CZECH TECHNICAL UNIVERSITY IN PRAGUE

FACULTY OF ELECTRICAL ENGINEERING

Department of Cybernetics

BACHELOR THESIS

Identifying Malicious Hosts by Aggregation of Partial Detections

Author: Ondřej Lukáš

Advisor: Ing. Sebastián García, Ph.D.

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student:	Ondřej L u k á š
Study programme:	Open Informatics
Specialisation:	Computer and Information Science
Title of Bachelor Project:	Identifying Malicious Hosts by Aggregation of Partial Detections

Guidelines:

1. Review the state-of-the art methods for malware network behaviors analyses with special attention to statistical data analysis and Bayesian statistics.
2. Propose and implement a Bayesian inference method to process incremental detections over time.
3. Experimentally evaluate the proposed solution on both equally distributed and real capture datasets.
4. Critically analyze the results and propose further extensions of the solution with respect to its applicability to StratosphereIPS project.

Bibliography/Sources:

[1] Daniel J. Burroughs, Linda F. Wilson and George V. Cybenko - Analysis of Distributed Intrusion Detection Systems Using Bayesian Methods - Thayer School of Engineering, Hannover, 2002
[2] Christopher Kruegel, Darren Mutz, William Robertson and Fredrik Valeur - Bayesian Event Classification for Intrusion Detection - University of California, Santa Barbara, 2003
[3] Chaker Katar - Combining Multiple Techniques for Intrusion Detection - IJCSNS International Journal of Computer Science and Network Security, February 2006
[4] Garcia, S. (2016) - Stratosphere Project - Retrieved January 24, 2016, from https://stratosphereips.org

Bachelor Project Supervisor: Ing. Sebastián García, Ph.D.

Valid until: the end of the summer semester of academic year 2017/2018

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 13, 2017

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor Ing. Sebastián García, Ph.D. for the continuous support, patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

My sincere thanks also go to Mgr. Jan Šochman, Ph.D. for helping me with the machine learning part of the thesis. Last but not the least, I would like to thank my family: my parents and my girlfriend Agnieszka for their support and understanding throughout the whole time I worked on this thesis.

Author statement

Abstract

Due to the variety of possible ways to attack a computer system, network intrusion detection has been always a very complex task. The main problem of detection tools is to balance the detection ratio with the errors. The cost of generating a false alarm can be prohibitive and should be avoided when possible. The increasing amount of attacks witnessed in the last few years makes it very necessary to have a detection tool for protecting the network. Stratosphere IPS is a free-software network intrusion detection tool which uses machine learning algorithms for identification of infected devices in the network. One of the downsides of the first version of Stratosphere IPS is that it detects individual connections and it, therefore, generates a lot of false alarms. This thesis proposes to design, implement and test a machine learning improvement of Stratosphere IPS which aggregates the partial detections of hosts and classifies them using the XGBoost algorithm to improve the overall performance of the tool. Our method is based on an additional layer of abstraction called Source Address layer which collects the partial data and pre-processes it or the classifier. Compared to the first version of Stratosphere IPS proposed extension results in 40% increase in accuracy and 26% improvement in the False Positive rate.

Keywords: Intrusion Protection System, Malware Detection, Machine Learning, XGBoost algorithm

Abstrakt

Ochrana počítačových sítí je v důsledku širokého spektra typů malware velmi obtížnou disciplínou. Strmý nárůst v počtu zařízení připojených k internetu v posledních letech vede ke zvyšující se poptávce po systémech na detekci útoků a na obranu před nimi. Stratosphere IPS je freeware využívající strojového učení k identifikaci infikovaných zařízení v síti. Jednou z jeho slabin je analýza založená na jednotlivých spojeních. V této práci jsme navrhli, naimplementovali a otestovali řešení ve formě agregace dílčích detekcí a jejich následném využití k identifikaci škodlivých hostitelů pomocí algoritmu XGBoost. Použitá metoda je založena na přidání vrstvy nazvané Source Address Layer, ve které jsou dílčí data shromažďována a zpracovávána tak, aby je bylo možné klasifikovat pomocí XGBoost algoritmu. Experimenty ukazují, že použití námi navržené metody zvyšuje přesnost detekcí o 40 % a současně snižuje míru nesprávně detekovaných adres o 26 %.

Klíčová slova: Ochrana počítačových sítí, Detekce malware, Strojové učení, algoritmus XGBoost

Contents

1. Introduction **1**
2. Related Work **2**
3. Stratosphere Linux IPS **3**
3.1. Matching a connection state and behavioral models 4
3.2. Interpreting a connection state as a Markov Chain 4
3.3. Implementation details of SLIPS 6
3.4. Limits of the current state of SLIPS 6
4. Source address layer **7**
4.1. Gathering information in Source Address Layer 8
5. Classification **9**
5.1. Features extracted in a Time Window 10
5.1.1. Connection ratio 10
5.1.2. Connection ratios sum and mean 11
5.2. Sliding Detection Window (SDW) 12
5.3. Vector of features 13
5.4. Classifiers 13
5.4.1. Naive Bayes 13
5.4.2. Support Vector Machine 14
5.4.3. XGBoost 14
6. Implementation **15**
6.1. Structure of classes in Source Address Layer 16
6.2. Alerts 16
6.3. WHOIS Information 17
6.4. Performance 17
6.5. Using SLIPS 17
7. Dataset **18**
7.1. Training Data 18
7.2. Testing Data 18
8. Experiments **19**
8.1. Training 20
8.1.1. SVM 21
8.1.2. XGBoost 22
8.2. Testing of performance 24
8.2.1. Performance comparison per TW 24
9. Results analysis **26**
9.1. Training 26

9.2. Performance per TW 27
9.3. Performance per capture 27
10. Conclusion 28
References 29
Appendix 31
Detailed results of testing 31
Commonly used symbols 38
Content of the CD 38

List of Figures

Labeling process in the original version....3
Key for coding flow as character used in Stratosphere IPS....5
An Example of Markov chain created from connection state "a, a, c+d+d+"....5
Matrix related to Markov Chain in Figure 2.....5
Process of grouping flows and connection by source address....7
Process of collecting information in Source Address Layer....8
Extraction of features for classification for a single host in a time window....9
Computation of connection ratio for connection state "C+a+a+A+H+e"....11
Extracting sum of CR from a single host....11
Extraction of feature vector from a single host....13
Workflow for processing binetflows with Source Address Layer....15
The Structure of classes in the Source. Address Layer....16
Learning Curves of Naive Bayes – TW only....20
Learning Curves of Naive Bayes - TW&SDW....20
Learning curves of SVM – TW only....21
Learning curves of SVM – TW&SDW....22
Cross-validation results for XGBoost using TW features only....22
Cross-validation results for XGBoost - results with TW&SDW....23
Learning curve of XGBoost – TW only....23
Learning curve of XGB - TW&SDW....24
Performance comparison (measured per time window)....25
Performance comparison (measured per capture)....26
Comparison of percentage improvement in quality measurements between individual classifiers per capture.....27

List of Tables

Thresholds for labeling flow based on its size....4
Thresholds for labeling the flow based on its duration....4
Thresholds for assigning periodicity labels based on the time difference.....4
Distribution of labels in the training set....18
Distribution of labels in the testing set....19
Cross-validation result SVM (parameters C, gamma) – TW only....21
Cross-validation result SVM (parameters C, gamma) – TW +SDW....21
Performance measurements per TW....24
Performance measurements per capture....25
Measurements per time window – Dataset 42....31
Measurements per capture – Dataset 42....32
Measurements per time window – Dataset 43....32
Measurements per capture– Dataset 43....33
Measurements per time window - Dataset 44....33
Measurements per capture- Dataset 44....34
Measurements per time window - Dataset 45....34
Measurements per capture- Dataset 45....35
Measurements - Dataset 46 (per TW)....35

Measurements per capture - Dataset 46 36
Measurements per time window- Dataset 48 36
Measurements per capture- Dataset 48 37
Measurements per time window- Dataset 51 37
Measurements per capture- Dataset 51 38

1. Introduction

One of the most common problems of intrusion detection systems is the production of false alarms and how to deal with them. It is vital not to overwhelm the users with reports and alarms, especially false ones because its time and resources are limited. At the one hand, a tool that generates too many false alarms consumes the resources of the users and endangers their trust. On the other hand, if no alarms are generated during an attack the users will simply stop using the tool. Stratosphere IPS for Linux (SLIPS) is a free software developed in the Stratosphere Lab of the CTU University in Prague [1]. SLIPS processes web flows, builds a connection from them and analyzes the behavior patterns in the connections.

The main topic of this thesis is to improve the results of Stratosphere Linux IPS tool to decrease the False Positive Rate of the detection. The latest version of SLIPS implements Markov Chain models for detection of malware behaviors in the network. SLIPS examines each connection separately which leads to increased amount of False Alarms as well as limited accuracy. Our goal is to design and test a system, which eliminates mentioned limits. This thesis builds on this foundation and extends it by an additional layer of abstraction called Source Address Layer. This technique allows deeper analysis as well as more precise classification of hosts in the monitored network. Since SLIPS only works with flows in the traffic the data in packets is not inspected or monitored in any manner.

We propose to build on the foundation of time windows (TW) which is being used by latest SLIPS. The time window is a set period in which decision about each host in the network must be made. Our goal is to extract as much information about the individual host in each TW and create a feature vector from the information. Afterward, we suggest using a machine learning classifier XGBoost [2] to classify the hosts as either “Normal” or “Malicious”. An additional step is to include close history in the decision-making process. We call this procedure Sliding Detection Window(SDW) and by default, it uses past 12 time windows (in other words last one hour of the traffic). The motivation for this extension is to evaluate each host with respect to its previous actions. Another advantage is softening sudden peaks and falls in the data.

The main contribution of this thesis is a new version of Stratosphere Linux IPS which implements the method described above. It is free and accessible at https://github.com/stratosphereips/StratosphereLinuxIPS. Our version of the SLIPS contains trained XGBoost algorithm which is ready to use. Improved alerting system is also included in the tool.

Another contribution of this thesis is an experimental evaluation of the accuracy of classification both in individual time windows and in sliding detection window. Comparison with the current version of SLIPS shows that the accuracy of decision has been increased from 0.5458 to 0.5504 with using TW only. If history is included with SDW average accuracy is 0.7693. The number of False Alarms is significantly reduced which improves the usability of the tool. For all experiments and training of the classifier CTU-13-Dataset [3] and other captures created in Stratosphere Project is used.

The remaining of this thesis is divided as follows: Section 3 describes the current version of SLIPS and its pros and cons. Section 4 describes our research on extending SLIPS and the proposed design. In Section 5 we show the suggested process of classification. Section 6 contains detail about implementation and performance. CTU-13-Dataset and other datasets used during the experiments are described in Section 7. In Sections 8 and 9 we show our experiments and analyze the results. Our conclusions are stated in Section 10.

2. Related Work

There has been done many experiments to reduce the False Positive rate in the malware detection. There are two main concepts of false alarms minimalization. The option is designing a more accurate system which is a complex task. Machine Learning algorithms often used for this purpose. Sumaiya and Aswani [4]successfully used a combination of advanced algorithms to improve the accuracy of the classification and reduce the size of dataset required for training. The rise in the popularity of the deep learning has brought attempts to use neural networks in the intrusion protection. Tang [5] used Deep Neural Network in the software defined networks, but so far with only average results. A different approach for improving the performance of Network Intrusion Detection Systems (NIDS) is incorporating additional sources of information as shown with Bayesian networks in [6]. The method of adaptive learning using the human verification has been proposed in 2004 by T. Pietraszek in [7]. The main concept is to gradually improve the performance of the classifier with manually verified samples.

Another approach to the reduction of the False Positive ratio is post-processing generated alerts before reporting to the user. T. Pietraszek and A. Tanner [8] described several ways of using machine learning algorithms in this area. Clustering is a typical example of that as shown in [9]. Another way of dealing with the problem is to filter the alerts using a data mining technique. Xiao and Li [10] show the usage of Outlier Detection results the reducing of the FP rate.

In our approach, we combine both ways described above. CH. Katar studied described several methods for combining of different models when designing in [11]. He concluded that even though some methods perform very well in certain classes of intrusion, they lack the accuracy in the global problem domain. Katar proposes combining the strength of several models to achieve the overall performance of the NIDS.

In our solution, we build on the foundation of the first version of Stratosphere IPS [1] which includes a system for detecting malware by matching behavioral models. Partial detections are clustered, and Extreme Boosting Classifier (XGBoost) [2] used for classification.

3. Stratosphere Linux IPS

Stratosphere IPS is a project to research, develop and verify methods to detect malware traffic in the network [12]. It was born in the Computer Science department of the Faculty of Electrical Engineering, CVUT in 2015 as the result of a Ph.D. thesis [1]. The algorithms developed in this project were first implemented in an intrusion prevention tool for Linux called SLIPS (Stratosphere Linux IPS). SLIPS implements these machine learning methods to identify the behavior of malware in the network. It is free software and its goal is to make available the advanced algorithms to the broad community of civil society organizations.

Among the main characteristics of SLIPS are that it receives bidirectional flow and it analyzes the traffic by separating the behaviors of each connection. SLIPS does not read nor receives packets from the network because its goal is to analyze high-level behaviors and not content. Currently, the bidirectional flows are generated by Argus [13].

A stream of flows is divided into time windows (TW). Flows are grouped by connections - each connection is defined by quadruple of Source Address, Destination Address, Destination Port, Protocol. After receiving, each flow is coded as a character based on its properties. Character obtained in that way is appended to the previous state of the connection. Afterward, the newly created state is compared to the behavioral models of malware.

Each of the models has been created from real malware and represents a certain type of attack. If any of the models matches, the connection is labeled as 'Malicious'. If no similarity is found, it's label is set to 'Normal'. When all flows in the TW are processed, each connection with label 'Malicious' is reported. Subsequently, any host with any 'Malicious' connection is marked as infected. The downside of using models of malware is that if the tool encounters the unknown type of attack, it has no way to recognize it. Such malware is labeled as 'Normal' which results in increased False Negatives.

