DBSCAN Algorithm as a means to protect the ATM Systems

Michal Maliszewski
University of Silesia, Faculty of Computer and Materials Science Security & Fraud Control Solutions
41-200 Sosnowiec, Poland
Email: maliszewski25@gmail.com

Steffen Pristerjahn
Diebold Nixdorf, Faculty of Computer and Materials Science
33106 Paderborn, Germany

Urszula Boryczka
University of Silesia, Faculty of Computer and Materials Science
41-200 Sosnowiec, Poland
Email: urszula.boryczka@us.edu.pl

Abstract—Automated teller machines are affected by two kinds of attacks: physical attacks, and software-based attacks, whereas the latter is becoming more and more popular every day. Most banks tend to look for a day-zero protection in order to secure their devices. Two most popular mechanisms are whitelisting and sandboxing. However, this kind of protection is not only hard to configure, but it also requires a deeper knowledge about software security as well as the information about the software currently installed on the device. The goal of this article is to present a possibility of using a modified DBSCAN algorithm to solve a complex configuration problem which is clustering programs within operating system. Results of the experimental studies show that this kind of automatic configuration can be even more precise than the basic clustering algorithms. As the results show great promise, it is all right to believe that the algorithms could be used in security products if additional security rules are taken into account.

Index Terms—Data clustering, security, sandbox, whitelisting, operating system, intrusion prevention, software security, ATM.

I. INTRODUCTION

As of this moment, there are over three million working automated teller machines around the globe. Each ATM allows the customers to perform various financial transactions without the need of interacting with a human cashier[7]. Millions of people use ATM every day. It is estimated that 85% of transactions are carried out using cash. That is a reason why the security of cash systems, such as ATMs, is so important nowadays. However, protecting an ATM against software-based attacks is a bit different than protecting a typical personal device. While maintaining the integrity of an operating system is still an important task in its own, preventing against yet unknown attacks, potentially new threads, and minimizing the effects of injected malwares that is even more important. Unknown attacks are the reason why day-zero protection is based on application whitelisting. The idea behind whitelisting is to provide protection against harmful applications and unauthorized demand for resources by listing all applications allowed to run. The list of resources accessible by programs is defined as well. Whitelisting is the polar opposite to blacklisting, which is a method used by most antivirus programs, intrusion detection/prevention systems, and spam filters[3]. In order to simplify a configuration process, all running applications are separated into virtual containers known as sandboxes. That method avoids configuring each program separately which gives better protection but is very hard to achieve. Thanks to sandbox approach the entire operating system will not be harmed if an untrusted application is used[8]. It is still possible to control the processes’ flow within the operating system by creating a limited number of sandboxes. Connections between clusters will be blocked if they are not allowed.

Finding out all the applications, together with sources they need to run on an ATM, is a difficult task. It can get even harder if there is a need to group them into sandboxes for closer encapsulation of the operating system. There are several questions that the person performing the configuration needs to answer. For example:

• What kind of programs should be included on an automated teller machine?
• How many groups should be created?
• What will happen if the configuration is either too restrictive or too unrestrictive?

An American company known as Diebold Nixdorf[4] provides a security solution that is able to simplify the configuration process[9]. Even though, many facilities and ready pre-configurations exist, the endpoint security policy must still be configured manually. Therefore, the configurer must take the machine type along with the software installed on it, into account when configuring the endpoint security policy, not to mention that it is obligatory to know what resources should be accessible by the installed software. Once this is all done, the configuration is placed on the ATM where it is monitored, and improved if need be. The configuration can be applied to the target device by using a dedicated software e.g. Diebold Nixdorf’s Vynamic Security - Intrusion Protection[10].

The idea presented in this article is to use a modified DBSCAN algorithm to prepare an initial configuration for DN Vynamic Security - Intrusion Protection. The authors strongly recommend to use the clustering algorithms to automate the configuration process. Even though the results of the previous research were promising, the authors agreed that there is still room for improvement[14].

