**SherLock vs Moriarty**

Kent Campbell, Ethan Dark, Kenny Freeman, Brandon Julian, Nick Milliken

**Abstract**

This project explores the idea that low level cellphone sensor data can predict whether a malicious activity is occurring or not. The project uses experimental data acquired for cyber security testing to follow as best possible, the published work from [Sherlock vs Moriarty](http://library.usc.edu.ph/ACM/SIGSAC%202017/aisec/p1.pdf). Multiple logistic regression models and cluster analysis models were run. Single predictor logistic models were proof of concept approaches. The complex logistic model was successful, though too late we realized we had passed a column that should not have been included. Cluster analysis models of the T2 accelerometer and geolocation data were attempted to try and visualize the stolen phone event and help continuous user authentication proof of concept.

**Business Understanding**

The motivation for the [Sherlock vs Moriarty](http://library.usc.edu.ph/ACM/SIGSAC%202017/aisec/p1.pdf) experiment is to help security professionals and academics create new methods of detecting malicious behavior in cell phones. This type of malware detection is important for many reasons, some of which include: attacks against personal devices, the overall expense of typical monitoring and high value asset protection.

The number of attacks against personal devices is ever increasing. This is due to smartphone capabilities rapidly expanding in functionality and with this comes more opportunities to access personal data. Security and device integrity is key for personal information from both the sellers and buyers point of view. A company will want to sell a product that they feel safe using and a buyer wants to know their personal data is safe.

Due to the ongoing arms race between attackers and defenders in the cyber security world there is already a huge market for cyber security. The problem with this market is that the cost of reliable or top of the line services will usually cost a premium that not everyone will have the luxury of affording or technical skill of installing.

Third, cell phones now contain most financial, personal, and frequently private personal information. Whether it be bank account information, social security numbers, website logins or important business files it is imperative to keep that all safe and secure.

This experimental data is created as a test bed for algorithm development and low level sensor understanding. This project seeks to model the hardware response to malicious software behavior using data that is inherent to modern phone functionality.

**Data Understanding**

The [Sherlock vs Moriarty](http://library.usc.edu.ph/ACM/SIGSAC%202017/aisec/p1.pdf) dataset is a long term smartphone experiment collecting sensor data of cellular devices while malware experiments are being conducted through. The full [SherLock](http://bigdata.ise.bgu.ac.il/sherlock/) dataset contains over 10 billion records and a subset of the data can be found on [Kaggle](https://www.kaggle.com/BGU-CSRC/sherlock). This data offers a groundbreaking view of signals and inputs within cell phones that had never been previously collected all together before. Some of the low level monitorable features that were collected included: whether or not the screen was one, device statistics down to the linux level, statistics from each running app, battery and power consumption, and accelerometer data.

The full dataset includes 13 different tables that all focus on different aspects of the android phones operating functions. For our project we used the following tables: Call Log, SMS Log, Screen Status, User Preference, App Packages, Moriarty, T0, T1, Wifi, Bluetooth, T2, T3, and T4. The range of tables allows us to mix and match based on what type of problems we were looking to solve and the Modeling Section will describe the sensor contents investigated.

Within the Sherlock vs Moriarty dataset there is data that belongs to two distinct types of sensors. The first is Push sensors which are event based and record the associated details when they happen such as an SMS arrival or the screen turning on. The second type is a Pull/Poll sensor which records at set intervals irrespective of Moriarty actions. For example some of the device statistics from each application running on the phone will be recorded every 5 seconds for months on end.

The information gathered throughout the experiment is designed for the predictability of Moriarty’s behavior (whether malicious activity is happening). Within the Moriarty dataset there are two important columns that two of the three models we designed will predict off of. These labeled classifications are the Action Type and Session Type which will either be labeled as ‘malicious’ or ‘benign’. Action Type is the specific action taken by Moriarty and Session Type is the windowed experiment time period; there can be many Action Types, with both ‘malicious’ or ‘benign’ behavior in a Session. We do not have the key to the Session Types, so the details of the experiment are beyond scope.

