Frequency analysis and online learning in malware detection

Huynh, Ngoc Anh

2019

Huynh, N. A. (2019). Frequency analysis and online learning in malware detection. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/93574>

<https://doi.org/10.32657/10356/93574>

NANYANG TECHNOLOGICAL UNIVERSITY

Frequency Analysis and Online Learning in Malware Detection

PhD Dissertation

By

Huynh Ngoc Anh

Supervisor: Dr. Ng Wee Keong

Co-supervisor: Dr. Jörn Kohlhammer

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

December, 2017

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

9/15/19

A handwritten signature in black ink, consisting of several loops and a long horizontal stroke extending to the right.

Huynh Ngoc Anh

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

15 / 9 / 2019

Ng Wee Keong

Authorship Attribution Statement

This thesis contains material from 4 papers published in the following peer-reviewed journal(s) and from papers accepted at conferences in which I am listed as an author.

Chapter 3 is published as N. A. Huynh, W. K. Ng, and H. G. Do. On periodic behavior of malware: experiments, opportunities, and challenges. In 2016 11th International Conference on Malicious and Unwanted Software (MALWARE), pages 1-8 (2016).

The contributions of the co-authors are as follows:

- A/P W. K. Ng provided the initial project idea.
- I designed the study of the malware samples and performed all the laboratory work at the School of Computer Science and Engineering.
- I prepared the manuscript drafts. The manuscript was revised by A/P W. K. Ng and Dr H. G. Do.

Chapter 4 is published as N. A. Huynh, W. K. Ng, A. Ulmer, and J. Kohlhammer. Uncovering periodic network signals of cyber attacks. In 2016 IEEE Symposium on Visualization for Cyber Security (VizSec), pages 1-8 (2016).

The contributions of the co-authors are as follows:

- A/P W. K. Ng and A/P J. Kohlhammer suggested the research idea, which is the collaboration between Nanyang Technical University and Technical University of Darmstadt.
- I implemented the prototype visualization and performed all the experiments.
- I wrote the drafts of the manuscript. The manuscript was revised together with A/P W. K. Ng, A. Ulmer, and A/P J. Kohlhammer.

Chapter 5 is published as N. A. Huynh, W. K. Ng, and K. Kanishka. A new adaptive learning algorithm and its application to online malware detection. In 20th International Conference on Discovery Science, DS 2017, pages 18-32 (2017).

The contributions of the co-authors are as follows:

- A/P W. K. Ng suggested the idea of using online machine learning for malware detection.

- I performed all the experiments to collect the behavioral artifacts generated by the malware samples using the facility provided by the DeterLab.
- I wrote the drafts of the manuscript. The manuscript was revised together with A/P W. K. Ng and Dr K. Kanishka.

Chapter 6 is published as N. A. Huynh, W. K. Ng, and K. Kanishka. Learning under concept drift with follow the regularized leader and adaptive decaying proximal. Journal of Expert Systems with Applications, volume 96, pages 49-63 (2018).

The contributions of the co-authors are as follows:

- A/P W. K. Ng suggested improving the idea in Chapter 5.
- I came up with adaptive mechanism of the algorithm and performed all the experiments to validate the idea.
- I wrote the drafts of the manuscript. The manuscript was revised together with A/P W. K. Ng and Dr K. Kanishka.

9/15/19

A handwritten signature in black ink, consisting of several loops and a long horizontal stroke extending to the right.

Huynh Ngoc Anh

Acknowledgments

First and foremost, I would like to thank my thesis advisor Dr. Ng Wee Keong, who has accepted me as a student to carry out research in Computer Science although my background was Electrical Engineering. The first two years was really a struggle for someone who ambitiously started his Ph.D. but could only make slow progress due to the transition from course-based learning to research-based learning. This time was full of doubts and ideas to quit but he was always patient in guiding me with constructive criticism and leaving enough room for my self-discovery.

I would like to extend my thanks to Dr. John and Alex as well as other colleagues in the Fraunhofer Institute for Graphical Data Processing. Following the cultural norm in Darmstadt IGD, I carried out a small but important experiment. I set a rule to work exactly 8 hours per weekday and no work in weekends. The important lesson was productivity. With the limited hours everyday, I was forced to better manage my time and avoid many sleepless nights, which was the practice till then. In addition, I would like to thank Dr. Kuijper for his helpful tips on research publication.

I would also like to thank Dr. Do and Dr. Ariyapala for fruitful discussions on my latest crazy ideas. This kind of discussions helps maintain the motivation needed to implement difficult research ideas.

Other contributions that cannot be skipped are the various comments by anonymous paper reviewers. They have tremendously helped me to revise my premature research ideas and learn to make the best out of sometimes painful criticism.

Although transparent but enormously helpful, I would like to thank the Nanyang Technological University and the Singapore Government for sponsoring the scholarship that covers all my expenses and tuition through the 4 years.

Last but not least, I would like to thank my beloved parents, who have always been supportive in any of my academic endeavors.

Contents

Acknowledgments	i
List of Figures	vii
List of Tables	ix
1 Introduction	1
1.1 Background	1
1.2 Periodic Behavior	3
1.3 Behavior Evolution	4
1.4 Contributions	5
1.5 Thesis Organization	7
1.6 List of Publication	8
2 Literature Survey	10
2.1 Introduction	10
2.2 Malware Periodic Behavior	12
2.2.1 Internal Host Aggregation	12
2.2.2 External Host Aggregation	13
2.2.3 Security Alert Verification	14
2.3 Malware Behavior Evolution	15
2.3.1 Batch Malware Detection	15
2.3.2 Online Malware Detection	16
2.3.3 Concept Drift Adaptation	18
2.3.4 Online Machine Learning	20
2.4 Summary	21

3	Are Malware Predictably Periodic?	23
3.1	Introduction	23
3.2	Problem Setting	25
3.3	Experiment	27
3.3.1	Malware Families	27
3.3.2	Methodology	31
3.4	Discussion	35
3.4.1	Malware Behavior	35
3.4.2	Behavior Analysis	37
3.5	Analyzing Traffic using Wavelet Transform	39
3.5.1	Frequency Analysis	40
3.5.2	Experiments with Wavelet Transform	44
3.5.3	Discussions	49
3.6	On Periodicity-Based Intrusion Detection	50
3.6.1	Opportunities	50
3.6.2	Challenges	53
3.7	Summary	54
4	Detecting Periodically-behaved Malware	56
4.1	Introduction	56
4.2	Problem Setting	58
4.2.1	Periodicity-based Intrusion Detection	59
4.2.2	Detection Result Verification	60
4.3	System Architecture	61
4.4	Periodicity	62
4.4.1	Frequency Analysis	62
4.4.2	Periodicity Measure	64
4.4.3	Characterization of Periodic Signals	65
4.5	Visualization	65
4.5.1	The Left Panel - Alert View (Panel A)	66

