client*, *bounty_guid*, *bid*, *verdict*, *metadata*)

Bases: *polyswarmclient.fast.transaction.PolySwarmTransactionRequest*

path

class polyswarmclient.fast.bountiesclient.**PostVoteTransactionRequest**(*client*, *bounty_guid*, *vote*)

Bases: *polyswarmclient.fast.transaction.PolySwarmTransactionRequest*

path

class polyswarmclient.fast.bountiesclient.**BountiesClient**(*client*)

Bases: object

polyswarmclient.fast.offersclient

Module Contents

polyswarmclient.fast.offersclient.**logger**

class polyswarmclient.fast.offersclient.**OffersClient**(*client*)

Bases: object

OffersClient to handle offers. Presently stores a given client and parameters.

polyswarmclient.fast.relayclient

Module Contents

polyswarmclient.fast.relayclient.**logger**

class polyswarmclient.fast.relayclient.**RelayDepositTransaction**(*client*, *amount*)

Bases: *polyswarmclient.ethereum.transaction.EthereumTransaction*

get_path(*self*)

get_body(*self*)

has_required_event(*self*, *transaction_events*)

class polyswarmclient.fast.relayclient.**RelayWithdrawTransactionRequest**(*client*, *amount*)

Bases: *polyswarmclient.fast.transaction.PolySwarmTransactionRequest*

path

class polyswarmclient.fast.relayclient.**RelayClient**(*client*)

Bases: object

polyswarmclient.fast.stakingclient

Module Contents

polyswarmclient.fast.stakingclient.**logger**

class polyswarmclient.fast.stakingclient.**StakingClient**(*client*)

Bases: object

polyswarmclient.fast.transaction

Module Contents

polyswarmclient.fast.transaction.**logger**

class polyswarmclient.fast.transaction.**PolySwarmTransactionRequest**(*client: Client*, *transaction: Transaction*)

Used to verify and post groups of transactions that make up a specific action.

For instance, when approving some funds to move, and calling a contract function that will consumer them.

path

Get the path of the route to build this transaction

Returns Polyswarmd path to get the transaction data

Return type str

sign_transaction(*self*)

Signs a transaction

Returns SignedTransaction

Package Contents

class polyswarmclient.fast.**BalanceClient**(*client*)

Bases: object

class polyswarmclient.fast.**BountiesClient**(*client*)
Bases: object

class polyswarmclient.fast.**OffersClient**(*client*)
Bases: object

OffersClient to handle offers. Presently stores a given client and parameters.

class polyswarmclient.fast.**RelayClient**(*client*)
Bases: object

class polyswarmclient.fast.**StakingClient**(*client*)
Bases: object

polyswarmclient.filters

Submodules

polyswarmclient.filters.bountyfilter

Module Contents

polyswarmclient.filters.bountyfilter.**logger**

polyswarmclient.filters.bountyfilter.**split_filter**(*ctx*, *param*, *value*)
Split some accept or exlcude arg from `key:value` to a tuple

Parameters

- **ctx** –
- **param** –
- **value** – list of exclude or accept values

Returns list[tuple] list of excludelaccept values as tuple key, value

class polyswarmclient.filters.bountyfilter.**BountyFilter**(*accept*, *reject*)
Bases: *polyswarmclient.filters.filter.MetadataFilter*

Takes two objects list[Filter], accept and reject These dicts are used to filter metadata json blobs. Each filter runs against given metadata, and is used to determine if this participant will respond to a bounty

is_allowed(*self*, *metadata*)
Check metadata against the accept and exclude filters, returning True if it passes all checks

Parameters **metadata** (*dict*) – metadata dict to test

Returns True if meets the conditions and passes the filter

Return type (bool)

polyswarmclient.filters.confidencefilter

Module Contents

polyswarmclient.filters.confidencefilter.**logger**

class polyswarmclient.filters.confidencefilter.**ConfidenceModifier**(*favor*, *penalize*)

Bases: *polyswarmclient.filters.filter.MetadataFilter*

modify(*self*, *metadata*, *confidence*)

Check metadata against the penalty and favor filters. Matching both bonus and penalty results offset

Parameters

- **metadata** (*any*) – metadata dict to test
- **confidence** (*float*) – confidence as returned by the Av engine

Returns confidence that is either more, same or less after comparing against bonus/penalty Filters

Return type (float)

polyswarmclient.filters.filter

Module Contents

polyswarmclient.filters.filter.**logger**

polyswarmclient.filters.filter.**parse_filters**(*ctx*, *param*, *value*)

Split some filters into a dict separated by type

Parameters

- **ctx** –
- **param** –
- **value** – list of 4 string tuples

Returns Dict where each key points to a list of Filters

Return type dict

class polyswarmclient.filters.filter.**FilterComparison**

Bases: enum.Enum

Enum of supported metadata comparisons

LT = <

LTE = <=

EQ = ==

GTE = >=

GT = >

CONTAINS = contains

STARTS_WITH = startswith

ENDS_WITH = endswith

REGEX = regex

__repr__(*self*)

static from_string(*value*)

class polyswarmclient.filters.filter.**Filter**(*key*, *comparison*, *target_value*)
Filter some metadata value

__eq__(*self*, *other*)

__repr__(*self*)

number_check(*self*, *value*)
Check a value as a number with GT, GTE, LT, or LTE comparisons

Parameters **value** (*str|int|float|bytes*) – Value to compare against

Returns: (bool) returns True if comparison matches

string_check(*self*, *value*)
Check a value as a string with EQ, CONTAINS, STARTS_WITH, ENDS_WITH, and REGEX comparisons

Parameters **value** (*str|int|float|bytes*) – Value to compare against

Returns: (bool) returns True if comparison matches

filter(*self*, *metadata*)
Take some metadata, and matches the given key against the target_value and comparison operator for this filter