Another critical label within the data is the ‘uuid’ which is located in every table. This ‘uuid’ column represents the unix millisecond timestamp of when the record was collected. These timestamps allow for cross analysis between tables and allow deeper analysis during the model building stage. In addition to the advantages these timestamps provide, they also may cause some issues when trying to merge datasets where every row doesn’t match up, which is the case for 99% of the data. Details of this issue will follow in the Data Preparation Section.

Data used for this project evolved over time. The project initially began using the Kaggle dataset linked above. This size of this dataset is 468.11 MB. Subtle differences were noticed and Dave Eargle, CU-Leeds faculty, offered access to a DropBox containing [2016-Q1 data](https://www.dropbox.com/sh/f562nmpexcibv40/AADGDWYaSwKdLOnnO3MGq0wna/2016_Q1?dl=0&subfolder_nav_tracking=1) (access granted with permission from DE). This data was the full dataset spanning this time period, but missing a key folder, ‘Apppackages.tsv’. However, in addition to the large size (some individually sized in the 10’s of GB) of these files, data corruption/formatting issues caused severe slowdown in progress. Steps taken for this ‘Extract-Transform-Load’ (ETL) are described below. Synchronously with the effort to remedy the mis-format issue, a parallel effort was underway to download a different time period, [2016-Q3](https://console.cloud.google.com/storage/browser/security-analytics-datasets/sherlock;tab=objects?prefix=&forceOnObjectsSortingFiltering=false) (access granted with permission from DE), that did not contain formatting issues and complete sensor tables from a google shared drive provided by David Eargle with data from the original researchers. Due to the large file sizes, time windows of 2 weeks were extracted from both datasets to get down to a reasonable level. All Kaggle data and 2 week windowed 2016-Q1 data were successfully uploaded to the Leeds’ cluster, but upload bandwidth and cluster instability prevented several of the larger 2016-Q3 files to be uploaded, though, some code development was conducted on this data in anticipation of eventually using this better suited dataset. Cluster access instability and performance instability challenged this project in terms of our accessibility to the data stored on the cluster and code development, hence the cobbled nature of our models. We apologize for this inconvenience.

**Data Preparation**

The initial Kaggle dataset was fit for immediate ingestion. However, ETL-style preparation was carried out in the MacOS environment for the larger files. To reduce file size to be uploadable to the Amazon cluster (constraint of bandwidth between home and cluster), a 2 week time window was extracted throughout the .csv or .tsv formatted data files. This extraction used the command structure:

***awk '{ if ($2 > <MINUUID> && $2 < <MAXUUID>) print $0 }' <INPUTFILE> > <OUTPUTFILE>***

With the minimum and maximum UUID corresponding to the beginning and ending Unix Seconds for the 2 week time period.

The T4 dataset originating from DE’s DropBox required reformatting with *sed*, as captured in the comment cell in the JupyterNB. This was done instead of REGEX within the PySpark environment since this was ‘one off’.

**sed 's/\\N/NULL/g'*<INPUTFILE>*|sed's/null/NULL/g'|sed's/\ MHz/\_MHz/g'|sed's/\ GHz/\_GHz/g'>tmp**

**sed's/-01 /-01\_/g'tmp|sed's/-02 /-02\_/g'|sed's/-03 /-03\_/g' | …**

**… sed's/-31 /-31\_/g'|sed's/ /,/g'|sed's/\_GHz//g'|sed's/\_MHz//g'|sed 's/\_/ /g'> *<OUTPUTFILE>***

Data ingestion into PySpark Notebook on the clusters, of both .csv or .tsv formats, was from files uploaded to the S3 storage from the command line. Once data was subsequently conditioned, data was exported in parquet format in the S3 storage.

