View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Universidade do Minho: RepositoriUM

Intelligent Data Leak Detection Through Behavioural Analysis

Ricardo Costeira
Informatics Department
University of Minho
Braga, Portugal
e-mail: rcosteira79@gmail.com

Henrique Santos
Information Systems Department
University of Minho
Guimarães, Portugal
e-mail: hsantos@dsi.uminho.pt

***Abstract*—In this paper we discuss a solution to detect data leaks in an intelligent and furtive way through a real time analysis of the user's behaviour while handling classified information. Data is based on experiences with real world use cases and a variety of data preparation and data analysis techniques have been tried. Results show the feasibility of the approach, but also the necessity to correlate with other security events to improve the precision.**

Keywords-Anomaly Detection; Machine Learning; Data Mining; Support Vector Machines.

I. INTRODUCTION

The evolution of technology and Information Systems has opened a phenomenal new world of possibilities for companies and organizations all over the globe, more dependent on data then ever. As an example, the advent of the Internet and development of distributed systems brought about the Cloud, which makes information accessible wherever the user is and whenever the user wants, as long as there is a connect to the service. However, this flexibility comes with some new threats related with the level of information exposure. Data leaks became one main concern a company can face. In the 2015 *Cost of Data Breach Study: Global Analysis* [1], which considers 350 companies in 11 countries, it is stated that 3.79 million dollars is the average total cost of data breaches, with a 23% increase since 2013. This study also points out that 47% of data leak incidents reported by the enquired companies are related to insider attacks.

To address data breaches, organizations have invested billions over the years mainly to secure their network perimeter with Firewalls and Intrusion Detection/Prevention systems, among other technologies. These solutions are by far insufficient, as corporate information keeps going out of the secure perimeter. As such, companies are now focusing on the concept of data-centric information protection [2]. RigthsWATCH [3], developed by Watchful Software, is an implementation of that concept. This type of system creates a protective bubble around data itself, implementing authorization rules defined by a given set of security policies. Besides, it also helps when a careless employee accidentally leaks information to the outside. However, as powerful and convenient it is, data-centric information protection is useless against a premeditated internal attack.

For this reason, behaviour analysis becomes an important concept to fight data leaks. By taking logged information about user interaction with protected data, it should be possible to create a behavioural profile, which can be used as a base of comparison during future interactions. In this paper, a framework for data leak detection through user behavior analysis is proposed. The framework collects RightsWATCH users' logs, crafting an individual behavioural profile from them. We then explored the capacity to distinguish between normal and abnormal behaviour. The research described here is part of the RightsWATCH development project.

This paper is organized into six main sections, the first of which is this introduction. The second section discusses the state of the art regarding Intrusion Detection systems, focusing mainly on Anomaly Detection. The third section provides a more detailed view of the proposed framework's architecture, and the fourth section describes the experimental environment. The fifth section discusses the obtained results, while the sixth and final section gathers the final conclusions and thoughts.

II. STATE OF THE ART

An intrusion detection system (IDS) is a tool capable of detecting possible security breaches on a system, by gathering and analysing security events. It can be designed to work with a wide range of information, such as logs from different sources (firewalls, OSs, etc.), application usage data, keyboard inputs, or network data packets. According to [4], an intrusion detection system should provide the following security functions: it has to *monitor* the computer or network, to *detect* possible threats and to *respond* to the possible intrusions.

Axelsson [5] developed a generalized IDS model, which is shown in Fig. 1: solid arrows represent data/control flow, while dotted arrows indicate a possible intrusion response. Axelsson's model is useful to describe the general, high level behaviour of an IDS, but a complete characterization and classification goes beyond this simple architecture, mainly due to the diverse technologies that can be implemented. Fig. 2 presents a more complete classification [6].

Copyright (c) IARIA, 2016. ISBN: 978-1-61208-512-8

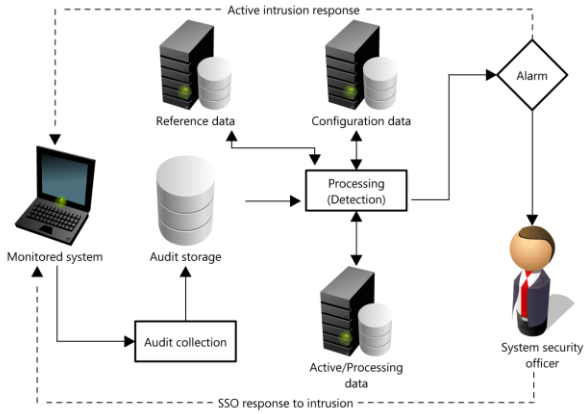

Figure 1. Generic model of an intrusion detection system.

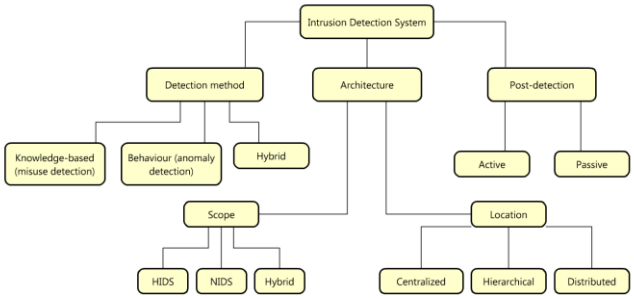

Figure 2. Characterization and classification of IDS.

This classification considers only the functional attributes of intrusion detection systems. But there is a non-functional one that should also be considered, the usage frequency [7], which is useful to distinguish an IDS from a scanner used for security assessment - the usage frequency provides a fair distinction between the two.

Each attribute has a very specific meaning, and is deeply related to the system purposes.

