Cisco Take Home Exam
Malware Detection

by

Amarjit Singh Dhillon

Tools used: PyCharm, LaTeX and Sublime Text

Table of Contents

List of Figures

Chapter 1

Problem Statement

You have a log file of JSON data that gives you information on files that were seen by users and whether it's safe or not (disposition). JSON format for data is given as follows:

```
ts:  timestamp
pt:  processing time
si:  session ID
uu:  user UUID
bg:  business UUID
sha: sha256 of the file
nm:  file name
ph:  path
dp:  disposition (valid values: MALICIOUS (1), CLEAN (2), UNKNOWN (3))
```

The JSON entries are not guaranteed to be valid. Your task is to write an application that processes each valid line in the file and then prints out a list of: sha, dp and count. Feel free to use any languages/tools you'd like. We like to see automated test coverage.

Ideally, you can send us a link to GitHub, GitLab, etc. You have one week from the time of receiving this email to submit your answer. Feel free to make reasonable assumptions.

Chapter 2

Code

Some of the assumptions made are

1. ts(timestamp) should be a valid time-stamp of 10 digits. It can't be null. As the precision of machine readable time-stamp is seconds, so we need to validate it ourself. For validating it, I converted it to a YYYY-MM-DD HH:MM:SS format using python's strftime() function. Further this human readable time is validated using the regex pattern written below. Pattern can further be complicated considering various special cases such as the leap year. But below regex is a basic one.

   ```
   ^\d\d\d\d-(0?[1-9]|1[0-2]) -(0?[1-9]|[12][0-9]|3[01])  (00|[0-9]|1[0-9]|2[0-3]):([0-9]|[0-5][0-9]):([0-9]|[0-5][0-9])$
   ```

2. pt(processing time) is a non-null integer lying in range of 1-100 values.

3. si(session ID), uu(user UUID) and bg(business UUID) is 36 digit long non-null having lowercase hexadecimal-alphanumeric chars of 32 length adhering to below mentioned regex pattern.

   ```
   ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}\$
   ```

4. sha(sha256 of the file) is 64 digit, non-null, lowercase alphanumeric chars having following pattern.

   ```
   ^[a-fA-F0-9]{64}$
   ```

5. nm is a filename which can consist of lowercase as well as uppercase alphanumeric chars with -(hyphen) and spaces allowed having below-written pattern. It can't be null. Also, the filenames in nm can have alphanumeric chars but no special chars.