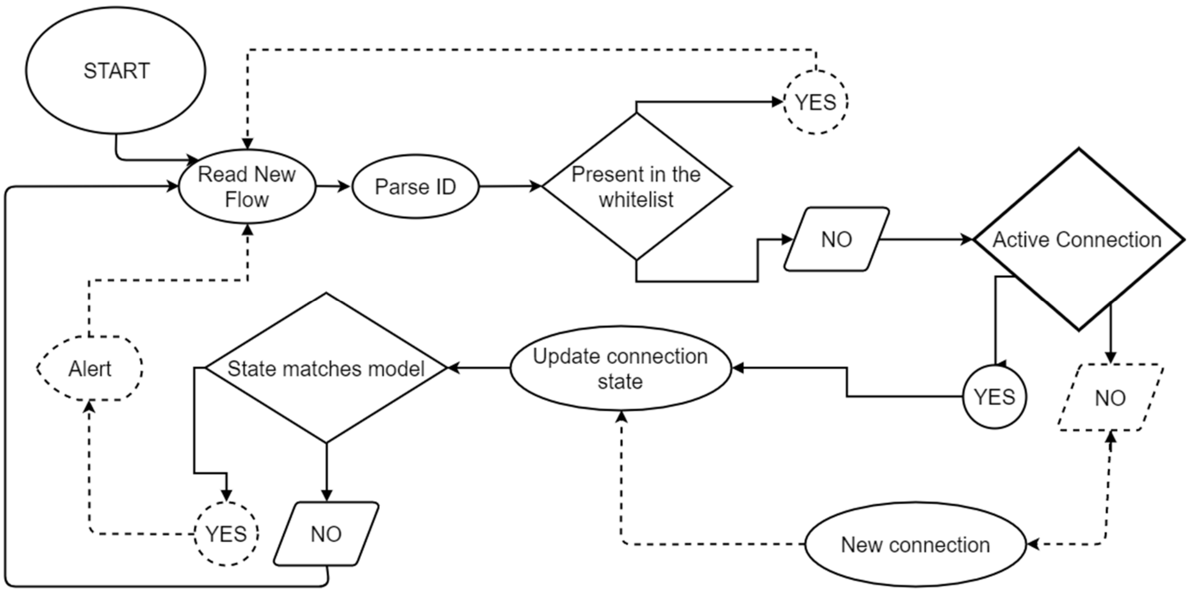


Figure 1: Labeling process in the original version

3.1. Matching a connection state and behavioral models

A connection must be transformed to the special form before it is compared to the behavioral models. Since the content of packets is not used in the Stratosphere project, only the flow properties are used for the transformation.

Size S (Bytes)	< 250	< 1100	>= 1100
Label	Small	Medium	Large

Table 1: Thresholds for labeling flow based on its size

Duration (s)	< 0.1	< 10	>= 10
Label	Short	Medium	Long

Table 2: Thresholds for labeling the flow based on its duration

Two previous flows are used for computing the periodicity of the current flow. More precisely, the timestamps of the flows. Let us call TS_t the timestamp of the current flow, TS_{t-1} timestamp of previous flow and TS_{t-2} the timestamps of the flow before.

Then Time difference TD of the current flow is computed using following formula:

Equation 1: Time difference of a flow

$$TD_t = |(TS_{t-1} - TS_{t-2}) - (TS_t - TS_{t-1})|$$

After obtaining the time difference, the label for periodicity can be assigned.

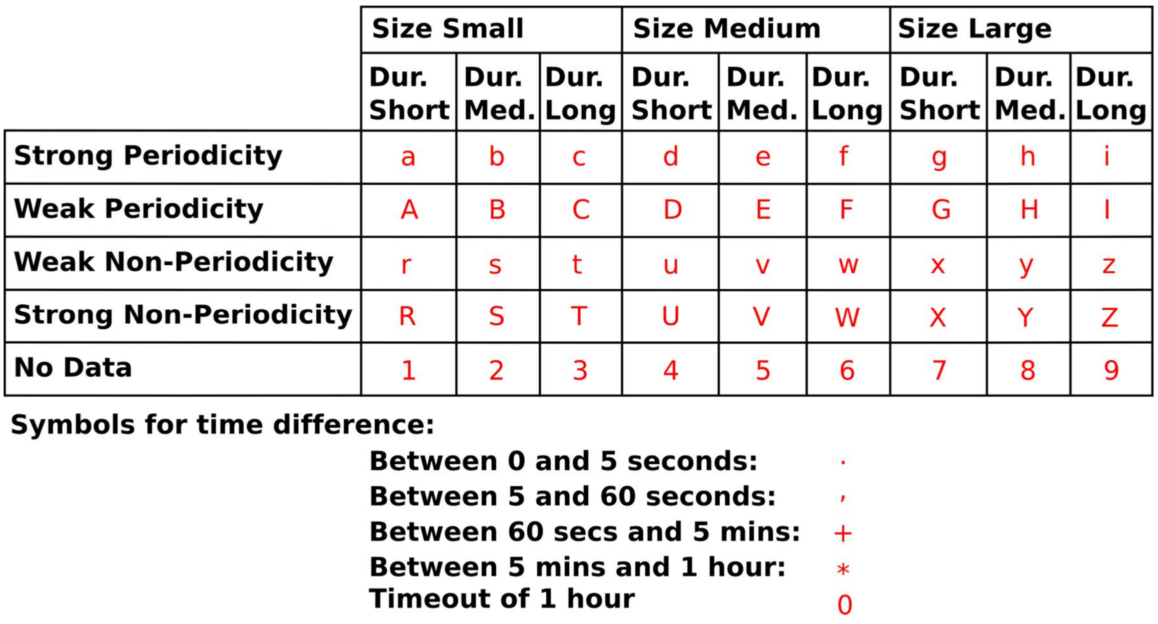
Time difference	< 1.05	< 1.3	< 5	>= 5
Periodicity	Strong Periodicity	Weak Periodicity	Weak Non-Periodicity	Strong Non-periodicity

Table 3: Thresholds for assigning periodicity labels based on the time difference.

3.2. Interpreting a connection state as a Markov Chain

When the class of each feature of the flow is known, it can be coded by character. String characters representing properties of the flow and its time difference define the state and behavior of the connection in time.

Given character and time difference are appended to the connection state. Including TD in the state is important to differentiate periodicity of 1 second from the periodicity of 1 month. Updated state of the connection is then compared to the behavioral model. Key in Figure 2 defines characters corresponding with each set of features.



	Size Small			Size Medium			Size Large		
	Dur. Short	Dur. Med.	Dur. Long	Dur. Short	Dur. Med.	Dur. Long	Dur. Short	Dur. Med.	Dur. Long
Strong Periodicity	a	b	c	d	e	f	g	h	i
Weak Periodicity	A	B	C	D	E	F	G	H	I
Weak Non-Periodicity	r	s	t	u	v	w	x	y	z
Strong Non-Periodicity	R	S	T	U	V	W	X	Y	Z
No Data	1	2	3	4	5	6	7	8	9

Symbols for time difference:

Between 0 and 5 seconds:	.
Between 5 and 60 seconds:	,
Between 60 secs and 5 mins:	+
Between 5 mins and 1 hour:	*
Timeout of 1 hour	0

Figure 2: Key for coding flow as character used in Stratosphere IPS

Connection state represented by a string of characters can be interpreted as first order Markov Chain model. That is a model of transition probabilities from one character to another. Let us consider connection with state "a, a, c+d+d+". Markov chain and corresponding matrix are shown in Figure 3 and Figure 4

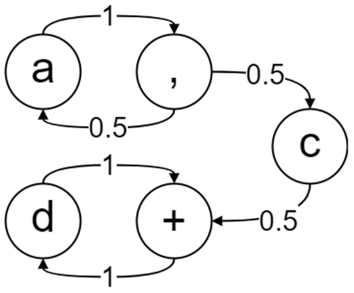
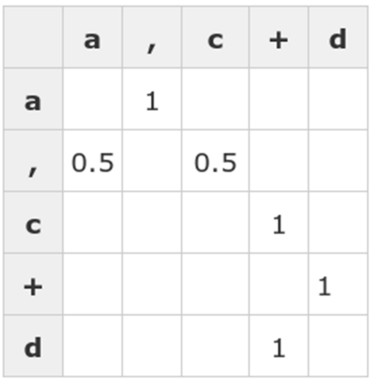


Figure 3: An Example of Markov chain created from connection state "a, a, c+d+d+".



	a	**,**	**c**	**+**	**d**
a		1			
,	0.5		0.5		
c				1	
+					1
d				1	

Figure 4: Matrix related to Markov Chain in Figure 2.

This method is used for detecting similarities between stored malware models and captured connections. Each model has its own threshold calculated using the probability of generating its original string using its Markov model. For each connection, the probability of generating its state (represented by string) is computed. If it overcomes the threshold of the model, the connection is reported as matching the model. Models for most common types of malware has been obtained in the Stratosphere project and are under constant improvement. However, there is always a delay between the occurrence of a new type of attack and the creation of the corresponding model.

3.3. Implementation details of SLIPS

SLIPS is not designed to create flows of the traffic. For that, it uses Argus which is an open source tool created by Carter Bullard [13]. Argus generates the bidirectional netflows which SLIPS can process afterward. Since SLIPS can be run in the real-time traffic it is necessary to read the flows as fast as possible. For that reason, there are two processes in the program. The purpose of the first is to read flows from Argus and store them in the Multiprocessing Queue. The second process reads flows from the queue, analyze them and builds connections from them. The final step is the compare the connection to the Markov models and generating alerts if the match is successful.

3.4. Limits of the current state of SLIPS

Processing flows in a way described suffer from several issues. Since reporting of the host as 'Malicious' depends only on the last label assigned to any of its connections, the progress of labeling is not considered at all. Therefore, every time the tool assigns label it considers the host as it was labeled for the first time. That way, a possible source of additional information is skipped. Secondly, the fact that result of every connection can lead to reporting the host as 'infected', leads to increased number of false positives. On the other hand, using Malware models means that only the known types of attacks are recognized. That means a high number of False Negatives and overall low accuracy of the detections. Additionally, connections with same host (same source address) are analyzed independently. That is another source of information being ignored. A side effect of alerting per connection detection is overwhelming the user with alerts. Since manual analysis of the potential malware takes a lot of time and effort, mistakes made by machine learning algorithms can be very costly.

4. Source address layer

Since the original version of SLIPS deals with connections separately, running the tool results in repeating detections for each time window. Two connections can share source address and yet differ in the final detection prediction. That is a huge problem for the users because such output is difficult to interpret. To solve this behavior, we propose to extend SLIPS in a way which eliminates such behavior. If several connections share a host it is possible to consider them as a part of the bigger object and work with it as one unit. Our method is based on the additional layer of abstraction called Source address layer. The main purpose of the layer is to gather and assemble and store pieces of data to create a complex image of the behavior of the host in the network in each time window. That corresponds with the main purpose of any network IDS - detection infected hosts.

The concept of merging objects that share some characteristic is already being used in the original version, where flows that share the source IP, destination IP, port and protocol ale combined to create a connection. The Same process can be applied to connection using source address as a key to group by. The motivation for cumulating the information is to improve the performance of detection through getting more data from the connection, such as relations between them and patterns in their states. We suggest extracting the information from each host as a vector of features and use machine learning algorithms as a tool for labeling the infected host. Our hypothesis is that our technique improves the performance of SLIPS while reducing the false alarms which we identified as one of the main issues.

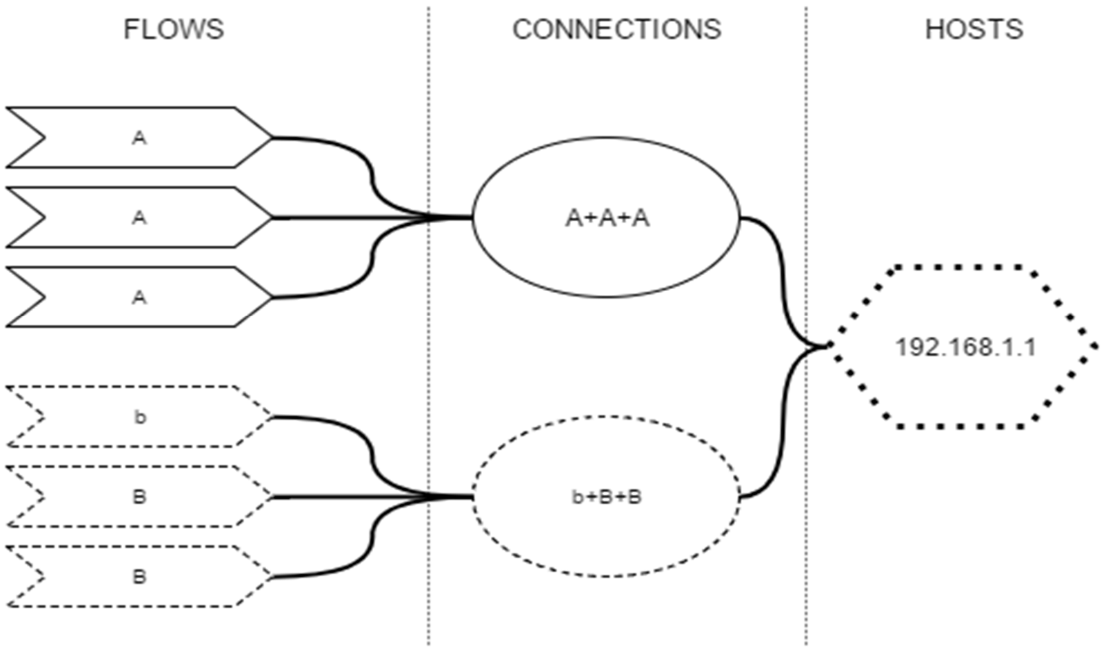


Figure 5: Process of grouping flows and connection by source address

Figure 5 shows how flows are grouped by connection as it happens in the original version. However, the result of matching each state to a Markov models is stored together with the connection state. That allows monitoring the progress of matching instead of the last result only is it is in the current method.