A. *Architecture*

The IDS architecture should be idealized considering three relevant factors: The source of the data to be analysed, the way tasks are being distributed and the processing components [6]. It is, as such, divided into two different categories: scope and location. Regarding its scope, an IDS can be host based (HIDS), network based (NIDS) or a **hybrid** system, comprising characteristics of both types.

Host based intrusion detection systems collect and analyse data from a single host. The detection software installed on the host is commonly known as agent. It can monitor a wide range of activities, such as application logs, changes in the file system, system integrity, use of resources, user access and interaction with the system, among many others.

Agents are developed to monitor servers, hosts, or even applications services. Due to the variety of implementations, de Boer and Pels consider four different main *HIDS* categories in [8]: Filesystem monitors, logfile analysers, connection analysers and kernel monitors.

The NIDS collects and analyses data from a network and usually focuses on the TCP/IP protocol. Data packets are sniffed through sensors positioned on “mission critical” spots. Sensors can be of two types [9]: appliance and software only.

In [9], the author describes some of the events detected by most of the NIDS: Application/Transport/Network layer reconnaissance and attacks, unexpected application services and policy violations.

The “location” in Fig. 2 refers to the placing of all the different modules that compose the IDS, comprising three types: centralized, where there is only one system (the manager) responsible for event analysis, detection, classification and system reaction; hierarchical, where more than one manager can exist; distributed, where there are several managers as well, but the data analysis and processing can also be done by any other component.

B. *Post-detection*

After detecting an intrusion, an IDS can perform either actively (if it reacts by its own), or passively (if it acts as a decision support system, by triggering alarms and/or notifications for the administrator).

C. *Detection Method*

Depending on the chosen methodology for analysing the audit data, IDSs can be categorized as knowledge based and/or behaviour based. Knowledge based IDSs are commonly referred to as misuse or signature detection systems, focused on attacks information, while behaviour based intrusion detection systems are usually known as anomaly detection systems (ADS), focused on information about the system behaviour [7]. The fusion of these detection methods into a hybrid system is possible.

An ADS is based on the premise that security breaches can be detected by monitoring audit data and searching for abnormal patterns of system usage, as Denning stated in [10]. The system starts by learning the general profile that describes a subject’s normal behavior – learning phase. Then, during normal work, the same features are captured and a profile is deduced in a similar way – detection phase. The working profile is then compared to the stored one searching for deviations, and if they occur, and are above a given threshold, the activity is considered a possible intrusion [5]. Note that the subject in this context refers to any resource capable of access and operate the information system, typically a user, or a process on behalf of a user.

Although the concept is fairly straightforward, the actual division between anomalies and normal data is quite challenging, generating frequently a high number of false positives, which is the main drawback of this approach.

Anomaly detection is the approach chosen for the problem at hands, mainly because there are no known patterns of possible attacks – and that seems very difficult to define. Looking for possible techniques to adopt in this case, the most commonly used are statistical methods [11][12] and data mining methods [13][14], the latter

usually dividing into classification or clustering [15][16], although other methods were also studied - expert systems [10][17][18], computer immunology [19], user intention identification [20][21], and a few other approaches, including combinations of different methods [11]. Following a clear tendency in the field, the scientific research potential and the authors' experience, classification was the technique selected for this work.

Classification commonly follows one of three approaches: Supervised, semi-supervised and unsupervised learning. These techniques are implemented under the assumption that a proper classifier, which can distinguish between normal and anomalous data, can be trained within the feature space available [22]. The model obtained through training can be one of two types: one-class and multi-class. In the first case, the training dataset has only one class label and the model consists of a discriminative boundary around the normal (labeled) instances, treating everything out of that boundary as anomalous. Multi-class methods are used when the training dataset has labels defining more than one normal class and, for each class, there will be a classifier. Some variants of this method associate a confidence level with the prediction made. The characteristics of the project under consideration points to one-class classification.

III. THE FRAMEWORK'S ARCHITECTURE

RightsWATCH logs every user's daily operation with protected data. The main challenge here is to discover which logs correspond to normal and abnormal activity – at first because there is no formal definition of what is normal and abnormal – and/or which features, among a large dataset, are relevant to catch dangerous activity. As already referred, the proposed anomaly detection framework follows a one-class classification model, which will be trained by a controlled dataset generated by regular user interaction (free of abnormal behaviour).

The proposed architectural design of the framework is depicted in Fig. 3 and a brief explanation of each component is given next. The *Reader* module is responsible for reading and preparing the input data. The raw data have to be previously selected by the developer and this information is embedded into the code, including formal interface rules. This way, the framework can be easily adapted to accept other data sources. The *Reader* is divided into two submodules. The first one is called *Preprocessor*, whose main function is to take raw data and prepare the desired features according to the classifier format requirements. Several logs are not in an adequate format, appearing as categorical or descriptive data. When dealing with categorical data, it is required to break it into *n* features, where *n* is the number of existent cases – a technique known as one-hot encoding.

The *Preprocessor* output feeds the second submodule, the *DatasetBuilder*, whose main function is to build the final dataset. This dataset aims to represent the user's behaviour, while preserving the user's privacy by completely transforming the data into something that cannot be tracked back to its original state - the dataset is fully composed of numeric values, whose relationship to the real data is completely eliminated.

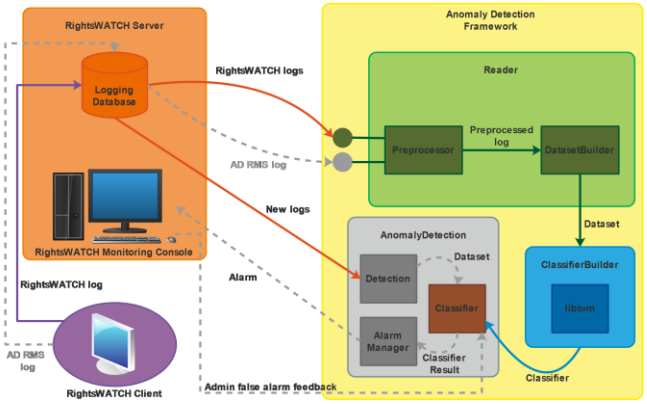

Figure 3. Architectural design of the framework.