4.5.2	The Center Panel - Time Series View (Panel B)	68
4.5.3	The Right Panel - Detail on Demand (Panel C)	70
4.6	Evaluation	72
4.6.1	Simulated Scenario	72
4.6.2	Real Scenario	74
4.7	Discussions	75
4.8	Summary	77
5	Predicting the Risk Level of Executables	79
5.1	Introduction	80
5.2	Problem Setting	82
5.3	Methodology	82
5.3.1	Batch Learning – LR	83
5.3.2	Online Learning – FTRL-DP	84
5.4	Data Collection	87
5.4.1	Malware Collection	87
5.4.2	Malware Execution	88
5.4.3	Feature Extraction	88
5.5	Evaluation	91
5.5.1	Experiment with LR and SVM	91
5.5.2	Experiment with FTRL algorithms	92
5.6	Discussion	93
5.6.1	Prediction Accuracy	93
5.6.2	Running Time	95
5.7	Summary	96
6	Learning under Concept Drift with FTRL	97
6.1	Introduction	98
6.2	Problem Setting	99
6.3	Methodology	101

6.3.1	Linearization of Loss Function	101
6.3.2	Decaying Proximal Regularization	103
6.3.3	Adaptive Decaying Rate	105
6.3.4	Space and Time Complexity of FTRL-ADP	107
6.4	Experiment	107
6.4.1	Synthetic Experiments	108
6.4.2	Real-world Experiments	114
6.5	Summary	118
7	Conclusion and Future Work	120
7.1	Conclusion	120
7.2	Future Work	121
	References	124
A	Appendix	139
A.1	Proof of Theorem 1	139
A.2	Proof of Theorem 2	140

List of Figures

1.1	Network Intrusion Detection System (NIDS).	4
1.2	Periodic traffic generated by Darkcomet malware.	4
2.1	Concept drift adaptation.	18
3.1	Network traffic generated by Darkcomet, Andromeda and Zeus.	25
3.2	Desktop control panel of RAT Darkcomet.	29
3.3	Web control panel of Trojan Neutrino.	30
3.4	Experiment setup.	32
3.5	A demonstration of Fourier transform.	34
3.6	Periodicity measures of all malware families.	35
3.7	Network traffic of Babylon, Tinba and Rustock.	36
3.8	Mexican Hat Wavelets.	41
3.9	Wavelet Transform.	43
3.10	DarkComet Traffic, without Noise - Signal B. A) Time domain plot. B) DFT plot. C) CWT plot using Mexican Hat wavelet. D) CWT plot using Morlet wavelet.	46
3.11	DarkComet Traffic with Noise Injected - Signal C. A) Time domain plot. B) DFT plot. C) CWT plot using Mexican Hat wavelet. D) CWT plot using Morlet wavelet.	47
3.12	Browsing traffic generated by a typical user in 3 days.	51
4.1	System architecture.	59
4.2	A demonstration of Fourier transform	62

4.3	Visualization model.	64
4.4	Visualization dashboard.	67
4.5	Alert post-processing.	69
4.6	Histogram of netflows.	71
4.7	The content of 4 netflows generated by trojan Dyre.	73
4.8	Netflows by Windows Update Service (A) and RSS Feed Reader (B). . .	75
5.1	Distribution of executables through time.	87
5.2	Distribution of risk level.	89
5.3	Error rates of FTRL-DP, FTRL-Proximal and LR	94
6.1	Error rates of the algorithms on 8 synthetic datasets.	111
6.2	Evolution of decaying rates of FTRL-ADP on 8 synthetic datasets. . .	112
6.3	Error rates of the algorithms on 6 real-world datasets.	115
6.4	Evolution of decaying rates of FTRL-ADP on 6 real-world datasets. . .	117
6.5	Error rates of all algorithms on two SEA datasets.	118

List of Tables

3.1	Periodicity measures and periods of all experimented malware families.	38
3.2	Periods of some good applications.	53
4.1	Periodicity of malware samples.	76
5.1	Running time of FTRL-DP, FTRL-Proximal and LR.	95
6.1	14 Evaluated datasets.	108
6.2	Error rates of all aglorithms on 8 synthetic datasets.	114
6.3	Error Rates of all Algorithms on 6 Real-world Datasets.	117

Abstract

Traditional antivirus products are signature-based solutions, which rely on a static database to perform detection. The weakness of this design is that the signatures may become outdated, resulting in the failure to detect new samples. The other method is behavior-based detection, which aims to identify malware based on their dynamic behavior. Behavior-based detection comes in two approaches. The first approach leverages on common known behaviors of malware such as random domain name generation and periodicity. The second approach aims to directly learn the behavior of malware from data using tools such as graph analytics and machine learning.

Behavior-based detection is difficult because we have to deal with intelligent and highly motivated attackers, who can change their strategy to maximize the chance of getting access to computer networks. We narrow our research to the domain of Windows malware detection and we are particularly interested in two approaches of behavior-based detection: periodic behavior and behavior evolution. Periodic behavior refers to the regular activities programmed by attackers such as periodic polling for server connection or periodic update of the victim machine's status. Behavior evolution refers to the change in behavior of malware over time.

In the **first approach**, we aim to exploit the periodic behavior for malware detection. The main analysis tool in this direction is Fourier transform, which is used to convert time domain signals into frequency domain signals. This idea is motivated by the fact that it is often easier to analyze periodic signals in the frequency domain than in the original time domain. Using Fourier transform, we propose a novel frequency-based periodicity measure to evaluate the regularity of network traffic. Another challenge in this direction is that, other than malware, most automatic services

of operating systems also generate periodic signals. To address this challenge, we propose a new visual analytics solution for effective alert verification.

In the **second approach**, we aim to develop adaptive learning algorithms to capture malware samples, whose behavior changes over time. We capitalize on the well-known online machine learning framework of Follow the Regularized Leader (FTRL). Our main contribution in this direction is the usage of an adaptive decaying factor to allow FTRL algorithms to better perform in environments with concept drifts. The decaying factor helps to increasingly discount the contribution of the examples in the past, thereby alleviating the problem of concept drifts. We advance the state of the art in this direction by proposing a new adaptive online algorithm to handle the problem of concept drift in malware detection. Our improved algorithm has also been successfully applied to other non-security domains.

1

Introduction

1.1 Background

Nowadays, more and more of our valuables such as money and personal information are digitalized. This is a treasure trove for hackers who look for financial gains from the Internet. Cyber-attacks although seem to be invisible hiding in the wires but have made huge impacts in the world. They target big companies and individuals alike. From the report by Hiscox Insurance¹, cyber-attacks have cost companies worldwide over \$450 billion in 2016. The rise of ransomwares which target individual is also alarming. Recently, the WannaCry² ransomware has infected more than 230k computers worldwide within a day – the largest cyber attack ever. In addition, the numbers from the Bureau of Labor Statistics (USA) show that there are around 209k cybersecurity positions to be filled in 2015 in the USA alone. Given the bigger scale of cyber-attack and the rising shortage of cyber analysis positions, the development of assisting tools or automated solutions are urgently needed.