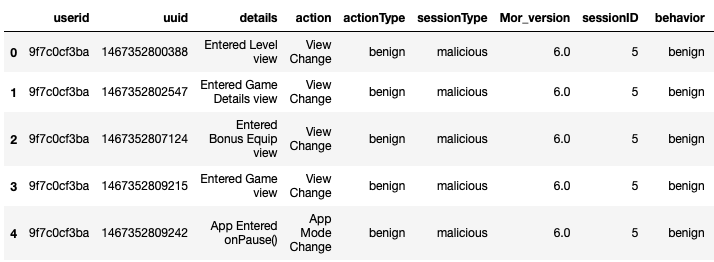
Most sensor tables required similar data conditioning steps. Column assignment, labeling and casting, along with integer conversion of the boolean ‘true’/’false’ and ‘malicious’/’benign’ labeling was usually conducted before merging for several of our team members.

The merging operations required more attention than expected. Initially, UUID, the Unix Second timestamp, was assumed to span uniquely across all tables as a linking key. However, this was not the case as the Push (i.e., Moriarty) sensors were timestamped with fractional seconds and the Pull sensors are at their sample rates. After much failed experimentation, an ‘outer’ merge approach was implemented and with a ‘forward fill’ philosophy. It is acknowledged this is error prone, but time constraints forced this decision.

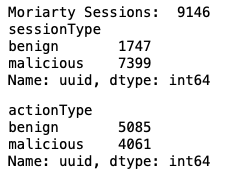
We believe the main representative Jupyter Notebook to evaluate is the Final\_Main\_PySpark.ipynb. This Notebook contains many of the project requirements.

**Modeling**

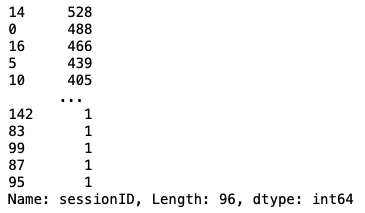
Exploratory Data Analysis (EDA) is always the first step towards understanding data before undertaking modeling. While we do know the researchers conducting the experiment know the exact time, type and purpose of the Moriarty events, we do not. So, to gain an understanding of the neerdowell, let’s look at the first few rows of the 2016-Q3 Moriarty Table (SherLock\_local\_PythonEDA):



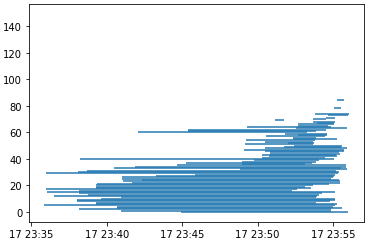
We see that within the 2016-Q3 Moriarty table, the breakout of benign and malicious actionTypes and sessionTypes were:



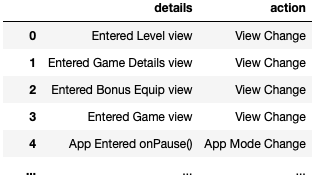
In the 2 week window of time, the top 5 sessions containing the most actions were 2016-Q3:



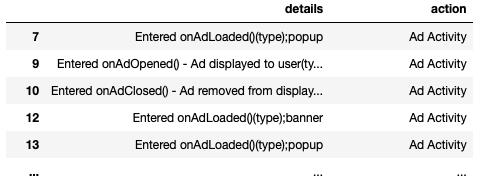
The sessions, spanning across all users, span (note pyplot unable to reconcile datetime format correctly):



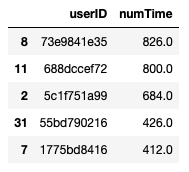
So, a few details of Benign actions looks like:



And Malicious:



For the participants in this Q3 time period, we see that Moriarty acted more than 800 times on some user phones:

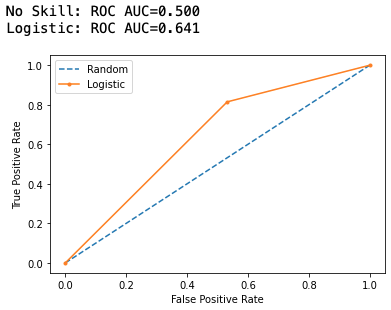


Several models were tested on various Push and Pull sensor tables. These models range from simple-proof of concept tests to more complex merges of data and algorithms. Starting with the simplest, but undertaken as a proof of concept is a single predictor of screenOn and Moriarty sessionType.