In Fig. 3 there is a second source of logs labeled as AD RMS logs (Active Directory Rights Management Services). It corresponds to an infrastructure provided by Microsoft, which gathers all the server and client technologies to support information protection through the use of rights management in an organization. RightsWATCH uses this technology, but the associated logs are not used, for now - the dashed and greyed arrow line indicates that.

The Dataset provider by the Reader module will be the input for the *ClassifierBuilder* module. This module analyses the data and creates a model of the user's behaviour. This model, or classifier, will be used as a reference for future comparisons. To perform this task, a classification algorithm is needed. Support Vector Machines (SVM) was the chosen one for the task, by wrapping the *libsvm* library provided, by Chang and Lin [23]. This library packs the standard SVM algorithm along with the most relevant variations, such as one-class SVM and SVM for regression. Another version of the library, available in the website, also packs the Support Vector Data Description (SVDD) algorithm. The reason for choosing this library has to do with the fact that since its inception in 2001, *libsvm* was successfully integrated and used in similar problems (the full list is available on *libsvm*'s website), appearing as one of the most promising approaches to the classification problem.

To perform the detection, two different SVM algorithms were investigated, although one of them to a much greater extent than the other (the standard binary SVM classification algorithm was also used in a set of preliminary tests). As referred above, these algorithms are the one-class classification algorithm and the SVDD. This second algorithm was only considered in the final stages of the investigation, mostly for performance comparison. Both of these relate to semi-supervised learning, and are the only SVM options available for this problem.

The secondary function of the *ClassifierBuilder* module is, in the validation stage and following a conservative approach, to test different variations of the detection method, as described in the next section.

Copyright (c) IARIA, 2016. ISBN: 978-1-61208-512-8

The final model will be stored and used by the *AnomalyDetection* module, which is divided into three submodules: *Detector*, *Classifier* and *AlarmManager*. The first submodule is responsible for getting logs from users in real-time, extract the features and prepare. After, the *Classifier* submodule evaluates the entry using the classifier obtained by the ClassifierBuilder module and produces a score result, which is outputted to the *AlarmManager* submodule. This submodule will react to the result, issuing an alarm to the administrator with the corresponding threat level. Shall it be the case of a false alarm, the administrator can mark the event as benign, which will also trigger an update of the classifier with the new information, to avoid similar mistakes in the future.

IV. DATASET CREATION AND FRAMEWORK DEVELOPMENT

RightsWATCH and the ADS framework discussed in this paper were developed at Watchful Software's office, facilitating the integration and assuring all the technical support necessary. All the testing was performed there, with data logs captured from 4 collaborators during 1 day (only 4 were chosen since only those exhibit what can be considered a typical work interaction with RightsWATCH). The available number of logs for user 1, user 2, user 3 and user 4 are, respectively, 5714, 3120, 2514 and 2365.

The logging information, initially stored in a dedicated database, is copied to a new table, called *LogTraining* (see Fig. 4), to keep original logs intact. Next we will describe in more detail the dataset creation, which, as already mentioned, raised several problems.

A. Dataset Creation

To data mining researchers, real world industrial databases can be considered one of their worst nightmares, and the reason is simple: real data is, most of the times, dirty and cluttered, and databases are not prepared at all for data mining. As this turned out to be the case, RightsWATCH logging table demanded for a preparation and cleansing process. The resulting table has thirty-eight dimensions (excluding primary key), filled with numerical, categorical and binary data. The table's structure, along with other new tables, is depicted in Fig. 4.

The most prominent problem is that thirty-five of the thirty-eight dimensions have missing values, blank spaces or null values, which affect every table record. Apart from this, there are inconsistencies between data, as well as information that should have been inserted into a different table cluttered in one single column. The logs containing email addresses that users chose as recipients in emails, are an example of the cluttered set of information – for each record, a user can have something like *email1;email2;...;emailn* for *n* "to" recipients.

The solution for these problems was quite straightforward. The missing values were substituted by actual default values, the inconsistent values were removed and the cluttered information was reorganised into different tables.

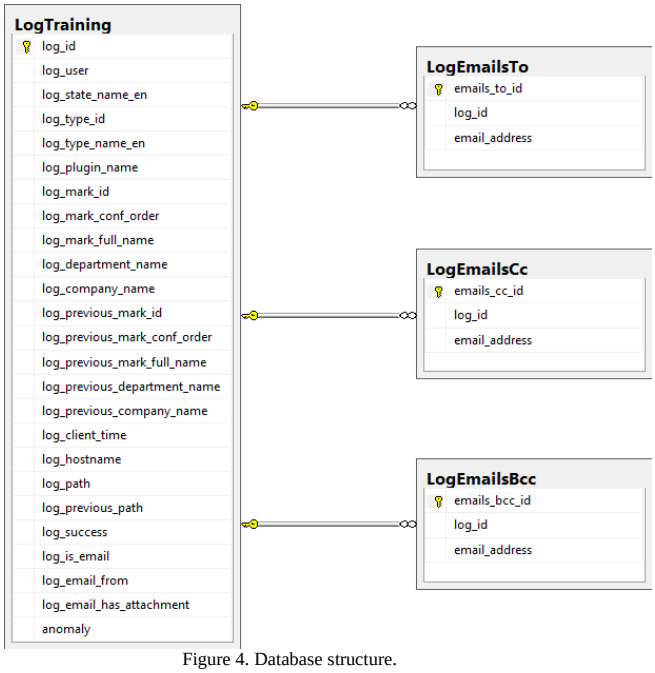

Figure 4. Database structure.