Signature-based approach towards malware detection have traditionally been the first choice in cyber defense [1, 2, 3]. In this approach, a new signature is developed to detect each new sample captured in the wild. For example, it looks for certain strings

¹<http://www.cnbc.com/2017/02/07/cybercrime-costs-the-global-economy-450-billion-ceo.html>.

²https://en.wikipedia.org/wiki/WannaCry_ransomware_attack

in the binary representation of the known malware samples (Snort [4]). This method is reliable in detecting known samples but can be evaded easily. An easy method to circumvent this approach is to use public tools to craft new samples based on existing ones [5]. For example, the XOR Obfuscation tool³ can be used to fabricate any number of new samples using just one template sample. The new samples and the template sample behave identically once installed but their binary representations are totally different. In another direction, behavior-based approach tries to detect malware based on how they behave. It is challenging because hackers are intelligent agents and can change their strategy to evade detection efforts.

Periodic Behavior. Malware samples have been known to exhibit periodic behavior [6]. This behavior was exploited for detection in various previous works [7], [8], which assume that any periodic signals captured at the network gateway are generated by malwares. This is a rather flawed assumption since certain OS services [9] are also known to generate periodic signals. Therefore, our first challenge is two-fold: improving periodicity-based detection techniques and effectively helping cyber analysts to verify the detected signals. Specifically, we narrow down our research to the problem of detecting periodic signals in network traffic. This is based on the reasonable assumption that malware samples must generate communication traffic with the attackers. The concrete problem definition and the approach are detailed in Section 1.2.

Behavior Evolution. Another challenge of behavior-based detection approach is the change in behavior of malware over time, which has been demonstrated in [10]. There are many works in the literature which use batch learning [10, 11, 12] for malware detection. The limitation of these works is that they do not consider the evolution of malware over time. This limitation has been pointed out in the field of Android malware detection [13]. Proper experiments, which take the time associated with each sample into consideration, shows that online learning method achieves

³https://en.wikipedia.org/w/index.php?title=XOR_cipher

superior performance compared to batch learning method. Online learning is advantageous in this case of malware detection due to their highly adaptive characteristic and the adversary nature of cyber attacks. Therefore, our second challenge is to develop adaptive learning method that can cope with evolving Windows malware behavior. The concrete problem and the approach are elaborated in Section 1.3.

1.2 Periodic Behavior

In the first problem, we aim to design and develop a Network Intrusion Detection System that can detect periodic signals left in network traffic by malware.

Network Intrusion Detection System (NIDS) is the device, which stands between a company network and the Internet (Figure 1.1). The company network is referred to as the internal network and the Internet the external network. The internal network and external network exchange data in the form of network packets, which are collectively referred to as network traffic. The job of NIDS is to inspect the network traffic to discover malicious intents contained in the network packets.

A network connection is defined as a 4-tuple: Source IP, Source Port, Destination IP and Destination Port. The plot in Figure 1.2 shows the network connections generated by the DarkComet malware⁴; the horizontal line of this plot represents time and the vertical line the number of TCP SYN network packets exchanged every half a second. From this plot, we can see that DarkComet generates TCP SYN packets in such a way that the intervals between consecutive packets follow the pattern: 3 second, 6 seconds, 12 seconds and 3 seconds again. This is an example of the “periodic patterns” in network traffic that we aim to detect.

This problem has two parts: the detection of periodic signals generated by malware and the verification of the detected signals. The challenge of the detection part is that we need to identify malicious periodic signals that are often buried in the highly noisy benign network traffic. To address this challenge, we propose a novel periodicity measure based on Fourier transform (Section 4.4).

⁴<https://threatpost.com/darkcomet-rat-used-new-attack-syrian-activists-081612/76919/>

Figure 1.1: Network Intrusion Detection System (NIDS).

Figure 1.2: Periodic traffic generated by Darkcomet malware.

The challenge of the verification part is the possible huge number of signals to be inspected by cyber analysts, who have limited time and concentration span. To address this challenge, we make the effective use of visualization (Section 4.5) to help cyber analyst quickly verify detection results by inspecting the deep packet content. These two parts work interactively in a unified visual analytics solution.

1.3 Behavior Evolution

In the second problem, we aim to develop novel adaptive online algorithms to detect malware, whose behavior may adversarially change over time.

We framed the problem of malware detection as the problem of regressing the risk level of an executable. We rely on the labels provided by 52 antivirus solutions on VirusTotal.com⁵. Formally, we aim to predict y , the percentage of the solutions that would flag a file as malicious, based on the input x_n which is the set of 482 hand-crafted features extracted from the reports provided by the Cuckoo sandbox [14]. The

⁵<https://virusshare.com>

semantic of the output can be thought of as the risk level of an executable.

The challenge of this problem is that the behavior of malware may change over time in an unpredictable way. Therefore, a model trained on past data may not be applicable to the present time. This problem is usually referred to as concept drift in the machine learning literature. To address this challenge, we propose a novel online algorithm based on the FTRL framework (Section 5.3.2.2). We then develop a fully adaptive version of this algorithm and apply the new version (Section 6.3) to various applications including non-security ones. The results show that our proposed algorithm achieves state-of-the-art performance in noisy settings with real drifts.

1.4 Contributions

To summarize, we make 3 key contributions in this thesis:

Contribution 1 We carry out a series of experiments with 31 malware toolkits commonly found in the wild and we find that 30/31 of them exhibit highly periodic behavior. This observation motivates our development of a measure to gauge the periodicity of a network traffic capture. However, the problem is that this measure alone would result in a high false positive rate since there are many other benign applications also exhibiting periodic networking behavior. This motivates us to develop a visual analytics solution to help cyber analyst quickly identify the true alerts. We evaluate the system on a public dataset of botnet and 7/16 samples were detected by the system. A close inspection of the failed cases shows that the presence of relevant malware samples in the dataset are too short for the periodic patterns to come up. This is also the main weakness of the detection system. There are two publications associated with this contribution [6, 15].

Contribution 2 We carry out an extensive experiment to collect the data generated by 100K Windows malware samples and propose an algorithm to help detecting evolving malwares. This is motivated by the fact that there lacks a big dataset on the behavior of Windows malware. Behavior-based detection is becoming more

and more important due to the sheer number of malware samples appearing everyday and the easiness of making new samples using off-the-shell toolkits. We use the opensource Cuckoo sandbox to execute the malware samples to obtain the reports. We then extract 482 features returned by the Cuckoo sandbox to make the dataset. In addition, we also propose a new online learning algorithm designed to detect the evolving malware samples. The proposed algorithm outperforms logistic regression, support vector machine, and other online learning algorithms on the same task. The associated publication is [16].

Contribution 3 We propose a novel adaptive online learning algorithm to address the problem of concept drift in many different domains, including non-security domains. We found out that concept drift is a common problem not just in security domains but also in any domain with time dimension such as weather prediction and electricity shortage detection. We evaluate the proposed algorithm on 8 synthetic datasets and 6 real datasets and find that the proposed algorithm is most advantageous in noisy environments with real concept drifts. We also provide a strong theoretical backing for the developed algorithm. Specifically, we prove that the algorithm ensures a sublinear bound on the prediction errors. The associated publication is [17].