**SIMPLE LOGISTIC REGRESSION:**

Model 1a,b:

The Python (SherLock\_local\_PythonEDA.ipynb) implementation shows that the screenOn predicting sessionType has almost no predictive capability, being almost a random predictor.



And for this data, the actual correlation of screenOn and sessionType is 0.15.

The PySpark implementation (FINAL\_cluster\_PySparkMain) :

The simple logistic regression in pyspark returned perfect results since it was run on consistent predictors. This was a trial and was removed after execution.

**COMPLEX LOGISTIC REGRESSION:**

Model 2: (Final\_Main\_PySpark.ipynb)

The first step of the logistic regression in an areaUnderROC of 79.88% and an areaUnderPR of 0.02%. After adding a cross validation parameter, the metric went up to 79.91%. Adding a chi square selector or a random forest classifier both resulted in 100% evaluation metrics.

**CLUSTERING:**

Model 1: T2 silhouette analysis using PySpark on the cluster (silhouette\_cluster\_final.ipynb)

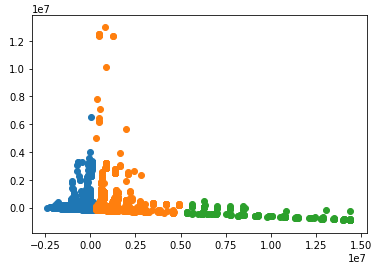
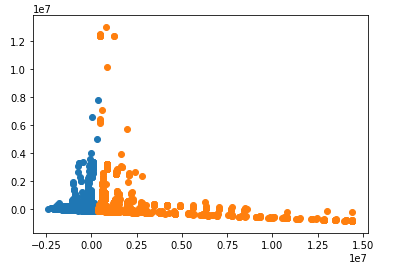
To help solve the problem of using unsupervised learning to predict whether a participant's phone had been stolen we decided to use KMeans clustering analysis. To do this the clustering model was based off of the data from the T2 dataset which mostly contained accelerometer and geolocation data. The hypothesis here was if we could see multiple clusters that didn’t hold much overlap, we could confidently say that one cluster represents normal activity and another could represent malicious activity. One approach to this problem is creating the clusters based solely on the accelerometer data and getting the Silhouette scores which measure how close each point in one cluster is to points in the neighboring clusters thus helping in figuring out clusters that are compact and well-spaced out. The silhouette scores can range from -1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters.

To begin this process all of the columns that are going to be used must be converted to the Integer Type from String Type and any null values must be removed. At this point the model this will be working with is 67,453 rows. Once that is done those columns must go through a Vector Assembler and transform all those columns into a single vector column. Next that vectorized column will be scaled using the Standard Scaler function and then fitted and transformed. At this point the data can be put through a Clustering Evaluator with the silhouette function included and tested.

The results from the evaluator produced eight unique clusters with a wide range of silhouette scores. There was one cluster with a very high score of .78 showing that it was truly unique and separated from the rest. There were three more clusters with a score of .4 or higher. The last four clusters had a score of .38 or lower and one even had a score of .18 which shows they were not as strong of clusters and could have overlap.