There are other features deserving particular attention, given their relevance and the amount of perturbation they may impose, namely: *log_path* and *log_previous_path*, contain a directory path of a protected file – since shared directories are avoided, this features may have an excessive weight in classification; *rule id*, that logs the id of a policy rule triggered by the user, appeared with some imprecisions due to system limitations encountered at the time. To perceive better their influence we decide to perform data analyses removing each and both from the complete dataset.

For the initial features selection and extraction stages, WEKA (Waikato Environment for Knowledge Analysis) was used, to assess the influence of each feature over the dataset. WEKA is a well-known machine learning tool wildly used for this type of data analysis and it includes *libsvm*. For each user's data we consider other users' data attacks (examples of bad behavior). Following regular recommendations, we perform a 10-cross validation operation (WEKA randomly divide the dataset in 10 subsamples, keeping one for model validation and the others for training; the process is repeated 10 times so that all subsamples are used for validation, in each time). The same process was performed with the original dataset and the ones obtained by removing the problematic features, as referred above. The results achieved are presented in tables I, II and III. Overall, the accuracy is not very high, which demands for more research.

With these results, it became obvious that a more formal way to assess feature quality was needed, which was done with PCA (Principal Components Analysis), a data analysis technique also included in WEKA.

PCA was run over the same dataset variants and it revealed that there are no outstanding features. There are, of course, features with larger coefficients (weight values)

Copyright (c) IARIA, 2016. ISBN: 978-1-61208-512-8

associated to them than others, but adjacent coefficients are close in value.

Table I. CLASSIFICATION ACCURACY RESULTS FOR THE FIRST VERSION OF THE DATASETS WITHOUT THE PATH FEATURES.

	Classification Accuracy Results			
Legitimate \ Anomalous	User 1	User 2	User 3	User 4
User 1		67.7%	79.5%	71.1%
User 2			70.9%	67.5%
User 3				73.1%

Table II. CLASSIFICATION ACCURACY RESULTS FOR THE VERSION OF THE DATASETS WITHOUT THE RULE ID FEATURE AND STILL WITHOUT THE PATH FEATURES.

	Classification Accuracy Results			
Legitimate \ Anomalous	User 1	User 2	User 3	User 4
User 1		67.7%	78.9%	70.8%
User 2			70.6%	67.6%
User 3				73.2%

Table III. CLASSIFICATION ACCURACY RESULTS FOR THE FIRST VERSION OF THE DATASETS WITH THE PATH FEATURES.

	Classification Accuracy Results			
Legitimate \ Anomalous	User 1	User 2	User 3	User 4
User 1		64.7%	77.6%	70.7%
User 2			68%	63.5%
User 3				71%

Contrary to what was expected, the path features do show some classification potential. On the other hand, the email features did introduce noise in the dataset, as expected. Also, the full set of email related features correspond to approximately thirty-six percent of the whole dataset.

In all PCA executions, the correlation between features was fairly poor. Except from the obvious correlations (for instance, the *mark email* action type feature is always heavily correlated with the *Microsoft Outlook plugin* feature), most of the features have shown low correlation values between each other.

Regarding the results, the only action decided was towards the recipient email addresses. As it became clear, their presence in the dataset is a prelude of an erroneous classifier. But simply discarding the emails seems a waste of possibly useful data. As such, a more conservative solution was adopted, performing the division of the recipient emails into their respective local and domain parts. The local part is discarded, while the domain is retained as a feature. Of course, this way it is not possible to distinguish, for example, if a user is sending one hundred emails to one hundred different *Gmail* addresses, or one hundred emails to the same *Gmail* address. Still, it is better than blinding the classifier with noise or not having any recipient email address information at all.

Finally, and to conclude this section, the features produced by *PCA* or, in other words, the principal components, are often used in place of the original features.

B. *Framework Conception*

The development of the *ADS* framework started with the *Reader* module and data transformation. The categorical feature that is not one-hot encoded is the *log_client_time*. It was rendered by two new features, *day_of_the_week* and *time_of_the_day*. Both features are numeric – *day_of_the_week* goes from zero (Sunday) to six (Saturday). It is important to note that *day_of_the_week* was only added later and, by that time, *time of the day* was also modified (these changes are explained in full detail on the next section).

Two modifications regarding user names had to be implemented, by privacy reasons. The first one was ignoring the *log_user* field - it is not associated with any type of behavior; the second one occurs at runtime, and it is the removal of the user's name from every directory path, whenever it is present. For instance, for a path such as "Users\John\Sales Report.docx", the final result is just "Users\Sales Report.docx".

After the preprocessing, the *Reader* module has to build the datasets. The ratio is 70% of the total data for training, and the remaining 30% for testing (following the recommendations given in *libsvm* documentation). The datasets are created in a sequential fashion – first the training set, and then the test set. This order has to be maintained, as the features of the test set will depend on the training set. In other words, the test set will only have features that are also available in the training set, (any new feature is excluded). This condition is needed since categorical features are divided into sets of new features through one-hot encoding, and there will be cases where some of these features will exist on the test set but not on the training set.

Next, at heart of the *ClassifierBuilder* module, we use *libsvm*, as already referred. It also has methods to test and cross-validate data, and since it performs a very time consuming task, it implements some inner and heavy loops in a paralyzed way using OpenMP (Open Multi-Processing).