Parameters **metadata** (*dict*) – Dict of k-v metadata values

Returns: (bool) True if it matches the filter

class polyswarmclient.filters.filter.**MetadataFilter**
Takes two objects list[Filter], accept and reject These dicts are used to filter metadata json blobs. Each filter runs against given metadata, and is used to determine if this participant will respond to a bounty

static pad_metadata(*metadata*, *min_length*)
Pad out the metadata list with None values to match a given length

Parameters

- **metadata** (*list[dict|None]*) – List of metadata dicts
- **min_length** (*int*) – Min size for the metadata list after padding

Returns list of metadata dicts, or None values

polyswarmclient.liveness

Submodules

polyswarmclient.liveness.exceptions

Module Contents

exception polyswarmclient.liveness.exceptions.**LivenessReadError**
Bases: *polyswarmclient.exceptions.PolyswarmClientException*

polyswarmclient.liveness.liveness

Module Contents

polyswarmclient.liveness.liveness.**logger**

class polyswarmclient.liveness.liveness.**Liveness**

last_iteration :int

average_wait :int

class polyswarmclient.liveness.liveness.**LivenessCheck**(*loop_iteration_threshold=5*, *average_wait_threshold=10*)

Bases: abc.ABC

get_liveness(*self*)

check(*self*)

Determine if participant is running smoothly, based on given inputs

class polyswarmclient.liveness.liveness.**Task**(*recorder*, *key*, *start_time*)

class polyswarmclient.liveness.liveness.**LivenessRecorder**

Bases: abc.ABC

waiting_task(*self*, *key*, *start_time*)

Add waiting task, but remove when done

Use: async with liveness.waiting_task(key, start_time):

Parameters

- **key** – task key
- **start_time** – start time, in any units (either block number or time)

polyswarmclient.liveness.local

Module Contents

polyswarmclient.liveness.local.**logger**

class polyswarmclient.liveness.local.**FileLock**(*fileno*)

Locks a file so that only LivelinessRecorder or LivelinessChecker can access at any moment

acquire(*self*)

release(*self*)

acquire_unix(*self*)

acquire_windows(*self*)

release_unix(*self*)

__enter__(*self*)

__exit__(*self*, *exc_type*, *exc_val*, *exc_tb*)

class polyswarmclient.liveness.local.**LocalLivenessCheck**(*loop_iteration_threshold=5*, *average_wait_threshold=10*)

Bases: *polyswarmclient.liveness.liveness.LivenessCheck*

Checks the liveness by reading a tempfile which should contain liveness information

get_liveness(*self*)

get_average_task_wait(*self*)

class polyswarmclient.liveness.local.**LocalLivenessRecorder**
Bases: *polyswarmclient.liveness.liveness.LivenessRecorder*

Record liveness data in a tempfile

write_sync(*self*, *content*)
Write the given content to the file at the given path.

Parameters **content** – content to write into the file

polyswarmclient.producer

Submodules

polyswarmclient.producer.base

Module Contents

polyswarmclient.producer.base.**logger**

polyswarmclient.producer.base.**WAIT_TIME = 20**

polyswarmclient.producer.base.**KEY_TIMEOUT = 10**

polyswarmclient.producer.base.**JOB_RESULTS_FORMAT = {}_{}_{}_results**

class polyswarmclient.producer.base.**Producer**(*client*, *redis_uri*, *queue*, *time_to_post*, *bounty_filter=None*, *confidence_modifier=None*, *rate_limit=None*)

polyswarmclient.producer.job

Module Contents

class polyswarmclient.producer.job.**JobRequest**

polyswarmd_uri :str

guid :str

index :int

uri :str

artifact_type :int

duration :int

metadata :Optional[Dict[str, Any]]

chain :str

ts :int

key

is_expired(*self*, *now=None*)

get_artifact_type(*self*)

asdict(*self*)

class polyswarmclient.producer.job.**JobResponse**

index :int

bit :bool

verdict :bool

confidence :float

metadata :str

asdict(*self*)

polyswarmclient.producer.jobprocessor

Module Contents

polyswarmclient.producer.jobprocessor.**logger**

class polyswarmclient.producer.jobprocessor.**PendingJob**(*key: str, jobs: List[JobRequest], future: Future*)

A wrapper around a list of Jobs that are processing in the backend

key :str

jobs :List[JobRequest]

results :Dict[int, ScanResult]

future :Future

times(*self*)

time_ratios(*self*)

__store_job_response(*self, response: JobResponse, confidence_modifier: Optional[ConfidenceModifier]*)

Converts a JobResponse to ScanResult with modified confidence. Stores at the correct index in internal results

Parameters

- **response** – JobResponse to conver
- **confidence_modifier** – an optional ConfidenceModifier to potentially change the confidence

Returns

is_done(*self*)

Checks all things to see if it is done :return: true if expired, or has all results

__is_expired(*self*)

Returns true if any of the jobs are expired

__has_all_results(*self*)
Returns true if all the jobs have a result

__finish(*self*)
Set the results in the future and mark done

class polyswarmclient.producer.jobprocessor.**JobProcessor**(*redis: Redis, queue: str, confidence_modifier: Optional[ConfidenceModifier], period: float = 0.25, redis_error_callback: Optional[Callable[[], Coroutine]] = None*)
Keeps track pending jobs, and polls the PendingJob results every period of time (.5 seconds)

redis_uri :str

confidence_modifier :Optional[ConfidenceModifier]

queue :str

period :float

pending_jobs :Dict[str, PendingJob]

job_lock :Optional[asyncio.Lock]

redis :Optional[Redis]

task

reset_callback

stop(*self*)
Stop processing jobs

Package Contents

class polyswarmclient.producer.**Producer**(*client, redis_uri, queue, time_to_post, bounty_filter=None, confidence_modifier=None, rate_limit=None*)

polyswarmclient.producer.**JOB_RESULTS_FORMAT = {}_{}_{}_results**

class polyswarmclient.producer.**JobRequest**

polyswarmd_uri :str

guid :str

index :int

uri :str

artifact_type :int

duration :int

metadata :Optional[Dict[str, Any]]

chain :str

ts :int

key

is_expired(*self*, *now=None*)

get_artifact_type(*self*)

asdict(*self*)

class polyswarmclient.producer.**JobResponse**

index :int

bit :bool

verdict :bool

confidence :float

metadata :str

asdict(*self*)

class polyswarmclient.producer.**JobProcessor**(*redis: Redis, queue: str, confidence_modifier: Optional[ConfidenceModifier], period: float = 0.25, redis_error_callback: Optional[Callable[[], Coroutine]] = None*)