The following tree puts our work in perspective in relations to other related works. The main theoretical tools that we used are highlighted in red.

In the first contribution, we study the periodic behavior of botnet and RAT toolkits widely found in the wild. We rely on frequency analysis to analyze the periodic patterns in the network traffic. Although effective in the cases when malware is present in the network long enough for the periodic patterns to show up, it can be bypassed if the attacker randomizes the connection period. In the second contribution, we study the behavior evolution of malware over time. We rely on Cuckoo sandbox to gather the features required for the dataset. In the third contribution, we extend the algorithm proposed in the second contribution and apply it to other domains.

1.5 Thesis Organization

The thesis is divided into 7 chapters. In the next chapter (Chapter 2), we review the important related work associated with our research. The rest of the thesis can be neatly divided into two parts corresponding to our two research directions.

- Periodic Behavior
 - In Chapter 3, we present an extensive experiment to study the prevalence of the periodic behavior of malware families captured in the wild. Specifically, we explore hacker forums and blogs to gather a set of 31 malware toolkits. These toolkits are often used by script kiddies and experienced hackers alike to quickly fabricate custom malware samples employed in cyber-attacks.
 - The results from Chapter 3 show that most of the malware families in the wild exhibit periodic behavior, which motivates the development in Chapter 4. Specifically, in Chapter 4, we present our visual analytics solution to help detect malware based on their periodic behavior. The solution also proposes a visualization that helps cyber analysis to effectively carry out the verification of detected signals.

- Behavior Evolution
 - In Chapter 5, we compare the performance of batch learning and online learning in the problem of malware detection. We first describe the experiment that we conduct to collect the dynamic behavioral data of 100k malware samples. We then propose a simple modification to an existing online algorithm and compare its performance to logistic regression. The choices of the algorithms are based on the similarity in their formulations for a fair comparison.
 - In Chapter 6, we extend the algorithm proposed in Chapter 5 and apply the improved algorithm to various applications including non-security ones such as weather prediction and flight delay prediction. This development was motivated by the observation that the problem of concept drift is ubiquitous and appears in many other application domains.

We conclude the thesis and present the future works in Chapter 7.

This thesis interpolates the materials from our 4 publications [6, 15, 16, 17]. To be specific, Chapter 3 uses the materials from [6], co-authored with Dr. Ng Wee Keong and Dr. Hoang Giang Do. Chapter 4 is adapted from the materials of [15], co-authored with Dr. Ng Wee Keong, Dr. John Kohlhammer, and Mr. Alex Ulmer. Chapter 5 and Chapter 6 are respectively adapted from [16] and [17], which are co-authored with Dr. Ng Wee Keong and Dr. Kanishka Ariyapala.

1.6 List of Publication

- **N. Anh Huynh**, W. Keong Ng, and K. Kanishka. Learning under concept drift with follow the regularized leader and adaptive decaying proximal. *Journal of Expert Systems with Applications*, volume 96, pages 49-63 (2018).
- **N. Anh Huynh**, W. Keong Ng, and K. Kanishka. A new adaptive learning algorithm and its application to online malware detection. In *20th International Conference on Discovery Science, DS 2017*, pages 18-32 (2017).

- **N. Anh Huynh**, W. Keong Ng, A. Ulmer, and J. Kohlhammer. Uncovering periodic network signals of cyber attacks. In 2016 IEEE Symposium on Visualization for Cyber Security (VizSec), pages 1-8 (2016).
- **N. A. Huynh**, W. K. Ng, and H. G. Do. On periodic behavior of malware: experiments, opportunities, and challenges. In 2016 11th International Conference on Malicious and Unwanted Software (MALWARE), pages 1-8 (2016).
- K. Ariyapala, H. G. Do, **H. N. Anh**, W. K. Ng, and M. Conti. A host and network based intrusion detection for Android smartphones. In 2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA), pages 849-854 (2016).

2

Literature Survey

2.1 Introduction

Since it was first introduced in the 1980s, intrusion detection systems (IDS) have become the most popular cyber defense technique. The research in this field can be divided into two approaches: signature-based and behavior-based. The signature-based approach focuses on what the binary representation of a malware sample looks like to carry out detection. It relies on certain static features of the suspicious file – for example, certain strings of characters or the hash value of the file [1, 2, 3]. All the signatures are stored in a database for reference. When a new suspicious sample emerges, the detection engine searches the database for a match. If a match is found the sample is declared malicious, and benign otherwise.

The signatures are created by human security analysts. This manual process is extensive in terms of human labor and time. In the old days [4], when the number of malware samples appearing every day is only handful, this method could keep up the speed of malware releases. However, nowadays, the number of new malicious files appearing every day is in millions. For example, the statistics by VirusTotal¹ reports around 1.7M new suspicious files with unique hash numbers daily. The human exten-

¹<https://www.virustotal.com>

sive process of developing new signatures clearly cannot keep up with this recent speed of new malware releases. Therefore, behavior-based detection approach to reduce the human effort in creating the signature is urgently needed.

Unlike signature-based approach, the behavior-based approach looks at what a sample does to make the decision. The behavior may be local on the host machine [26] or occurs at the network gateway [27]. [28] and [29] noted that botnets often generate strangely looking domain names which no human would use (such as jgelvgde.net [29]) to bypass firewalls. Yadav et al. [28] proposed to analyze statistical properties of URLs to identify randomly generated domain names. In another direction, [7] and [8] noted that malware often generates periodic signals in network traffic. Hubballi et al. [7] proposed to use the standard deviation of the temporal distance between consecutive TCP SYN packets as a proxy measure for periodicity. These two known behaviors of malware can also be considered as relaxed signatures, which are applicable to a wide range of malware samples and not too restricted like the previous approach. However, they are still a knowledge-based approach.

On the other hands, the other approach in behavior-based detection aims to learn the behavior of malware from the data. Researchers have been experimenting with techniques borrowed from graph analytics [30], Bayesian network [31] and machine learning [10]. Anderson et al. used Markov chain to model the instruction traces created by programs [32]. Various graph kernels are used to evaluate the similarity between different instruction traces. Zhang et al. used a graphical probabilistic model to reason about security events [33]. The proposed model incorporates IDS alerts into the attack graph thereby providing a unifying framework to reason about the security of a network. Saxe et al. [10] derived a set of behavioral features such as the number of packets generated and the size of the file. After that, a neural network with 2 hidden layers was used to classify malicious and benign samples.

Our scope is to advance the state-of-the-art in behavior-based malware detection. Specifically, we narrow down our research to two notable behaviors of malware: periodic behavior and behavior evolution. In Section 2.2, we review the literature on

periodic behavior of malware by introducing two ways to formulate the problem of detecting periodic signals left by malware in network traffic. We also touch on the issue of alert verification, which is crucial in computer security due to the semantic gap in operational cyber defense [34]. In Section 2.3, we review previous works on the application of machine learning to learn the behavior of malware from data. We also review relevant online learning methods which we use to address the evolution in behavior of malware over time.