Apart from its main classification function, the *ClassifierBuilder* also implements new methods for performance measurement and classifier quality assessment. Essentially, the classification results regarding false/true positives/negatives are used to build a confusion matrix, and then combined together to compute the precision and recall values. These indicators evaluate different aspects of the classifier. Precision, or *positive predictive value*, is obtained with the division of true positives by the sum of all the examples that were considered positive (i.e., true positives and false positives) and represents the accuracy. It can be though as a numerical representation of the model's *exactness*. Conversely, recall or *sensitivity* is calculated by dividing true positives by the sum of true positives with false negatives, which can be seen as the model ability to classify observations from a class as cases from that actual class. It can be understood as the classifier's *completeness*. Both values vary between zero and one – zero being the worst

Copyright (c) IARIA, 2016. ISBN: 978-1-61208-512-8

case, and one the best case. Low precision can be a sign of a large number of false positives, and low recall can mean that the classifier is detecting too many false negatives. As such, the ideal is to have both values as closer to one as possible. Both measures are used to determine the F1 score, which can be interpreted as the weighted average of precision and recall, and its computation is achieved through the harmonic mean (F1 score = $2 \cdot (precision \cdot recall)/(precision+recall)$). F1 score also varies between zero and one, with similar interpretation. These performance indicators are commonly used with anomaly detection solutions.

Finally, the *ClassifierBuilder*, was designed to test the two SVM algorithms, as stated before, and using all the four kernels available in *libsvm* - linear, RBF, polynomial and sigmoid kernels. This is achieved through a loop that performs a grid search over the algorithm parameters – *C* for *SVDD* and ν for one-class – and kernel parameter γ (except, of course, for the linear kernel, that does not use γ). Besides, the algorithm also performs a k-fold cross validation over the dataset (as explained before), looking for the best classifier provided by each kernel. The results of the four classifiers are then evaluated, and stored along with the parameters used. Since this is done for each user, it is not viable to actually use this in a real world environment (besides, the grid search takes too much time). Still, this configuration will be maintained at the prototype level for research purposes.

V. RESULTS

For the testing methodology, it was decided to follow nearly the procedure suggested by *libsvm* authors, the difference being that all the available kernels are used.

Two very similar training datasets were used for each user, built from the same logs, which means that in total every user will have two classifiers. The difference between the two datasets is a subset of features – those features that are completely unique to each user, namely hostname and user email addresses, were removed from one dataset. Note that in a regular situation, these features would be vital in the dataset - for instance, it is possible to use the hostname to cover the anomalous cases where the user account is used on an unknown machine. In this case, as most of the test sets were composed by data from other users, labeled as anomalous behaviors, the presence of that information would introduce a strong bias on the results. For similar reasons, the username feature was completely excluded from both datasets as well.

For this testing phase, the default values for the one-class classification parameters - 0.5 to ν and $1/features$ to γ – were used. For the polynomial kernel we keep the default parameter value. Dataset *A* is the complete one, and dataset *B* is the one without unique features. The values for precision, recall and F1 scores were computed only for the attacks with all the available test data, and are available on tables IV and V (for visualisation purposes, every decimal value was rounded from six to two decimal places).

Let the focus be on dataset *A* results. The number of final features per user vary - 1781, 819, 377 and 761 for users 1, 2, 3 and 4, respectively. As it was expected, most of the classifiers that were trained with full featured datasets were heavily influenced by the unique features. Otherwise, it would be virtually impossible for any classifier to achieve a classification accuracy of 100%, as it happens in so many cases. However, note that this is a good thing, considering the true nature of the problem: the idea is not to have the classifier distinguishing between users, but instead having it identifying anomalous behaviour of the same user and, from that point of view, the classifier ability to decide that an example is anomalous if the hostname or the email address that the user wields are different from the usual, is very important. Regardless, it seems that most of the classifiers whilst easily recognising other users, struggle to recognise the user itself. A particularly noteworthy case of this is User 2. As a clear case of the closed world assumption point of view discussed earlier, this user has got the lowest score when tested against itself no matter the kernel used and, as a consequence, had the lowest F1 score too, given the low recall results. Also, notice that regardless of the unrealistically high classification values that this user achieved with the test dataset that comprises every user, the F1 score values resist this tendency and provide for a more grounded analysis, which ends up proving the advantage of using such a metric. As for the kernels, the *RBF* kernel achieved the overall worst results, while both linear and sigmoid kernels appear to perform better in this case.

The results are slightly different for dataset *B*. With a minimal decrease in features, the importance of the missing features becomes evident, as the overall results are much lower in quality. With this dataset, the classifiers had trouble both in distinguishing between behaviors and in recognising the target user. In addition, these results also prove that the default *SVM* parameters are, in this case, far from optimal. Finally, the kernels exhibit similar results between them, with the linear kernel slightly outstanding itself from the rest, always with the highest F1 score for all users. Note that in contrast with dataset *A*, the recall values for dataset *B* were always higher than the corresponding precision - not due to an increase in recall values, but instead because of a drastic decrease in precision.

After testing the kernels, the next challenge is to search for the best (ν, γ) combination, through cross-validation. To do that, the same 70%-30% division was made for each user. The training sets were used in a 10-fold cross validation, for a grid search on twelve values for ν and fifteen values for γ (again, following *libsvm*'s documentation recommendations). In the end, the best performing parameters combination for each dataset is stored, and then used to create new classification models. Then, each model is tested. The resulting performance metrics are presented on tables VI and VII.

At a first glance, and judging by the accuracy values, it seemed that these parameters would push the framework

Copyright (c) IARIA, 2016. ISBN: 978-1-61208-512-8

into producing overly *permissive* classifiers. This has to do with the way that one-class *SVM* works: tampering with the *v* parameter has a direct impact on the number of observations that are accepted - the lower the value, the more permissive the resulting classifier will be.