Density-based spatial clustering of applications with noise (DBSCAN) is a density-based clustering algorithm proposed
II. RELATED WORK

Literature offers a lot of suggestions for using clustering algorithms to detect and categorize malware, such as Ensemble Clustering for Internet Security Applications by W. Zhuang, Y. Ye, Y. Chen and T. Li[19] in which authors proposed an approach to an automatic categorization system of phishing websites or malware. An ATM fraud detection system was the basis for Model-Based Anomaly Detection for Discrete Event Systems[12] written by T. Klerx, M. Anderka, H. K. Bning and S. Priesterjahn. Unsupervised anomaly detection in network intrusion detection using clusters[13] written a few years earlier by K. Leung and C. Leckie propose a way of unsupervised anomaly detection to avoid the need of supporting large database of threads which is expensive to produce and provide maintenance for. The need of zero-day protection becomes crucial due to rapid growth in the amount of malicious software. Detection of zero-day malware based on the analysis of opcode sequences[20] written by M. Zolotukhin and T. Hämäläinen is a promising alternative as their blacklisting approach is supported by SVM clustering. However, due to the need of fully controlling the processes flow within the operating system, it may not be enough.

Authors express the opinion that whitelisting is the future of software security. The last few years have shown that this notion is being widely exported. For example, in The Promise of whitelisting by Steve Mansfield-Devin[15]. Some researchers are focusing on the areas where whitelisting can be used. For example, C. Bildsten in Application Whitelisting: Smartphones in High Security Environments[2] or R.Barbosa, R.Sadre, A.Pras in Flow whitelisting in SCADA networks, where the authors propose a way to improve the security of SCADA networks by using flow whitelisting[1]. On the other hand, clustering algorithms are widely used in intrusion detection systems (a blacklisting approach), e.g. cyber attacks detection in PCS systems described in A clustering-based approach to detect cyber attacks in process control systems by I. Kiss, B. Genge and P. Haller[11].

A similar approach was presented in 2016 by S.I. Monterrosa Reyes, G. R. Salgado and J. P. Ortega in Defining Adaptive Whitelists by Using Clustering Techniques, a Security Application to Prevent Toll Fraud in VoIP Networks, where the authors used K-means to generate whitelists with call destinations[17], however this method was used in case of VoIP Networks. Moreover, whitelisting mechanism is adaptive, which will be difficult to use in case of ATM security (e.g. newly installed programs should not be automatically allowed to run because the security policy must be updated before, which must be somehow confirmed).

In this paper we present a way how to use DBSCAN algorithm for automatic configuration of software security system based on whitelisting and sandboxing ideas. This approach allows users to provide just one test run on configured ATM to secure their devices with real zero-day protection.

III. CLUSTERIZATION OF EVENTS

On the basis of a defined prevention/detection policy applied to the currently protected system, solutions like Dynamic Security - Intrusion Protection can control and monitor what programs are running on the operating system. This results in the controlling program providing detailed log files with exact list of all processes running during that time, as well as sources which they are trying to use. Even though the data includes a lot of useful information, only a few truly matters when it comes to clustering processes.

As in the authors’ previous research, the data for these tests were taken from an ATM with Windows 7 SP 1 32-bit OS, and includes two three-hour test runs with Vynamic Security - Intrusion Protection (which uses whitelisting approach together with sandboxing) in its monitoring mode (every process is monitored but neither is yet blocked). That gives two datasets, where each record is defined as a pair of the process and an argument, which is equal to process and resources used by it (like file or registry) plus additional attributes. The number of records was limited to approximately 3000 per run due to the fact that the authors had to be able to manually prepare the correct configuration and evaluate the obtained data in relation to the actual security needs. Moreover, please keep in mind that the amount of applications, able to continuously run on an ATM’s operating system, is limited as well. An exemplary dataset is presented on Fig. 1 and contains 9 of 33 available attributes.

Most of the pre-processing steps were done by using filtering mechanisms prepared by the authors. Once prepared, they can also be used for new datasets, and with high profitability, they will be sufficient in most cases. If there is a need for some special event filtering, existing mechanism can be easily extended by a rule based on regular expressions. A list of described pre-processing steps used in the clustering process can be found below:

- Noise Removal - this includes records generated by Windows explorer or a set of programs from /system32 directory.
- Duplicates Removal - there was no need to group the same process - target pair more than once.
- Unnecessary Events Removal - events different than requests for access to files or registries were omitted.
Fig. 1. A dataset created from log files

- Data attributes were filtered dynamically - meaning, that a subset of attributes was selected for clustering process to improve the quality of the solution, however a larger subset was used for pre- and post-processing due to the fact that some of the information wasn’t given directly as an attribute, and had to be read from multiple attributes, instead.