4.1. Gathering information in Source Address Layer

In the Subsection 3.1, we discussed drawbacks of the original version. We propose to use advanced machine learning algorithms together with the data aggregated in the Source Address Layer to improve the overall performance of the SLIPS detection abilities. Classification is being done on the host level which means we need to create a process of summarizing information stored in the Source Address layer in a form which can be used by the classifier. For this purpose, a vector of features is being used. All features are based on the result of the comparison to the malware models. However, they are not tight to any specific type. That is an advantage for the further development of the SLIPS because additional means of similar behavior comparison can be easily plugged into the system.

SLIPS works on a time window basis which means, that at the end of each TW (5 minutes by default) it labels every host. For the classification set of features is extracted from the host, which describes the behavior of the host in the TW. In addition to the activities in the current TW, the past behavior is considered as well. The Sliding Detection Window (SDW) is used for collecting information about the behavior of a host in a fixed number of previous time windows. In the default settings, we use 12 previous TW. The features from the current time window and the SDW are stored in a vector based on which the host is labeled.

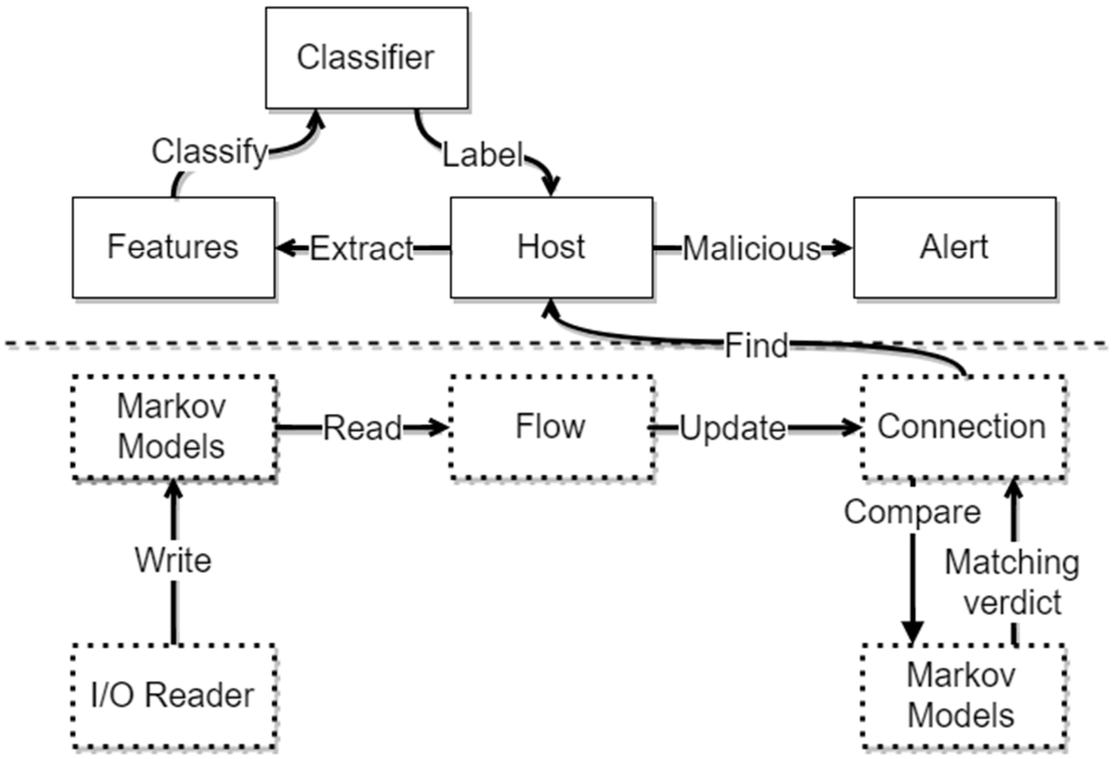


Figure 6: Process of collecting information in Source Address Layer

In Figure 6 we show how the structure of original SLIPS is used in the information gathering process. Original parts are marked with dashed line. The centerpiece of the Source Address layer is the host in which the data from the original version is stored and pre-processed to a form which a classifier can work with. Potential alerts are also gathered in the host.

5. Classification

In the 3.4 we discussed drawbacks of the original version. We propose to use advanced machine learning algorithms together with the data aggregated in the Source Address Layer to improve the overall performance of the SLIPS detection abilities. Classification is being done on the host level which means we need to create a process of summarizing information stored in the Source Address layer in a form which can be used by the classifier. For this purpose, a vector of features is being used. All features are based on the result of the comparison to the malware models. However, they are not tight to any specific model. That is an advantage for the further development of the SLIPS because additional models of malware or other means of behavioral detection can be easily plugged into the system.

Slips works on a time window basis which means, that at the end of each TW (5 minutes by default) it labels every host. For classification set of features, which describe the behavior of the host in the TW, is extracted. In addition to the activities in the current TW, past behavior is considered as well. Sliding Detection Window (SDW) is used for collecting information about the behavior of a host in a set number of previous time windows. In default settings, we use 12 previous TW. Features from the current time window and the SDW are stored in a vector based on which the host is labeled.

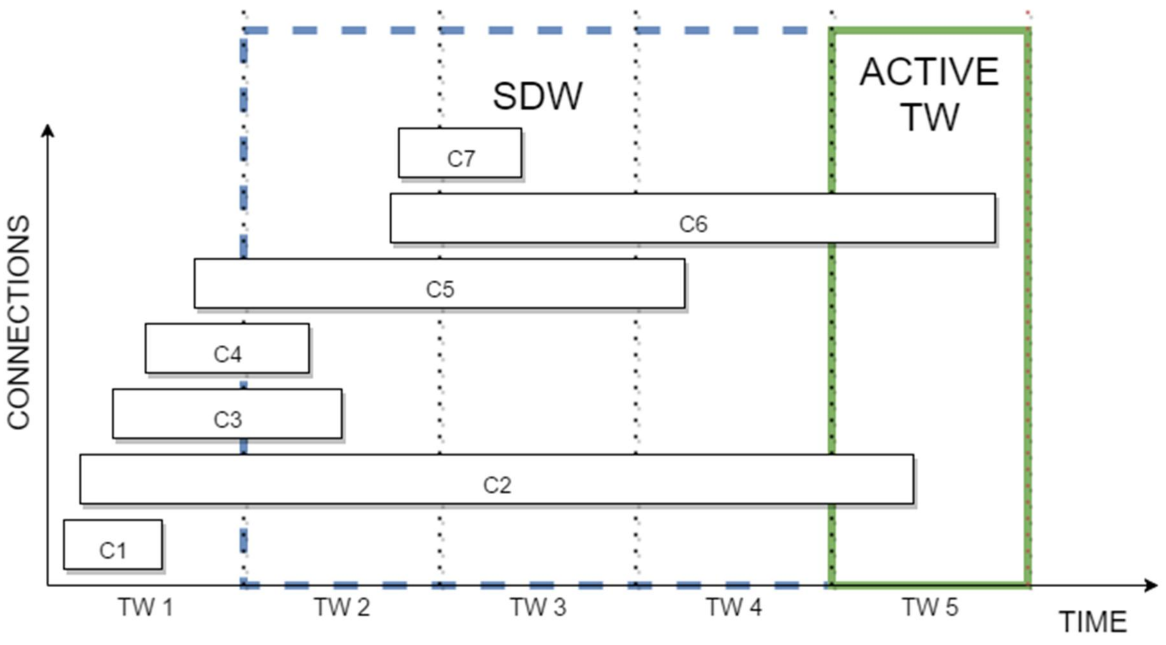


Figure 7: Extraction of features for classification for a single host in a time window

In Figure 7, we show an example of Sliding Detection Window in a single host. For the sake of simplicity, we decreased the width of the SDW to 3. The important property of the SDW is that it changes over time so only the features from the short-term history are included.

5.1. Features extracted in a Time Window

To describe the behavior and properties of a host in 1 time window we use following a set of features:

- **(i)** Amount of connections
- **(ii)** Sum of connection ratios
- **(iii)** Mean of connection ratios
- **(iv)** Amount of connections without any matching state
- **(v)** Amount of connections with at least one matching state
- **(vi)** Number of connections with all matching states
- **(vii)** Percentage of connections without any matching state
- **(viii)** Percentage of connections with at least one matching state
- **(ix)** Percentage of connections with all matching states
- **(x)** Number of new connections states created in this TW
- **(xi)** Number of new connections states matching any model
- **(xii)** Percentage of new connections states matching any model

In the following Subsections, we describe and explain each of the features.

5.1.1. Connection ratio

One of the most important features is a sum of Connection ratios (CR). The CR is a property of individual connections and describes how much does the connection match the Markov models. In other words, it is an amount of successful comparisons with any model in the TW over the number of all comparisons in the TW. Remember that one comparison is done for each new flow assigned to the connection.

Equation 2: Matching function of state and model

$$f\colon S\times M \to \{0,1\}, \qquad f(s,m) = \begin{cases} 1, & s \sim m \\ 0, & otherwise \end{cases}$$

Matching functions f describes the situation when connection state s matches malware model m. It is a core part of the formula for Connection ratio:

Equation 3: Connection ratio

$$cr\colon C \to \langle 0; 1\rangle, \qquad cr(c) = \frac{\sum_{i=1}^{S}\sum_{j=1}^{M} f(s_i, m_j)}{|S|}$$

Connection ratio shows us the overall matching of the connection and Markov models. It is common that malicious connections tend to have more states (they consist of more flows). That is the reason use a percentage instead of sum.

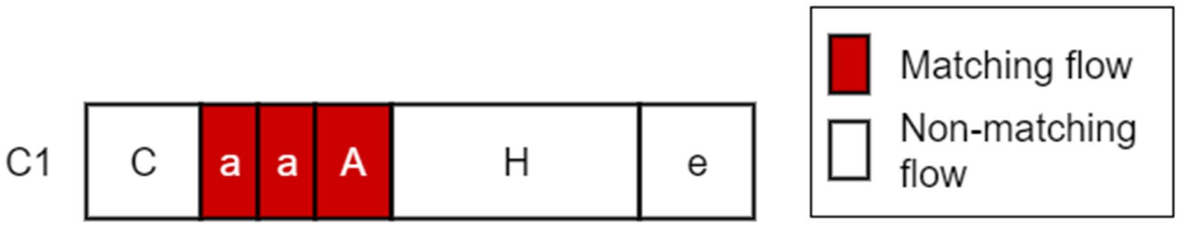


Figure 8: Computation of connection ratio for connection state "C+a+a+A+H+e"

Connection C1 in Figure 8 has 6 states which are compared to the malware behavioral model. Only 3 of the states match from which we can say that $cr(C_1) = \frac{3}{6} = 0.5$. It is important to note that if $cr(C) = 0$ it does not mean that there is no malicious behaviror in connection C. Each of the models is tailored to detect a specific type of malware behavior. Therefore, if the behavior of the connection differs from the models it might be labeled clean even though it is malicious.

5.1.2. Connection ratios sum and mean

When CR for each of the connections is known, we can compute the sum and mean of connection ratios. The sum of the CRs shows how the overall matching of the host with the models.

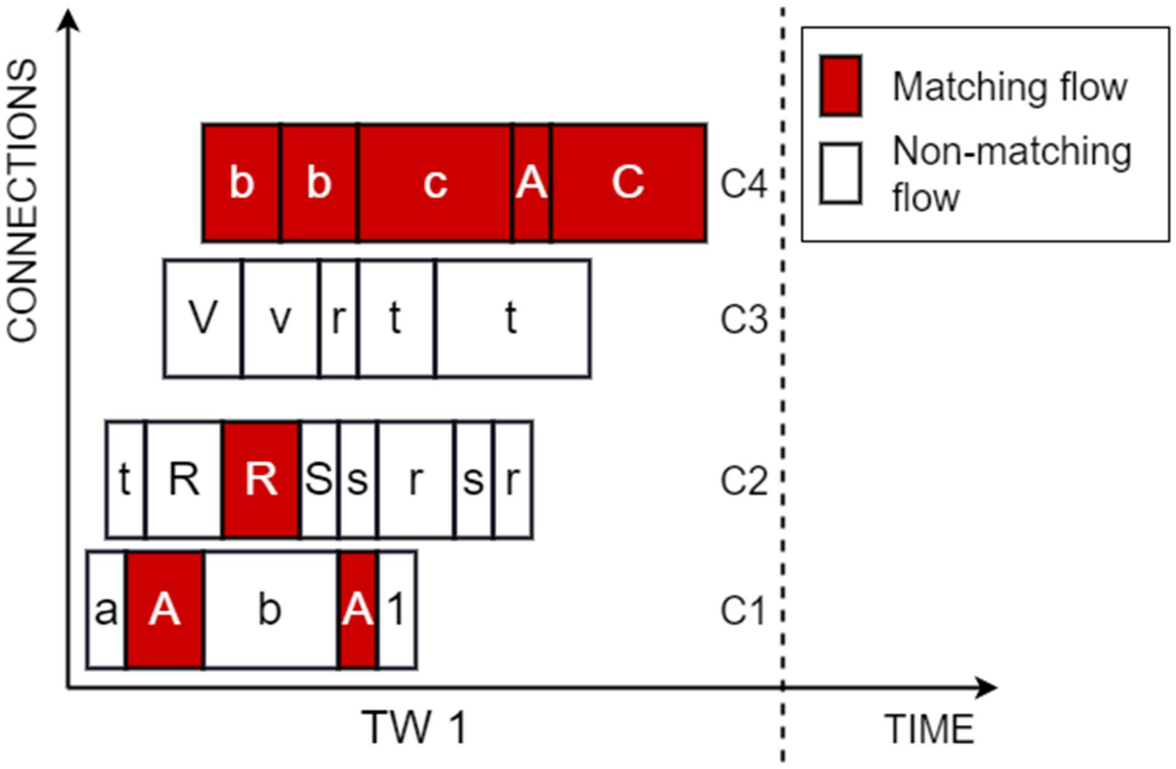


Figure 9: Extracting sum of CR from a single host

However, only the sum is not enough. Let us consider two hosts, both with 2 connections. Host A has one connection with all states matching some of the models and connection that has no such states. The sum of the CR of host A is 1. Host B has two connections with CR = 0.5. That means the sum of CR for B is also 1. Using mean of the connection ratios as one of the features increases the separability of connections. For host in Figure 9, the sum of CR = 1.5250 and mean of CR = 0.3813.