Dataset *A* still demonstrates the importance of the user unique features. Also outstanding is the clear difference in the ability to correctly classify both anomalous and legitimate instances between the linear kernel and the remaining three. The quality of this kernel was already noticed before, but not to this extent. For every user, this kernel attained the best F1 score. In fact, the F1 scores for this kernel are, for every user, better than those reported in the results before the grid search. Although, it was the only kernel to achieve such feat, as the remaining F1 scores are mostly low. Finally, the recall values for this dataset were substantially higher than before, due to the classifier accepting more samples as normal ones (i.e., being more permissive).

The results for dataset *B* turned out to be even worse than before. Without the unique features, and with such forgiving *v* parameters, the classifiers declare a large part of the attacking users' instances as legitimate. In this dataset, apparently, no kernel stands out from the others, and even with the high recall values, the F1 scores are simply unacceptable.

At a first glance, this investigation could end here. The linear kernel achieved better results with dataset *A*, after the grid search. However, this does not prove that the linear classifier will do a good job detecting anomalies. In fact, without the unique features (dataset *B*), the classifier does not perform so well. What if the user himself leaked valuable information, from his usual machine, with his usual email? Thanks to the unique features, the classifier would likely detect the resulting log as legitimate. As such, it is not time to finish, but to stop for a while. Time to pause and think about the causes of the results obtained with dataset *B*. Was it the classification algorithm, or the data itself? It might have been both.

The *v* parameter controls how many observations get misclassified, and how many turn into support vectors. For instance, if the *v* parameter is set to 0.1, it is guaranteed that at most 10% of the training instances will be misclassified, and at least 10% of them will become support vectors. Before the grid search, *v* was the default *libsvm* value of 0.5. This is a conservative value, which created classifiers incapable of correctly recognising a legitimate user by setting a high upper bound for the outlier ratio. Then, the grid search chose the parameters that allowed for the highest F1 scores. With such low *v* values, the upper bound of outliers decreased, therefore letting more examples fall on the correct side of the hyperplane. On the other side, assuming too small outlier ratios, can easily allow anomalous instances to fall on the legitimate side of the hyperplane, which indeed happened. Of course, the γ parameter also influences everything as well. This parameter defines how *far* the influence of a single training example reaches: low values mean *far* and high values mean *close*. Before the grid search, the value was dictated by the number of features, which means that it varied, depending on the dataset, between ≈ 0.0006 and ≈ 0.001. After the grid search, most of the γ parameters achieved values that were higher, which in turn diminished the influence of the training instances. All of this leads to one thought: The grid search method might prove effective with two class problems, but the same might not apply to one-class classification, given the actual nature of the optimisation problem, when performing it with cross-validation.

Taking this into consideration, a new grid search was performed: The classifiers were tested against data from both the user and the other users and, instead of returning the parameters with the highest F1 score, the algorithm was modified to output all the data it produces for each parameter combination, so that each of the 552 classifiers (12 for the linear kernel and 180 for each of the other kernels) could be individually examined. The perusal of the data confirmed the worst: with dataset *B* there were no cases where any of the classifiers managed to successfully separate anomalous from legitimate records. The classifiers mostly bounced between the two extremes - either classifying most of the cases as legitimate or as anomalous. When this is not the case, the classification accuracy values just revolve around the 50% mark, with very low standard deviation values.

With these results, it was inevitable to think about the quality of the data. One of the first impressions that arose when contacting with the available data for the first time, during the data cleansing and data selection phases, was that it would be possible it be too **fine grained** for the *SVM* algorithm. The features are poorly correlated between them, and with the naked eye, at least, it is next to impossible to distinguish between examples, if we ignore users' unique features. Although this granularity-level issue is not confirmed, the obtained results do allow to consider it as a reason for the classifiers poor quality. As such, it seems appropriate to rethink the way the dataset is built, and how the information is used. Hence, the next section describes the final dataset transformation.

Table IIV. PERFORMANCE METRICS FOR DATASET A.

	User 1 Metrics			User 2 Metrics			User 3 Metrics			User 4 Metrics		
Kernels	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Linear	1	0.65	0.79	1	0.11	0.19	1	0.53	0.69	1	0.58	0.73
RBF	0.16	0.68	0.26	0.10	0.25	0.14	1	0.52	0.68	0.24	0.74	0.37
Polynomial	0.26	0.84	0.39	1	0.09	0.17	1	0.55	0.71	1	0.25	0.40
Sigmoid	1	0.65	0.78	1	0.11	0.20	1	0.46	0.63	1	0.58	0.74

Copyright (c) IARIA, 2016. ISBN: 978-1-61208-512-8

Table V. PERFORMANCE METRICS FOR DATASET B

	User 1 Metrics			User 2 Metrics			User 3 Metrics			User 4 Metrics		
Kernels	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Linear	0.14	0.65	0.24	0.08	0.57	0.14	0.22	0.53	0.31	0.06	0.55	0.11
RBF	0.14	0.78	0.24	0.07	0.71	0.14	0.18	0.41	0.25	0.06	0.71	0.10
Polynomial	0.15	0.76	0.25	0.07	0.49	0.12	0.22	0.51	0.30	0.05	0.21	0.08
Sigmoid	0.14	0.65	0.24	0.08	0.53	0.13	0.22	0.47	0.30	0.06	0.60	0.11

Table VI. PERFORMANCE METRICS FOR DATASET A AFTER GRID SEARCH.

	User 1 Metrics			User 2 Metrics			User 3 Metrics			User 4 Metrics		
Kernels	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Linear	0.99	0.97	0.97	1	0.2	0.33	0.99	0.99	0.99	0.99	0.99	0.99
RBF	0.17	0.98	0.29	0.04	0.54	0.08	0.17	0.97	0.28	0.06	0.99	0.11
Polynomial	1	0.95	0.98	1	0.19	0.32	0.2	0.72	0.31	1	0.97	0.98
Sigmoid	0.17	0.99	0.3	0.07	0.48	0.12	0.24	0.99	0.39	0.06	1	0.11

Table VII. PERFORMANCE METRICS FOR DATASET B AFTER GRID SEARCH.