Keeps track pending jobs, and polls the PendingJob results every period of time (.5 seconds)

redis_uri :str

confidence_modifier :Optional[ConfidenceModifier]

queue :str

period :float

pending_jobs :Dict[str, PendingJob]

job_lock :Optional[asyncio.Lock]

redis :Optional[Redis]

task

reset_callback

stop(*self*)

Stop processing jobs

class polyswarmclient.producer.**PendingJob**(*key: str, jobs: List[JobRequest], future: Future*)

A wrapper around a list of Jobs that are processing in the backend

key :str

jobs :List[JobRequest]

results :Dict[int, ScanResult]

future :Future

times(*self*)

time_ratios(*self*)

__store_job_response(*self, response: JobResponse, confidence_modifier: Optional[ConfidenceModifier]*)
Converts a JobResponse to ScanResult with modified confidence. Stores at the correct index in internal results

Parameters

- **response** – JobResponse to conver
- **confidence_modifier** – an optional ConfidenceModifier to potentially change the confidence

Returns

is_done(*self*)
Checks all things to see if it is done :return: true if expired, or has all results

__is_expired(*self*)
Returns true if any of the jobs are expired

__has_all_results(*self*)
Returns true if all the jobs have a result

__finish(*self*)
Set the results in the future and mark done

polyswarmclient.ratelimit

Submodules

polyswarmclient.ratelimit.abstractratelimit

Module Contents

class polyswarmclient.ratelimit.abstractratelimit.**AbstractRateLimit**
Bases: abc.ABC

Abstract class for building a limit for scans, based on a third party requirement (Such as api key limit for a vendor) Allows different implementations

polyswarmclient.ratelimit.redis

Module Contents

polyswarmclient.ratelimit.redis.**logger**

polyswarmclient.ratelimit.redis.**SECONDS = 1**

polyswarmclient.ratelimit.redis.**MINUTELY_SECONDS = 60**

polyswarmclient.ratelimit.redis.**HOURLY_SECONDS**

polyswarmclient.ratelimit.redis.**DAILY_SECONDS**

class polyswarmclient.ratelimit.redis.**RedisKeyManager**
Bases: abc.ABC

get_key(*self, prefix*)

get_expiration(*self*)

class polyswarmclient.ratelimit.redis.**DateKeyManager**(*expiration*, *format_str*)
Bases: *polyswarmclient.ratelimit.redis.RedisKeyManager*

get_key(*self*, *prefix*)

get_expiration(*self*)

class polyswarmclient.ratelimit.redis.**DailyKeyManager**
Bases: *polyswarmclient.ratelimit.redis.DateKeyManager*

class polyswarmclient.ratelimit.redis.**HourlyKeyManager**
Bases: *polyswarmclient.ratelimit.redis.DateKeyManager*

class polyswarmclient.ratelimit.redis.**MinutelyKeyManager**
Bases: *polyswarmclient.ratelimit.redis.DateKeyManager*

class polyswarmclient.ratelimit.redis.**SecondlyKeyManager**
Bases: *polyswarmclient.ratelimit.redis.DateKeyManager*

class polyswarmclient.ratelimit.redis.**RedisRateLimit**(*redis*, *queue*, *limit*, *key_manager=None*)
Bases: *polyswarmclient.ratelimit.abstractratelimit.AbstractRateLimit*

Third Party limitation where redis is used to track a daily scan limit. Keys are based on the current date, and will expire the next day.

This implementation is used in the producer and worker since they use Redis already.

key

set_redis(*self*, *redis*)

class polyswarmclient.ratelimit.redis.**RedisDailyRateLimit**
Bases: *polyswarmclient.ratelimit.redis.RedisRateLimit*

1.6.2 Submodules

polyswarmclient.abstractambassador

Module Contents

polyswarmclient.abstractambassador.**logger**

polyswarmclient.abstractambassador.**BOUNTY_QUEUE_SIZE**

polyswarmclient.abstractambassador.**MAX_BOUNTIES_IN_FLIGHT**

polyswarmclient.abstractambassador.**MAX_BOUNTIES_PER_BLOCK**

polyswarmclient.abstractambassador.**BLOCK_DIVISOR**

class polyswarmclient.abstractambassador.**QueuedBounty**(*artifact_type*, *amount*, *ipfs_uri*, *duration*, *api_key=None*, *metadata=None*)
Bases: object

__repr__(*self*)

class polyswarmclient.abstractambassador.**AbstractAmbassador**(*client*, *testing=0*, *chains=None*, *watchdog=0*, *submission_rate=0*)

Bases: abc.ABC

classmethod connect(*cls*, *polyswarmd_addr*, *keyfile*, *password*, *api_key=None*, *testing=0*, *chains=None*, *watchdog=0*, *submission_rate=0*)

Connect the Ambassador to a Client.

Parameters

- **polyswarmd_addr** (*str*) – URL of polyswarmd you are referring to.
- **keyfile** (*str*) – Keyfile filename.
- **password** (*str*) – Password associated with Keyfile.
- **api_key** (*str*) – Your PolySwarm API key.
- **testing** (*int*) – Number of testing bounties to use.
- **chains** (*set(str)*) – Set of chains you are acting on.

Returns Ambassador instantiated with a Client.