- Process - target pairs were converted to lower case because nominal metrics were used in data clustering.

Prepared data was clustered by the following algorithms:

- DBSCAN I (used parameters: minPts = 2, $\varepsilon = 2.0$).
- DBSCAN II (used parameters: minPts = 1, $\varepsilon = 2.0$).
- DBSCAN III (parameters used in DBSCAN II were used here as well. However, more attributes were used).
- Mod DBSCAN (parameters used in DBSCAN II were used here as well).
- X-means (used parameters as in X-means but nominal to numerical conversion was used as described in authors previous article[14]).

During the authors’ first approach to clustering of ATMs’ processes described in Basic clustering algorithms used for monitoring the processes of the ATM’s OS[14] an extended variant of K-means has been selected (X-means) because of the need to automatically select numbers of clusters. The authors decided to compare the results against DBSCAN approach. Four different variants were tested in this case.

The important part of the research, as well as one of the most complex, was to find the proper configuration variants that could be used for comparison. Based on authors’ experience we estimated that the minimum cluster number is three, which from our experience, is the minimum for our datasets. Moreover, the desirable number of groups was estimated between 6-8, where eight uses the best capabilities of used security solution (Technically, DN’s Dynamic Security - Intrusion Protection allows to configure up to eight custom sandboxes which means maximum of eight groups in a clustering process).

Mod DBSCAN’s iterative approach was supposed to find the optimum value of $\varepsilon$. The iteration step was set to 0.1 (smaller iteration changed the grouping result to a very small extent and additionally increased the duration of whole process). Tuning of the algorithm continued until the next result was worse than the previous result. Such approach was enough for the test dataset. However, for larger amount of data (the order of millions of records) there may be a need to improve the process due to performance issues and susceptibility to local maxima.

IV. ACCURACY AND ALGORITHM MODIFICATIONS

As mentioned before, the iterative approach was applied to find the optimum $\varepsilon$ value. The abstract DBSCAN algorithm can be described by following steps:[18]

1) Find the $\varepsilon$ (eps) neighbors of every point and identify the core points with more than minPts neighbors.
2) Find the connected components of core points on the neighbor graph while ignoring all non-core points.
3) Assign each non-core point to a nearby cluster if the cluster is an $\varepsilon$ neighbor. Otherwise, assign it to noise.

The following modifications were proposed in order to adapt the algorithm to the approach of processes whitelisting:

1) Find the $\varepsilon$ neighbors of every point, and identify the core points with more than minPts neighbors only when they are not already mentioned in the existing core points.
2) Find the connected components of core points on the neighbor graph while ignoring all non-core points.
3) Assign each non-core point to a nearby cluster if the cluster is an $\varepsilon$ neighbor and is not a resource for other core point.

In order to keep the highest possible encapsulation within the operating system, the authors decided to remove the noise from the modified DBSCAN (as minPts = 1). As mentioned in Section III, the data was pre-processed and there is no need for the algorithm to check for noise again. On the other hand, tight encapsulation forces the authors to keep all the data under control. Therefore, the noise should not exist.

Last year, Intrusion Protection changed the default behavior of security engine defining all unknown software as blocked (instead of a very limited default sandbox), which gave an opportunity to replace the pre-processing (e.g. filters) with
extended noises recognition inside the modified version of DBSCAN.

It seldom happens that there is a pair of core points \( p \) and \( p' \) with bidirectional connections (e.g. Two processes working together in a challenge-response way, but due to the fact that they were designed asynchronously, they are forced to use their APIs for processing). Situation like this can cause problems for most clustering algorithms. An additional step was implemented in order to avoid creating two different clusters for \( p \) and \( p' \). This additional step checks if a given element is not a part of any other cluster already. In other words, if any of the core points do not contain this point as a resource (every record in a dataset is a combination of the program itself and the resource it tries to access). To provide a way to quickly check of already defined core points (those processed before the current one), the authors defined each cluster as a set data type (that uses a hash table) that provides a lookup with an average computational complexity of \( O(1) \) (or \( O(n) \) in the worst case).

