Outlier Detection for Memory Accesses

VINAY AWASTHI* and WOODROW WANG*, CS221 Stanford Univ., USA

Presenting a way to detect program execution out-lier behavior, based on how pages/memory zones are accessed, during program execution.

Additional Key Words and Phrases: datasets, neural networks, outlier detection

ACM Reference Format:
Vinay Awasthi and Woodrow Wang. 2020. Outlier Detection for Memory Accesses. *ACM Trans. Graph.* 37, 4, Article 111 (August 2020), 5 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The way program starts and ends had well defined memory access patterns. These memory read and write access patterns can be pre analyzed into well defined clusters. These clusters can then be passed on to "`IOT`"devices. IOT devices are, now a days, ubiquitous with very little real-time safety measures builtin as software installed can only protect against known threats.

With new threats being discovered daily, IOT installers are left with only few choices.

- `Obfuscate`: Wind-river - closed source.
- `Update Often`: IOS/Android patches.
- `Deny Access`: Hand carry USB stick, once in 2 weeks, to Oil Riggs, to get sensitive drilling data. (Exxon/Shell).

In all of above mentioned security modes, IOT devices in question, need constant care. Operators, SW/HW service providers, supply chain logistics all need to sign off at hundreds of places to ensure safety.

With trillion of these devices coming online with arrival of 5G networks, in next two to four years, this approach, will not scale and once breached, there is no way to stop/correct data flow out of these devices.

One of the ways to manage risk is to tell device, what is a normal mode of operation. We propose a low cost out-lier detection based on memory access that is specialized for a given device and its use-case. This way device can take corrective actions when those parameters of normal mode execution are compromised.

Since IOT Devices only run one key application such as, collecting sensor data, making credit card transactions, and doing basic analytics, before uploading data to cloud for offline analytics (Banks running fraud detection) [2]. The "`Outlier detection`" will monitor, specific for that IOT device, application, memory-access zones. Upon detecting violations, it will trigger faults – "Anomaly Faults" to desired *executive-layer* for further action.

Imagine a case, where Point of sale system processes credit cards at certain rates for various hours of the day when business in open. If POS system detects, large set of small transactions, it can alerts banks, shop owner as transactions are being attempted rather than after the fact. Following story is illustrative of how low quality cyberattacks are bankrupting small businesses. https://www.wsj.com/articles/a-small-business-with-no-working-website-felled-by-a-cyberattack-11559490543

2 TECHNOLOGY OVERVIEW

As noted in the introduction, the "`Outlier Detection`", here is demonstrated on memory access clusters identifications for Java Virtual Machine based application, a common choice for IOT devices.

I will use PyOD[4] https://github.com/yzhao062/pyod to analyze memory access and create clusters of accesses for those memory accesses highlighting and correlating zones of memory along with program execution status as program runs through out the day.

There are three types of access patterns that trigger flurry of activity — a start up time activity, when program is loading shared libraries, compiling JITed code or running interpreter, then — *analysis - normal*a steady state execution state, where program (service) is now executing its intended function. Finally a terminating state, — *analysis - terminal* here IOT device attempts to get to good known state, by sending all pending data to cloud, before shutting down.

In this project, we will focus on *analysis - normal* and *analysis - terminal*. Data set is unlabeled. While performing hypothesis (classification about normal execution), we assume that data was collected in safe environment and data is free of any anomaly.

2.1 Cluster Classification

"`Out-lier Detection`" program, in classification phase, analyzes memory accesses *sampled at various rates* corresponding to most used parameters, generating a list of clusters and respective sizes used (2 MB pages, 64 MB zones etc..). These normal execution access classification parameters the will be passed on to IOT Device as part of SW update/installation.

Almost all attacks, start with installing a backdoor to analyze files and scan servers (open ports, ARP (Address Resolution Protocols) tables, virus scanning services running) to see what is running when and where first. Most of these back-doors go undetected for upto atleast 8-9 months before any attack is carried out, attackers analyze network and its vulnerabilities and first thing attackers do is kill all virus scanning services, services sending log files up, triggering alarms etc... These kinds of access can easily be triggered as anomalous using this scheme. use the `Execution` Anomaly Detection:

*Both authors contributed equally to this research.