Return type *AbstractAmbassador*

static generate_metadata(*content*)

Generate a bunch of metadata for a given bytestream from a file

Parameters **content** – bytes-like object (or string)

Returns dictionary of metadata about a file

run(*self*)

Run the Client on all of our chains.

polyswarmclient.abstractarbiter

Module Contents

polyswarmclient.abstractarbiter.**logger**

class polyswarmclient.abstractarbiter.**AbstractArbiter**(*client*, *testing=0*, *scanner=None*, *chains=None*, *artifact_types=None*)

Bases: object

classmethod connect(*cls*, *polyswarmd_addr*, *keyfile*, *password*, *api_key=None*, *testing=0*, *scanner=None*, *chains=None*, *artifact_types=None*)

Connect the Arbiter to a Client.

Parameters

- **polyswarmd_addr** (*str*) – URL of polyswarmd you are referring to.
- **keyfile** (*str*) – Keyfile filename.
- **password** (*str*) – Password associated with Keyfile.
- **api_key** (*str*) – Your PolySwarm API key.
- **testing** (*int*) – Number of testing bounties to use.

- **scanner** (AbstractScanner) – Scanner for scanning artifacts
- **chains** (*set (str)*) – Set of chains you are acting on.
- **artifact_types** (*list (ArtifactType)*) – List of artifact types you support

Returns Arbiter instantiated with a Client.

Return type *AbstractArbiter*

run (*self*)
Run the Client on the Arbiter's chains.

polyswarmclient.abstractmicroengine

Module Contents

polyswarmclient.abstractmicroengine.**logger**

class polyswarmclient.abstractmicroengine.**AbstractMicroengine**(*client*, *testing=0*, *scanner=None*, *chains=None*, *artifact_types=None*, *bid_strategy=None*, *bounty_filter=BountyFilter(None, None)*, *confidence_modifier=ConfidenceModifier(None, None)*)

Bases: object

classmethod connect(*cls*, *polyswarmd_addr*, *keyfile*, *password*, *api_key=None*, *testing=0*, *scanner=None*, *chains=None*, *artifact_types=None*, *bid_strategy=None*, *bounty_filter=BountyFilter(None, None)*, *confidence_modifier=ConfidenceModifier(None, None)*)
Connect the Microengine to a Client.

Parameters

- **polyswarmd_addr** (*str*) – URL of polyswarmd you are referring to.
- **keyfile** (*str*) – Keyfile filename.
- **password** (*str*) – Password associated with Keyfile.
- **api_key** (*str*) – Your PolySwarm API key.
- **testing** (*int*) – Number of testing bounties to use.
- **scanner** (Scanner) – *Scanner* instance to use.
- **chains** (*set (str)*) – Set of chains you are acting on.
- **artifact_types** (*list (ArtifactType)*) – List of artifact types you support
- **bid_strategy** (BidStrategyBase) – Bid Strategy for bounties
- **bounty_filter** (BountyFilter) – Filters to accept/reject artifacts
- **confidence_modifier** (ConfidenceModifier) – Filters to adjust confidence based on metadata

Returns Microengine instantiated with a Client.

Return type *AbstractMicroengine*

run(*self*)
Run the *Client* on the Microengine's chains.

polyswarmclient.abstractscanner

Module Contents

polyswarmclient.abstractscanner.**logger**

class polyswarmclient.abstractscanner.**ScanResult**(*bit=False*, *verdict=False*, *confidence=1.0*, *metadata=Verdict().set_malware_family('').json()*)
Bases: object

Results from scanning one artifact

__repr__(*self*)

class polyswarmclient.abstractscanner.**ScanMode**
Bases: enum.Enum

Denote whether the Scanner is using asynchronous or synchronous scan

SYNC = 0

ASYNC = 1

class polyswarmclient.abstractscanner.**AbstractScanner**(*mode: ScanMode = ScanMode.ASYNC*)
Base *Scanner* class. To be overwritten with other scanning logic.

This class offers two scan options, which can be specified by passing a ScanMode enum value as *mode*. It uses asynchronous scan by default.

The function *scan_async* is a coroutine function where everything called from this function must be async compatible, That means it uses only non-blocking IO, and runs nothing cpu-bound, like hash functions.

The function *scan_sync* is a synchronous function where anything goes. It is called in a ThreadPoolExecutor so it is compatible with the worker that uses asyncio.

Overwriting *scan* directly is deprecated.

get_executor(*self*)

scan_sync(*self*, *guid*, *artifact_type*, *content*, *metadata*, *chain*)
Override this to implement custom synchronous scanning logic

Parameters

- **guid** (*str*) – GUID of the bounty under analysis, use to track artifacts in the same bounty
- **artifact_type** (*ArtifactType*) – Artifact type for the bounty being scanned
- **content** (*bytes*) – Content of the artifact to scan
- **metadata** (*dict*) – Metadata dict from the ambassador
- **chain** (*str*) – What chain are we operating on

Returns Result of this scan

Return type *ScanResult*

polyswarmclient.backoff_wrapper

Module Contents

polyswarmclient.backoff_wrapper.**logger**

class polyswarmclient.backoff_wrapper.**BackoffWrapper**(*func*, ***kwargs*)
Uses a generator to create backoff times

This is for use in functions that don't return. In our use case, listen_for_events is not supposed to return, but exists as a long running Task. Using the decorator, the backoff time grows forever, because the function is only called one time. So each error causes a longer and longer timeout.