   ```
   ^[a-zA-Z-  ]+\.[a-z43]+$
   ```

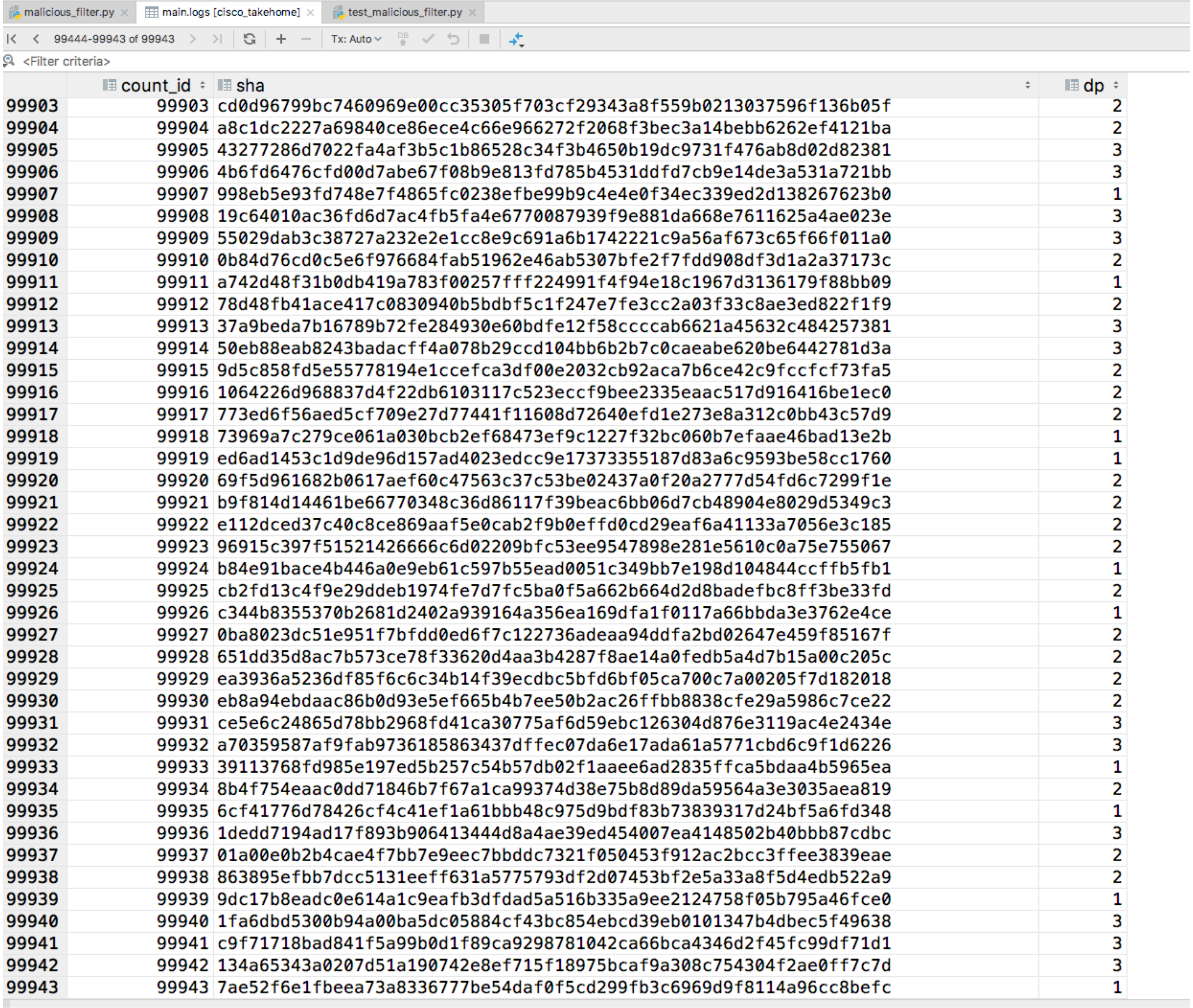
6. pg is a file directory path structure having following pattern.

   ```
   ^[a-zA-Z-  ]+\.[a-z43]+$
   ```

7. dp is a value which can be 1 or 2 or 3. It can't be null.