Authors' address: Vinay Awasthi, v365747@stanford.edu; Woodrow Wang, wwang153@stanford.edu, CS221 Stanford Univ., P.O. Box 1212, Palo Alto, California, USA, .

- `Severity`: If we have incorrectly classified something, we can overwrite this in next release.
- `Time`: Most attacks take place in the morning (5 a.m. Saturday) or on the weekends.
- `Privilege Level`: Many banks are now seeing attacks from the inside (Wells Fargo, BOFA etc..).

For all of these attacks, ideally no SW is needed as these patterns can be directly encoded to MMU/FPGAs.

`Out of Scope, in this paper..`

- `FPGA`: How MMU + FPGA will monitor zonal accesses, preventing scans.
- `Bypassing Mechanisms`: Quick way to disable noisy neighbor warnings.
- `Power`: IOT devices are very sensitive to power usage, this area is not covered here.
- `Networking`: Updating other IOT devices in neighborhood (broadcast Severity, Time, Privilege Level, cluster/size list to get ready to defend (other can shutdown ethernet or only allow access from few ports), once attack is discovered on one device.

2.2 Algorithm Used

I started with kNN then moved to LSTM-RNN [3] as kNN did not yield good clustering information. kNN is not helpful in time-series data with multiple local clusters (where loops are executing, accessing memory often before moving on to next page.

With (LSTM-RNN) Long Short-Term Memory Recurrent Neural Network (PyTorch implementation), I was able to predict correct program behavior provided I selected my window size carefully.

Frequently-used parameters, or combinations of parameters, include:

- `kNN`: Suitable for a "scattered data not time series data" I tired k = 10000, for a data window of 50,000 and even that was not helpful in identifying localities of accesses.
- `LSTM-RNN`: When windows size was small (2K records), this worked beautifully. Training model did not take long and predictions were spot-on.
- `kNN multi run clustering`: I tried Average of Maximum (AOM), Maximum of Averages (MOA), averaging multiple clusters sizes (ranging from K = 10 to K = 200) runs. These runs also did not provide any meaningful out-lier detection.

3 ADAPTATION

Literature Review:

http://web.mst.edu/ wjiang/SkNN-ICDE14.pdf

While this paper does not cover issue at hand directly, it does cover areas on how to get information in an encrypted environments. As Intel, AMD etc.. move towards secure encrypted virtualizations (SEV). Direct access to data may not remain possible. This paper shows how one can run secure kNN in encrypted environments. This paper lists how to compute secure squared euclidean distance, perform secure multiplications and finally how to compute secure minimums. With these 3 operations, one can implement SkNN in memory encrypted environments.

Charu C Aggarwal. Outlier analysis. In Data mining, 75–79. Springer, 2015 [1] This book lists many methods on how to perform outlier analysis. This book is used by PyOD as a reference to implement 20 or so methods to perform outlier analysis.

https://archive.siam.org/meetings/sdm10/tutorial3.pdf
By Hans-Peter Kriegel, Peer Kroger, Arthur Zimek.2010. This is a very well written presentation on various methods to do out-lier detection under un-supervised learning environments.

Dataset: I collected this data using Intel PMU performance counters on Large Java (function as a service) System (very similar to SAP) that uses 63+ Java Virtual Machines from 2 different vendors (Rocket Java Virtual Machine(JVM) and Oracle's internal JVMs along with 2 others JVMs that could not be ported by Oracle so these are used as is...). This data was collected to analyze heap/cpu usage 2 years ago.

I cleaned up data (using various shell commands such as tr, grep, uniq etc...) to only include data from few JVMs and reduce data from 60GB to 1048570 lines highlighting various memory accesses for 2 phases of application run. I am using first set to train and other set to test for out-lier detection. Cleaning up data was the most time consuming and delicate step of this whole analysis.

Baseline - I am using JVM (Java Virtual Machine) heap accesses as in cluster and set everything else as out of cluster data accesses. I then ran kNN using 20 or so sets using various k values ranging from 20 to 200. I can see JVM application's kernel memory accesses (higher memory address ranges) as well as, Java interpreter runs that uses different memory addresses as well as compiled function running correctly and satisfying expectations of outlier detection when JVM ran in a different phase (termination phase).

