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, gaze detection, text tagging

ACM Reference Format:
Vinay Awasthi and Woodrow Wang. 2020. Outlier Detection for Memory Accesses. *ACM Trans. Graph.* 37, 4, Article 111 (August 2020), 3 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`: Windriver - closed source.
- `Update Often`: IOS/Android patches.
- `Deny Access`: Handcarry 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 memoery access that is specialized for a given device and its usecase. This way device can take corrective actions when those parametrs 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). 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 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. Fianlly 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, analyze 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.

```
classification:
\UPD-0[Access-Zones]{Sizes}
```

Almost all attacks, styart with installing a backdoor to analyze files and scan servers to see what is running when and where first. Most of these backdoors go undetected and before any attack actually takes place, attackers analyze network and its vulnerabilities. These kinds of access can easily be triggered as anomalous. use the `Execution` Anomaly Detection:

- `Severity`: If we have incorrectly classified something, we can overwrite this in next release.

*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, .

0730-0301/2020/8-ART111 $15.00
https://doi.org/10.1145/1122445.1122456

- `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 diable noisy warmings.
- `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 am implementing kNN *K Nearest neighbors* https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

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

- `anonymous,review`: Suitable for a "double-blind" conference submission. Anonymizes the work and includes line numbers. Use with the `\acmSubmissionID` command to print the submission's unique ID on each page of the work.
- `authorversion`: Produces a version of the work suitable for posting by the author.
- `screen`: Produces colored hyperlinks.

This document uses the following string as the first command in the source file:

```
\documentclass[acmtog]{acmart}
```

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 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 Oracle's Human Capital Management System (very similar to SAP) that uses 63+ Java Virtual Machines from 2 different vendors (Rocket 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 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).

Main approach - Idea here is to perform kNN analysis for IOT application running one Java application setting up ranges of expected memory accesses. Then show when other memory accesses happen, model can recognize and flag those as outliers and may stop application from running again indicating compromised IOT environment. I am going to try 3 methods of performing outlier analysis and report results.

These three methods are:

- `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.

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 - I am working on data analysis.

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.

References -

https://github.com/yzhao062/anomaly-detection-resources

4 KEY LEARNINGS

"`Outlier Detection`" key learnings wrt to JVM usage..

5 CONCLUSIONS

Here `Outlier Detection` outcome and future directions will be discussed.

6 ACKNOWLEDGMENTS

Identification of funding sources and other support, and thanks to individuals and groups that assisted in the research and the preparation of the work should be included in an acknowledgment section, which is placed just before the reference section in your document.

This section has a special environment:

```
\begin{acks}
...
\end{acks}
```

so that the information contained therein can be more easily collected during the article metadata extraction phase, and to ensure consistency in the spelling of the section heading.

Authors should not prepare this section as a numbered or unnumbered `\section`; please use the "`acks`" environment.

7 APPENDICES

If your work needs an appendix, add it before the "`\end{document}`" command at the conclusion of your source document.

Start the appendix with the "`appendix`" command:

```
\appendix
```

and note that in the appendix, sections are lettered, not numbered. This document has two appendices, demonstrating the section and subsection identification method.

ACKNOWLEDGMENTS

To Robert, for the bagels and explaining CMYK and color spaces.

REFERENCES

A RESEARCH METHODS

A.1 Part One

A.2 Part Two

B ONLINE RESOURCES