2.2 Malware Periodic Behavior

Depending on the ways that the time series are constructed, the problem of periodicity-based intrusion detection can be formulated as a signal identification problem or a signal detection problem.

2.2.1 Internal Host Aggregation

In this approach, the intrusion detection problem is formulated as a signal detection problem. The task is to identify whether a signal is highly periodic using some sort of periodicity measure.

AsSadhan et al. [19] studied the effectiveness of Fourier transform when it is applied to the problem of detecting periodic patterns hidden in network traffic. Various levels of noise were incrementally injected to a purely periodic signal. The finding, as expected, was that the significance of the highest peak in the frequency spectrum and the signal to noise ratio form a reciprocal relationship. However, in this work, both local IP addresses (representing internal hosts) and remote IP addresses (external hosts) were discarded, and as a result, this method could only confirm the existence of periodic signal in network traffic. There was no direct action to circumvent the infection implied by detection result.

In a similar manner but motivated by the fact that some malwares are designed to beacon back very infrequently in order to evade detection, Chen et al. [8] aimed to detect periodic series of events which occur at very low rates. The author proposed an

algorithm to incrementally increase the duration of the time series in order to increase the chance of detecting hidden periodic signals. However, this work and the work of AsSadhan et al. had the same drawback that detection result doesn't suggest any countermeasure action.

The previous works were based on Fourier transform and suffered the same problem of time localization: it is not possible to infer the temporal occurrences of detected periodic patterns with the sole help of frequency spectrums. Motivated by this drawback, Bartlett et al. [9] chose another frequency transform, Discrete Wavelet transform. Unlike Fourier transform, the result of Discrete Wavelet transform was an energy map which contains information on both frequencies and temporal occurrences of hidden periodic signals. However, this work also had the same drawback as in the works of AsSadhan et al. [19] and Chen et al. [8].

The current literature on this approach leaves open the issue of detection result verification, which is highly important in operational settings due to the semantic gap in intrusion detection [34]. We are going to address this issue in Section 4.5 with the use of interactive visualization.

2.2.2 External Host Aggregation

In this approach, the intrusion detection problem is formulated as a signal identification problem. The task is to extract periodic patterns buried in the traffic noise generated by ordinary users.

Hubballi et al. [7] defined the variance of the inter-arrival times of TCP SYN ACK packets as the measure to evaluate the periodicity of each time series, constructed from a network traffic. This method was a simplistic one and might fail if malware generates complicated periodic patterns, as observed in the experiments, which could not be identified by the simple variance of inter-arrival times.

Qiao et al. [35] used pattern mining to look for hidden regional periodic patterns in network traffic. There were key steps in the proposed algorithm: first, candidate periods are generated by using circular autocorrelation; second, the mined regional

patterns are then assessed for periodicity by using an intensity function of periodicity. The detection was carried out by comparing the obtained scores.

In a different direction, Garcia et al. [18] aimed to identify malicious netflows in network traffic by modeling behavior of malwares using Markov chain. Among the three features that the author used to characterize a chain of netflows, periodicity was a key feature. To measure periodicity, the author used a simple difference in the inter-arrival time of consecutive netflows. The drawback of this technique is that it cannot measure the periodicity of a complicated periodic traffic, as evidenced in the experiments (Fig. 3.7).

In summary, a few simple methods were proposed in the literature to measure the periodicity of network traffic. The drawback of these methods is that they would fail to detect complicated periodic patterns, generated by some malware families (such as Rustock). Therefore, in our future works, we are going to adapt our periodicity measure proposed in Section 4.4 to address this issue.

2.2.3 Security Alert Verification

The main problem which a security analyst usually encounters in an operational setting is the overwhelming amount of log data. Investigating every log entry is infeasible. Therefore, automatic detection methods are necessary. However, due to the semantic gap in computer security [34], suspicious activities may not always be malicious and a human analyst is still needed to do the final check on every alert. This is usually done through visualization, which is the highest bandwidth pathway to the human brain [36]. Visualization is used to communicate the context of each alert to the analyst for quick decision making.

To address the problem of insider threat detection, Legg et al. [37] made use of a PCA-based anomaly detector to identify anomalous activities. The visualization component features a visual analytics dashboard to support the verification of generated alerts as well as to explore the original records. The visualization enables interpreting the result of the anomaly detector by linking detection result and user profile.

For the problem of network traffic anomaly detection, Best et al. [38] employed a simple statistical model to detect anomalous behavior. The model checks every current data point with the minute-of-the-week stable historical record for anomalies. The author then used a visual analytics environment to help explore the generated alerts as well as the original data with the use of a flexible hierarchical grouping technique.

In rating fraud detection, Webga et al. [39] proposed to assess the suspicious level of each user based on the average rating and the number of ratings. If the suspicious level of some user is measured to be more than a certain threshold, an alert will be sent to the visualization requesting further investigation by the analyst. During the analysis, the analyst can interactively perform singular value decomposition to refine the result.

2.3 Malware Behavior Evolution

The idea of a completely automated detection solution that can detect novel samples by modeling the behavior of previous malware samples has been intensively researched. The modeling techniques previously used vary from graph analytics [32, 40], Bayesian network [41, 42] to machine learning methods [10, 12]. Specifically, in the machine learning approach, previous works in the literature are most notably different in the set of features used to perform classification.

2.3.1 Batch Malware Detection

The analysis of suspicious executables for extracting the features used for automated classification can be broadly divided into two types: static analysis and dynamic analysis. In static analysis, the executables are not required to be executed and only features extracted directly from the executables are used for classification such as file size, readable strings, binary images or n-gram words, etc [43, 44, 45]. On the other hand, dynamic analysis usually requires the execution of the executables to collect generated artifacts for feature extraction. Dynamic features can be extracted from

host-based artifacts such as system call traces, dropped files or modified registries [46, 47, 48, 49, 50]. Dynamic features can also be extracted from network traffic such as the frequency of TCP packets or UDP packets [51, 52]. While static analysis is highly vulnerable to obfuscation attacks, dynamic analysis is more robust to obfuscation. The other advantage of dynamic analysis is that it would impose more cost on malware developers since the creation of a new malware with completely different dynamic behavior is time-consuming while it is fairly easy to just obfuscate existing samples by using widely available automated tools such as crypters or packers [5].

To improve detection accuracy, the consideration of a variety of dynamic features of different types is recently gaining attention due to the appearance of highly effective automated malware analysis sandboxes. In their work, Mohaisen et al. [53] studied the classification of malware samples into malware families by leveraging on a wide set of features (network, file system, registry, etc) provided by the proprietary AutoMal Sandbox. In a similar spirit, Korkmaz et al. [12] used the open-source Cuckoo sandbox to extract a bigger set of features aiming to classify between traditional malware and non-traditional malware, which are the samples involved in past Advanced Persistent Threat campaigns. The general conclusion in these papers is that combining dynamic features of different types tends to improve the detection accuracy.

