```
import logging
import os
import shutil
import time
import uuid
import netaddr
from datetime import datetime, timedelta
from collections import deque

import adal  #pylint: disable=E0401
import gevent
import requests
import yaml
import ujson as json
from gevent.queue import Queue, Empty, Full
from netaddr import IPNetwork
from requests.exceptions import RequestException,
HTTPError

from minemeld.ft import ft_states  #pylint:
disable=E0401
from minemeld.ft.base import _counting  #pylint:
disable=E0401
from minemeld.ft.actorbase import ActorBaseFT
#pylint: disable=E0401
from minemeld.ft.table import Table #pylint:
disable=E0401

LOG = logging.getLogger(__name__)
AUTHORITY_BASE_URL =
'https://login.microsoftonline.com'
AUTHORITY_URL =
'https://login.microsoftonline.com/{}'
RESOURCE =
'https://securitycenter.onmicrosoft.com/windowsatpse
rvice'
ENDPOINT_URL =
'https://partnerstifrontend-eus-prd.trafficmanager.n
et/threatintel/info'

MM_2_WDATP_TYPE = {
    'sha1': 'FileSha1',
```

```
        'sha256': 'FileSha256',
        'IPv4': 'IpAddress',
        'domain': 'DomainName',
        'URL': 'Url'
    }
    WD_ATP_TIINDICATORS_ENDPOINT =
    'https://api.securitycenter.windows.com/api/indicato
    rs/import'

    class AuthConfigException(RuntimeError):
        pass

    class WDATPResponseException(RuntimeError):
        pass

    class Output(ActorBaseFT):
        def __init__(self, name, chassis, config):
            self._queue = None

            super(Output, self).__init__(name, chassis,
             config)

            self._push_glet = None
            self._checkpoint_glet = None
            self.api_client_id = str(uuid.uuid4())
            self.sequence_number = 0

        def configure(self):
            super(Output, self).configure()

            self.queue_maxsize = int(self.config.get(
            'queue_maxsize', 100000))
            if self.queue_maxsize == 0:
                self.queue_maxsize = None
            self._queue = Queue(maxsize=self.
            queue_maxsize)

            self.client_id = self.config.get(
            'client_id', None)
            self.client_secret = self.config.get(
            'client_secret', None)
            self.tenant_id = self.config.get(
```

```
            'tenant_id', None)
67
68          self.sender_id = self.config.get(
            'sender_id', 'minemeld')
69
70          self.side_config_path = self.config.get(
            'side_config', None)
71          if self.side_config_path is None:
72              self.side_config_path = os.path.join(
73                  os.environ['MM_CONFIG_DIR'],
74                  '%s_side_config.yml' % self.name
75              )
76
77          self._load_side_config()
78
79      def _load_side_config(self):
80          try:
81              with open(self.side_config_path, 'r')
                as f:
82                  sconfig = yaml.safe_load(f)
83
84          except Exception as e:
85              LOG.error('%s - Error loading side
                config: %s', self.name, str(e))
86              return
87
88          client_id = sconfig.get('client_id', None)
89          if client_id is not None:
90              self.client_id = client_id
91              LOG.info('{} - client_id set'.format(
                self.name))
92
93          client_secret = sconfig.get('client_secret'
            , None)
94          if client_secret is not None:
95              self.client_secret = client_secret
96              LOG.info('{} - client_secret set'.
                format(self.name))
97
98          tenant_id = sconfig.get('tenant_id', None)
99          if tenant_id is not None:
100             self.tenant_id = tenant_id
```

```
101             LOG.info('{} - tenant_id set'.format(
                self.name))
102
103     def _saved_state_restore(self, saved_state):
104         super(Output, self)._saved_state_restore(
            saved_state)
105
106         self.api_client_id = saved_state.get(
            'api_client_id', None)
107         self.sequence_number = saved_state.get(
            'sequence_number', None)
108
109         LOG.info('{} - saved state: api_client_id:
            {} sequence_number: {}'.format(
110             self.name,
111             self.api_client_id,
112             self.sequence_number
113         ))
114
115     def _saved_state_create(self):
116         sstate = super(Output, self).