5.2. Sliding Detection Window (SDW)

When the extracting of features from a host, each TW is treated as an independent unit. However, the connection often last longer the 1 time window. In those cases, part of the information is being lost with the end of the time window. To prevent it, we propose the concept of the sliding frame in which information about previous actions of the host is gathered as a complement to the features extracted in the current TW. It is called Sliding Detection Window (SDW) and it is used for adding the short-term history in the decision-making process. Estimation of the ideal width of the SDW is a complex task which requires a variety of experiments. In this paper, we used a fixed width of both TW and SDW. A 5-minute time window is a default option of the original SLIPS. For the SDW we use the last 12 TW which is equal to the one hour of traffic. Any changes in the widths of either TW or SDW require re-training the classifier.

SDW collects the following information:

- **(i)** Amount of connections without any matching state
- **(ii)** Amount of connections with at least one matching state
- **(iii)** Number of connections with all matching states
- **(iv)** Percentage of connections without any matching state
- **(v)** Percentage of connections with at least one matching state
- **(vi)** Percentage of connections with all matching states

Features from the SDW are appended to the vector of features from the current time window and used as an input for the classifier. In Section 8, we show the comparison between classification using time window extended with SDW and using TW only.

5.3. Vector of features

When TW finishes, a vector of features is extracted from each host which has a connection updated state updated. The vector contains 18 elements – 12 features from the TW and 6 obtained from the SDW.

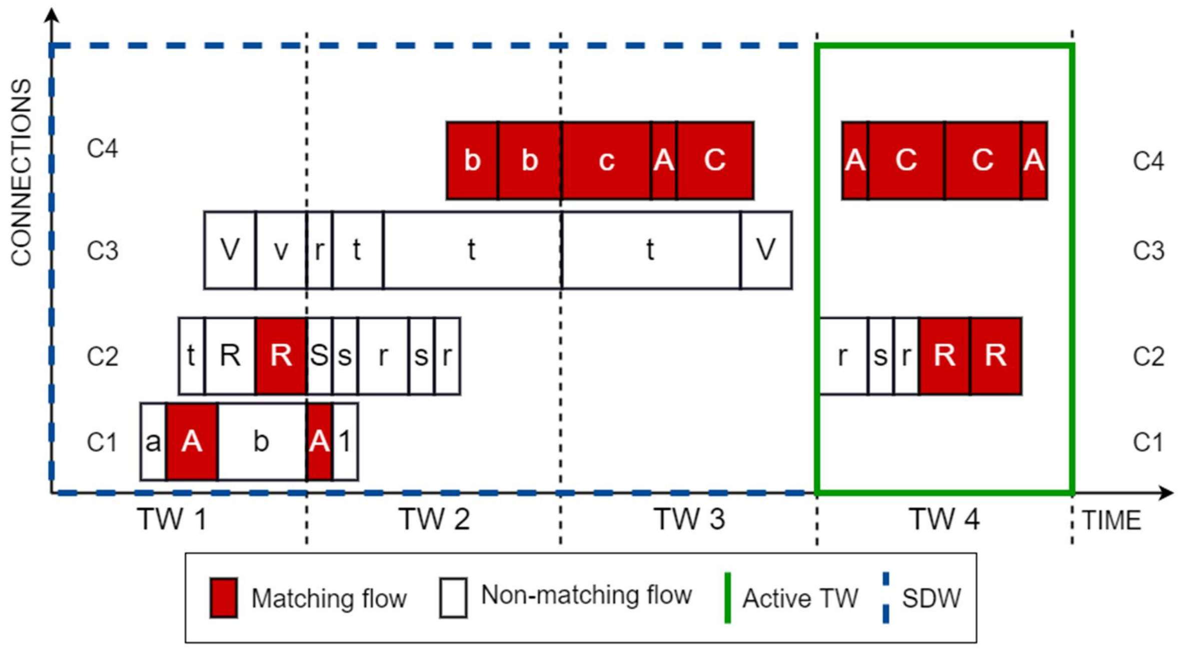


Figure 10: Extraction of feature vector from a single host

If we analyze the host from Figure 10 we obtain the following vector of features: (2; 1.4; 0.7; 0; 2; 1; 0; 1; 0.5; 9; 6; 0.6667; 1; 3; 1; 0.25; 0.75; 0.25). First 12 items belong to the current time window; the rest is extracted from the SDW of width 3.

5.4. Classifiers

Machine learning algorithms can be used for classifying the host based on the set of featured gathered in both time window and SDW. For that, a set of training samples is required to create the model. Statistical methods are used to estimate the parameters of the model. In our experiments, we used three types of classifiers. First is based on the Bayesian decision theory. It has been used in the similar problem with notable results. The second type of classifier is the Support Vector Machine (SVM) [14]. SVM is a robust classifier which can be used even in the cases with non-separable data. The last classifier we use is an algorithm called XGBoost [2] which is a shortcut for Extreme Gradient Boosting. XGBoost was introduced in 2016 and it combines features of several classifiers. We use XGBoost for its scalability and ability to generalize which is useful with the complex task such as malware detection.

5.4.1. Naive Bayes

Naïve Bayes (NB) method belongs to the family of algorithms based on the studies of Thomas Bayes (1702-1761). The core idea in the Bayesian classification is that there is a hypothesis that a sample belongs to a certain class. Evidence is then gathered and used for computing probability that the hypothesis is correct.

5.4.2. Support Vector Machine

Support Vector Machine (SVM) is a learning algorithm introduced by Vapnik and Chervonenkis [15] in 1979. However, improved version [14] using soft margin for dealing with outliers is currently the most commonly used.

It is capable of both linear and non-linear classification. SVM tries to find the hyperplane which divides the training samples in such way, that the margin between the samples and the hyperplane. The motivation behind that is the increase generalization abilities of the model. SVM is capable of both linear and non-linear classification. If the data is not linearly separable, Kernel trick [16] transform the data to a higher dimension. Although it helps the separation, it also influences the generalization model. In our experiments, we use Radial Basis function as a kernel function [17].

5.4.3. XGBoost

XGBoost (Extreme Gradient Boost) algorithm [2] which was introduced in 2016 and it has won several ML competitions since. It is based on the Gradient boosting Machine [18] introduced by Freidman in 1999. The algorithm combines techniques known from other classifiers such as prevention of overfitting with regularized model (used in Regularized Greedy Forrest [19]). Because of the variety of the types of malware, it is difficult to find a model with enough generalization abilities needed classifying different types of Malware. Another benefit of XGBoost is speed. The algorithm is designed to maximize efficiency and can be run in parallel. When running SLIPS online reducing the time complexity of the program is very important since lots of data needs to be processed in real-time.

6. Implementation

We used Python for implementing all the routines and algorithms. It is a simple yet extremely powerful tool especially suited for machine learning problems. For implementing the machine learning parts we used Scipy [20], XGBoost [2] and scikit-learn [21]. Those are highly optimized libraries which help to speed up the run of the tool. Matplotlib [22] was used to generate graphs and figures. Source code with all can be downloaded and used for free from **https://github.com/ondrej-lukas/StratosphereLinuxIPS**

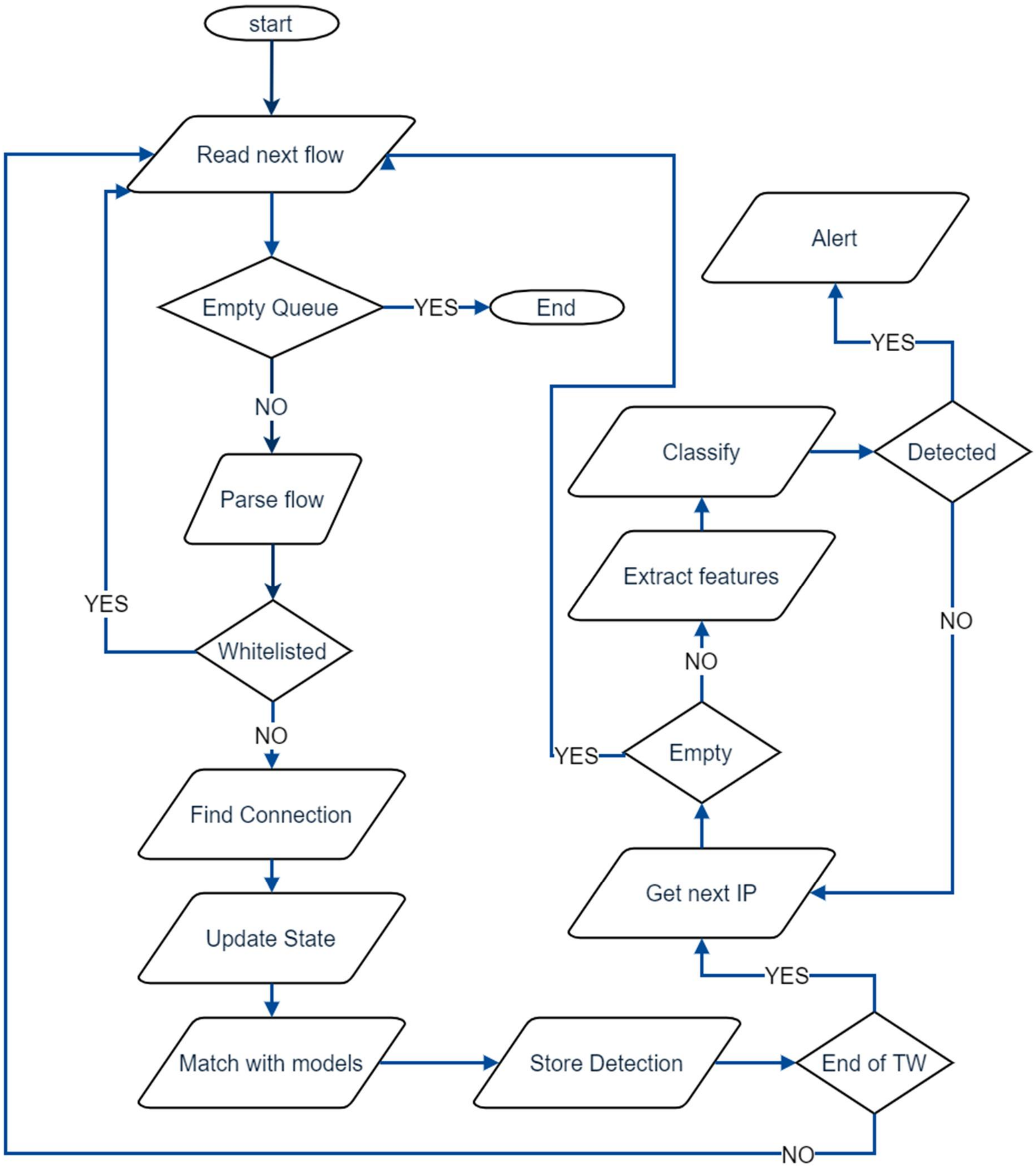


Figure 11: Workflow for processing binetflows with Source Address Layer

6.1. Structure of classes in Source Address Layer

Source Address Layer is built on the foundation of the original version. It uses the same interface for reading and parsing bidirectional flows. A system for updating states of the connection and its matching with malware models remains unchanged.

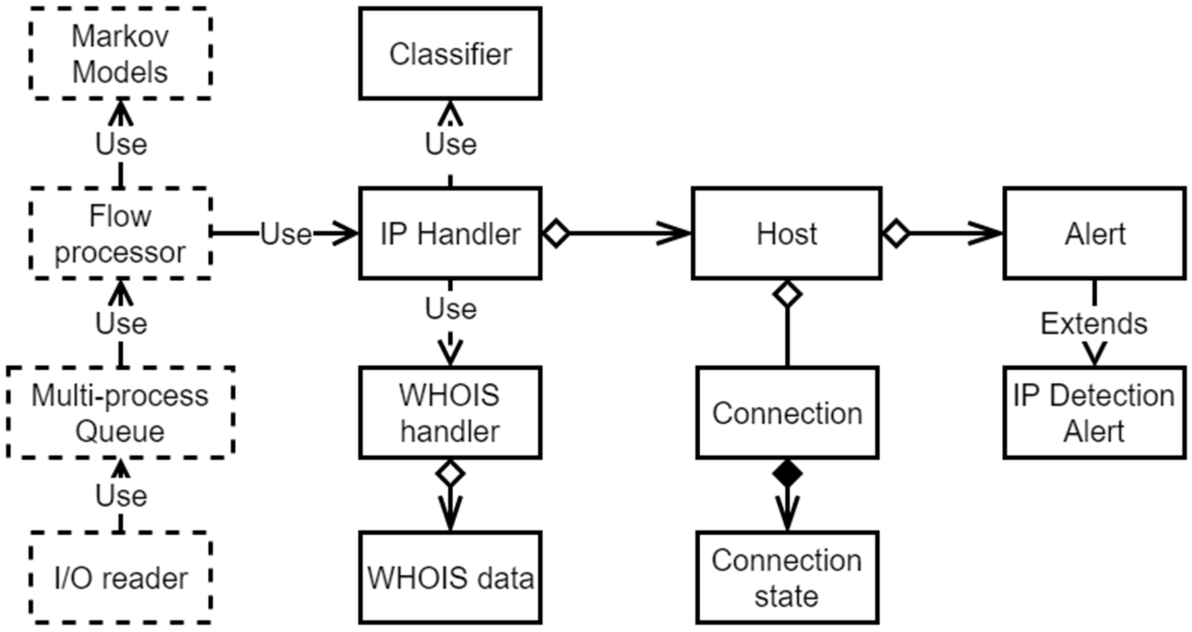


Figure 12: The Structure of classes in the Source. Address Layer

The core piece of the Sour Address Layer is an IPHandler class. It is responsible for storing the SourceAddress objects and information within them such as partial detection from Markov models, labels from the XGBoost and alerts corresponding to the. It serves as a bridge between the new functionality and structures of original SLIPS. Classifier class is used for the labeling. It is not designed for training which means pre-trained classifier should be provided in a form of serialized object.