In light of these considerations, we decided to use the Cuckoo sandbox to execute and extract the behavioral data generated by a collection of more than 100k suspicious executables. Our feature set is similar to that of Korkmaz et al. although we address a different problem: classifying between malicious and benign executables. Our work also differs from previous work as we additionally formulate this problem as an online learning problem rather than just a batch learning one (Section 5.2).

2.3.2 Online Malware Detection

Although previous studies [10, 52, 53] on the application of machine learning to the problem of malware detection have been reporting promising high accuracy, these results are not directly relevant to the real practice of malware detection. The reason

for this claim is that temporal information of the executables is often ignored in most of these works, resulting in the fact that the algorithms are only able to recognize unknown executables, but not new ones [10]. However, in practice, we are more interested in detecting new malware samples, not just unknown ones. We argue that the right way to evaluate a machine learning algorithm in this practice of malware detection is to construct the train set and the test set by splitting the dataset using some dividing time stamp. The reason for this approach is the evolutionary behavior of malware samples, which is rather different from other disciplines such as object detection or sentiment analysis, in which the objects to be detected are effectively fixed.

The importance of the time dimension has been extensively studied in the domain of Android malware detection. In their paper [54], Allix et al. studied the effect of history on the biased results of existing works on the application of machine learning to the problem of malware detection. The author notes that most existing works evaluate their methodology by randomly picking the samples for the train set and the test set. The conclusion is that this procedure usually leads to much higher accuracy than the cases when the train set and the test set are historically coherent. The author argues that this result is misleading as it is not useful for a detection approach to be able to identify randomly picked samples but fail to identify new ones.

To address this problem of history relevance, Narayanan et al. [13] have developed an online detection system, called DroidOL, which is based on the online Passive Aggressive (PA) algorithm. The novelty of this work is the use of an online algorithm that may improve the performance compared to batch learning algorithms due to the ability of online algorithms to adapt to the change in behavior of malware. The result turns out to be the case as the online PA algorithm results in a much higher accuracy (20%) compared to the typical setting of batch learning and 3% improvement in the settings when the batch model is frequently retrained.

Figure 2.1: Concept drift adaptation.

2.3.3 Concept Drift Adaptation

In the presence of concept drift, the predictive model is necessary to be updated from time to time. There are two different approaches towards model update: active and passive. These two approaches differ in how often the model gets updated. In the active approach [55], the update only takes place on the trigger by a drift detector, which is usually good at detecting substantial changes (abrupt drifts). However, drift detectors are often insensitive to small but consistent changes caused by gradual drifts. On the other hand, the model is updated much more often in the passive approach [56, 57] since an update is triggered on every new example. As a result, passive approach is often more sensitive to gradual drifts than active approach.

The general framework of active approach is shown in Figure 2.1. There are two key components to handle drifts in this framework: the component to detect concept drift (Change Detector) and the component to cope with detected drifts (Adaptation). The former component detects drifting concepts by monitoring the prediction error and/or the data generating process [55]. The second component then defines different ways to cope with drifts such as windowing, weighing or random sampling [58].

There are two different approaches towards concept drift detection: supervised and unsupervised. In the supervised approach, the changes in the prediction error are monitored for concept drifts. Drift Detection Method (DDM [55]) keeps track of the minimum false positive rate and its standard deviation. A drift is detected

if the current rate significantly deviates from the minimum rate. To address the insensitiveness of DDM to gradual drifts, Early Drift Detection Method [59] proposes to instead monitor the maximum distance between prediction errors and its standard deviation. A drift is signaled if the current distance significantly deviates from the maximum distance. In another direction, the Page-Hinkley test [60] is proposed to detect any significant discrepancy between the cumulative accuracy difference and the minimum difference.

In unsupervised drift detection, the underlying distribution of the data generating process (or the input) is monitored for concept drift. Masud et al. [61] aim to watch out for sizable regions of an unknown class using outlier detection techniques. Similarly, Sethi et al. [62] leverage on clustering techniques to look for unknown regions. In another direction, independent features are monitored in two different chunks for change of concept: the current chunk and the reference chunk. Lee et al. [63] use Pearson correlation to evaluate the correlation between the current chunk and the reference chunk. On the other hand, Ditzler et al. [64] use Hellinger distance to achieve the same purpose. In general, supervised drift detection methods are more accurate than unsupervised methods at the cost of labeled data.

In passive drift adaptation, there are two main categories of passive learners: ensemble learner and single learner. The SEA ensemble classifier [65] is among the first proposals for ensemble learners. The ensemble creates a new learner for every new batch of data. To make room for new learners, existing learners are removed based on age or prediction accuracy. In another direction, Pocock et al. [66] propose the Online Nonstationary Boosting Algorithm (ONSBoost), which is adapted from the Online Boosting Algorithm [67] to address the challenges in non-stationary settings. Similarly, Kolter et al. [68] proposes the Dynamic Weighted Majority (DWM) algorithm. Unlike ONSBoost, DWM has a variable number of learners.

Compared to the ensemble approach, the single learner approach is often less computationally expensive. Domingos et al. [69] proposes Concept Drift Very Fast Decision Tree (CDVFDT) to mine nonstationary streaming data. This algorithm is

adapted from the Very Fast Decision Tree, which is a popular tree-based solution for stream data mining. CDVFDT uses an adaptive window size to cut off least recent learning examples [70]. In another direction, Lim et al. [71] propose a modification to the originally stationary Extreme Machine Learning. The basic idea of this work and similar works [72, 73] in this approach is to use a fixed or variable forgetting factor to discount the contribution of previous learning examples.

2.3.4 Online Machine Learning

Online algorithms such as PA [21] and CWL [22] also belong to the single learner approach. The basic idea in these algorithms is the objective to find the weight that minimizes the loss associated with the most recent example. Certain constraints are enforced on the weight to stabilize the solution. Passive Aggressive (PA) algorithm [21] ensures that the most current example is correctly classified. The weight, which least deviates from the most recent weight, is the solution. PA-I and PA-II improve on PA by introducing the loss associated with the misclassified example into the objective function. Confidence Weighted Learning (CWL) algorithm [22] relaxes the requirement of PA, only ensuring that the most current example is correctly classified within a certain probability. The objective is to minimize the Kullback-Leibler divergence between the new weight distribution and the most recent one. Like PA I and PA II, the Soft Confidence Weighted Learning algorithm [74] proposes to modify the objective of CWL by introducing the loss associated with the misclassified example into the objective function.

These algorithms follow the Mirror Gradient Descent style [75], which only considers the loss associated with the most recent example. As a result, they are robust under concept drift but prone to prediction errors caused by noise.