This adds an ability to reset the backoff, so your long running function can reduce the timeout after success

reset(*self*)

polyswarmclient.bidstrategy

Module Contents

class polyswarmclient.bidstrategy.**BidStrategyBase**(*min_bid_multiplier=None*, *max_bid_multiplier=None*)

polyswarmclient.config

Module Contents

polyswarmclient.config.**logger**

polyswarmclient.config.**validate_apikey**(*ctx*, *param*, *value*)
Validates the API key passed in through click parameters

polyswarmclient.config.**init_logging**(*loggers*, *log_format*, *loglevel=logging.WARNING*)
Logic to support JSON logging.

class polyswarmclient.config.**LoggerConfig**(*loggers*, *log_format*, *log_level=logging.WARNING*)

LEVELS

configure(*self*)

set_level(*self*, *new_level*)

polyswarmclient.corpus

Module Contents

polyswarmclient.corpus.**logger**

polyswarmclient.corpus.**MALICIOUS_BOOTSTRAP_URL**

polyswarmclient.corpus.**ARCHIVE_PASSWORD**

class polyswarmclient.corpus.**DownloadToFileSystemCorpus**(*base_dir=None*)
Bases: object

mal_path

benign_path

download_and_unpack(*self*)

_get_pth_listing(*self*, *p*)

get_malicious_file_list(*self*)

get_benign_file_list(*self*)

download_truth(*self*)

polyswarmclient.events

Module Contents

polyswarmclient.events.**logger**

class polyswarmclient.events.**Callback**
Bases: object

Abstract callback class which is the parent to a number of child callback classes to be used in different scenarios.

Note: Classes which extend *Callback* are expected to impliment the *run* method.

register(*self*, *f*)
Register a function to this Callback.

Parameters **f** (*function*) – Function to register.

remove(*self*, *f*)
Remove a function previously assigned to this Callback.

Parameters **f** (*function*) – Function to remove.

class polyswarmclient.events.**OnRunCallback**
Bases: *polyswarmclient.events.Callback*

Called upon entering the event loop for the first time, use for initialization

class polyswarmclient.events.**OnStopCallback**
Bases: *polyswarmclient.events.Callback*

Called when the client is stopping

This can happen on errors, or due to a signal

class polyswarmclient.events.**OnNewBlockCallback**
Bases: *polyswarmclient.events.Callback*

Called upon receiving a new block, scheduled events triggered separately

class polyswarmclient.events.**OnNewBountyCallback**
Bases: *polyswarmclient.events.Callback*

Called upon receiving a new bounty

class polyswarmclient.events.**OnNewAssertionCallback**
Bases: *polyswarmclient.events.Callback*

Called upon receiving a new assertion

class polyswarmclient.events.**OnRevealAssertionCallback**
Bases: *polyswarmclient.events.Callback*

Called upon receiving a new assertion reveal

class polyswarmclient.events.**OnNewVoteCallback**
Bases: *polyswarmclient.events.Callback*

Called upon receiving a new arbiter vote

class polyswarmclient.events.**OnQuorumReachedCallback**
Bases: *polyswarmclient.events.Callback*

Called upon a bounty reaching quorum

class polyswarmclient.events.**OnSettledBountyCallback**
Bases: *polyswarmclient.events.Callback*

Called upon a bounty being settled

class polyswarmclient.events.**OnDeprecatedCallback**
Bases: *polyswarmclient.events.Callback*

Called upon the BountyRegistry contract being deprecated

class polyswarmclient.events.**OnInitializedChannelCallback**
Bases: *polyswarmclient.events.Callback*

Called upon a channel being initialized

class polyswarmclient.events.**Schedule**
Bases: object

Generic Schedule class. Uses a PriorityQueue data structure to store Events.

empty(*self*)
Return True if the queue is empty.

Returns Is the queue empty.

Return type boolean

peek(*self*)
Return True if the queue is empty.

Returns Tuple at the front of the queue if the queue is full, else *None*.

Return type (block, event)

get(*self*)
Pop the lowest valued block in the queue.

Returns The lowest valued block in the PriorityQueue.

Return type (block, event)

put(*self*, *block*, *event*)
Add a tuple (block, event) to the PriorityQueue. Block signifies the priority of the event.

class polyswarmclient.events.**Event**(*guid*)
Bases: object

Generic Event class. Stores GUID and can compare for equality and order Events.

Parameters **guid** (*str, None*) – GUID of the event.

__eq__(*self*, *other*)

__lt__ (*self*, *other*)

class polyswarmclient.events.**RevealAssertion**(*guid*, *index*, *nonce*, *verdicts*, *metadata*)
Bases: *polyswarmclient.events.Event*

An assertion scheduled to be publically revealed

Parameters

- **guid** (*str*) – GUID of the bounty being asserted on
- **index** (*int*) – Index of the assertion to reveal
- **nonce** (*str*) – Secret nonce used to reveal assertion
- **verdicts** (*List[bool]*) – List of verdicts for each artifact in the bounty
- **metadata** (*str*) – Optional metadata

class polyswarmclient.events.**OnRevealAssertionDueCallback**
Bases: *polyswarmclient.events.Callback*

Called when an assertion is needing to be revealed

class polyswarmclient.events.**VoteOnBounty**(*guid*, *votes*, *valid_bloom*)
Bases: *polyswarmclient.events.Event*

A scheduled vote from an arbiter :param guid: GUID of the bounty being voted on :type guid: str :param votes: List of votes for each artifact in the bounty :type votes: List[bool] :param valid_bloom: Is the bloom filter submitted with the bounty valid :type valid_bloom: bool

class polyswarmclient.events.**OnVoteOnBountyDueCallback**
Bases: *polyswarmclient.events.Callback*

Called when a bounty is needing to be voted on

class polyswarmclient.events.**SettleBounty**(*guid*)
Bases: *polyswarmclient.events.Event*

A bounty scheduled to be settled :param guid: GUID of the bounty being asserted on :type guid: str

class polyswarmclient.events.**OnSettleBountyDueCallback**
Bases: *polyswarmclient.events.Callback*

Called when a bounty is needing to be settled

class polyswarmclient.events.**WithdrawStake**(*amount*)
Bases: *polyswarmclient.events.Event*

A scheduled stake withdrawal from an arbiter

Parameters **amount** (*int*) – Amount to withdraw from stake

class polyswarmclient.events.**OnWithdrawStakeDueCallback**
Bases: *polyswarmclient.events.Callback*