            _saved_state_create()
117
118         sstate['api_client_id'] = self.api_client_id
119         sstate['sequence_number'] = self.
            sequence_number
120
121         return sstate
122
123     def _saved_state_reset(self):
124         super(Output, self)._saved_state_reset()
125
126         self.api_client_id = str(uuid.uuid4())
127         self.sequence_number = 0
128
129     def connect(self, inputs, output):
130         output = False
131         super(Output, self).connect(inputs, output)
132
133     def initialize(self):
134         pass
135
```

```
136         def rebuild(self):
137             pass
138 
139         def reset(self):
140             pass
141 
142         def _get_auth_token(self):
143             if self.client_id is None:
144                 LOG.error('{} - client_id not set'.
                    format(self.name))
145                 raise AuthConfigException('{} -
                    client_id not set'.format(self.name))
146             if self.client_secret is None:
147                 LOG.error('{} - client_secret not set'.
                    format(self.name))
148                 raise AuthConfigException('{} -
                    client_secret not set'.format(self.name
                    ))
149             if self.tenant_id is None:
150                 LOG.error('{} - tenant_id not set'.
                    format(self.name))
151                 raise AuthConfigException('{} -
                    tenant_id not set'.format(self.name))
152 
153             context = adal.AuthenticationContext(
154                 AUTHORITY_URL.format(self.tenant_id),
155                 validate_authority=self.tenant_id !=
                    'adfs',
156                 api_version=None
157             )
158 
159             token = context.
                acquire_token_with_client_credentials(
160                 RESOURCE,
161                 self.client_id,
162                 self.client_secret
163             )
164 
165             if token is None or 'accessToken' not in
                token:
166                 LOG.error('{} - Invalid token or
                    accessToken not available'.format(self.
```

```
                name))
167             raise RuntimeError('{} - Invalid token
                or accessToken not available'.format(
                self.name))
168
169         return token['accessToken']
170
171     def _get_endpoint_orgid(self, token):
172         # this should look like
173         # {
174         # u'AadTenantId':
            u'bb19bb5c-0e8d-4a73-bd6b-d015b298ecd7',
175         # u'ServiceUri':
            u'https://partnerstifrontend-eus-prd.traffic
            manager.net/threatintel/indicators',
176         # u'ServiceType': 1,
177         # u'WdAtpOrgId':
            u'55c01df7-a1eb-4eae-ae3b-a9b423d07d72'
178         # }
179         result = requests.get(
180             ENDPOINT_URL,
181             headers={
182                 'Authorization': 'Bearer {}'.format
                    (token),
183                 'Content-Type': 'application/json'
184             }
185         )
186         result.raise_for_status()
187
188         result = result.json()
189         LOG.debug('{} - endpoints: {}'.format(self.
            name, result))
190
191         if result.get('AadTenantId', None) != self.
            tenant_id:
192             raise AuthConfigException('{} -
                Endpoint response AadTenantId differs
                from tenant_id: {}'.format(self.name,
                result))
193
194         endpoint = result.get('ServiceUri', None)
195         if endpoint is None:
```

```
196                 raise AuthConfigException('{} -
                    Endpoint response missing ServiceUri
                    field: {}'.format(self.name, result))
197
198             org_id = result.get('WdAtpOrgId', None)
199             if endpoint is None:
200                 raise AuthConfigException('{} -
                    Endpoint response missing WdAtpOrgId
                    field: {}'.format(self.name, result))
201
202             return endpoint, org_id
203
204         def _push_indicators(self, token, endpoint,
            org_id, indicators):
205             # DEPRECATED
206
207             # message = {
208             #     'Id': self.api_client_id,
209             #     'SequenceNumber':
                self.sequence_number,
210             #     'SenderId': self.sender_id,
211             #     'Indicators': list(indicators),
212             #     'WdAtpOrgId': org_id
213             # }
214
215             # LOG.debug(message)
216
217             # result = requests.post(
218             #     endpoint,
219             #     headers={
220             #         'Content-Type':
                    'application/json',
221             #         'Authorization': 'Bearer
                    {}'.format(token)
222             #     },
223             #     json=message
224             # )
225
226             # LOG.debug(result.text)
227
228             # result.raise_for_status()
229             raise WDATPResponseException('This output
```

```
            node is deprecated: please switch to
            OutputBatch')
230
231         def _push_loop(self):
232             while True:
233                 msg = self._queue.get()
234
235                 artifacts = deque()
236                 artifacts.append(msg)
237
238                 try:
239                     while len(artifacts) < 511:
240                         artifacts.append(self._queue.