On the other hand, the FTRL framework considers all previous learning examples in the objective function [75], aiming to derive the best model in hindsight. Regularized Dual Averaging [25] tries to minimize the linearized losses associated with all previous learning examples. It regularizes the solution using L_1 -norm and an adap-

tive L_2 -norm. Similarly, FTRL-Proximal [76] uses a per-coordinate learning schedule to regularize the proximal terms. As FTRL-derived algorithms factor in all previous losses, they are less responsive to prediction errors than Mirror Gradient Descent style algorithms.

Mirror Descent algorithms and FTRL algorithms are proved to be equivalent and only different in the regularization scheme used [23]. Mirror Descent algorithms are responsive to changes, hence not robust to noise because of their complete forgetting mechanism. On the other hand, the latter approach of FTRL is more conservative and slow in adaptation to concept drifts due to the contribution of all examples. In this aspect, TDAP [24] strikes the balance between these two approaches via the usage of a decaying factor.

Our approach further develops the idea in TDAP. Specifically, we propose a mechanism to adaptively control the decaying rate. This adaptive mechanism eliminates the need to manually tune the decaying rate (Section 6.3.3).

2.4 Summary

We identify 3 research gaps in the literature which precisely correspond to our 3 key contributions in this thesis:

- The issue of alert verification in periodicity-based intrusion detection. This issue is brought about by the semantic gap in operational cyber security [34]. To the best of our knowledge, it has not been addressed by any researchers. We address this issue by proposing a visual analytic solution that makes the effective use of visualization for quick verification of raised alerts.
- The lack of a dataset on the dynamic behavior of malware. Although there are a few repositories that share raw malware samples²³, they are only static files and don't contain the dynamic behavior of the binaries. We carry out an

²<https://virusshare.com>

³<https://zeltser.com/malware-sample-sources/>

extensive experiment to collect the dynamic behavior of 100k malware samples and donate it to the UCI portal for machine learning datasets.

- The evolution of the behavior of malware over time. This characteristic behavior of malware has recently been raised by a few researchers [10, 77]. It has been addressed by Narayanan et al. [13] in the field of Android malware detection. However, we are the first to address this issue in the field of Windows malware detection by using a novel online learning algorithm.

3

Are Malware Predictably Periodic?

We notice, in our experiments with three malware samples of Darkcomet, Andromeda, and Zeus, that all three of them generate highly periodic network traffic. While there are a few works on intrusion detection, which are based on this periodic behavior of malware, the prevalence of this behavior among the malware families in the wild is unknown. In this chapter, we conduct an extensive experimental study to investigate the networking behavior of 31 different malware families, which belong to two large classes of malware families commonly found in the wild: Remote Administration Tools and Botnet Trojans. We would like to address the question that how likely a malware captured in the wild would generate periodic traffic. This study is the first part of the **Contribution 1** of the thesis. It finds out that 30 out of 31 (96.77%) experimented malware families are found to leave periodic traces in network traffic when they communicate with command and control servers.

3.1 Introduction

Figure 3.1 shows the histograms of network traffic generated by Darkcomet¹, Andromeda [78] and Zeus [79]. The traffic dumps are captured in the duration of 10

¹<https://en.wikipedia.org/wiki/DarkComet>

minutes when the malware samples have already established connections to remote servers. Upon inspection of packet payloads, we are able to find out that the machine infected by Darkcomet keeps sending keep-alive signals to attacker’s server following the pattern: 3s, 6s, 12s, and then 3s again. This keep-alive activity is clearly reflected on the periodic histogram of network traffic generated by Darkcomet. Similarly, the pattern generated by Andromeda is found to be 3s, 6s and 60s. The pattern generated by Zeus is the most complicated and found to be 3s, 6s, 12s, 3s, 6s, and 17s; this pattern is repeated 10 times then paused for 1 minute before starting all over again.

In practice, the simplest way to implement a keep-alive mechanism between a client and a server is that of periodic polling. In this technique, the client periodically sends keep-alive signals to the server at fixed interval to update the server on the current status of the infected machine or to download new version of the malware. As a result, these periodic activities leave highly periodic traces in network traffic. In our opinion, malware author chooses this method to control their malware because of its ease of programming and maintenance.

There are a few works in the literature on intrusion detection with detection techniques based on this periodic networking behavior of malware [7, 9, 19, 80]. In all these works, the authors make the same assumption on the periodic networking behavior of malware and the problem of intrusion detection is reduced to the problem of detecting periodic signals hidden in network traffic. This approach has two known weaknesses. The first weakness is that malware authors may change the mechanism that malware communicates to attacker’s server by randomizing the polling interval. The second weakness is that there are many other good applications that also leave periodic patterns in network traffic (Table 3.2).

This chapter aims to address the first weakness by studying the network traffic generated by 31 different malware families. Our principle research question is “How likely does a malware captured in the wild generate periodic patterns in network traffic?” Positive answer to this question will not only confirm a naive communication mechanism commonly used by attackers but also enable the development of

Figure 3.1: Network traffic generated by Darkcomet, Andromeda and Zeus.

corresponding detection techniques. The direct consequence is that malware authors would be forced to devise more complicated mechanisms for infected machines to communicate with attacker’s servers.

The contribution of this chapter includes:

- An empirical study on the networking behavior of malware in the wild, specifically their periodic networking behavior. The experiments are elaborated in Section 3.3 and experimental results are reported in Section 3.4.
- An analysis of the opportunities as well as the challenges on the development of detection algorithms based on this periodic behavior of malware (Section 3.6). This is motivated by experimental results, which suggest highly periodic behavior among malware families in the wild.

Before the details of the experiments are presented, in Section 3.2, we state the research questions being addressed and briefly analyze the requirements for the experiments.

3.2 Problem Setting

In this chapter, we set out to address two questions:

- (i) How likely does a malware captured in the wild generate periodic patterns in network traffic?
- (ii) How to detect these periodic patterns if they exist?

To address the first question, the experiments with malware samples must satisfy the following requirements:

- R-1: The study must experiment with a large coverage of malware families in the wild for the result to be applicable to many malware families.
- R-2: The method to carry out the experiments must ensure that the malware samples would behave in the way that they are supposed to behave in the wild.
- R-3: There must be a sound technique to provide an objective measurement for the periodicity of network traffic generated by malware samples.

For requirement R-1, we conduct experiments on two large classes of malware families: Remote Administration Tools and Botnet Trojans. These two malware family classes make up the most part of malware frequently used in the past²³ as well as recent cyber attacks [81, 82]. For requirement R-2, we use physical machines instead of virtual machines and separate the environment to execute the malware from the environment to capture generated network traffic. This requirement is necessary because some malware samples [83] are found to generate random traffic to confuse analyst if they detect that they are running in a virtual machine. For requirement R-3, we develop an algorithm, based on Fourier transform, to assess the periodicity of captured network traffic. This periodicity measure serves as an objective measure for periodic traffic, supporting the subjective identification performed by experimenters.