After running kmean with various k numbers [10..200] on full Million+ dataset for 4 hours using 100% CPU and over 2 GB memory, it became clear, that kNN can not be used realistically for 2 reasons.

- `Poor Classification`:
- `Computation and Memory`: Compute and memory usage is just too high for this methodology to be of any use at IOT class device which is expected to detect out-lier as it happens.

I then reduced this data set in to a sliding window set of 50,000 samples each. My compute issues evaporated but I realized that this time-series data does not show any meaningful clusters when applied, with-out much post processing, (raw data can be processed into BINs of access-frequencies to show memory pages accessed often and do clustering then). Using this approach, will require offline processing, (to create BINS for histogram analysis), which will defeat the objective of, stopping the attack, as program deviates from its intended use.

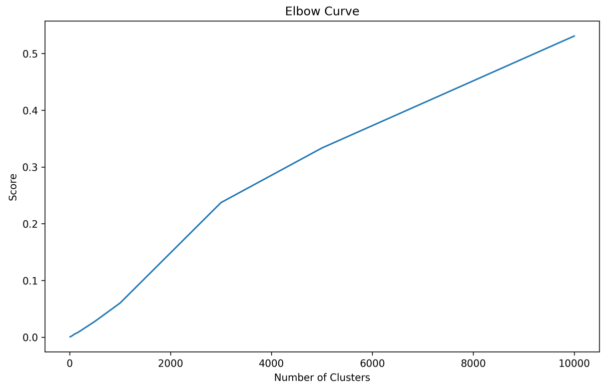
I also ran kmean clustering on this 50,000 reduced data window, thinking I may be able to utilize sliding window method for local analysis, with k value up-to 10,000 and did not see any elbow.

Results: Baseline ***kNN*** approach - kNN was performed in various ways to detect clusters. First I tried various cluster size values all the way upto 10,000. I also tried running kNN many times over various k values ***Model Combination*** and averaging distances to see if clusters could show up. These three methods were:

- kNN: Using 20 different values of k ranging from 20 to 200 and then taking maximum, median and average distances based nearest neighbor detection.
- MOA: Maximum of Averages.
- AOM: Average of Maximums.

I also realized that knowing about data is probably the most important aspect of data analysis. Tools are trying their best, making various assumptions along the way to show what they can, within the parameters passed.

I noticed this clearly when I plotted charts with various scales such as log to see data that was previously not obvious.



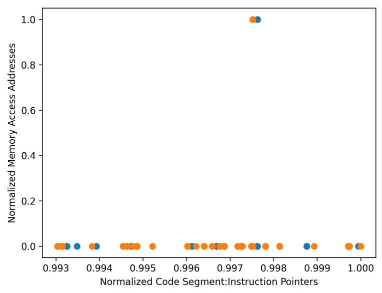
Obviously, no elbow value is in chart above, regardless of k value used. At this point, it became clear that kNN will not work here.

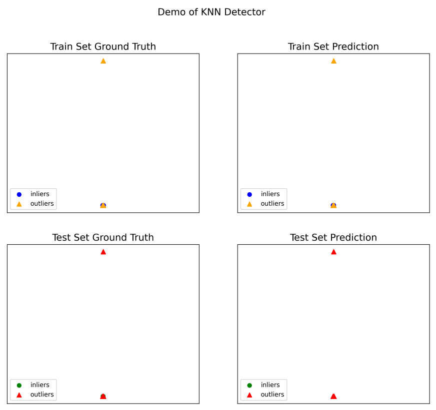
Evaluation Metric - All memory accesses beyond a threshold are tagged as outside cluster of normative behavior. I am still running my program runs and characterizing data.

Results & Analysis - Using 20 kNN i got following outcome. Computing took about 50+ minutes taking 3+ GB memory on Intel Ice-Lake processor.