Called when a an arbiter needs to withdraw stake (due to deprecation)

polyswarmclient.exceptions

Module Contents

exception polyswarmclient.exceptions.**PolyswarmClientException**
Bases: Exception

polyswarm-client related errors

exception polyswarmclient.exceptions.**ApiKeyException**
Bases: *polyswarmclient.exceptions.PolyswarmClientException*

Used an API key when not communicating over https, without explicitly allowing

exception polyswarmclient.exceptions.**InvalidBidError**
Bases: *polyswarmclient.exceptions.PolyswarmClientException*

Fault in bid logic that resulted in a bid that is not between the min and max values provided by polyswarmd

exception polyswarmclient.exceptions.**LowBalanceError**
Bases: *polyswarmclient.exceptions.PolyswarmClientException*

Not enough NCT to complete the requested action

exception polyswarmclient.exceptions.**TransactionError**
Bases: *polyswarmclient.exceptions.PolyswarmClientException*

A transaction failed

exception polyswarmclient.exceptions.**InvalidMetadataError**
Bases: *polyswarmclient.exceptions.PolyswarmClientException*

Metadata does not match the valid schema

exception polyswarmclient.exceptions.**RateLimitedError**
Bases: *polyswarmclient.exceptions.PolyswarmClientException*

Hit the rate limit from polyswarmd

exception polyswarmclient.exceptions.**NonceDesyncError**
Bases: *polyswarmclient.exceptions.PolyswarmClientException*

Got a nonce too low or too high error

exception polyswarmclient.exceptions.**ReceiptError**
Bases: *polyswarmclient.exceptions.PolyswarmClientException*

Failed to get receipt from polyswarmd

exception polyswarmclient.exceptions.**UnsupportedHashError**
Bases: *polyswarmclient.exceptions.PolyswarmClientException*

Raised when a hash doesn't match the format of a hash we use

exception polyswarmclient.exceptions.**ScannerSetupFailedError**
Bases: *polyswarmclient.exceptions.PolyswarmClientException*

Scanner reported an unsuccessful setup

class polyswarmclient.exceptions.**FatalError**(*message=''*, *exit_code=0*)
Bases: click.ClickException

exception polyswarmclient.exceptions.**SecurityWarning**
Bases: Warning

Warnings about disabled security features.

polyswarmclient.log_formatter

Module Contents

class polyswarmclient.log_formatter.**ExtraTextFormatter**
Bases: logging.Formatter

Custom formatter that adds extra fields to the message string

format(*self*, *record*)
Takes a LogRecord and sets record.message = record.msg % record.args This one does some extra work. It searches the record dict for some extra keys we use in the client. (specified as extra= in the logger statement) If it finds one, it grabs the dict and adds an extra %s arg to record.msg, and the dict value to the record.args tuple.

class polyswarmclient.log_formatter.**JSONFormatter**
Bases: pythonjsonlogger.jsonlogger.JsonFormatter

Class to add custom JSON fields to our logger. Presently just adds a timestamp if one isn't present and the log level. INFO: https://github.com/madzak/python-json-logger#customizing-fields

add_fields(*self*, *log_record*, *record*, *message_dict*)

polyswarmclient.parameters

Module Contents

class polyswarmclient.parameters.**Parameters**(*p*)
Bases: object

Trivial wrapper around a dict but protected via a RWLock to allow updates

polyswarmclient.request_rate_limit

Module Contents

polyswarmclient.request_rate_limit.**logger**

polyswarmclient.request_rate_limit.**RATE_LIMIT_SLEEP = 2**

class polyswarmclient.request_rate_limit.**RequestRateLimit**(*event*, *lock*)

polyswarmclient.utils

Module Contents

polyswarmclient.utils.**logger**

polyswarmclient.utils.**TASK_TIMEOUT = 1.0**

polyswarmclient.utils.**MAX_WAIT**

polyswarmclient.utils.**MAX_WORKERS = 4**

polyswarmclient.utils.**DEPRECATION_MESSAGE = polyswarm-client 2.x.x is deprecated. However,**

Action is required to continue using 2.x.x.

If your build is based off the polyswarm-client:latest docker image, you MUST change to another tag. Tag *:2* will include all future fixes for polyswarm-client 2.x.x.

polyswarmclient.utils.**to_bytes**(*value*)

polyswarmclient.utils.**sha3**(*seed*)

polyswarmclient.utils.**int_to_bytes**(*i*)

polyswarmclient.utils.**int_from_bytes**(*b*)

polyswarmclient.utils.**bool_list_to_int**(*bs*)

polyswarmclient.utils.**int_to_bool_list**(*i*, *expected_size*)

polyswarmclient.utils.**guid_as_string**(*guid*)

polyswarmclient.utils.**calculate_commitment**(*account*, *verdicts*, *nonce=None*)

polyswarmclient.utils.**configure_event_loop**()

polyswarmclient.utils.**asyncio_join**()
Gather all remaining tasks, assumes loop is not running

polyswarmclient.utils.**asyncio_stop**()
Stop the main event loop

polyswarmclient.utils.**check_response**(*response*)
Check the status of responses from polyswarmd

Parameters **response** – Response dict parsed from JSON from polyswarmd

Returns True if successful else False

Return type (bool)

polyswarmclient.utils.**is_valid_sha256**(*uri*)
Ensure that a given uri is a valid sha256 hash by checking length, and converting to an int

Parameters **uri** (*str*) – uri to validate

Returns is this valid

Return type bool

polyswarmclient.utils.**is_valid_ipfs_uri**(*ipfs_uri*)
Ensure that a given ipfs_uri is valid by checking length and base58 encoding.

Parameters **ipfs_uri** (*str*) – ipfs_uri to validate

Returns is this valid?