When the flow is read and input for the IP Handler is a tuple containing the flow, and the result of matching from the Markov models (partial detection). The tuples are stored in the SourceAddress for analysis at the end of the time window. When TW finishes IPHandler selects hosts which state has been changed in the TW and extract the feature vectors. List of vectors is processed by the classifier which returns a list of labels. If the address is labeled ‘Malicious’ instance of the Alert object is created and stored in the Source Address.

6.2. Alerts

When a host is labeled as ‘Malicious’ user should be notified. Alerts in the original version were reduced to a simple print in the console. We introduced the basic Alert class and its derivate IPDetectionAlert. An instance of IPDetectionAlert is created upon any detection.

Once there is a host labeled as ‘Malicious’, the user is notified. In the original SLIPS, alerts were printed on the standard output. Because of the simplicity of this solution we created the Alert object which serves as a container for as much information about the detection as possible. If the detection should be checked manually, it is vital to provide the user with as much information as possible to make the verification simpler. Alerts include the source, time, WHOIS information about the destination address if such is available and the vector of features which was used for the classification. That is essential if we want to use the manual verification to re-train the classifier with additional data.

When the run of the program ends (both at the end of the capture or when interrupted) all alerts are stored in the log file as well as shown in the console.

6.3. WHOIS Information

One of the features SLIPS helps the users to verify detections is a WHOIS service[1]. It is free online databases containing the information about owners of individual domains. There is a library for Python which can search the online database but if there is a high number of queries for the database the response time rises. To prevent the creation of bottleneck because of the secondary feature of low importance, we created a WhoisHandler class which has an offline file containing the whois information and serves as a proxy for getting it. Before the query to the online database is created WhoisHandler searches the data stored locally for the presence of the information wanted. If the search is not the successful query is sent and the result is stored for further use. When the program starts, the content of the local WHOIS file is loaded into Python dictionary which eliminates any delay when searching for the information. File containing WHOIS information is distributed together with the rest of the scripts.

6.4. Performance

Since SLIPS is designed to run in real time, it is required to process a large amount of data very fast. There are hundreds of connections to be classified in each TW. Scipy library together with the scikit-learn provides us with highly optimized tools for data processing as well as a variety of machine learning algorithms. Because of the grouping on several levels, plenty of searching in the data structures is required. Python Dictionaries are very suitable for that as their average time complexity for most operations is O (1)[2].

6.5. Using SLIPS

SLIPS can be used both for real-time detection and analysis of the binetflow files. For the real-time detection, Argus tool must be installed and running to process the traffic and create the binetflow file.

Example of running SLIPS with real time data from Argus:

```
ra -F [slipsfolder]/ra.conf -n -Z b -S 127.0.0.1:902 | python slips.py -
f [models folder] -v 2 -d
```

An example of an offline run of slips with verbosity 2 and included WHOIS info:

```
cat [filename] | python slips.py -f [models folder] -v 2 -d
```

List of all available parameters can be displayed with the command:

```
python slips.py -h
```

[1] https://pypi.python.org/pypi/ipwhois
[2] https://wiki.python.org/moin/TimeComplexity

7. Dataset

For both training of the classifier and evaluation of its accuracy, choice of data is essential. Data used in all our experiments comes from the Stratosphere Project. For testing CTU-13-Dataset [3] has been used. It consists of a variety of different Malware scenarios as well as normal hosts and background traffic. That is useful for simulation of regular traffic. All datasets we use in our experiments are labeled based on deep manual analysis of the captures.

7.1. Training Data

For most machine learning algorithm structure of the training data is essential. We combined real captures of a variety of types of malware in different scenarios to avoid overfitting to just one kind of malware. Normal samples are included in the training set as well. Those were obtained by monitoring the traffic of common users in the university network as well as from the normal hosts included in the malware captures. Deeper information about the datasets can be found in the description at **https://stratosphereips.org/category/dataset.html**.

The balance of the samples in the training set is another key factor of learning. If the set it significantly imbalanced, learning of the classifier is biased towards the class with more samples in the set. Distribution of samples among each class is shown in the following table. The sample is a vector of features extracted from a host in one time window.

Dataset	Normal samples	Malicious Samples
CTU-Malware-Capture-Botnet-100	1,537	4,869
CTU-Malware-Capture-Botnet-119-2	1,048	5,226
CTU-Malware-Capture-Botnet-221-2	8,141	807
CTU-Malware-Capture-Botnet-244-1	1,495	7,801
CTU-Normal-3-Public	750	0
CTU-Normal-4-only-DNS	22	0
CTU-Normal-5	159	0
CTU-Normal-6-filtered	73	0
CTU-Malware-Capture-Botnet-50 (part)	272	0
CTU-Malware-Capture-Botnet-47 (part)	101	35
TOTAL	13,598	18,738

Table 4: Distribution of labels in the training set.

7.2. Testing Data

For testing, we use data from the CTU-13-Dataset. Each of them is a real capture of a malware executed in the Stratosphere Project. Type of malware being executed differ among dataset which simulates the real network traffic. The original dataset contains a lot of background traffic. For such traffic, the real label is very difficult to assign because on limited information about the host. Therefore, we filtered all background traffic from all datasets and for all experiments, we used only data from hosts which are manually checked.

Datasets used for testing:

Dataset	Normal samples	Malicious Samples
CTU-Malware-Capture-Botnet-42	192	58
CTU-Malware-Capture-Botnet-43	67	43
CTU-Malware-Capture-Botnet-44	998	184
CTU-Malware-Capture-Botnet-45	142	33
CTU-Malware-Capture-Botnet-46	17	5
CTU-Malware-Capture-Botnet-48	14	2
CTU-Malware-Capture-Botnet-51	147	214
TOTAL	1,577	539

Table 5: Distribution of labels in the testing set

8. Experiments

The main goal of the experiments is to measure the difference in the quality of detection of the original version of SLIPS and the implementation we propose. Secondly, we want to determine if classification based on Sliding Detection Window leads to better results compared to the usage of time window features only. Additionally, we show the process of training the classifier which is a part of the new version of SLIPS. For training, cross-validation and measurements of the performance we use tools from the scikit-learn library. Plots and graphs are created by Matplotlib [22] library.

Since the original SLIPS classifies each connection separately, we need to define a measurement which allows us to compare it to the system which is making a decision per host. For the original SLIPS, we consider host as detected (labeled as 'Malicious') in the time window if there is at least detected connection of the host in the TW. With that, we can compare the classification in each TW. A second measurement is the number of hosts reported over all time windows. The host is considered reported as 'Malicious' if it is detected in at least one TW. Each experiment is performed for the original version of SLIPS and both TW and SDW trained the classifier.

To demonstrate the process of training we show data from cross-validation which was used for tuning the parameters of the classifiers. 5-folds with Leave on out technique. Accuracy was the criterion used in the finding the optimal parameters. Because of the time complexity of the cross-validation. We used only 10 000 samples in the cross-validation.

Settings of parameters obtained from the cross-validation are used for the learning curves. Those show how the size of the testing dataset influences the performance.

8.1. Training

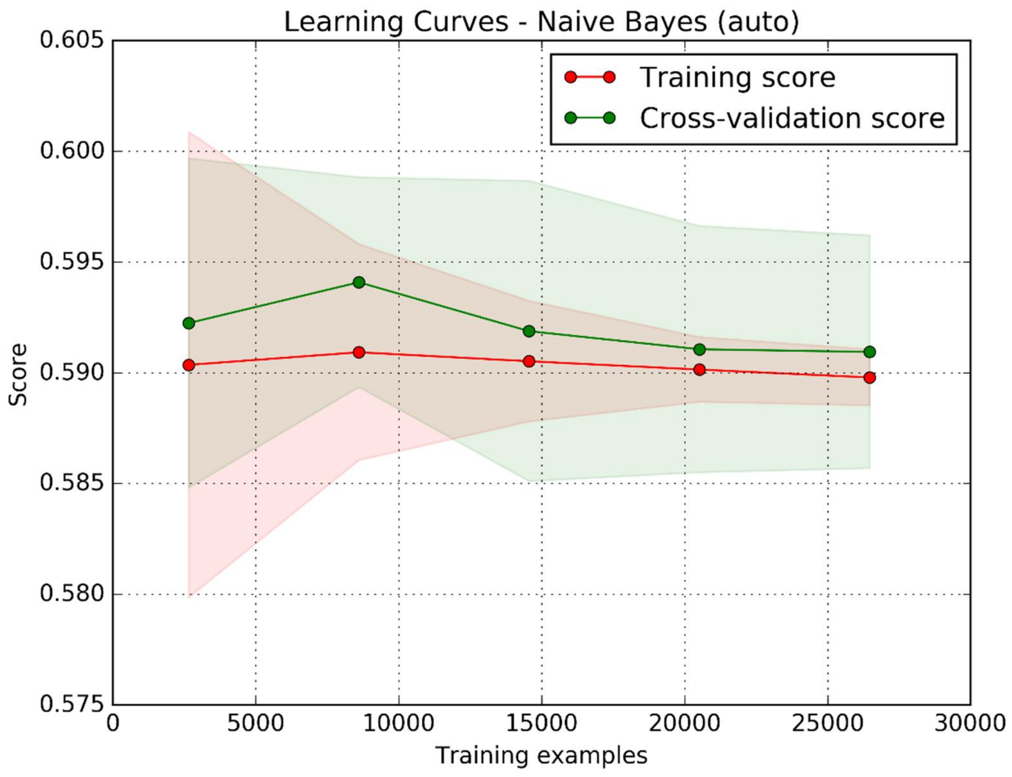


Figure 13: Learning Curves of Naive Bayes – TW only

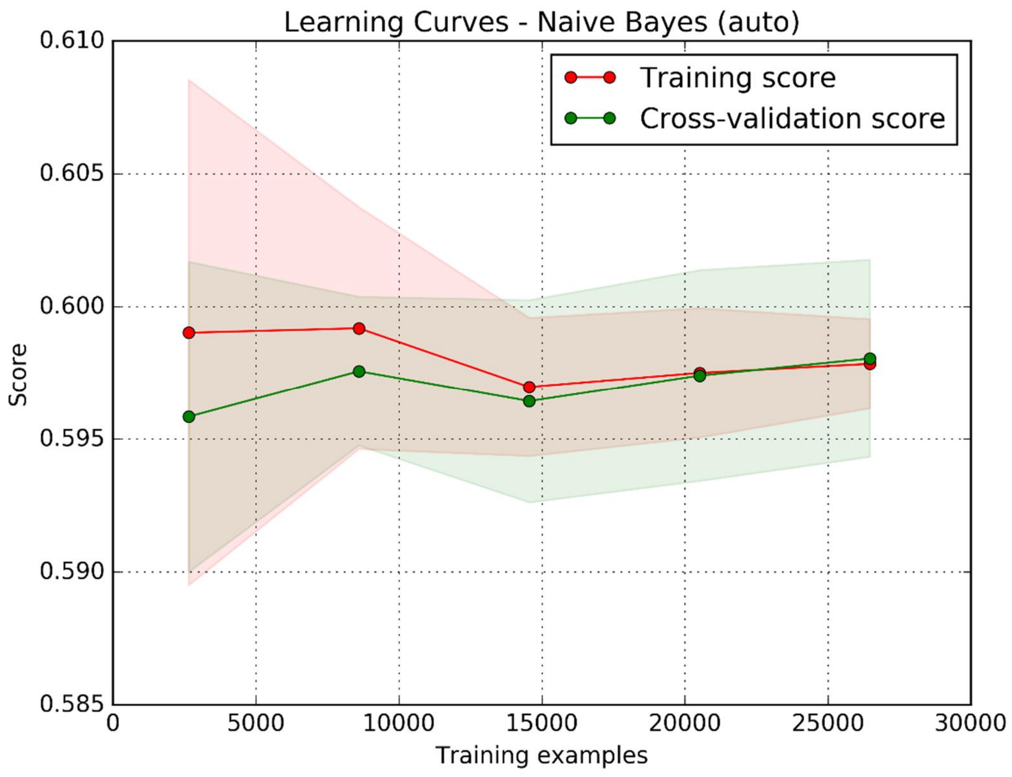


Figure 14: Learning Curves of Naive Bayes - TW&SDW

8.1.1. SVM

	gamma = 0.001	gamma = 0.0001	gamma = 1e-05
C = 1	0.630	0.627	0.597
C = 10	0.639	0.628	0.622
C = 100	0.806	0.638	0.629
C = 1000	0.826	0.645	0.634

Table 6: Cross-validation result SVM (parameters C, gamma) – TW only

	gamma = 0.001	gamma = 0.0001	gamma = 1e-05
C = 1	0.622	0.615	0.580
C = 10	0.618	0.610	0.610
C = 100	0.733	0.620	0.615
C = 1000	0.818	0.617	0.620