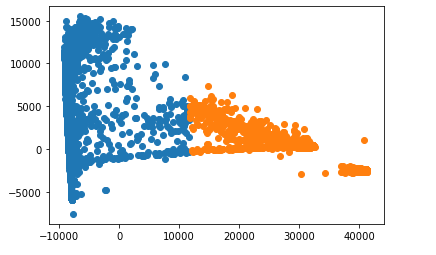
Model 2: T2 Accelerometer Clustering and PCA analysis (T2ClusterAnalysis.ipynb)

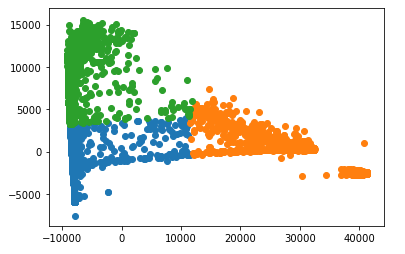
Using a T2 dataset sample of user 97bb95f55a spanning the course of a month of the experiment we conducted KMeans clustering analysis to see if the event of the participants phone being stolen could be identified and if the hypothesis of continuous authentication could be valid. Due to the immense dimensionality of the dataset of over 200 features, clustering using the majority of features must be accomplished with principal component analysis. We proceeded with unscaled singular value decomposition to apply to KMeans clustering. With K = 2 and also K = 3 the following plots are created.



Without time as an observable feature and a perfect understanding of the principal loading scores the meaning of this visualization can be misleading. However, at first glance this may be indicative of the stolen phone event. If the loading scores of PC 2 show a large degree of dependence on accelerometer data then the spike visible along the Y axis may be representative of the event and the variation on the X axis may be changes in phone use habit of an unauthorized user. The separation of the clusters could be the breaking of continuous authentication and the settling of new habits of the new user.

Removing features with the suffix ‘Gyroscope’, and ‘Pressure’ chosen for their low loading scores and clustering again at K = 2 and 3 results in the following graphs.





These clusters may be less of a proof of concept compared to the previous ones that incorporate all the features. There are visually less obvious patterns that would be indicative of the scenario. Referencing the paper *‘SherLock vs Moriarty: A Smartphone Dataset for Cybersecurity Research’*, “(1) the extracted motion features are fed one-by-one into a stream clustering algorithm, (2) the sequence of outputted cluster IDs are applied to a Markov chain to compute transition probabilities, and (3) the anomaly scores are computed with the outputted transition probabilities.” (Page 8, Section 4.2.2). Their clustering approach was extremely complex and utilized all motion features. Drilling down further on loading scores here would be fruitless in the effort to mimic their results.

**Evaluation**

Simple Logistic Model Python Evaluation:

This proof of concept has the expected predictive power in alignment with the poor correlation 0.15 between the screenOn and sessionType. Naive intuition was that Moriarty would have more actions when the user was away or phone was not in use.

Simple Logistic Model Pyspark Evaluation:

The Pyspark implementation was meant as a trial model to see how to code in predictions. The model evaluator showed no flaws in the data which was expected with only one predictor. Now that we knew the basics of how to run models on our dataset, we were able to add complexity.

Complex Logistic Model Pyspark Evaluation:

Since Pyspark machine learning does not handle nulls well, we started by manipulating code to fill in nulls. Since the target is observed in instances, it was safe to assume all null values in the target were zero. However, for CPU and ScreenOn, instances are recorded when the value changes. That being said, we needed to add a forward fill. We created a function that would iterate through the dataset and pull the last value into null values for each column. After having a full dataset, we were able to begin our machine learning.

To begin running the model, we needed to make sure all the predictors were numeric. We had one categorical variable, ‘Action’. We were able to include ‘Action’ in the model by first assigning each factor level an index with StringIndexer and then simulating dummy coding with OneHotEncoder. Now that all predictors were numeric and continuous, we applied VectorAssembler to add all values to a ‘features’ column. To wrap up the first part of the model, we took a LogisticRegression on the target variable and split the dataset to fit and transform a pipeline. The evaluations were promising with an areaUnderROC of 80%, but there was more tuning to include.