Return type bool

class polyswarmclient.utils.**AsyncArtifactTempfile**(*blob: 'bytes' = None*, *filename: 'str' = None*, *mode: 'str' = 'w+b'*)

asynchronous ctxmgr for temporary artifacts

Notes:

The following underlying file's methods are awaited:

flush, peek, read, seek, write

You can use the object like an ordinary context manager or supply the binary blob to be written as the first argument

```
>>> blob = b'hello world'
>>> async with AsyncArtifactTempfile(blob) as f:
>>>     with open(f.name, 'rb') as of:
```

(continues on next page)

(continued from previous page)

```
>>>         of.read()
b'hello world'
>>> async with AsyncArtifactTempfile() as f:
>>>     await f.write(blob)
>>>     await f.read()
b'hello world'
```

The underlying file is always deleted after ctxmgr exits

__del__(*self*)

__getattr__(*self*, *name*)

polyswarmclient.utils.**return_on_exception**(*exceptions=(Exception,)*, *default=None*)

polyswarmclient.utils.**finalize_polyswarmd_addr**(*polyswarmd_addr*, *api_key*, *allow_key_over_http*, *insecure_transport*)

polyswarmclient.utils.**fill_scheme**(*polyswarmd_addr*, *insecure_transport*)

1.6.3 Package Contents

class polyswarmclient.**BidStrategyBase**(*min_bid_multiplier=None*, *max_bid_multiplier=None*)

class polyswarmclient.**NonceManager**(*client*, *chain*)

Manages the nonce for some Ethereum chain

static find_gaps(*nonces*)

Finds any gaps between base nonce and the last nonce in the given nonces list.

Parameters **nonces** (*list[int]*) – list of nonces being checked

Returns (list[int]): Any missing nonces between base_nonce and the last given nonce

exception polyswarmclient.**RateLimitedError**

Bases: *polyswarmclient.exceptions.PolyswarmClientException*

Hit the rate limit from polyswarmd

class polyswarmclient.**LocalLivenessRecorder**

Bases: *polyswarmclient.liveness.liveness.LivenessRecorder*

Record liveness data in a tempfile

write_sync(*self*, *content*)

Write the given content to the file at the given path.

Parameters **content** – content to write into the file

class polyswarmclient.**BackoffWrapper**(*func*, ***kwargs*)

Uses a generator to create backoff times

This is for use in functions that don't return. In our use case, listen_for_events is not supposed to return, but exists as a long running Task. Using the decorator, the backoff time grows forever, because the function is only called one time. So each error causes a longer and longer timeout.

This adds an ability to reset the backoff, so your long running function can reduce the timeout after success

reset(*self*)

class polyswarmclient.**RequestRateLimit**(*event*, *lock*)

polyswarmclient.**logger**

polyswarmclient.**MAX_ARTIFACTS = 256**

polyswarmclient.**RATE_LIMIT_SLEEP = 2.0**

polyswarmclient.**MAX_BACKOFF = 32**

class polyswarmclient.**Client**(*polyswarmd_addr*, *keyfile*, *password*, *api_key=None*, *tx_error_fatal=False*)

Bases: object

Client to connected to a Ethereum wallet as well as a polyswarmd instance.

Parameters

- **polyswarmd_addr** (*str*) – URI of polyswarmd you are referring to.
- **keyfile** (*str*) – Keyfile filename.
- **password** (*str*) – Password associated with keyfile.
- **api_key** (*str*) – Your PolySwarm API key.
- **tx_error_fatal** (*bool*) – Transaction errors are fatal and exit the program
- **insecure_transport** (*bool*) – Allow insecure transport such as HTTP?

run(*self*, *chains=None*)

Run the main event loop

Parameters **chains** (*set(str)*) – Set of chains to operate on. Defaults to {'home', 'side'}

clear_sub_clients(*self*)

create_ethereum_sub_clients(*self*, *chains*)

create_fast_sub_clients(*self*, *chains*)

static to_wei(*amount*, *unit='ether'*)

static from_wei(*amount*, *unit='ether'*)

static _check_status_for_rate_limit(*status*)

schedule(*self*, *expiration*, *event*, *chain*)

Schedule an event to execute on a particular block

Parameters

- **expiration** (*int*) – Which block to execute on
- **event** (Event) – Event to trigger on expiration block
- **chain** (*str*) – Which chain to operate on

1.7 liveness

1.7.1 Submodules

liveness.__main__

Module Contents

liveness.__main__.**main**(*log*, *log_format*, *loop_update_threshold*, *average_bounty_wait_threshold*)

1.8 microengine

1.8.1 Subpackages

microengine.bidstrategy

Submodules

microengine.bidstrategy.aggressive

Module Contents

class microengine.bidstrategy.aggressive.**BidStrategy**
Bases: *polyswarmclient.BidStrategyBase*

microengine.bidstrategy.conservative

Module Contents

class microengine.bidstrategy.conservative.**BidStrategy**
Bases: *polyswarmclient.BidStrategyBase*

microengine.bidstrategy.default

Module Contents

class microengine.bidstrategy.default.**BidStrategy**
Bases: *polyswarmclient.BidStrategyBase*

1.8.2 Submodules

microengine.__main__

Module Contents

microengine.__main__.**logger**

microengine.__main__.**choose_backend**(*backend*)
Resolves microengine name string to implementation

Parameters **backend** (*str*) – Name of the backend to load, either one of the predefined implementations or the name of a module to load (module:ClassName syntax or default of module:Microengine)

Returns Microengine class of the selected implementation

Return type (Class)

Raises **(Exception)** – If backend is not found

microengine.__main__.**choose_bid_strategy**(*bid_strategy*)
Resolves bid strategy name string to implementation

Parameters **bid_strategy** (*str*) – Name of the bid strategy to load, either one of the predefined implementations or the name of a module to load (module:ClassName syntax or default of)

Returns Microengine class of the selected implementation

Return type (Class)

Raises **(Exception)** – If backend is not found

microengine.__main__.**main**(*log*, *client_log*, *polyswarmd_addr*, *keyfile*, *password*, *api_key*, *backend*, *testing*, *insecure_transport*, *allow_key_over_http*, *chains*, *log_format*, *artifact_type*, *bid_strategy*, *accept*, *exclude*, *filter*, *confidence*)
Entrypoint for the microengine driver

microengine.clamav

Module Contents

microengine.clamav.**logger**

microengine.clamav.**CLAMD_HOST**

microengine.clamav.**CLAMD_PORT**

microengine.clamav.**CLAMD_TIMEOUT = 30.0**

class microengine.clamav.**Scanner**
Bases: *polyswarmclient.abstractscanner.AbstractScanner*

class microengine.clamav.**Microengine**(*client*, *testing=0*, *scanner=None*, *chains=None*, *artifact_types=None*, ***kwargs*)
Bases: *polyswarmclient.abstractmicroengine.AbstractMicroengine*

Microengine which scans samples through clamd.