                            get_nowait())
241                 except Empty:
242                     pass
243
244                 while True:
245                     result = None
246
247                     try:
248                         LOG.info('{} - Sending {}:{}'.
                            format(self.name, self.
                            api_client_id, self.
                            sequence_number))
249                         # DEPRECATED - no need to get
                            the token
250                         # token = self._get_auth_token()
251                         token = 'DEPRECATED'
252                         LOG.debug('{} - token: {}'.
                            format(self.name, token))
253
254                         # DEPRECATED
255                         #endpoint, org_id =
                            self._get_endpoint_orgid(token)
256                         #LOG.debug('{} - endpoint: {}
                            WdAtpOrgId:
                            {}'.format(self.name,
                            endpoint, org_id))
257
258                         # self._push_indicators(
259                         #     token=token,
```

```
260                         #     endpoint=endpoint,
261                         #     org_id=org_id,
262                         #     indicators=artifacts
263                         # )
264                         self._push_indicators(None,
                            None, None, None)
265
266                         self.sequence_number += 1
267                         self.statistics['indicator.tx']
                             += len(artifacts)
268                         break
269
270                 except gevent.GreenletExit:
271                     return
272
273                 except RequestException as e:
274                     LOG.error('{} - error
                            submitting indicators - {}'.
                            format(self.name, str(e)))
275
276                     if result is not None and
                            result.status_code >= 400 and
                            result.status_code < 500:
277                         LOG.error('{}: error in
                                request - {}'.format(self.
                                name, result.text))
278                         self.statistics[
                                'error.invalid_request'] +=
                                 1
279                         break
280
281                     self.statistics['error.submit']
                             += 1
282                     gevent.sleep(60)
283
284                 except AuthConfigException as e:
285                     LOG.exception('{} - Error
                            submitting indicators - {}'.
                            format(self.name, str(e)))
286                     self.statistics['error.submit']
                             += 1
287                     gevent.sleep(60.0)
```

```
288
289                 except WDATPResponseException as e:
290                     LOG.exception('{} - error
                    submitting indicators - {}'.
                    format(self.name, str(e)))
291                     self.statistics['error.submit']
                     += 1
292                     break
293
294                 except Exception as e:
295                     LOG.exception('{} - error
                    submitting indicators - {}'.
                    format(self.name, str(e)))
296                     self.statistics['error.submit']
                     += 1
297                     gevent.sleep(120.0)
298
299             gevent.sleep(0.1)
300
301     def _encode_indicator(self, indicator, value,
        expired=False):
302         type_ = value['type']
303
304         description = '{} indicator from {}'.format(
305             type_,
306             ', '.join(value['sources'])
307         )
308         external_id = '{}:{}'.format(type_,
            indicator)
309         expiration = datetime.utcnow() + timedelta(
            days=365)
310         if expired:
311             expiration = datetime.fromtimestamp(0)
312         expiration = expiration.isoformat()
313
314         result = {
315             'Description': description,
316             'Confidence': value['confidence'],
317             'ExternalId': external_id,
318             'IndicatorExpirationDateTime':
                expiration
319         }
```

```
320
321             if type_ == 'URL':
322                 result['Url'] = indicator
323             elif type_ == 'domain':
324                 result['DNSDomainName'] = indicator
325             elif type_ == 'md5':
326                 result['FileMD5'] = indicator
327             elif type_ == 'sha256':
328                 result['FileSha256'] = indicator
329             elif type_ == 'IPv4':
330                 if '-' in indicator:
331                     a1, a2 = indicator.split('-', 1)
332                     #indicator = netaddr.IPRange(a1,
                        a2).cidrs()[0]
333                     r =list(netaddr.IPRange(a1, a2))
334                     indicator = [str(i) for i in r]
335
336                 #parsed = netaddr.IPNetwork(indicator)
337                 #if parsed.size == 1:
338                 result['NetworkDestinationIPv4'] = str(
                    indicator)
339                 #else:
340                 #
                    result['NetworkDestinationCidrBlock']
                    = str(indicator)
341
342             else:
343                 self.statistics['error.unhandled_type']
                     += 1
344                 raise RuntimeError('{} - Unhandled {}'.