Combining 20 kNN detectors

Combination by Average ROC:0.2772, precision @ rank n:0.9701
Combination by Maximization ROC:0.2772, precision @ rank n:0.9701
Combination by AOM ROC:0.2772, precision @ rank n:0.9701
Combination by MOA ROC:0.2772, precision @ rank n:0.9701

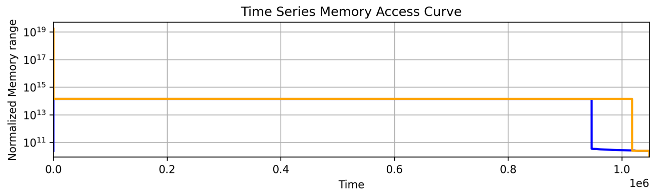


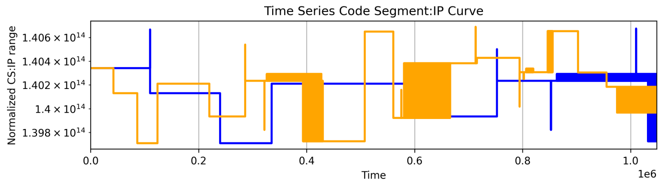


As charts indicate, these outliers are densely packed, so I need to do time-series analysis of these out-liers. Future Work - My next steps are to try LOF (density based outlier detection). I am inclined to try this method as when **loop optimizations**, take place compilers ensure data accesses remain local. This way if I see **hot pages** access, I can characterize it as app with N hot loops and if i happen to encounter an unknown program with characteristics, I can call it out as an outlier.

Code https://github.com/v365747/Outlier/blob/main/od.py and od_knn.py are 2 programs to run LSTM and kNN respectively.

Main LSTM-RNN approach - Long Short Term Memory Networks (LSTM-RNN) seems to be the next model of choice among researchers of time series anomaly detection. PyTorch has good set already implemented RNN-LSTM. First let us just look how JVM Code Segment and Memory access profile looks for training and test data.





Code Segments above show big variation as JVM runs interpreter, analyzes hot code, compiles hot code and finally then installs those translations in memory which then execute to access memory where data is for normal execution.

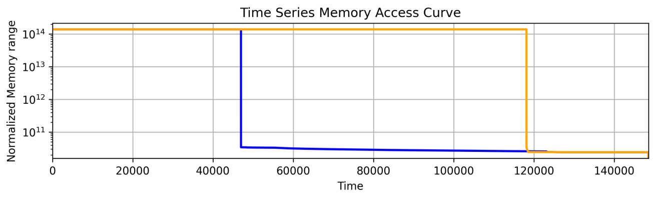
Since I ran JVMs in different states (steady state (training), during program termination (test), we are seeing large variation in $CODE\_SEGMENTS : IP$ values between two runs.

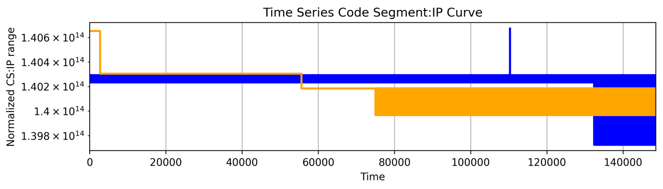
However for LSTM-RNN analysis first chart is showing that memory access remain fairly consistent (i.e. even in interpreter mode, data was continued to be accessed from similar memory regions).

This means we can train our RNN-LSTM based on steady state and if program behaves anomalously RNN will trigger out-lier trajectory of memory accesses behavior.

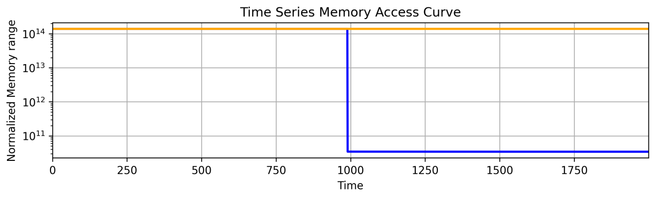
First, I will reduce data-set from 1 Million+ to 148570 items to actually focus on training/testing using LSTM-RNN as earlier data-set has no variation.

Here is how reduce data as part of, a sliding window method, while sampling memory accesses looks like. I am not using CS:IP, just showing it for completeness.