	User 1 Metrics			User 2 Metrics			User 3 Metrics			User 4 Metrics		
Kernels	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Linear	0.16	0.98	0.27	0.07	0.97	0.14	0.11	0.99	0.2	0.05	0.99	0.1
RBF	0.16	0.98	0.28	0.07	0.99	0.13	0.09	0.99	0.17	0.05	0.99	0.1
Polynomial	0.16	0.93	0.27	0.08	0.94	0.14	0.16	0.72	0.27	0.06	0.95	0.1
Sigmoid	–	0	–	0.07	1	0.13	0.09	1	0.17	0.05	1	0.1

A. *Dataset Refactoring*

The current dataset is composed of *point anomaly* records. Following the reasoning explained before, the framework will now aggregate data points into larger sets, thus producing new features that, in essence, **summarise** the attribute values from every example that comprises the given aggregated set, this way creating a different, *collective anomaly* dataset.

The log aggregation process was performed considering each set of logs that was generated in an hour. In other words, the set of logs generated in an hour became one collective log. The starting time was defined by the very first log. This time frame was chosen for two reasons. Firstly, it is feasible that a user generates more than one log in an hour. Secondly, it is still a small enough time frame to allow for mitigating measures in case of a data leak.

With the aggregation, the number of available observations diminished significantly. Users 1, 2, 3 and 4 now have, respectively, 1576, 1722, 469 and 1466 total observations. The number of features has also changed, as it was expected with the changes regarding the path features, not to mention the fact that the features themselves are different. Using the same distinction between datasets, dataset *A* now consists of 642, 355, 485 and 398 records for the four users (in that order), while dataset *B* contains 635, 349, 482 and 394 records. Note that with this dataset, the training set of user 3 will have a number of features higher than the number of observations, which can reshape the final results.

The testing process was the same one used before. The 70%-30% division between training and test sets still remains, but note that this division was established only in the individual logs. A lot of work would be needed in order to apply this process to the new data, as it is created dynamically. Fortunately, with the division between the individual logs, the final ratio of the new data between training and test sets is roughly 80%-20% for users 1 and 3, and 75%-25% for users 2 and 4, which are still acceptable boundaries. Also, this will be an opportunity to test the impact of different division ratios between the data.

Despite the expectative, results are not very different from those obtained before – for that reason we think it is not necessary to present them in new tables. Next we will discuss the small details that deserve some attention.

Dataset *A* exhibits the same high precision and low recall values, while dataset *B* displays the same drop in precision, while maintaining the recall values. Even the different kernels performed in an identical fashion. On the other hand, the variation between the F1 scores of both dataset types is not wide enough to allow for any kind of conclusion just yet. Since the default parameters were used, these results are not unexpected.

The grid search for this dataset was slightly different from before. This time, every parameter combination was manually examined. Also, instead of performing the search through cross-validation, the algorithm tested each of the 552 classifiers with the test sets that contain information about all users. This was done so that the obtained parameters were more adequate to the validation process. Indeed, in a normal situation, the parameters would be generated via cross-validation, as the data belonging to the user would be the only data used. However, it was already

Copyright (c) IARIA, 2016. ISBN: 978-1-61208-512-8

seen that the classifiers are able to accept logs as being legitimate, and thus the purpose now is to search for parameters that can aid the classifiers into defining the best plane possible between both legitimate and anomalous instances. Concerning dataset *A*, the only thing to point is the superior performance of the polynomial kernel over all the others. This kernel managed to obtain good results even when the others wavered.

Recall that the only reason for working on a completely different dataset had to do with dataset *B* disappointing results. As it turns out, the new dataset *B* error metric results are similar to the old ones - although, from a more optimistic point of view, when looking at both the classification and confusion matrices results, these classifiers performed clearly better, with no exception. In fact, these are good news, since there are still so many possible ways to perform the log aggregation that were not investigated. In other words, one (or more than one) of these other options might prove itself to be more successful.

A curious pattern emerged with both datasets. With only two exceptions - users 1 and 3 always scored the top classification results. This is most certainly related to the data division between training and test sets. Recall that these two users are the ones with the 805-20% division – more data to train the classifier and less data to test it. As it is known, the more data there is, the more accurate should the classifier be. However, this same pattern is noted, although to a lesser extent, on dataset *B* classification result, which might suggest that there is more to it than the data division.

Finally, a last attempt was made with the SVDD method, using the initial datasets, *A* and *B*. However, the obtained results were worst most of the times. Besides, it suggests that this algorithm has a stronger resistance to the unique features, as results from both datasets were quite similar. This is actually a good omen if we think in terms of generalisation capabilities.

Now the only question remaining is whether the optimal parameters are able to improve the classifier's accuracy or not. Similarly to what happened before, both datasets produced similar results, with dataset *A*'s classifiers sometimes underachieving when compared to those of dataset *B*, but most of the times surpassing them and, in some instances, by a large margin. With dataset *A*, the *RBF* kernel was always ahead of the other three, while this distinction was not so clear with dataset *B*, where it was even surpassed by the polynomial kernel on the tests with user 4.

A noticeable aspect of the tests with this classification method is that, for each user, the *C* regularisation parameter is mostly constant throughout the different kernels, which leads to the belief that this regularisation parameter is stronger and more influential than the *v* on the one-class classification method, which might be directly involved in this method's resistance to the unique variables. In other words, it is possible that the classifiers produced by *SVDD* are more stable than the ones generated through the one-class classification *v-SVM*, but this result require more research.