                    format(self.name, type_))
345
346             return result
347
348         def _checkpoint_check(self, source=None, value=
            None):
349             t0 = time.time()
350
351             while ((time.time() - t0) < 30) and self.
                _queue.qsize() != 0:
352                 gevent.sleep(0.5)
353             self._push_glet.kill()
```

```
354
355             LOG.info('{} - checkpoint with {} elements
                in the queue'.format(self.name, self._queue
                .qsize()))
356             super(Output, self).checkpoint(source=
                source, value=value)
357
358         @_counting('update.processed')
359         def filtered_update(self, source=None,
            indicator=None, value=None):
360             try:
361                 self._queue.put(
362                     self._encode_indicator(indicator,
                        value, expired=False),
363                     block=True,
364                     timeout=0.001
365                 )
366             except Full:
367                 self.statistics['error.queue_full'] += 1
368
369         @_counting('withdraw.processed')
370         def filtered_withdraw(self, source=None,
            indicator=None, value=None):
371             if value is None:
372                 self.statistics['error.no_value'] += 1
373                 return
374
375             try:
376                 self._queue.put(
377                     self._encode_indicator(indicator,
                        value, expired=True),
378                     block=True,
379                     timeout=0.001
380                 )
381             except Full:
382                 self.statistics['error.queue_full'] += 1
383
384         @_counting('checkpoint.rx')
385         def checkpoint(self, source=None, value=None):
386             self.state = ft_states.CHECKPOINT
387             self._checkpoint_glet = gevent.spawn(
388                 self._checkpoint_check,
```

```
389                 source,
390                 value
391             )
392
393         def mgmtbus_status(self):
394             result = super(ActorBaseFT, self).
                mgmtbus_status()
395             result['sub_state'] = 'ERROR'
396             result['sub_state_message'] = 'This node
                is deprecated'
397
398             return result
399
400         def length(self, source=None):
401             return self._queue.qsize()
402
403         def start(self):
404             super(Output, self).start()
405
406             self._push_glet = gevent.spawn(self.
                _push_loop)
407
408         def stop(self):
409             super(Output, self).stop()
410
411             if self._push_glet is not None:
412                 self._push_glet.kill()
413
414             if self._checkpoint_glet is not None:
415                 self._checkpoint_glet.kill()
416
417         def hup(self, source=None):
418             LOG.info('%s - hup received, reload side
                config', self.name)
419             self._load_side_config()
420
421         @staticmethod
422         def gc(name, config=None):
423             ActorBaseFT.gc(name, config=config)
424             shutil.rmtree(name, ignore_errors=True)
425
426
```

```
427   class OutputBatch(ActorBaseFT):
428       def __init__(self, name, chassis, config):
429           self._queue = None
430
431           super(OutputBatch, self).__init__(name,
              chassis, config)
432
433           self._push_glet = None
434           self._checkpoint_glet = None
435
436       def configure(self):
437           super(OutputBatch, self).configure()
438
439           self.queue_maxsize = int(self.config.get(
              'queue_maxsize', 100000))
440           if self.queue_maxsize == 0:
441               self.queue_maxsize = None
442           self._queue = Queue(maxsize=self.