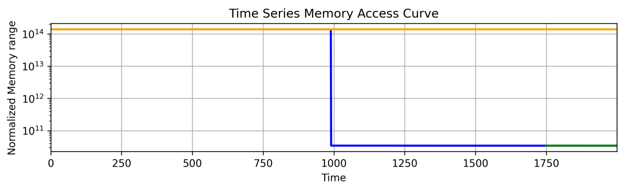


After running for 4 hours and not seeing even one epoch completing, it became clear that I need to seriously reduce my dataset as RNN-LSTM is very compute intensive when it comes to training. I then reduced my dataset to 2000 samples, where I can see memory data access diverging, as shown below.



Using this updated method, program is taking about 10 minutes and here are the results. Blue line is an actual intended memory access. Orange line is what is observed in test (where I need to flag invalid accesses). Green line is showing last 250 predicted values based on LSTM-RNN.

As you can see, model is accurately predicting accesses and will be able to flag incorrect trajectory of invalid memory accesses as they happen.



References -

- PyOD: https://github.com/yzhao062/anomaly-detection-resources
- PyTorch:https://pytorch.org

4 KEY LEARNINGS

"`Outlier Detection`": Get to know your data as much as you can. Analyze it in various ways to figure out various aspects as to how it changes and how to responds to various features. Tools make many assumptions and can only do so much, it is this first analysis that can save lot of time as then one can decide what tool to use.

5 CONCLUSIONS

Here `Outlier Detection` I will like to implement this for other applications to see whether, once trained, how accurately this model predicts for some other applications which are not based on Java Virtual Machine. I will also propose this method at Intel Corporation for "Proof Of Concept" work are independent vector to implement security.

6 ACKNOWLEDGMENTS

I will like to thank Woodrow Wang, Dorsa Sadigh and Percy Liang for teaching CS221, providing encouragement, reviewing out-lier detection methods, suggesting LSTM-RNN to use in this case, and finally doing all this under COVID-19 pandemic.

A RESOURCES

A.1 Code

All code is at *https://github.com/v365747/Outlier*.
od_knn.py is main driver file to run kNN using train.txt, test.txt as input. od.py is main driver file to run LSTM-RNN using train4.txt, test4.txt as input.

A.2 Data

- `Data Github`:Data is checked in at *https://github.com/v365747/Outlier*.
- `Training Data`:Training data is DLA_do_object_1.csv.Z (1.5 million samples)
- `kNN Training Data`:train.txt (50,000 samples)
- `LSTM Training Data`:train4.txt (2,000 samples)
- `Test Data`:Test data is DLA_OBJ_ITER_1.csv.Z (1.5 million samples)
- `kNN Test Data`:test.txt (50,000 samples)
- `LSTM Test Data`:text4.txt (2,000 samples)

B ONLINE RESOURCES

- `PyOD`:https://pyod.readthedocs.io/en/latest/example.html.

- `Elbow Detection`:https://towardsdatascience.com/time-series-of-price-anomaly-detection-13586cd5ff46
- `LSTM-RNN`:https://stackabuse.com/time-series-prediction-using-lstm-with-pytorch-in-python/
- `PyTorch`:https://pytorch.org/tutorials/beginner/nlp/

REFERENCES

[1] Charu C Agarwal. 2013. Outlier Analysis. *Springer* 1, 1 (Jan. 2013), 36–44. https://www.amazon.com/Outlier-Analysis-Charu-C-Aggarwal/dp/1461463955

[2] Yujing Wang Chao Yi Congrui Huang Xiaoyu Kou Tony Xing Mao Yang Jie Tong Qi Zhang Hansheng Ren, Bixiong Xu. 2019. *Time-Series Anomaly Detection Service at Microsoft.* Retrieved 2019 from https://arxiv.org/pdf/1906.03821.pdf

[3] Alex Shertinsky. 2020. Fundamentals of Recurrent Neural Network and Long Short-Term Memory Network. In *Machine Learning (Machine Learning 2020).* arXiv.org, New York, NY, 226–236. https://doi.org/10.1016/j.physd.2019.132306

[4] Zheng Li Yue Zhao, Zain Nasrullah. 2019. *PyOD: A Python Toolbox for Scalable Outlier Detection.* Retrieved 2019 from https://www.jmlr.org/papers/volume20/19-011/19-011.pdf