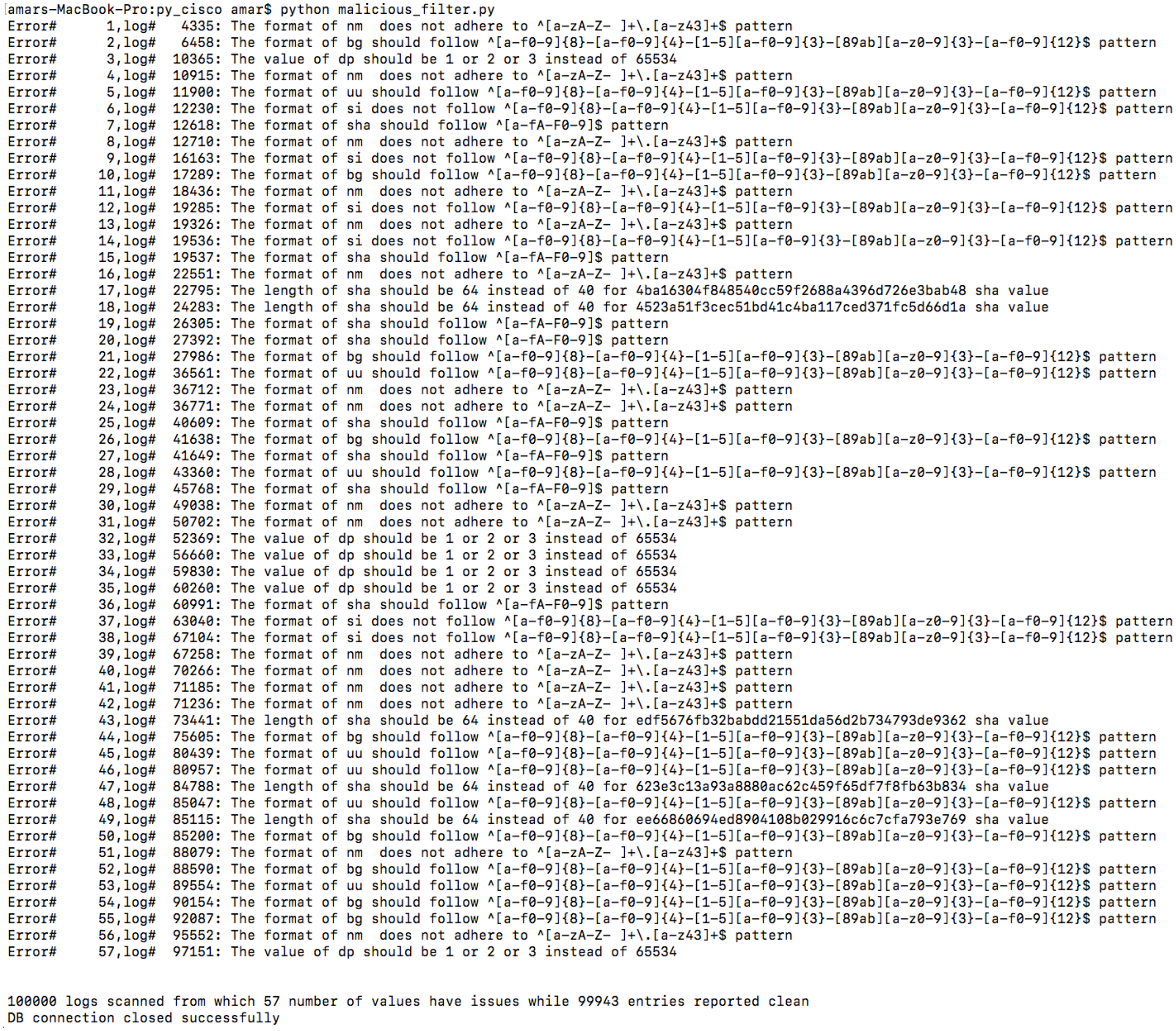
Note: As there are large number of values so, I have appended the valid values to sqlite database, as the terminal instance will not show all the values if we scroll up to see all of them in case of a large log values. As the invalid values are smaller in number, so I show them on terminal. Results are shown below:

Total logs	Clean logs	Invalid logs
100000	99943	57



	count_id	sha	dp
99903	99903	cd0d96799bc7460969e00cc35305f703cf29343a8f559b0213037596f136b05f	2
99904	99904	a8c1dc2227a69840ce86ece4c66e966272f2068f3bec3a14bebb6262ef4121ba	2
99905	99905	43277286d7022fa4af3b5c1b86528c34f3b4650b19dc9731f476ab8d02d82381	3
99906	99906	4b6fd6476cfd00d7abe67f08b9e813fd785b4531ddfd7cb9e14de3a531a721bb	3
99907	99907	998eb5e93fd748e7f4865fc0238efbe99b9c4e4e0f34ec339ed2d138267623b0	1
99908	99908	19c64010ac36fd6d7ac4fb5fa4e6770087939f9e881da668e7611625a4ae023e	3
99909	99909	55029dab3c38727a232e2e1cc8e9c691a6b1742221c9a56af673c65f66f011a0	3
99910	99910	0b84d76cd0c5e6f976684fab51962e46ab5307bfe2f7fdd908df3d1a2a37173c	2
99911	99911	a742d48f31b0db419a783f00257fff224991f4f94e18c1967d3136179f88bb09	1
99912	99912	78d48fb41ace417c0830940b5bdbf5c1f247e7fe3cc2a03f33c8ae3ed822f1f9	2
99913	99913	37a9beda7b16789b72fe284930e60bdfe12f58ccccab6621a45632c484257381	3
99914	99914	50eb88eab8243badacff4a078b29ccd104bb6b2b7c0caeabe620be6442781d3a	3
99915	99915	9d5c858fd5e55778194e1ccefca3df00e2032cb92aca7b6ce42c9fccfcf73fa5	2
99916	99916	1064226d968837d4f22db6103117c523eccf9bee2335eaac517d916416be1ec0	2
99917	99917	773ed6f56aed5cf709e27d77441f11608d72640efd1e273e8a312c0bb43c57d9	2
99918	99918	73969a7c279ce061a030bcb2ef68473ef9c1227f32bc060b7efaae46bad13e2b	1
99919	99919	ed6ad1453c1d9de96d157ad4023edcc9e17373355187d83a6c9593be58cc1760	1
99920	99920	69f5d961682b0617aef60c47563c37c53be02437a0f20a2777d54fd6c7299f1e	2
99921	99921	b9f814d14461be66770348c36d86117f39beac6bb06d7cb48904e8029d5349c3	2
99922	99922	e112dced37c40c8ce869aaf5e0cab2f9b0effd0cd29eaf6a41133a7056e3c185	2
99923	99923	96915c397f51521426666c6d02209bfc53ee9547898e281e5610c0a75e755067	2
99924	99924	b84e91bace4b446a0e9eb61c597b55ead0051c349bb7e198d104844ccffb5fb1	1
99925	99925	cb2fd13c4f9e29ddeb1974fe7d7fc5ba0f5a662b664d2d8badefbc8ff3be33fd	2
99926	99926	c344b8355370b2681d2402a939164a356ea169dfa1f0117a66bbda3e3762e4ce	1
99927	99927	0ba8023dc51e951f7bfdd0ed6f7c122736adeaa94ddfa2bd02647e459f85167f	2
99928	99928	651dd35d8ac7b573ce78f33620d4aa3b4287f8ae14a0fedb5a4d7b15a00c205c	2
99929	99929	ea3936a5236df85f6c6c34b14f39ecdbc5bfd6bf05ca700c7a00205f7d182018	2
99930	99930	eb8a94ebdaac86b0d93e5ef665b4b7ee50b2ac26ffbb8838cfe29a5986c7ce22	2
99931	99931	ce5e6c24865d78bb2968fd41ca30775af6d59ebc126304d876e3119ac4e2434e	3
99932	99932	a70359587af9fab9736185863437dffec07da6e17ada61a5771cbd6c9f1d6226	3
99933	99933	39113768fd985e197ed5b257c54b57db02f1aaee6ad2835ffca5bdaa4b5965ea	1
99934	99934	8b4f754eaac0dd71846b7f67a1ca99374d38e75b8d89da59564a3e3035aea819	2
99935	99935	6cf41776d78426cf4c41ef1a61bbb48c975d9bdf83b73839317d24bf5a6fd348	1
99936	99936	1dedd7194ad17f893b906413444d8a4ae39ed454007ea4148502b40bbb87cdbc	3
99937	99937	01a00e0b2b4cae4f7bb7e9eec7bbddc7321f050453f912ac2bcc3ffee3839eae	2
99938	99938	863895efbb7dcc5131eeff631a5775793df2d07453bf2e5a33a8f5d4edb522a9	2
99939	99939	9dc17b8eadc0e614a1c9eafb3dfdad5a516b335a9ee2124758f05b795a46fce0	1
99940	99940	1fa6dbd5300b94a00ba5dc05884cf43bc854ebcd39eb0101347b4dbec5f49638	3
99941	99941	c9f71718bad841f5a99b0d1f89ca9298781042ca66bca4346d2f45fc99df71d1	3
99942	99942	134a65343a0207d51a190742e8ef715f18975bcaf9a308c754304f2ae0ff7c7d	3
99943	99943	7ae52f6e1fbeea73a8336777be54daf0f5cd299fb3c6969d9f8114a96cc8befc	1