Table 7:Cross-validation result SVM (parameters C, gamma) – TW +SDW

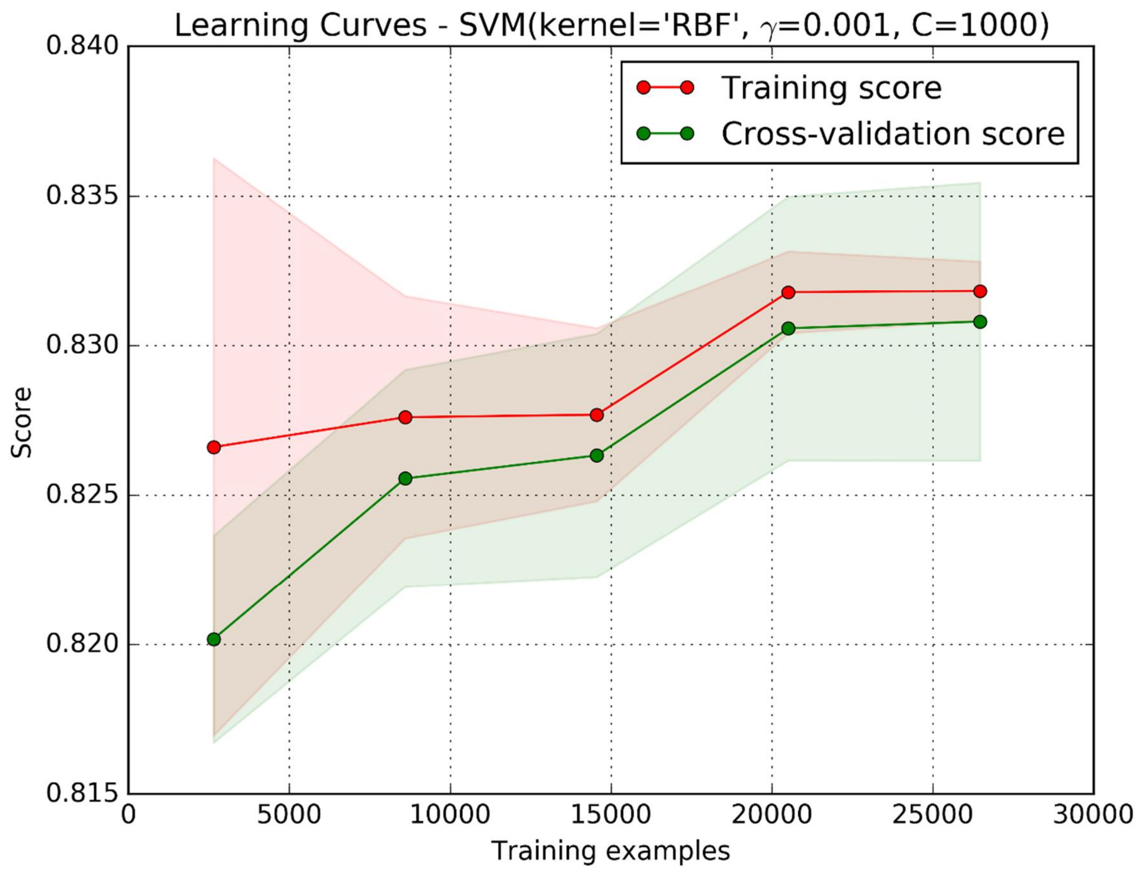


Figure 15: Learning curves of SVM – TW only

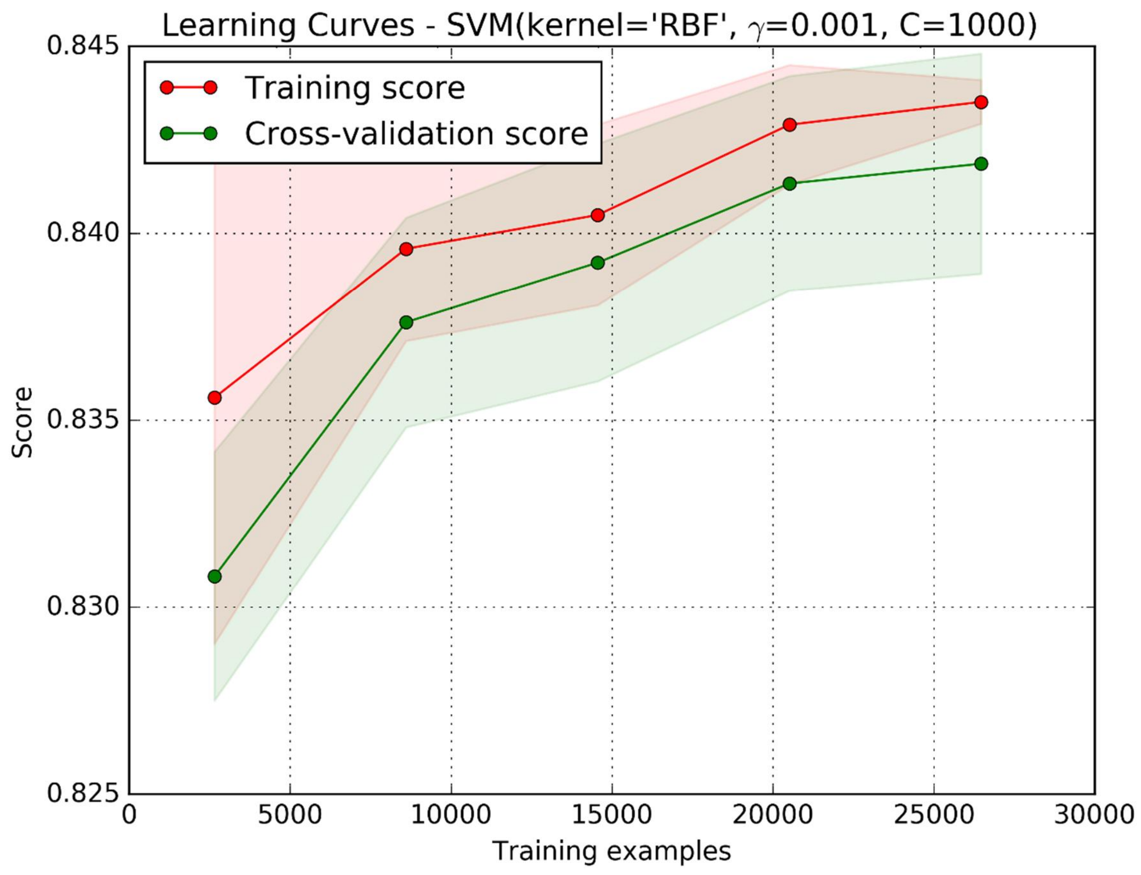


Figure 16: Learning curves of SVM – TW&SDW

8.1.2. XGBoost

Gamma= 0.1				Gamma= 0.2			
estimators	lr =0.2	lr=0.3	lr = 0.4	estimators	lr =0.2	lr=0.3	lr = 0.4
400	0.8250	0.8230	0.8270	400	0.8250	0.8200	0.8220
600	0.8250	0.8230	0.8270	600	0.8250	0.8200	0.8220
800	0.8250	0.8230	0.8270	800	0.8250	0.8200	0.8220
1,000	0.8250	0.8230	0.8270	1,000	0.8250	0.8200	0.8220
Gamma = 0.3				Gamma = 0.4			
estimators	lr =0.2	lr=0.3	lr = 0.4	estimators	lr =0.2	lr=0.3	lr = 0.4
400	0.8270	0.8180	0.8250	400	0.8180	0.8150	0.8170
600	0.8270	0.8180	0.8250	600	0.8180	0.8150	0.8170
800	0.8270	0.8180	0.8250	800	0.8180	0.8150	0.8170
1,000	0.8270	0.8180	0.8250	1,000	0.8180	0.8150	0.8170

Figure 17: Cross-validation results for XGBoost using TW features only[3]

[3] lr is a shortcut for Learning rate

Gamma= 0.1				Gamma= 0.2			
estimators	lr =0.2	lr=0.3	lr = 0.4	estimators	lr =0.2	lr=0.3	lr = 0.4
400	0.8220	0.8170	0.8130	400	400	0.8200	0.8170
600	0.8220	0.8170	0.8130	600	600	0.8200	0.8170
800	0.8220	0.8170	0.8130	800	800	0.8200	0.8170
1,000	0.8220	0.8170	0.8130	1,000	1,000	0.8200	0.8170
Gamma = 0.3				Gamma = 0.4			
estimators	lr =0.2	lr=0.3	lr = 0.4	estimators	lr =0.2	lr=0.3	lr = 0.4
400	0.8200	0.8230	0.8270	400	0.8220	0.8200	0.8250
600	0.8200	0.8230	0.8270	600	0.8220	0.8200	0.8250
800	0.8200	0.8230	0.8270	800	0.8220	0.8200	0.8250
1,000	0.8200	0.8230	0.8272	1,000	0.8220	0.8200	0.8250

Figure 18: Cross-validation results for XGBoost - results with TW&SDW

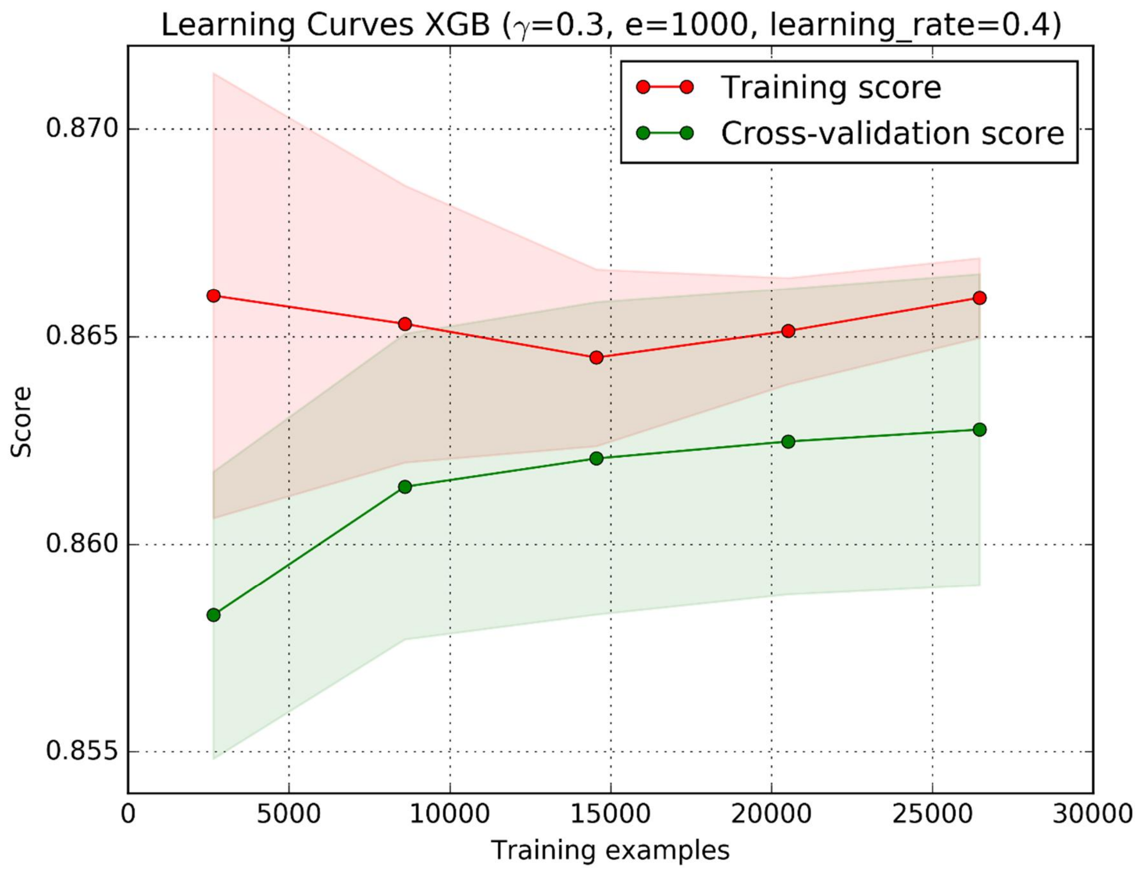


Figure 19: Learning curve of XGBoost – TW only

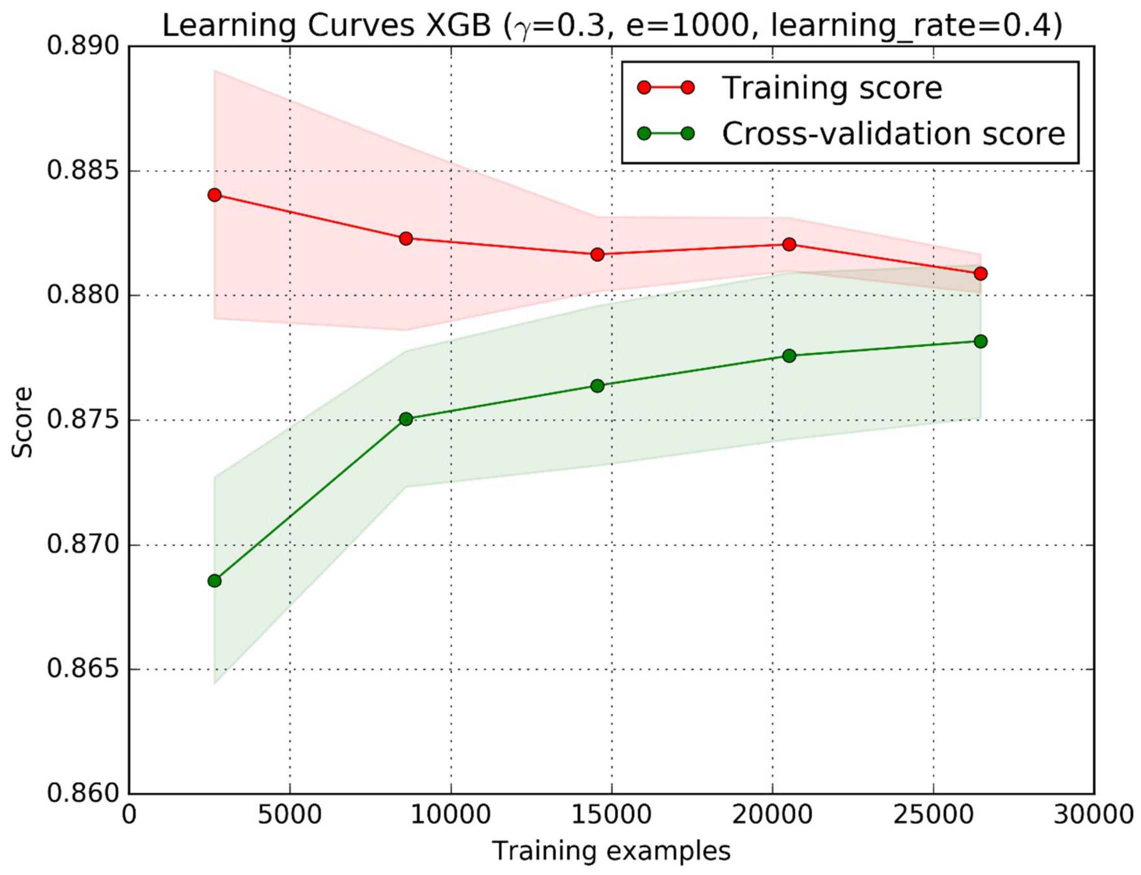


Figure 20: Learning curve of XGB - TW&SDW

8.2. Testing of performance

8.2.1. Performance comparison per TW

	TW&SDW	TW only	Original
True positive rate	0.2206	0.2206	0.0021
True negative rate	0.9362	0.9250	0.9014
Precision	0.5755	0.5311	0.3663
Negative predictive value	0.7638	0.7624	0.4159
False positive rate	0.0638	0.0750	0.0986
False discovery value	0.4245	0.4689	0.6337
False negative rate	0.7794	0.7794	0.9979
Accuracy	0.7445	0.7359	0.4171
F1-score	0.2965	0.2832	0.0041