Parameters

- **client** (*Client*) – Client to use
- **testing** (*int*) – How many test bounties to respond to
- **chains** (*set[str]*) – Chain(s) to operate on

microengine.eicar

Module Contents

microengine.eicar.**logger**

microengine.eicar.**EICAR**

class microengine.eicar.**Scanner**
Bases: *polyswarmclient.abstractscanner.AbstractScanner*

scan_sync(*self*, *guid*, *artifact_type*, *content*, *metadata*, *chain*)
Scan an artifact

Parameters

- **guid** (*str*) – GUID of the bounty under analysis, use to track artifacts in the same bounty
- **artifact_type** (*ArtifactType*) – Artifact type for the bounty being scanned
- **content** (*bytes*) – Content of the artifact to be scan
- **metadata** (*dict*) –
- **chain** (*str*) – Chain we are operating on

Returns Result of this scan

Return type *ScanResult*

class microengine.eicar.**Microengine**(*client*, *testing=0*, *scanner=None*, *chains=None*, *artifact_types=None*, ***kwargs*)
Bases: *polyswarmclient.abstractmicroengine.AbstractMicroengine*

Microengine which tests for the EICAR test file.

Parameters

- **client** (*Client*) – Client to use
- **testing** (*int*) – How many test bounties to respond to
- **chains** (*set[str]*) – Chain(s) to operate on

microengine.multi

Module Contents

microengine.multi.**logger**

microengine.multi.**BACKENDS**

class microengine.multi.**Scanner**
Bases: *polyswarmclient.abstractscanner.AbstractScanner*

class microengine.multi.**Microengine**(*client*, *testing=0*, *scanner=None*, *chains=None*, *artifact_types=None*, ***kwargs*)
Bases: *polyswarmclient.abstractmicroengine.AbstractMicroengine*

Microengine which aggregates multiple sub-microengines

microengine.producer

Module Contents

microengine.producer.**logger**

microengine.producer.**REDIS_ADDR**

microengine.producer.**QUEUE**

microengine.producer.**RATE_LIMIT**

microengine.producer.**TIME_TO_POST_ASSERTION = 6**

microengine.producer.**KEY_TIMEOUT = 20**

class microengine.producer.**Microengine**(*client*, *testing=0*, *scanner=None*, *chains=None*, *artifact_types=None*, *bid_strategy=None*, ***kwargs*)

Bases: *polyswarmclient.abstractmicroengine.AbstractMicroengine*

microengine.scratch

Module Contents

microengine.scratch.**logger**

class microengine.scratch.**Scanner**

Bases: *polyswarmclient.abstractscanner.AbstractScanner*

scan_sync(*self*, *guid*, *artifact_type*, *content*, *metadata*, *chain*)

Scan an artifact

Parameters

- **guid** (*str*) – GUID of the bounty under analysis, use to track artifacts in the same bounty
- **artifact_type** (*ArtifactType*) – Artifact type for the bounty being scanned
- **content** (*bytes*) – Content of the artifact to be scan
- **metadata** (*dict*) –
- **chain** (*str*) – Chain we are operating on

Returns Result of this scan

Return type *ScanResult*

class microengine.scratch.**Microengine**(*client*, *testing=0*, *scanner=None*, *chains=None*, *artifact_types=None*, ***kwargs*)

Bases: *polyswarmclient.abstractmicroengine.AbstractMicroengine*

Scratch microengine is the same as the default behavior.

Parameters

- **client** (*Client*) – Client to use
- **testing** (*int*) – How many test bounties to respond to
- **chains** (*set[str]*) – Chain(s) to operate on

microengine.yara

Module Contents

microengine.yara.**logger**

microengine.yara.**RULES_DIR**

class microengine.yara.**Scanner**

Bases: *polyswarmclient.abstractscanner.AbstractScanner*

scan_sync(*self*, *guid*, *artifact_type*, *content*, *metadata*, *chain*)

Scan an artifact with Yara.

Parameters

- **guid** (*str*) – GUID of the bounty under analysis, use to track artifacts in the same bounty
- **artifact_type** (*ArtifactType*) – Artifact type for the bounty being scanned
- **content** (*bytes*) – Content of the artifact to be scan
- **metadata** (*dict*) –
- **chain** (*str*) – Chain we are operating on

Returns Result of this scan

Return type *ScanResult*

class microengine.yara.**Microengine**(*client*, *testing=0*, *scanner=None*, *chains=None*, *artifact_types=None*, ***kwargs*)

Bases: *polyswarmclient.abstractmicroengine.AbstractMicroengine*

Microengine which matches samples against yara rules

`polyswarm-client` is a convenient library on which to build PolySwarm market participants.

Here you'll find auto-generated documentation based on the `polyswarm-client` source code.

Consulting these low level details is unnecessary for developing a successful PolySwarm participant. Most users will instead want to consult the PolySwarm Documentation.

Developers interested in low level `polyswarm-client` details can do so by navigating to "API Reference" on the left.

PYTHON MODULE INDEX

W

INDEX

Symbols

A

B

C

D

E

F

G

J

K

L

M

N

O

P

Q

R

S

T

U

V

W