Figure 2.1: A Screenshot of clean values saved in database



```
amars-MacBook-Pro:py_cisco amar$ python malicious_filter.py
Error#     1,log#   4335: The format of nm  does not adhere to ^[a-zA-Z- ]+\.[a-z43]+$ pattern
Error#     2,log#   6458: The format of bg should follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#     3,log#  10365: The value of dp should be 1 or 2 or 3 instead of 65534
Error#     4,log#  10915: The format of nm  does not adhere to ^[a-zA-Z- ]+\.[a-z43]+$ pattern
Error#     5,log#  11900: The format of uu should follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#     6,log#  12230: The format of si does not follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#     7,log#  12618: The format of sha should follow ^[a-fA-F0-9]$ pattern
Error#     8,log#  12710: The format of nm  does not adhere to ^[a-zA-Z- ]+\.[a-z43]+$ pattern
Error#     9,log#  16163: The format of si does not follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    10,log#  17289: The format of bg should follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    11,log#  18436: The format of nm  does not adhere to ^[a-zA-Z- ]+\.[a-z43]+$ pattern
Error#    12,log#  19285: The format of si does not follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    13,log#  19326: The format of nm  does not adhere to ^[a-zA-Z- ]+\.[a-z43]+$ pattern
Error#    14,log#  19536: The format of si does not follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    15,log#  19537: The format of sha should follow ^[a-fA-F0-9]$ pattern
Error#    16,log#  22551: The format of nm  does not adhere to ^[a-zA-Z- ]+\.[a-z43]+$ pattern
Error#    17,log#  22795: The length of sha should be 64 instead of 40 for 4ba16304f848540cc59f2688a4396d726e3bab48 sha value
Error#    18,log#  24283: The length of sha should be 64 instead of 40 for 4523a51f3cec51bd41c4ba117ced371fc5d66d1a sha value
Error#    19,log#  26305: The format of sha should follow ^[a-fA-F0-9]$ pattern
Error#    20,log#  27392: The format of sha should follow ^[a-fA-F0-9]$ pattern
Error#    21,log#  27986: The format of bg should follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    22,log#  36561: The format of uu should follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    23,log#  36712: The format of nm  does not adhere to ^[a-zA-Z- ]+\.[a-z43]+$ pattern
Error#    24,log#  36771: The format of nm  does not adhere to ^[a-zA-Z- ]+\.[a-z43]+$ pattern
Error#    25,log#  40609: The format of sha should follow ^[a-fA-F0-9]$ pattern
Error#    26,log#  41638: The format of bg should follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    27,log#  41649: The format of sha should follow ^[a-fA-F0-9]$ pattern
Error#    28,log#  43360: The format of uu should follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    29,log#  45768: The format of sha should follow ^[a-fA-F0-9]$ pattern
Error#    30,log#  49038: The format of nm  does not adhere to ^[a-zA-Z- ]+\.[a-z43]+$ pattern
Error#    31,log#  50702: The format of nm  does not adhere to ^[a-zA-Z- ]+\.[a-z43]+$ pattern
Error#    32,log#  52369: The value of dp should be 1 or 2 or 3 instead of 65534
Error#    33,log#  56660: The value of dp should be 1 or 2 or 3 instead of 65534
Error#    34,log#  59830: The value of dp should be 1 or 2 or 3 instead of 65534
Error#    35,log#  60260: The value of dp should be 1 or 2 or 3 instead of 65534
Error#    36,log#  60991: The format of sha should follow ^[a-fA-F0-9]$ pattern
Error#    37,log#  63040: The format of si does not follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    38,log#  67104: The format of si does not follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    39,log#  67258: The format of nm  does not adhere to ^[a-zA-Z- ]+\.[a-z43]+$ pattern
Error#    40,log#  70266: The format of nm  does not adhere to ^[a-zA-Z- ]+\.[a-z43]+$ pattern
Error#    41,log#  71185: The format of nm  does not adhere to ^[a-zA-Z- ]+\.[a-z43]+$ pattern
Error#    42,log#  71236: The format of nm  does not adhere to ^[a-zA-Z- ]+\.[a-z43]+$ pattern
Error#    43,log#  73441: The length of sha should be 64 instead of 40 for edf5676fb32babdd21551da56d2b734793de9362 sha value
Error#    44,log#  75605: The format of bg should follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    45,log#  80439: The format of uu should follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    46,log#  80957: The format of uu should follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    47,log#  84788: The length of sha should be 64 instead of 40 for 623e3c13a93a8880ac62c459f65df7f8fb63b834 sha value
Error#    48,log#  85047: The format of uu should follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    49,log#  85115: The length of sha should be 64 instead of 40 for ee66860694ed8904108b029916c6c7cfa793e769 sha value
Error#    50,log#  85200: The format of bg should follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    51,log#  88079: The format of nm  does not adhere to ^[a-zA-Z- ]+\.[a-z43]+$ pattern
Error#    52,log#  88590: The format of bg should follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    53,log#  89554: The format of uu should follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    54,log#  90154: The format of bg should follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    55,log#  92087: The format of bg should follow ^[a-f0-9]{8}-[a-f0-9]{4}-[1-5][a-f0-9]{3}-[89ab][a-z0-9]{3}-[a-f0-9]{12}$ pattern
Error#    56,log#  95552: The format of nm  does not adhere to ^[a-zA-Z- ]+\.[a-z43]+$ pattern
Error#    57,log#  97151: The value of dp should be 1 or 2 or 3 instead of 65534


100000 logs scanned from which 57 number of values have issues while 99943 entries reported clean
DB connection closed successfully
```