Table 8: Performance measurements per TW

	TW&SDW	TW only	Original
True positive rate	0.6404	0.8571	0.4286
True negative rate	0.7366	0.3333	0.6429
Precision	0.6278	0.4432	0.2667
Negative predictive value	0.8972	0.9000	0.7381
False positive rate	0.2634	0.6667	0.3571
False discovery value	0.3722	0.5568	0.7333
False negative rate	0.3596	0.1429	0.5714
Accuracy	0.7693	0.5504	0.5458
F1-score	0.5541	0.5528	0.2619

Table 9: Performance measurements per capture

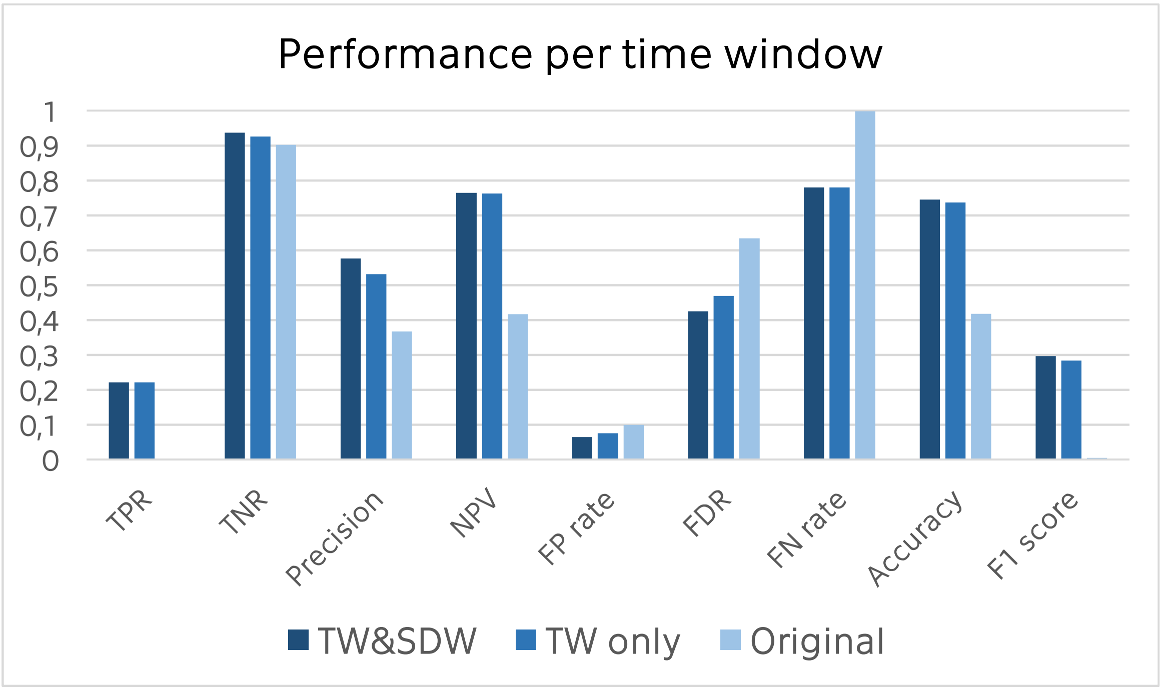


Figure 21: Performance comparison – measurements per time window

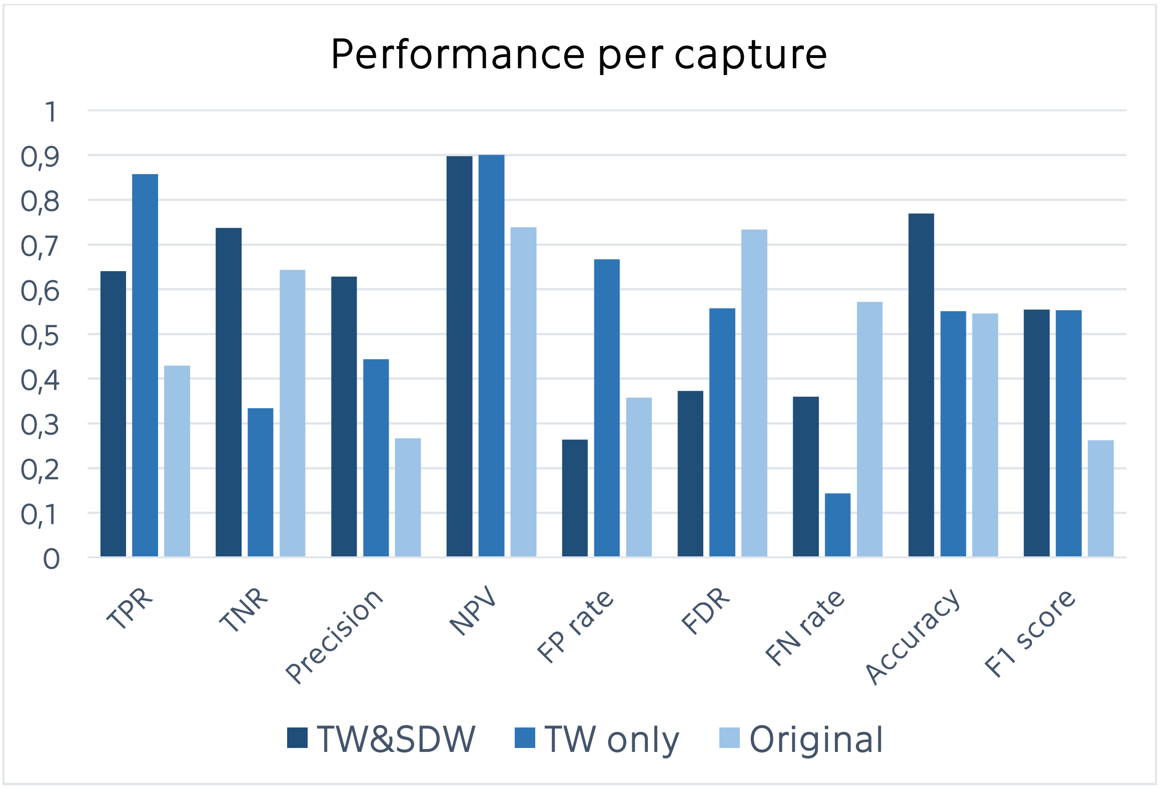


Figure 22: Performance comparison – measurements per capture

9. Results analysis

Apart from accuracy, the most watched measurement for network protection systems is a number of False positives due to the costs connected to them. Because of that, we focus mainly on three types of measurement in the testing: False Positive rate (also known False alarm ratio), Accuracy and F1 score which is a weighted average of precision and recall. The F1 score shows the balance between False positives and False negatives. When tuning the parameters of the classifiers we used accuracy as a leading measure of quality.

9.1. Training

From the Learning curves, we see that accuracy of Naïve Bayes in much lower than other classifiers. There are several reasons for that. Firstly, the Bayesian method requires prior probabilities of each class. However, estimation of this values in the real world in nearly impossible. We can also see that the variance of in the results of NB. Naive Bayes method assumes of independence of features. When processing partial detections from Markov models such condition is violated.

SVM and XGBoost have similar results. Analysis of the learning curves indicates that with more training samples, results of both algorithms could improve. When comparing the variance of the results, XGBoost outruns the SVM. Higher variance is caused by overfitting the model to the training data. That is undesirable behavior. After the examination of the learning process, XGBoost is suggested to be used for classification in SLIPS.

9.2. Performance per TW

The graph in Figure 21 shows that the similarities in the performance of both classifiers using SDW and using features from TW only. However, compared to the original version, we can see improvement in most of the quality measurements. We can see that using only the features from the time window increases the volume of correctly identified attacks. When the history is included with the SDW, accuracy is yet increased while FP rate drops by 14%. Comparison of SDW with the original version is even more obvious.

The increase in accuracy is 78% in favor of SDW and the F1 score changed from 0.0041 for original version to 0.2965 with SDW which more than 70 times better result. The main reason for this is an enormous number of false negatives of the original version. In other words, the original version is too moderate. However, it is important to note that when measured per time window is disadvantageous for the original version because of assigning a label to a host based on each of its connections.

9.3. Performance per capture

When comparing results at the end of a capture we can see that using features from TW only increases the True Positive rate significantly.

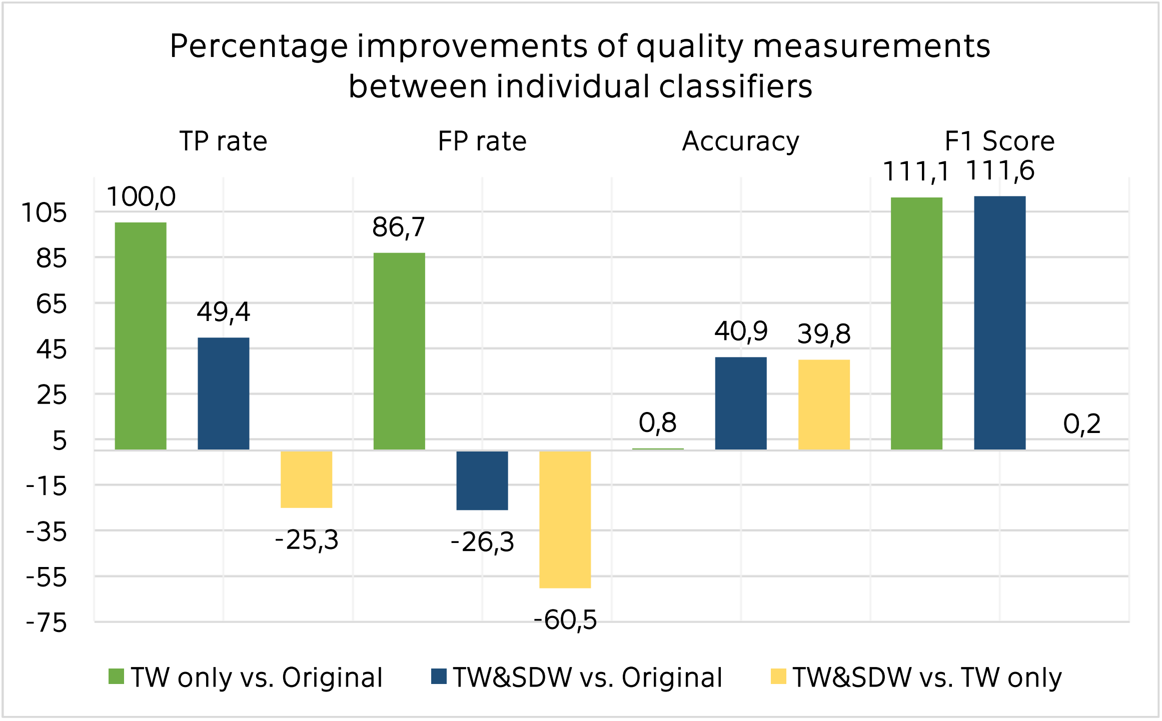


Figure 23: Comparison of percentage improvement in quality measurements between individual classifiers per capture.

However, Figure 23 shows that cost for high TP rate is an increase in the FP rate, which is increased by 86% compared to the original version. The difference in the accuracy in the original version and usage of the time window only is minimal. From the user point of view, false alarms are a great issue which means that using the classification based on single time window is not a valid replacement of the current approach.

Results of the classifier based on the SDW are much more promising. Detections of SDW are 40% more accurate than either in original version or when using single

TW only. There is also a significant drop in the False positive ratio – more half of the value of the single TW classifier and 26% decrease compared to the original SLIPS.

The results support the hypothesis that records of short-term history can help to improve the performance of SLIPS both in the more accurate detections and reduced amount of false alarms.

10. Conclusion

In this thesis, we presented a possible extension of the Stratosphere IPS tool based on aggregation of partial detection and its impact on the performance of the tool. We

described the designed and implementation details of Source Address Layer with and its relation to the existing version of the SLIPS.

As part of the technique, we defined a set of features which define the behavior of a host in one time window and presented their extraction. We also presented the concept of Sliding Detection Window and its impact on the quality of detection. We showed the process of aggregating and processing partial detections and demonstrated how the processed data can be used with the machine learning algorithms. We examined the performance of different types of classifiers when using only data from single time window and when the short-term from SDW history is included.

