Sign in

thinkst / opencanary Public

Notifications Fork 360 Star 2.3k

Code Issues Pull requests 4 Discussions Actions Projects Wiki Security

opencanary / opencanary / logger.py

jayjb Add ignore_localhost for portscan c85d092 · 3 years ago

294 lines (257 loc) · 10.5 KB

```
1   from __future__ import print_function
2   import simplejson as json
3   import logging.config
4   import socket
5   import hpfeeds
6   import sys
7
8   from datetime import datetime
9   from logging.handlers import SocketHandler
10  from twisted.internet import reactor
11  import requests
12
13  from opencanary.iphelper import *
14
15  class Singleton(type):
16      _instances = {}
17      def __call__(cls, *args, **kwargs):
18          if cls not in cls._instances:
19              cls._instances[cls] = super(Singleton, cls).__call__(*args, **kwargs)
20          return cls._instances[cls]
21
22  def getLogger(config):
23      try:
24          d = config.getVal('logger')
25      except Exception as e:
26          print("Error: config does not have 'logger' section", file=sys.stderr)
```

```
            exit(1)

        classname = d.get('class', None)
        if classname is None:
            print("Logger section is missing the class key.", file=sys.stderr)
            exit(1)
```

opencanary / opencanary / logger.py ↑ Top

Code Blame Raw

```
        kwargs = d.get('kwargs', None)
        if kwargs is None:
            print("Logger section is missing the kwargs key.", file=sys.stderr)
            exit(1)
        try:
            logger = LoggerClass(config, **kwargs)
        except Exception as e:
            print("An error occured initialising the logger class", file=sys.stderr)
            print(e)
            exit(1)

        return logger

    class LoggerBase(object):
        LOG_BASE_BOOT                   = 1000
        LOG_BASE_MSG                    = 1001
        LOG_BASE_DEBUG                  = 1002
        LOG_BASE_ERROR                  = 1003
        LOG_BASE_PING                   = 1004
        LOG_BASE_CONFIG_SAVE            = 1005
        LOG_BASE_EXAMPLE                = 1006
        LOG_FTP_LOGIN_ATTEMPT           = 2000
        LOG_HTTP_GET                    = 3000
        LOG_HTTP_POST_LOGIN_ATTEMPT     = 3001
        LOG_SSH_NEW_CONNECTION          = 4000
        LOG_SSH_REMOTE_VERSION_SENT     = 4001
        LOG_SSH_LOGIN_ATTEMPT           = 4002
        LOG_SMB_FILE_OPEN               = 5000
        LOG_PORT_SYN                    = 5001
        LOG_PORT_NMAPOS                 = 5002
        LOG_PORT_NMAPNULL               = 5003
        LOG_PORT_NMAPXMAS               = 5004
        LOG_PORT_NMAPFIN                = 5005
        LOG_TELNET_LOGIN_ATTEMPT        = 6001
```

```
73          LOG_HTTPPROXY_LOGIN_ATTEMPT                 = 7001
74          LOG_MYSQL_LOGIN_ATTEMPT                     = 8001
75          LOG_MSSQL_LOGIN_SQLAUTH                     = 9001
76          LOG_MSSQL_LOGIN_WINAUTH                     = 9002
77          LOG_TFTP                                    = 10001
78          LOG_NTP_MONLIST                             = 11001
79          LOG_VNC                                     = 12001
80          LOG_SNMP_CMD                                = 13001
81          LOG_RDP                                     = 14001
82          LOG_SIP_REQUEST                             = 15001
83          LOG_GIT_CLONE_REQUEST                       = 16001
84          LOG_REDIS_COMMAND                           = 17001
85          LOG_TCP_BANNER_CONNECTION_MADE              = 18001
86          LOG_TCP_BANNER_KEEP_ALIVE_CONNECTION_MADE   = 18002
87          LOG_TCP_BANNER_KEEP_ALIVE_SECRET_RECEIVED   = 18003
88          LOG_TCP_BANNER_KEEP_ALIVE_DATA_RECEIVED     = 18004
89          LOG_TCP_BANNER_DATA_RECEIVED                = 18005
90          LOG_USER_0                                  = 99000
91          LOG_USER_1                                  = 99001
92          LOG_USER_2                                  = 99002
93          LOG_USER_3                                  = 99003
94          LOG_USER_4                                  = 99004
95          LOG_USER_5                                  = 99005
96          LOG_USER_6                                  = 99006
97          LOG_USER_7                                  = 99007
98          LOG_USER_8                                  = 99008
99          LOG_USER_9                                  = 99009
100
101         def sanitizeLog(self, logdata):
102             logdata['node_id'] = self.node_id
103             logdata['local_time'] = datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S.%f")
104             logdata['utc_time'] = datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S.%f")
105             logdata['local_time_adjusted'] = datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f")
106             if 'src_host' not in logdata:
107                 logdata['src_host'] = ''
108             if 'src_port' not in logdata:
109                 logdata['src_port'] = -1
110             if 'dst_host' not in logdata:
111                 logdata['dst_host'] = ''
112             if 'dst_port' not in logdata:
113                 logdata['dst_port'] = -1
114             if 'logtype' not in logdata:
115                 logdata['logtype'] = self.LOG_BASE_MSG
116             if 'logdata' not in logdata:
117                 logdata['logdata'] = {}
118             return logdata
```

```
119
120 class PyLogger(LoggerBase):
121     """
122     Generic python logging
123     """
124     __metaclass__ = Singleton
125
126     def __init__(self, config, handlers, formatters={}):
127         self.node_id = config.getVal('device.node_id')
128
129         # Build config dict to initialise
130         # Ensure all handlers don't drop logs based on severity level
131         for h in handlers:
132             handlers[h]["level"] = "NOTSET"
133
134         logconfig = {
135             "version": 1,
136             "formatters" : formatters,
137             "handlers": handlers,
138             # initialise all defined logger handlers
139             "loggers": {
140                 self.node_id : {
141                     "handlers": handlers.keys()
142                 }
143             }
144         }
145
146         try:
147             logging.config.dictConfig(logconfig)
148         except Exception as e:
149             print("Invalid logging config", file=sys.stderr)
150             print(type(e))
151             print(e)
152             exit(1)
153
154         # Check if ignorelist is populated
155         self.ignorelist = config.getVal('ip.ignorelist', default='')
156
157         self.logger = logging.getLogger(self.node_id)
158
159     def error(self, data):
160         data['local_time'] = datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S.%f")
161         msg = '[ERR] %r' % json.dumps(data, sort_keys=True)
162         print(msg, file=sys.stderr)
163         self.logger.warn(msg)
```

```
164
165         def log(self, logdata, retry=True):
166             logdata = self.sanitizeLog(logdata)
167             # Log only if not in ignorelist
168             notify = True
169             if 'src_host' in logdata:
```

```
221             self.host=str(host)
222             self.port=int(port)
223             self.ident=str(ident)
224             self.secret=str(secret)
225             self.channels=map(str,channels)
226             hpc=hpfeeds.new(self.host, self.port, self.ident, self.secret)
227             hpc.subscribe(channels)
228             self.hpc=hpc
229
230         def emit(self, record):
231             try:
232                 msg = self.format(record)
233                 self.hpc.publish(self.channels,msg)
234             except:
235                 print("Error on publishing to server")
236
237     class SlackHandler(logging.Handler):
238         def __init__(self,webhook_url):
239             logging.Handler.__init__(self)
240             self.webhook_url=webhook_url
241
242         def generate_msg(self, alert):
243             msg = {}
244             msg['pretext'] = "OpenCanary Alert"
245             data=json.loads(alert.msg)
246             msg['fields']=[]
247             for k,v in data.items():
248                 msg['fields'].append({'title':k, 'value':json.dumps(v) if type(v) is dict else v})
249             return {'attachments':[msg]}
250
251         def emit(self, record):
252             data = self.generate_msg(record)
253             response = requests.post(
254                 self.webhook_url, json=data
255                 )
```

```
256             if response.status_code != 200:
257                 print("Error %s sending Slack message, the response was:\n%s" % (response.status_code,
258
259     class TeamsHandler(logging.Handler):
260         def __init__(self,webhook_url):
261             logging.Handler.__init__(self)
262             self.webhook_url=webhook_url
263
264         def message(self, data):
265             message = {
266                 "@type": "MessageCard",
267                 "@context": "http://schema.org/extensions",
268                 "themeColor": "49c176",
269                 "summary": "OpenCanary Notification",
270                 "title": "OpenCanary Alert",
271                 "sections": [{
272                     "facts": self.facts(data)
273                 }]
274             }
275             return message
276
277         def facts(self, data, prefix=None):
278             facts = []
279             for k, v in data.items():
280                 key = str(k).lower() if prefix is None else prefix + '__' + str(k).lower()
281                 if type(v) is not dict:
282                     facts.append({"name": key, "value": str(v)})
283                 else:
284                     nested = self.facts(v, key)
285                     facts.extend(nested)
286             return facts
287
288         def emit(self, record):
289             data = json.loads(record.msg)
290             payload = self.message(data)
291             headers = {'Content-Type': 'application/json'}
292             response = requests.post(self.webhook_url, headers=headers, json=payload)
293             if response.status_code != 200:
294                 print("Error %s sending Teams message, the response was:\n%s" % (response.status_code,
```