Figure 2.2: A Screenshot of invalid values shown in terminal

Chapter 3

Unit Test

The unit test suite does 2 things

1. Assert that the valid logs return True when passed to the test_json_validator().

2. Assert that the in-valid logs return False when passed to the test_json_invalidator().

Thus there are 2 functions one of which generate the valid set of values and other one will generate invalid logs. The valid logs are generated using following code

```
ts  = calendar.timegm(time.gmtime())                                         # generate value of ts
sha = ''.join(random.choice(('abcdef') + string.digits) for i in range(64))  # generate value of sha
pt  = random.randint(1,100)                                                  # generate value of pt
si_uu_bg = ''.join(random.choice(('abcdef') + string.digits) for i in range(8)) + '-' \
           + ''.join(random.choice(('abcdef') + string.digits) for i in range(4)) + '-' \
           + ''.join(random.choice('12345') for i in range(1)) + ''.join(
           random.choice(('abcdef') + string.digits) for i in range(3)) + '-' \
           + ''.join(random.choice('89ab') for i in range(1)) + ''.join(
           random.choice(string.ascii_lowercase + string.digits) for i in range(3)) + '-' \
           + ''.join(random.choice(('abcdef') + string.digits) for i in range(12))  # generate value of si, uu and bg

nm  = ''.join(random.choice(string.ascii_letters) for i in range(10)) + '- ' + '.' + \
        ''.join(random.choice(('43') + string.ascii_lowercase) for i in range(4))  # generate value of nm

ph  = ''.join(random.choice(string.ascii_letters) for i in range(10)) + '- /' + '.'+\
      ''.join(random.choice(('43') + string.ascii_lowercase) for i in range(4))    # generate value of ph

dp  = random.randint(1,3)                                                    # generate value of dp
```

```
valid_log_entry = {'ts': ts,'pt': pt,'si': si_uu_bg,'uu': si_uu_bg,'bg': si_uu_bg,'sha': sha,'nm': nm,'ph': ph,'dp': dp}
self.valid_log = valid_log_entry
```