The next step we took was hyperparameter tuning with a CrossValidator. However, adding this barely increased our accuracy. To try to impactfully increase accuracy, we included ChiSqSelector for train-validation splitting. Choosing five top features, this model skyrocketed our accuracy. The areaUnderROC went all the way to 100% by this point. The RandomForestClassifier had the same effect. With five trees, the areaUnderROC was 100% again.

Looking back on the evaluation of this model, we regret including ‘Action’ in our list of predictors. The moriarty dataframe was malicious events only, so obviously a column coming from that dataframe would always output accurate results. With time and cluster limitations, we were not able to rerun the model with the variable removed.

Silhouette clustering evaluation:

With four unique clusters it seems as if there could be some malicious activity occurring but there must be more ETL work done to get accurate results. In this case the data actually contained multiple users which could easily produce a variety of clusters along with this model only using the accelerometer data and no geolocation data. If more data from the study was accessible then the model could run based on one user and more columns to produce silhouette scores that could accurately predict whether something malicious is occurring.

T2 Accelerometer Clustering and PCA analysis Evaluation:

The original plots could possibly be visually misleading due to the dimensionality of the data. As people who are aware of the scenario of the stolen phone it is easy to pick a pattern in the data and assign meaning. But without applied knowledge of the loading scores of the principal components a visual conclusion should probably not be reached. The original researchers were able to create the continuous authentication algorithm using a much larger sample than we did, nearly the entire set from over a year's worth of data. That entire set was clustered using a tool called pcStream (created by most of the authors of the Sherlock Vs Moriarty paper) and then an incrementally updated Markov chain was used to create the transitional probabilities given the last seen input. This technique is a little more advanced than KMeans clustering so it would be surprising if our model is representative of the event.

In retrospect it would have been interesting to try out the tool pcStream and see if the visual representation (or numerical insights) of the data would have been at all different. pcStream would have been able to incorporate a temporal aspect to the data that KMeans cannot. It is also possible that the visualization of pcStream would be identically uninterpretable since the dimensionality would be similar to KMeans, thus requiring PCA again or some other form of linear dimension control.

**Deployment**

The implications of the results above prove that with the right diagnostics and monitoring software it is possible to track malicious activity on the metadata level as well as physical level. Although the experiment calls for an extra battery to provide power for the monitoring ability, it shows that the concept of full monitorability is possible and could be feasible to implement in the future as devices advance in capabilities. The foundation of this type of project hinges on the data being usable with the ability to combine all 13 dataset. If this data monitoring program was to be used in the future then functional changes to how the data is stored would need to occur. As of now the data can be challenging to merge and when you are searching for deeper insights such as clustering or advanced modeling then limiting the ETL process should be a top of mind focus.