VI. CONCLUSIONS AND FUTURE WORK

As the obtained results suggest, and even if they are less conclusive then expected, it is possible to define a user behaviour pattern based on the features generated by *RightsWATCH*, which can be used to identify possible data leaks linked to abnormal behaviour. Regardless, there is still so much more to do. With the knowledge about what has already been done and how, it becomes easy to define a high level roadmap for the framework. The first step is to test additional different options for log aggregation – tests for aggregating logs considering different time frames, considering a fixed number of logs and considering the use of a sliding window. If the classifier's performance does not improve, it might be advisable to start testing with different classification methods, other than *SVM*. There is actually an alternative machine learning technique that worth to compare with *SVM* (the only reason why it was not tested already is because it would give birth to a whole new project). This technique is called online learning. Online or incremental (online learning and incremental learning are considered to be the same thing as often as they are not) machine learning is similar to the standard "offline" machine learning, with the difference that the model is updated after the initial training, as new data points arrive. This would allow the framework to continuously improve the classifier, even if the user changed is behavioural pattern. In a very interesting article, Laskov suggests a way to extend the already known one-class and *SVDD* classification methods to this learning method [24], but of course, there are many more implementation suggestions, considering both linear and non-linear kernels.

ACKNOWLEDGMENTS

This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the Project Scope UID/CEC/00319/2013.

REFERENCES

[1] "2015 cost of data breach study: Global analysis," Ponemon Institute LLC, 2308 US 31 North, Traverse City, Michigan 49686 USA, Tech. Rep., May 2015.

[2] J. Bayuk, "Data-centric security," Computer Fraud & Security, vol. 2009, no. 3, 2009, pp. 7 – 11.

[3] Watchful software's RightsWATCH. https://www.watchfulsoftware.com/en. Retrieved: August, 2016.

[4] D. V. C. Venkaiah, D. M. S. Rao, and G. J. Victor, "Intrusion detection systems - analysis and containment of false positives alerts," International Journal of Computer Applications, vol. 5, no. 8, August 2010, pp. 27–33, published by Foundation of Computer Science.

[5] S. Axelsson, "Research in intrusion-detection systems: A survey," 1998.

[6] A. Pathan, The State of the Art in Intrusion Prevention and Detection. Taylor and Francis, January 2014.

Copyright (c) IARIA, 2016. ISBN: 978-1-61208-512-8

[7] H. Debar, M. Dacier, and A. Wespi, “Towards a taxonomy of intrusiondetection systems,” Computer Networks, vol. 31, no. 8, 1999, pp. 805–822.
[8] P. de Boer and M. Pels, “Host-based intrusion detection systems,” Amsterdam University, 2005.
[9] K. Scarfone and P. Mell, “Guide to intrusion detection and prevention systems (idps),” NIST Special Publication, vol. 800, no. 2007, 2007, p. 94.
[10] D. E. Denning, “An intrusion-detection model,” Software Engineering, IEEE Transactions on, no. 2, 1987, pp. 222–232.
[11] H. S. Javitz and A. Valdes, “The sri ides statistical anomaly detector,” in Research in Security and Privacy, 1991. Proceedings., 1991 IEEE Computer Society Symposium on. IEEE, 1991, pp. 316–326.
[12] C. Manikopoulos and S. Papavassiliou, “Network intrusion and fault detection: a statistical anomaly approach,” Communications Magazine, IEEE, vol. 40, no. 10, 2002, pp. 76–82.
[13] W. Lee and S. J. Stolfo, Data mining approaches for intrusion detection. Defense Technical Information Center, 2000.
[14] L. Portnoy, E. Eskin, and S. Stolfo, “Intrusion detection with unlabeled data using clustering,” in Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001, 2001, pp. 5–8.
[15] T. Lane and C. E. Brodley, “An application of machine learning to anomaly detection,” in Proceedings of the 20th National Information Systems Security Conference, vol. 377. Baltimore, USA, 1997.
[16] T. Shon and J. Moon, “A hybrid machine learning approach to network anomaly detection,” Information Sciences, vol. 177, no. 18, 2007, pp. 3799–3821.
[17] H. S. Vaccaro and G. E. Liepins, “Detection of anomalous computer session activity,” in Security and Privacy, 1989. Proceedings., 1989 IEEE Symposium on. IEEE, 1989, pp. 280–289.
[18] C. Dowell and P. Ramstedt, “The computerwatch data reduction tool,” in Proceedings of the 13th National Computer Security Conference. University of California, 1990, pp. 99–108.
[19] S. Forrest, S. A. Hofmeyr, and A. Somayaji, “Computer immunology,” Communications of the ACM, vol. 40, no. 10, 1997, pp. 88–96.
[20] P. Spirakis, S. Katsikas, D. Gritzalis, F. Allegre, J. Darzentas, C. Gigante, D. Karagiannis, P. Kess, H. Putkonen, and T. Spyrou, “Securenet: A network-oriented intelligent intrusion prevention and detection system,” Network Security Journal, vol. 1, no. 1, 1994.
[21] T. Spyrou and J. Darzentas, “Intention modelling: approximating computer user intentions for detection and prediction of intrusions.” in SEC, 1996, pp. 319–336.
[22] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey,” ACM Computing Surveys (CSUR), vol. 41, no. 3, 2009, p. 15.
[23] C.-C. Chang and C.-J. Lin, “Libsvm: a library for support vector machines,” ACM Transactions on Intelligent Systems and Technology (TIST), vol. 2, no. 3, 2011, p. 27.
[24] P. Laskov, C. Gehl, S. Kruger, and K.-R. M ¨ uller, “Incremental support vector learning: Analysis, implementation and applications,” The Journal of Machine Learning Research, vol. 7, 2006, pp. 1909–1936.

Copyright (c) IARIA, 2016. ISBN: 978-1-61208-512-8