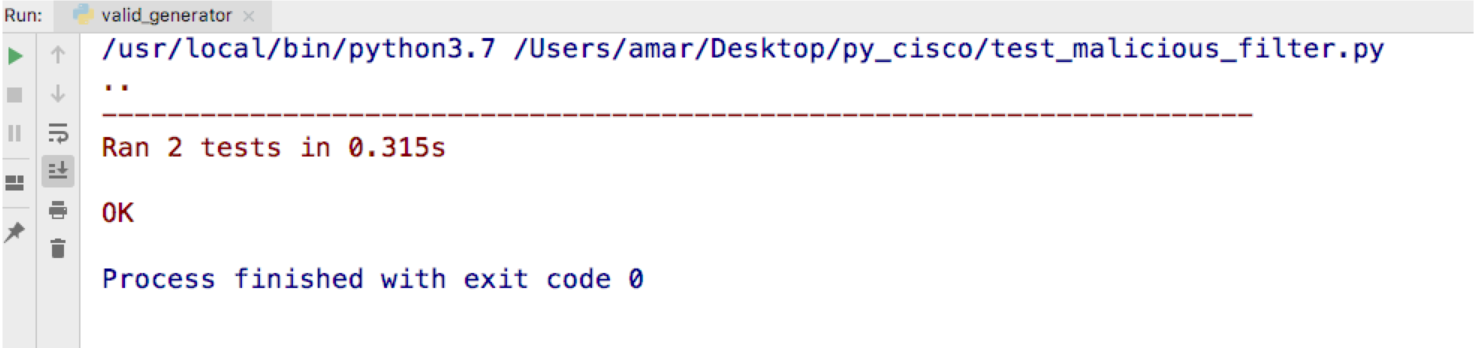
Also, the invalid values are generated using below code

```
 #   ---------- Generating invalid logs ----------
in_ts = random.randint(19, 199)                                                                              # generate value of ts
in_sha = ''.join(random.choice(('qwertyuiopasddg') + string.digits) for i in range(62))                     # generate value of sha
in_pt = random.randint(100, 200)                                                                             # generate value of pt
in_si_uu_bg = ''.join(random.choice(('qwertyuiopasddg') + string.digits) for i in range(8)) + '-' \
                + ''.join(random.choice(('qwertyuiopasddg') + string.digits) for i in range(4)) + '-' \
                + ''.join(random.choice('1234234234245') for i in range(21)) + \
                ''.join(random.choice(('qwertyuiopasddg') + string.digits) for i in range(31)) + '-' \
                + ''.join(random.choice('qwertyuiopasddg') for i in range(17)) +\
                ''.join(random.choice(string.ascii_lowercase + string.digits) for i in range(3)) + '-' \
                + ''.join(random.choice(('qwertyuiopasddg') + string.digits) for i in range(12))            # generate value of si, uu and bg

in_nm = ''.join(random.choice(string.ascii_letters) for i in range(140)) + '- ' + '.' + \
        ''.join(random.choice(('qwertyuiopasddg') + string.ascii_lowercase) for i in range(42))             # generate value of nm

in_ph = ''.join(random.choice(string.ascii_letters) for i in range(50)) + '- /' + '.' + \
        ''.join(random.choice(('qwertyuiopasddg') + string.ascii_lowercase) for i in range(46))             # generate value of ph

in_dp = random.randint(1, 3)                                                                                 # generate value of dp

invalid_log_entry = {'ts': in_ts, 'pt': in_pt, 'si': in_si_uu_bg, 'uu': in_si_uu_bg, 'bg': in_si_uu_bg, 'sha': in_sha, 'nm': in_nm,'ph': in_ph, 'dp':
    in_dp}

self.invalid_log = invalid_log_entry
```

Then both of these values are tested x(10000) number of times as shown below

```
def test_json_validator(self):
    for i in range(10000):
        self.assertTrue(malicious_filter.json_validator(self.valid_log))                                         # testing the correct values

def test_json_invalidator(self):
    for i in range(10000):
        self.assertFalse(malicious_filter.json_validator(self.invalid_log))                                    # testing the in-correct values
```

A passed test case is shown below.



```
/usr/local/bin/python3.7 /Users/amar/Desktop/py_cisco/test_malicious_filter.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.315s

OK

Process finished with exit code 0
```

Figure 3.1: A Screenshot of test case pass in which both valid and invalid tests are passed