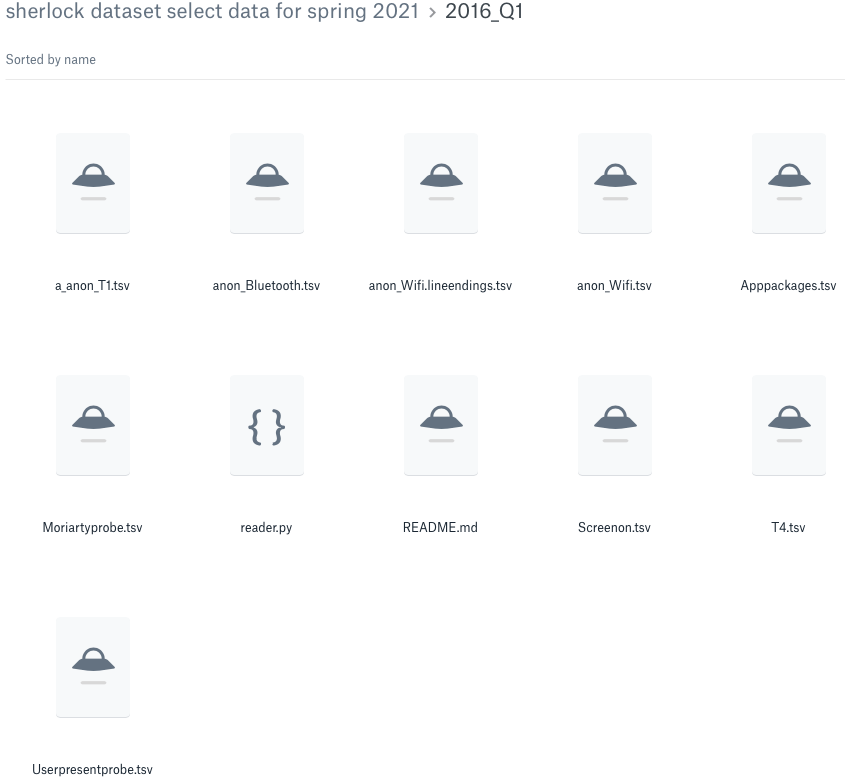
Even though we were unsuccessful in a true proof of concept of our own user authentication through cluster analysis; The importance of this concept is still massive. The implications of being able to track, save, and develop an authentication algorithm purely using motion sensor data produced by habit are next-level. If the issue of data collection and storage can be solved for the everyday user to allow the algorithm to be developed - personal computing (physical) security can approach being perfectly secure. The authors of *‘SherLock vs Moriarty: A Smartphone Dataset for Cybersecurity Research’* were successful in an experiment and the next step would be deployment of just this section of their research. Without the other bulky data collection the issue of battery consumption might be negligible and this concept can be deployed to a larger audience and studied long-term.

The project tested the technical waters and with more time, we feel we could refine our existing models and pursue other approaches that would isolate the signal that is Moriarty in the multitude of sensor data.

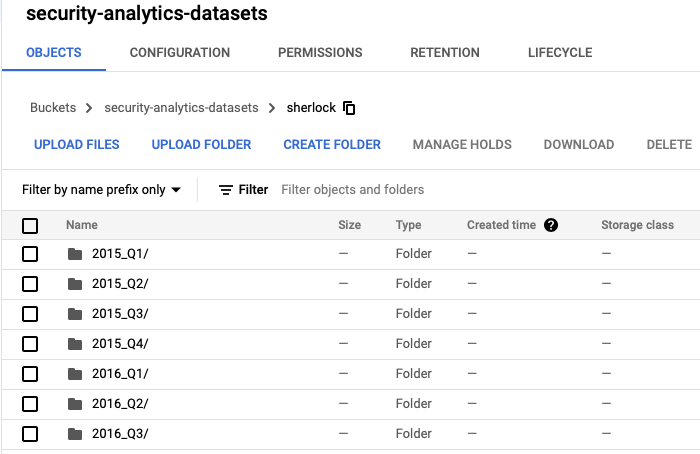
**Data:**

[Kaggle](https://www.kaggle.com/BGU-CSRC/sherlock) (On cluster)

DropBox: (On cluster)

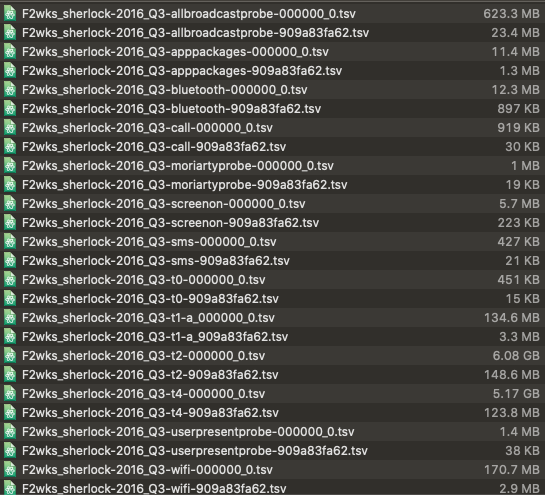


GoogleDrive:



2016-Q1 Pieces on cluster, S3

2016-Q3 (full 2 week filter = ‘000000’ series & user)

****