The main contribution of the thesis is a method of grouping connections by hosts in the Source Address Layer and gathered information to improve the quality of detections infected hosts. We experimented with several classifiers such as Support Vector Machine(SVM) and XGBoost (XGB) and showed the quality of their predictions using the CTU-13 Dataset. Based on the experiments, we picked the XGBoost algorithm, tuned its parameters and trained it. Serialized trained XGB, which was published together with the new SLIPS, is another contribution of the thesis. It is ready to use for users without any inside knowledge. We also created a new alerting system for SLIPS which simplifies the output of the program for a user and allows easier verification of the detections.

To verify the impact of our technique we designed a set of test in which we compared performance in detecting an infected host of the original SLIPS and our extended version. The tests proved that by using the aggregated data processed by the Source Address layer accuracy of detections is increased by 40% while the False positive rate decreased by 30%. F1 score, which is another measure of quality shows an increase over 110%. Therefore, we proved that our approach helps to solve some of the main flaws of the original SLIPS.

During the testing, we experimented with the inclusion of the short-term history of host behavior in the classification process and proved, that by using SDW to gather information about the close history, both precision and accuracy of detections are improved.

In the future, we would like to extend SLIPS in a way of variable width of both time windows and SDW. Also, we would like to focus on the system of using the user-verified alerts to re-training the classifier and boosting its performance. Another possible way of continuation of the research is a potential usage of neural networks and deep learning in the SLIPS.

References

[1] S. Garcia, Identifying Modeling and Detecting Botnet Behaviors in the Network, Tandil, Buenos Aires, Argentina: UNICEN, 2014.

[2] T. Chen and C. Guestrin, XGBoost: A Scalable Tree Boosting System, University of Washington, 2016.

[3] S. Garcia, M. Grill, J. Stiborek and A. Zunino, "An empirical comparison of botnet detection methods," *Computers and Security Journal, Elsevier,* pp. 100-123, 2014.

[4] S. T. Ikram and C. K., "Improving Accuracy of Intrusion Detection Model Using PCA and optimized SVM," *Journal of Computing and Information Technology,* vol. 24, no. 2, p. 133–148, 2016.

[5] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi and M. Ghogho, "Deep learning approach for Network Intrusion Detection in Software Defined Networking," in *Wireless Networks and Mobile Communications (WINCOM), 2016*, Fez, Morroco, 2016.

[6] C. Kruegel, D. Mutz, W. Robertson and ValeurFredrik, "Bayesian Event Classification for Intrusion Detection," in *Computer Security Applications Conference*, Las Vegas, NV, USA, USA, 2003.

[7] P. T., "Using Adaptive Alert Classification to Reduce False Positives in Intrusion Detection," *Recent Advances in Intrusion Detection,* vol. 3224, pp. 102-124, 2004.

[8] T. Pietraszek and A. Tanner, "Data mining and machine learningd Towards reducing false positives in intrusion detection," *Information Security Technical Report,* vol. 10, no. 3, pp. 169-183, 2005.

[9] R. Perdisci, G. Giacinto and F. Roli, "Alarm clustering for intrusion detection systems in computer networks," *Engineering Applications of Artificial Intelligence ,* no. 19, pp. 429-438, 2006.

[10] F. Xiao and X. Li, "Using Outlier Detection to Reduce False Positives in Intrusion Detection," in *IFIP International Conference on Network and Parallel Computing*, Shanghai, China, China, 2008.

[11] Katar and Chaker, "Combining Multiple Techniques for Intrusion Detection," *International Journal of Computer Science and Network Security,* vol. 6, no. 2, pp. 208-218, 2006.

[12] S. Garcia, "Stratosphere IPS Project," [Online]. Available: https://stratosphereips.org/. [Accessed 2 20 2017].

[13] QoSient, "Argus," QoSient, LLC, 2000. [Online]. Available: https://qosient.com/argus/. [Accessed 17 3 2017].

[14] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning* , vol. 20, no. 3, p. pp 273–297, 1995.

[15] V. Vapnik and A. Chervonenkis, Theory of Pattern Recognition, Moscow: Nauka, 1979.

[16] Y. Cho and L. K. Saul, "Kernel Methods for Deep Learning," in *Advances in Neural Information Processing Systems 22*, Vancouver, B.C., Canada, 2009.

[17] B. Schölkopf, K.-K. Sung, C. J. C. Burges, F. Girosi and V. Vapnik, "Comparing Support Vector Machines with Gaussian Kernels to Radial Basis Function Classifiers," *IEEE TRANSACTIONS ON SIGNAL PROCESSING,* vol. 45, no. 11, pp. 2758-2765, 1997.

[18] J. H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," *The Annals of Statistics,* vol. 29, no. 5, pp. 1189-1232, 2001.

[19] T. Zhang and R. Johnson, "Learning nonlinear functions using regularized greedy forest," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 36, no. 5, 2014.

[20] E. Jones, E. Oliphant, P. Peterson and e. al, "SciPy: Open Source Scientific Tools for Python," 2001. [Online]. Available: http://www.scipy.org/ . [Accessed 30 4 2017].

[21] P. Fabian and Et Al., "Scikit-learn: Machine Learning in Python," in *Journal of Machine Learning Research*, Brookline, MA, Microtome Publishing, 2011, pp. 2825-2830.

[22] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering,* no. 9, pp. 90-95, 2007.

[23] G. P. Spathoulas and S. K. Katsikas, "Reducing false positives in intrusion detection systems," *Computers & Security,* no. 29, pp. 35-44, 2010.

[24] W. Meng, X. Luo, S. Furnell and J. Zhou, Protecting Mobile Networks and Devices: Challenges and Solutions, Auerbach Publications, November 17, 2016.

Appendix

In the first part of the appendix, results of individual experiments are listed. The second part contains a table of symbols which are commonly used throughout the thesis and their explanation. The content of the CD attached to the thesis is described in the last part of the appendix.

Detailed results of testing

In Subsection 8.2 only the average results were shown. In this section of appendix results of experiments in individual datasets are listed. Data is divided by the time of experiment: measurements per TW means that the confusion matrix was updated in every time window while in the results per capture confusion matrix was created after all input file was processed. All datasets are part of CTU-13-Dataset. Numbers in the dataset name correspond with the labels used in CTU-13-Dataset. For example, Dataset 42 is equal to CTU-Malware-Capture-Botnet-42.

	TW&SDW	TW only	Original
TPR	0.4828	0.4828	0.0027
TNR	0.9896	0.9115	0.3333
Precision	0.9333	0.6222	0.9355
NPV	0.8636	0.8537	0.0001
FP rate	0.0104	0.0885	0.6667
FDR	0.0667	0.3778	0.0645
FN rate	0.5172	0.5172	0.9973
Accuracy	0.8720	0.8120	0.0027
F1 score	0.6364	0.5437	0.0053

Table 10: Measurements per time window – Dataset 42

	TW&SDW	TW only	Original
TPR	1.0000	1.0000	1.0000
TNR	0.6667	0.3333	0.3333
Precision	0.5000	0.3333	0.3333
NPV	1.0000	1.0000	1.0000
FP rate	0.3333	0.6667	0.6667
FDR	0.5000	0.6667	0.6667
FN rate	0.0000	0.0000	0.0000
Accuracy	0.7500	0.5000	0.5000
F1 score	0.6667	0.5000	0.5000

Table 11: Measurements per capture – Dataset 42

	TW&SDW	TW only	Original
TPR	0.8605	0.4186	0.0118
TNR	0.9254	0.9701	0.9921
Precision	0.8810	0.9000	0.8852
NPV	0.9118	0.7222	0.1619
FP rate	0.0746	0.0299	0.0079
FDR	0.1190	0.1000	0.1148
FN rate	0.1395	0.5814	0.9882
Accuracy	0.9000	0.7545	0.1700
F1 score	0.8706	0.5714	0.0233

Table 12: Measurements per time window – Dataset 43

	TW&SDW	TW only	Original
TPR	1.0000	1.0000	1.0000
TNR	0.5000	0.0000	0.5000
Precision	0.5000	0.3333	0.5000
NPV	1.0000	NaN	1.0000
FP rate	0.5000	1.0000	0.5000
FDR	0.5000	0.6667	0.5000
FN rate	0.0000	0.0000	0.0000
Accuracy	0.6667	0.3333	0.6667
F1 score	0.6667	0.5000	0.6667

Table 13: Measurements per capture– Dataset 43

	TW&SDW	TW only	Original
TPR	0.0870	0.0163	0.0000
TNR	0.9940	0.9930	0.9909
Precision	0.7273	0.3000	0.0106
NPV	0.8552	0.8456	0.2755
FP rate	0.0060	0.0070	0.0091
FDR	0.2727	0.7000	0.9894
FN rate	0.9130	0.9837	1.0000
Accuracy	0.8528	0.8409	0.2748
F1 score	0.1553	0.0309	0.0001

Table 14: Measurements per time window - Dataset 44

	TW&SDW	TW only	Original
TPR	1.0000	1.0000	1.0000
TNR	0.6667	0.3333	0.6667
Precision	0.5000	0.3333	0.5000
NPV	1.0000	1.0000	1.0000
FP rate	0.3333	0.6667	0.3333
FDR	0.5000	0.6667	0.5000
FN rate	0.0000	0.0000	0.0000
Accuracy	0.7500	0.5000	0.7500
F1 score	0.6667	0.5000	0.6667

Table 15: Measurements per capture- Dataset 44

	TW&SDW	TW only	Original
TPR	0.0909	0.3333	0.0000
TNR	1.0000	0.8592	0.9969
Precision	1.0000	0.3548	0.0000
NPV	0.8256	0.8472	0.6892
FP rate	0.0000	0.1408	0.0031
FDR	0.0000	0.6452	1.0000
FN rate	0.9091	0.6667	1.0000
Accuracy	0.8286	0.7600	0.6878
F1 score	0.1667	0.3438	0.0000

Table 16: Measurements per time window - Dataset 45

	TW&SDW	TW only	Original
TPR	1.0000	1.0000	0.0000
TNR	1.0000	0.3333	0.3333
Precision	1.0000	0.3333	0.0000
NPV	1.0000	1.0000	0.5000
FP rate	0.0000	0.6667	0.6667
FDR	0.0000	0.6667	1.0000
FN rate	0.0000	0.0000	1.0000
Accuracy	1.0000	0.5000	0.2500
F1 score	1.0000	0.5000	0.0000

Table 17: Measurements per capture- Dataset 45

	TW&SDW	TW only	Original
TPR	0.0000	0.2000	0.0000
TNR	1.0000	1.0000	1.0000
Precision	NaN	1.0000	NaN
NPV	0.7727	0.8095	0.6438
FP rate	0.0000	0.0000	0.0000
FDR	NaN	0.0000	Nan
FN rate	1.0000	0.8000	1.0000
Accuracy	0.7727	0.8182	0.6438
F1 score	0.0000	0.3333	0.0000

Table 18: Measurements - Dataset 46 (per TW)

	TW&SDW	TW only	Original
TPR	0.0000	1.0000	0.0000
TNR	1.0000	1.0000	1.0000
Precision	NaN	1.0000	NaN
NPV	0.7500	1.0000	0.7500
FP rate	0.0000	0.0000	0.0000
FDR	NaN	0.0000	NaN
FN rate	1.0000	0.0000	1.0000
Accuracy	0.7500	1.0000	0.7500
F1 score	0.0000	1.0000	0.0000

Table 19: Measurements per capture - Dataset 46

	TW&SDW	TW only	Original
TPR	0.0000	0.0000	0.0000
TNR	0.9286	0.8571	1.0000
Precision	0.0000	0.0000	NaN
NPV	0.8667	0.8571	0.9022
FP rate	0.0714	0.1429	0.0000
FDR	1.0000	1.0000	NaN
FN rate	1.0000	1.0000	1.0000
Accuracy	0.8125	0.7500	0.9022
F1 score	0.0000	0.0000	0.0000

Table 20: Measurements per time window- Dataset 48

	TW&SDW	TW only	Original
TPR	0.0000	0.0000	0.0000
TNR	0.6667	0.3333	1.0000
Precision	0.0000	0.0000	NaN
NPV	0.6667	0.5000	0.7500
FP rate	0.3333	0.6667	0.0000
FDR	1.0000	1.0000	NaN
FN rate	1.0000	1.0000	1.0000
Accuracy	0.5000	0.2500	0.7500
F1 score	0.0000	0.0000	0.0000

Table 21: Measurements per capture- Dataset 48

	TW&SDW	TW only	Original
TPR	0.0607	0.0935	0.0000
TNR	0.9796	0.8844	0.9963
Precision	0.8125	0.5405	0.0000
NPV	0.4174	0.4012	0.2388
FP rate	0.0204	0.1156	0.0037
FDR	0.1875	0.4595	1.0000
FN rate	0.9393	0.9065	1.0000
Accuracy	0.4349	0.4155	0.2386
F1 score	0.1130	0.1594	0.0000

Table 22: Measurements per time window- Dataset 51

	TW&SDW	TW only	Original
TPR	1.0000	1.0000	0.0000
TNR	0.3333	0.0000	0.6667
Precision	0.8333	0.7692	0.0000
NPV	1.0000	NaN	0.1667
FP rate	0.6667	1.0000	0.3333
FDR	0.1667	0.2308	1.0000
FN rate	0.0000	0.0000	1.0000
Accuracy	0.8462	0.7692	0.1538
F1 score	0.9091	0.8696	0.0000

Table 23: Measurements per capture- Dataset 51

Commonly used symbols

Symbol	Explanation
SLIPS	Stratosphere Intrusion Protection System for Linux
TW	Time window
SDW	Sliding detection window
SVM	Support Vector Machine
Cr	Connection Ratio
NB	Naive Bayes
XGBoost	Extreme Gradient Boosting algorithm
TPR	True positive rate
TNR	True negative rate
NPR	Negative predicted value
FDR	False discovery rate
FP	False positive
FN	False negative

Content of the CD

Attached CD contains the text of this thesis, all python scripts, and source code. Additionally, trained ready-to-use XGBoost classifier in a form of serialized object and WHOIS file containing used as a local storage for whois information. Two binetflows for testing are included together with their description. Lastly, results of experiments and corresponding graphs and plots are stored on the CD.