              queue_maxsize)
443
444           self.client_id = self.config.get(
              'client_id', None)
445           self.client_secret = self.config.get(
              'client_secret', None)
446           self.tenant_id = self.config.get(
              'tenant_id', None)
447           self.action = self.config.get('action',
              'Alert')
448           self.severity = self.config.get('severity',
               None)
449
450           self.side_config_path = self.config.get(
              'side_config', None)
451           if self.side_config_path is None:
452               self.side_config_path = os.path.join(
453                   os.environ['MM_CONFIG_DIR'],
454                   '%s_side_config.yml' % self.name
455               )
456
457           self._load_side_config()
458
459       def _load_side_config(self):
```

```
460             try:
461                 with open(self.side_config_path, 'r')
                    as f:
462                     sconfig = yaml.safe_load(f)
463
464             except Exception as e:
465                 LOG.error('%s - Error loading side
                    config: %s', self.name, str(e))
466                 return
467
468             client_id = sconfig.get('client_id', None)
469             if client_id is not None:
470                 self.client_id = client_id
471                 LOG.info('{} - client_id set'.format(
                    self.name))
472
473             client_secret = sconfig.get('client_secret'
                , None)
474             if client_secret is not None:
475                 self.client_secret = client_secret
476                 LOG.info('{} - client_secret set'.
                    format(self.name))
477
478             tenant_id = sconfig.get('tenant_id', None)
479             if tenant_id is not None:
480                 self.tenant_id = tenant_id
481                 LOG.info('{} - tenant_id set'.format(
                    self.name))
482
483             action = sconfig.get('action', None)
484             if action is not None:
485                 self.action = action
486                 LOG.info('{} - action set'.format(self.
                    action))
487
488         def connect(self, inputs, output):
489             output = False
490             super(OutputBatch, self).connect(inputs,
                    output)
491
492         def _initialize_table(self, truncate=False):
493             self.table = Table(name=self.name, truncate
```

```
            =truncate)
494
495         def initialize(self):
496             self._initialize_table()
497
498         def rebuild(self):
499             self._initialize_table(truncate=(self.
                last_checkpoint is None))
500
501         def reset(self):
502             self._initialize_table(truncate=True)
503
504         def _get_auth_token(self):
505             if self.client_id is None:
506                 LOG.error('{} - client_id not set'.
                    format(self.name))
507                 raise AuthConfigException('{} -
                    client_id not set'.format(self.name))
508             if self.client_secret is None:
509                 LOG.error('{} - client_secret not set'.
                    format(self.name))
510                 raise AuthConfigException('{} -
                    client_secret not set'.format(self.name
                    ))
511             if self.tenant_id is None:
512                 LOG.error('{} - tenant_id not set'.
                    format(self.name))
513                 raise AuthConfigException('{} -
                    tenant_id not set'.format(self.name))
514
515             context = adal.AuthenticationContext(
516                 AUTHORITY_URL.format(self.tenant_id),
517                 validate_authority=self.tenant_id !=
                    'adfs',
518                 api_version=None
519             )
520
521             token = context.
                acquire_token_with_client_credentials(
522                 RESOURCE,
523                 self.client_id,
524                 self.client_secret
```

```
525             )
526
527             if token is None or 'accessToken' not in
                token:
528                 LOG.error('{} - Invalid token or
                    accessToken not available'.format(self.
