Product Solutions Resources Open Source Enterprise Pricing Sign in Sign up

thinkst / opencanary Public

Notifications Fork 361 Star 2.3k

Code Issues Pull requests 4 Discussions Actions Projects Wiki Security 2 Insights

opencanary / opencanary / logger.py

jayjb Add ignore_localhost for portscan c85d092 · 3 years ago History

```python
from __future__ import print_function
import simplejson as json
import logging.config
import socket
import hpfeeds
import sys

from datetime import datetime
from logging.handlers import SocketHandler
from twisted.internet import reactor
import requests

from opencanary.iphelper import *

class Singleton(type):
    _instances = {}
    def __call__(cls, *args, **kwargs):
        if cls not in cls._instances:
            cls._instances[cls] = super(Singleton, cls).__call__(*args, **kwargs)
        return cls._instances[cls]

def getLogger(config):
    try:
        d = config.getVal('logger')
    except Exception as e:
        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)

```

Files

a0896ad

Go to file

- bin
- data
- docs
- opencanary
  - data
  - modules
  - test
  - __init__.py
  - config.py
  - honeycred.py

opencanary / opencanary / logger.py Top

Code Blame 294 lines (257 loc) · 10.5 KB Raw

```python
def getLogger(config):
    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
```

- iphelper.py
- logger.py
- .gitignore
- Dockerfile.latest
- Dockerfile.stable
- LICENSE
- README.md
- docker-compose.yml
- opencanary.service
- requirements.txt
- run.sh
- setup.py

```python
    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
    LOG_HTTPPROXY_LOGIN_ATTEMPT                 = 7001
    LOG_MYSQL_LOGIN_ATTEMPT                     = 8001
    LOG_MSSQL_LOGIN_SQLAUTH                     = 9001
    LOG_MSSQL_LOGIN_WINAUTH                     = 9002
    LOG_TFTP                                    = 10001
    LOG_NTP_MONLIST                             = 11001
    LOG_VNC                                     = 12001
    LOG_SNMP_CMD                                = 13001
    LOG_RDP                                     = 14001
    LOG_SIP_REQUEST                             = 15001
    LOG_GIT_CLONE_REQUEST                       = 16001
    LOG_REDIS_COMMAND                           = 17001
    LOG_TCP_BANNER_CONNECTION_MADE              = 18001
    LOG_TCP_BANNER_KEEP_ALIVE_CONNECTION_MADE   = 18002
    LOG_TCP_BANNER_KEEP_ALIVE_SECRET_RECEIVED   = 18003
    LOG_TCP_BANNER_KEEP_ALIVE_DATA_RECEIVED     = 18004
    LOG_TCP_BANNER_DATA_RECEIVED                = 18005
    LOG_USER_0                                  = 99000
    LOG_USER_1                                  = 99001
    LOG_USER_2                                  = 99002
    LOG_USER_3                                  = 99003
    LOG_USER_4                                  = 99004
    LOG_USER_5                                  = 99005
    LOG_USER_6                                  = 99006
    LOG_USER_7                                  = 99007
    LOG_USER_8                                  = 99008
    LOG_USER_9                                  = 99009

    def sanitizeLog(self, logdata):
        logdata['node_id'] = self.node_id
        logdata['local_time'] = datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S.%f")
        logdata['utc_time'] = datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S.%f")
        logdata['local_time_adjusted'] = datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f"
        if 'src_host' not in logdata:
            logdata['src_host'] = ''
        if 'src_port' not in logdata:
            logdata['src_port'] = -1
        if 'dst_host' not in logdata:
            logdata['dst_host'] = ''
        if 'dst_port' not in logdata:
            logdata['dst_port'] = -1
        if 'logtype' not in logdata:
            logdata['logtype'] = self.LOG_BASE_MSG
        if 'logdata' not in logdata:
            logdata['logdata'] = {}
        return logdata

class PyLogger(LoggerBase):
    """
    Generic python logging
    """
    __metaclass__ = Singleton

    def __init__(self, config, handlers, formatters={}):
        self.node_id = config.getVal('device.node_id')

        # Build config dict to initialise
        # Ensure all handlers don't drop logs based on severity level
        for h in handlers:
            handlers[h]["level"] = "NOTSET"

        logconfig = {
            "version": 1,
            "formatters" : formatters,
            "handlers": handlers,
            # initialise all defined logger handlers
            "loggers": {
                self.node_id : {
                    "handlers": handlers.keys()
                }
            }
        }

        try:
            logging.config.dictConfig(logconfig)
        except Exception as e:
            print("Invalid logging config", file=sys.stderr)
            print(type(e))
            print(e)
            exit(1)

        # Check if ignorelist is populated
        self.ignorelist = config.getVal('ip.ignorelist', default='')

        self.logger = logging.getLogger(self.node_id)

    def error(self, data):
        data['local_time'] = datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S.%f")
        msg = '[ERR] %r' % json.dumps(data, sort_keys=True)
        print(msg, file=sys.stderr)
        self.logger.warn(msg)

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

```
        self.host=str(host)
        self.port=int(port)
        self.ident=str(ident)
        self.secret=str(secret)
        self.channels=map(str,channels)
        hpc=hpfeeds.new(self.host, self.port, self.ident, self.secret)
        hpc.subscribe(channels)
        self.hpc=hpc

    def emit(self, record):
        try:
            msg = self.format(record)
            self.hpc.publish(self.channels,msg)
        except:
            print("Error on publishing to server")

class SlackHandler(logging.Handler):
    def __init__(self,webhook_url):
        logging.Handler.__init__(self)
        self.webhook_url=webhook_url

    def generate_msg(self, alert):
        msg = {}
        msg['pretext'] = "OpenCanary Alert"
        data=json.loads(alert.msg)
        msg['fields']=[]
        for k,v in data.items():
            msg['fields'].append({'title':k, 'value':json.dumps(v) if type(v) is dict e
        return {'attachments':[msg]}

    def emit(self, record):
        data = self.generate_msg(record)
        response = requests.post(
            self.webhook_url, json=data
            )
        if response.status_code != 200:
            print("Error %s sending Slack message, the response was:\n%s" % (response.s

class TeamsHandler(logging.Handler):
    def __init__(self,webhook_url):
        logging.Handler.__init__(self)
        self.webhook_url=webhook_url

    def message(self, data):
        message = {
            "@type": "MessageCard",
            "@context": "http://schema.org/extensions",
            "themeColor": "49c176",
            "summary": "OpenCanary Notification",
            "title": "OpenCanary Alert",
            "sections": [{
                "facts": self.facts(data)
            }]
        }
        return message

    def facts(self, data, prefix=None):
        facts = []
        for k, v in data.items():
            key = str(k).lower() if prefix is None else prefix + '__' + str(k).lower()
```

```
                if type(v) is not dict:
                    facts.append({"name": key, "value": str(v)})
                else:
                    nested = self.facts(v, key)
                    facts.extend(nested)
            return facts

        def emit(self, record):
            data = json.loads(record.msg)
            payload = self.message(data)
            headers = {'Content-Type': 'application/json'}
            response = requests.post(self.webhook_url, headers=headers, json=payload)
            if response.status_code != 200:
                print("Error %s sending Teams message, the response was:\n%s" % (response.s
```