The main goal of the clustering solution for this problem was to ensure that the created segmentation provides as much encapsulation as possible. This means, no correlations between clusters. On the other hand, higher number of clusters means less elements inside, which in sandboxing approach provides less damage for a system if it is infected by malware. Based on these requirements, we define the criteria in Fig. 2.

\[
P(x) = \begin{cases} 
0, & x < 3 \\
\frac{n(A) - n_c}{n(A)}, & x \geq 3 
\end{cases}
\]

Fig. 2. Accuracy measure where \( P(x) \) - accuracy, \( x \) - number of clusters, \( n(A) \) - cardinality and \( n_c \) sum of edges between elements that do not belong to the same group

V. RESULTS

Unlike in the previous research, the authors worked on the nominal representation of the data instead of converting them to numerical values. Because there was no pre-processing, DBSCAN showed its advantage as it was able to find non-linearly separable clusters. The number of created clusters is shown in Fig. 3.

All variants of DBSCAN were able to create almost maximum number of groups. Please keep in mind that creating more than eight clusters is not an issue as long as small clusters can be later easily merged together. This will, of course, lower the encapsulation, gives the elements in merged groups more access to the sources (each element will have access to resources of A and B if both clusters are combined). Due to technical limitation, it is fully acceptable.

\( X \)-means algorithm was not able to create more than 2 clusters. Therefore, it did not meet the required minimum number of groups. Thanks to the conversion of nominal data to numerical (exactly as in the previous research[14]) the authors were able to increase the number of clusters (see \( X \)-means II).

Unfortunately, the quality of grouping was slightly worse than the results achieved by DBSCAN variants.

Tabular form of the results validated against criteria presented in Fig. 2 is presented in Table I.

Results achieved by DBSCAN III variant seems to be the worst from all tested DBSCAN variants. However, thanks to the extended attributes this version was able to correctly recognize the type of requested resources. In other words, it was able to recognize if there was a request for an access to either file or registry. Split like this has created additional connections between clusters, and as a result - less restrictive encapsulation. Sadly, from the technical point of view - both system registries and files are treated the same way during creation of security policy (there are no separated sandboxes for registries or files). Taking into account the above facts, it is completely justified to mark such connections as potential security leak and therefore lower the quality of results.

As described in section IV, the test datasets were bidirectional connections between core points. In situation such as this the proposed DBSCAN modifications were able to increase the quality of clustering. However, the authors cannot guarantee that there will be no case that did not appear in the test data.

The effectiveness of the solution is promising. Even without modifications, DBSCAN (variants I and II) achieved satisfactory results with an accuracy of 80-90%. Please keep in mind, that only the quality of segmentation was evaluated. Performance of the clustering algorithms was not a part of this research due to difficulty of getting a very large set of real data from ATMs.
Fig. 4 shows the segmentation of core processes working on an ATM. These processes are mostly responsible for banking transactions, marketing content, security, or allowing maintenance. Results are presented in the form of a number of programs working on ATM OS and their resources per cluster. Disproportion in groups’ size is completely normal and is caused by the shell (main) application of the ATM system, which enables withdrawals (It is also the most popular system events creator). Moreover, it is often the one asking for most resources.

VI. CONCLUSIONS AND FURTHER WORK

DBSCAN algorithm seems to be a better solution for automatic security configuration of ATM than $X$-means. It also achieves better results than algorithms tested by the authors in their previous research.

The iterative way of finding $\varepsilon$ was sufficient for the test dataset. However, it should be improved e.g. using heuristics. Moreover, as long as there is no need to detect outliers inside the algorithm (instead pre-processing), the static value of $\minPts = 1$ can be used.

DBSCAN is able to group data correctly even if core points $p$ and $p'$ have correlations to points $p''$ and $p'''$ and $p' \sim p'''$.

Unmodified DBSCAN has difficulty with the correct classification of bidirectional connections.

All kinds of DBSCAN were able to recognize the difference between event types (access to different types of resources such as files or registries).

Due to non-deterministic nature (at least not completely), border points that are achievable from more than one cluster, can be a part of each of these clusters. This is not a problem as long as the security tools allow for flexibility in sandboxes configuration.

Further steps for the authors will be to solve existing problems (like iterative approach to find $\varepsilon$) and prepare a prototype of automatic security policy configurator. This will require further exploration of this topic as there are some post-processing steps that should be done in case if the authors want to achieve a fully secure solution. The authors are also planning to test the prototype on real machines as well as to check the performance of the whole process (pre-processing, clustering and post-processing) on large amount of data.

REFERENCES