                    name))
529                 raise RuntimeError('{} - Invalid token
                    or accessToken not available'.format(
                    self.name))
530
531             return token['accessToken']
532
533         def _push_indicators(self, token, indicators):
534             message = {
535                 'Indicators': list(indicators)
536             }
537
538             LOG.debug(message)
539
540             result = requests.post(
541                 WD_ATP_TIINDICATORS_ENDPOINT,
542                 headers={
543                     'Content-Type': 'application/json',
544                     'Authorization': 'Bearer {}'.format
                    (token)
545                 },
546                 json=message
547             )
548
549             LOG.debug(result.text)
550
551             result.raise_for_status()
552
553             # Check the status of the submitted
                indicators
554             # NOTE: if the indicator contains a range
                split by _encode_indicators, a partial
                submission might go through
555             # i.e. 192.168.0.1-192.168.0.3 can be
                split in 192.168.0.1/32 and 192.168.0.2/31
556             # the first might go through, the second
```

```
            might return error
557         # This output node doesn't check for this
            condition (although the error counters are
            correctly updated)
558
559         result = result.json()
560         if not result or  '@odata.context' not in
            result or result['@odata.context'] !=
            'https://api.securitycenter.windows.com/api/
            $metadata#Collection(microsoft.windowsDefend
            erATP.api.ImportIndicatorResult)':
561             raise WDATPResponseException(
                'Unexpected response from WDATP API')
562
563         if 'value' not in result:
564             raise WDATPResponseException('Missing
                value from WDATP API result')
565
566         for v in result['value']:
567             if 'indicator' not in v or 'isFailed'
                not in v:
568                 raise WDATPResponseException(
                    'Missing indicator values from
                    WDATP response')
569             LOG.debug('{} - Got result for
                indicator {}: isFailed is {}'.format(
                self.name, v['indicator'], v["isFailed"
                ]))
570             if not v["isFailed"]:
571                 # Success!
572                 self.statistics['indicator.tx'] += 1
573             else:
574                 failReason = v['failureReason'] if
                    'failureReason' in v else 'Unknown'
575                 LOG.error('{}: error submitting
                    indicator {}: {}'.format(self.name,
                     v['indicator'], failReason))
576                 self.statistics['error.submit'] += 1
577
578     def _push_loop(self):
579         while True:
580             msg = self._queue.get()
```

```
581
582                 artifacts = deque()
583                 artifacts.append(msg)
584
585                 try:
586                     while len(artifacts) < 50:
587                         artifacts.append(self._queue.
                            get_nowait())
588                 except Empty:
589                     pass
590
591                 while True:
592                     retries = 0
593
594                     try:
595                         LOG.info('{} - Sending {}
                            indicators'.format(self.name,
                            len(artifacts)))
596                         token = self._get_auth_token()
597                         LOG.debug('{} - token: {}'.
                            format(self.name, token))
598
599                         self._push_indicators(
600                             token=token,
601                             indicators=artifacts
602                         )
603                         # Counter already incremented
                            in push_indicators
604                         #
                            self.statistics['indicator.tx']
                            += len(artifacts)
605                         break
606
607                     except gevent.GreenletExit:
608                         return
609
610                     except HTTPError as e:
611                         LOG.error('{} - error
                            submitting indicators - {}'.
                            format(self.name, str(e)))
612                         status_code = e.response.
                            status_code
```

```
613
614                     if status_code >= 400 and
                        status_code < 500:
615                         LOG.error('{}: error in
                            request - {}'.format(self.
                            name, e.response.text))
616                         self.statistics[
                            'error.invalid_request'] +=
                             1
617                         break
618
619                     self.statistics['error.submit']
                         += 1
620                     gevent.sleep(60)
621
622                 except AuthConfigException as e:
623                     LOG.exception('{} - Error
                        submitting indicators - {}'.
                        format(self.name, str(e)))
624                     self.statistics['error.submit']
                         += 1
625                     gevent.sleep(60.0)
626
627                 except WDATPResponseException as e:
628                     LOG.exception('{} - error
                        submitting indicators - {}'.
                        format(self.name, str(e)))
629                     self.statistics['error.submit']
                         += 1
630                     break
631
632                 except Exception as e:
633                     LOG.exception('{} - error
                        submitting indicators - {}'.
                        format(self.name, str(e)))
634                     self.statistics['error.submit']
                         += 1
635                     retries += 1
636                     if retries > 5:
637                         break
638                     gevent.sleep(120.0)
639
```

```
640             gevent.sleep(0.1)
641
642         def _encode_indicator(self, indicator, value,
            expired=False):
643             type_ = MM_2_WDATP_TYPE.get(
644                 value['type'],
645                 None
646             )
647             if type_ is None:
648                 self.statistics['error.unhandled_type']
                     += 1
649                 raise RuntimeError('{} - Unhandled {}'.
                    format(self.name, type_))
650
651             if value['type'] == 'IPv4' and '-' in
                indicator:
652                 a1, a2 = indicator.split('-', 1)
653                 #r = netaddr.IPRange(a1, a2).cidrs()
654                 r =list(netaddr.IPRange(a1, a2))
655                 indicators = [str(i) for i in r]
656
657             else:
658                 indicators = [indicator]
659
660             description = '{} indicator from {}'.format(
661                 type_,
662                 ', '.join(value['sources'])
663             )
664             title = 'MineMeld - {}'.format(indicator)
665
666             creation = datetime.utcnow()
667             creation = creation.isoformat() + 'Z'
668
669             expiration = datetime.utcnow() + timedelta(
                days=365)
670             if expired:
671                 expiration = datetime.fromtimestamp(0)
672             expiration = expiration.isoformat() + 'Z'
                # expiration is always in UTC
673
674             result = []
675             for i in indicators:
```

```
676                 d = dict(
677                     indicatorValue=i,
678                     indicatorType=type_,
679                     title=title,
680                     description=description,
681                     creationTimeDateTimeUtc=creation,
682                     expirationTime=expiration,
683                     action=self.action
684                 )
685                 if self.severity is not None:
686                     d['severity'] = self.severity
687 
688                 result.append(d)
689 
690             return result
691 
692         def _checkpoint_check(self, source=None, value=
        None):
693             t0 = time.time()
694 
695             while ((time.time() - t0) < 30) and self.
            _queue.qsize() != 0:
696                 gevent.sleep(0.5)
697             self._push_glet.kill()
698 
699             LOG.info('{} - checkpoint with {} elements
            in the queue'.format(self.name, self._queue
            .qsize()))
700             super(OutputBatch, self).checkpoint(source=
            source, value=value)
701 
702         @_counting('update.processed')
703         def filtered_update(self, source=None,
        indicator=None, value=None):
704             try:
705                 for i in self._encode_indicator(
                indicator, value, expired=False):
706                     self._queue.put(
707                         i,
708                         block=True,
709                         timeout=0.001
710                     )
```

```
711             except Full:
712                 self.statistics['error.queue_full'] += 1
713
714         @_counting('withdraw.processed')
715         def filtered_withdraw(self, source=None,
            indicator=None, value=None):
716             if value is None:
717                 self.statistics['error.no_value'] += 1
718                 return
719
720             try:
721                 for i in self._encode_indicator(
                    indicator, value, expired=True):
722                     self._queue.put(
723                         i,
724                         block=True,
725                         timeout=0.001
726                     )
727             except Full:
728                 self.statistics['error.queue_full'] += 1
729
730         @_counting('checkpoint.rx')
731         def checkpoint(self, source=None, value=None):
732             self.state = ft_states.CHECKPOINT
733             self._checkpoint_glet = gevent.spawn(
734                 self._checkpoint_check,
735                 source,
736                 value
737             )
738
739         def length(self, source=None):
740             return self._queue.qsize()
741
742         def start(self):
743             super(OutputBatch, self).start()
744
745             self._push_glet = gevent.spawn(self.
                _push_loop)
746
747         def stop(self):
748             super(OutputBatch, self).stop()
749
```

```
750             if self._push_glet is not None:
751                 self._push_glet.kill()
752
753             if self._checkpoint_glet is not None:
754                 self._checkpoint_glet.kill()
755
756             self.table.close()
757
758         def hup(self, source=None):
759             LOG.info('%s - hup received, reload side
                config', self.name)
760             self._load_side_config()
761
762         @staticmethod
763         def gc(name, config=None):
764             ActorBaseFT.gc(name, config=config)
765             shutil.rmtree(name, ignore_errors=True)
766
```