²<http://hackforums.net/showthread.php?tid=4935588>

³<http://www.kernelmode.info/forum/viewtopic.php?f=16&t=2680>

3.3 Experiment

3.3.1 Malware Families

We experiment with two large classes of malware families commonly found in the wild: Remote Administration Tools (RAT) and Botnet Trojan (shortly referred to as Trojan). We obtain samples of these malware families from two types of sources in the wild: hacker forums such as **hackerforums.net** or **leakforums.net** and online malware analyzers such as **malwr.com** or **virusshare.com**. From our understanding, RATs and Trojans are the two most discussed classes of malware families in hacker forums. There is another class of malware families, which includes malware samples used in past Advanced Persistent Threat campaigns such as Stuxnet or Duqu [84, 85]. These malware families are not addressed in this study due to their less popularity.

RATs are originally designed with the good purpose to remotely administer a machine but later used for criminal purpose. Most known RATs use raw TCP and UDP as the communication protocol. These protocols allow real-time communication between infected machine and the remote server controlled by attackers. Examples of RATs include Darkcomet and jRat⁴. Trojans, on the other hand, use HTTP as the protocol to communicate. The communication between infected machine and remote server is limited to GET or POST requests initiated by infected computer. This is a disadvantage of HTTP protocol as the remote server cannot directly send files or commands to the infected machine without the infected machine having initiated the communication beforehand. However, the advantage of HTTP protocol is that of stealthiness. Zeus and Conficker [86] are examples of Trojans.

3.3.1.1 Remote Administration Tools

The toolkit of a RAT comprises two components: server program and client program. Server program is designed to run on a remote server, controlled by attackers, and serve two purposes: generating client program and managing infected machines. Server program usually has a nice Desktop Graphical User Interface (Figure 3.2) to

⁴<https://github.com/java-rat>

ease the management in case of a lot of machines being compromised. Figure 3.2, taken from a hacker blog⁵, shows the control panel of Darkcomet which displays information such as IP address, OS or RAM usage of infected machines.

Client program is configured and generated by server program. Common configuration options include domain name of the remote server, which hosts the server program, and the time interval for client program to regularly check in. The generated client program is the actual malware sample to be distributed to victim machines via various channels such as drive-by download or phishing schemes. Because the client program has to regularly update the status of infected machine to the remote server, the network traffic generated by the client program is usually highly periodic, whose period is precisely the preset time interval. This is the reason why most RATs leave periodic patterns in network traffic.

We obtained samples of RATs from a thread on hackforums.net⁶, which gathers a collection of paid as well as free RATs commonly sought after by criminals to carry out cyber attacks. The most notable RAT among these is Darkcomet, which is involved in a few scandals such as the surveillance of Syrian activists by the government⁷ or recently the campaign which takes advantages of the attack on the Charlie Hebbo magazine to spread malware⁸. Darkcomet first appeared in 2008 and is still a favorite RAT commonly used by attackers due to its stability and rich feature set.

3.3.1.2 Botnet Trojans

The toolkit of a Trojan comprises three components: Trojan builder, Trojan sample, and web control panel. Trojan builder is designed to configure and generate Trojan samples, which are the actual malware samples to be distributed to victim computers. However, unlike RATs, infected machines are not managed using the Trojan builder but the web control panel. The web control panel (Figure 3.3) is most commonly written in PHP language and hosted on a web server. Figure 3.3, also taken from

⁵<http://hack1291.blogspot.de/2011/10/darkcomet-rat-v42.html>

⁶<http://hackforums.net/showthread.php?tid=4935588>

⁷<https://threatpost.com/darkcomet-rat-used-new-attack-syrian-activists-081612/76919/>

⁸<http://securityaffairs.co/wordpress/32332/cyber-crime/criminals-je-suis-charlie-darkcomet.html>

Figure 3.2: Desktop control panel of RAT Darkcomet.

a hacker blog⁹, shows the web control panel of Trojan Neutrino which displays the commands to be executed by infected machines in the next check-in.

From network intrusion detection perspective, the use of HTTP protocol makes it more challenging to detect the traffic generated by Trojans as it is very hard to determine whether an HTTP request serves normal or malicious purposes without checking the underlying semantic of the request. However, the drawback of HTTP protocol is that remote server cannot directly initiate a communication with infected machines, especially the case that the infected machines sit behind a Network Address Translation device. The most straight-forward way for an infected machine to maintain established connection to a remote server is that of periodic polling. In most cases, Trojans are found to update status of infected computer to web control panel by periodically sending HTTP requests. This is the reason why most Trojans also leave highly periodic patterns in network traffic like RATs.

Depending on the availability of the complete toolkits and time of prevalence, we divide the class of Botnet Trojans into three different sub-classes: Cracked Old Trojans, Uncracked Old Trojans, and Recent Trojans. Old Trojans are the Trojans which

⁹<http://www.xylibox.com/2015/01/neutrino-bot.html>

Figure 3.3: Web control panel of Trojan Neutrino.

prevail more than 2 years ago as reported in a thread on kernelmode.info¹⁰. These Old Trojans had been popular at most 2 years ago and replaced by Recent Trojans, which are extracted from two recent reports from Symantec and SecureWorks [81, 82]. Among the Old Trojans, some of them are cracked meaning that we can obtain their complete toolkits. The rest of the Old Trojans, which we can only obtain the samples in the wild, falls under the sub-class of Uncracked Old Trojans.

Cracked Old Trojans The complete toolkits of the Trojans in this subclass are obtained from hackforums.net, leakforums.net and nulled.io. The most notable Trojan of this subclass is Zeus, which first appeared in 2007 and became widespread in 2009, infecting around 3.6 million PCs in US alone. Zeus source code was leaked in 2010, which then led to a lot of Zeus variants such as Citadel or Gameover Zeus.

Uncracked Old Trojans The difference between the Trojans in this subclass and the Cracked Old Trojans is that their complete toolkits are not available in the wild since they are not yet cracked. To obtain the full toolkit, we would have to pay Trojan authors in the black market. Therefore, only Trojan samples are obtainable, from malwr.com and virusshare.com, which are the places for users to submit suspicious

¹⁰<http://www.kernelmode.info/forum/viewtopic.php?f=16&t=2680>

binaries for automatic analysis.

Recent Trojans The names of the Trojans in this subclass are extracted from two recent reports by Symantec and Dell SecureWorks [81, 82]. These Trojans are still very active at the moment and their samples are obtained from malwr.com and virussshare.com.

In summary, we are able to obtain and experiment with 31 different malware families: 6 RATs, 5 Cracked Old Trojans, 10 Uncracked Old Trojans, and 10 Recent Trojans. We believe that this malware collection cover most common malware families found in the wild. The complete list of experimented malware families is shown in Table 3.1.

3.3.2 Methodology

3.3.2.1 Experiment Setup

To ensure that the malware behave in the way they would if they infect actual victim machines in the wild, we decide to use physical machines instead of virtual machines to carry out the experiments. This requirement is essential as some malware samples do have virtualization detection capability [83] that allows them to determine whether the running environment is virtual or real. The consequence is that they may delete themselves or generate random traffic attempting to confuse intrusion detection efforts. Figure 3.4 shows the experiment setup, which involves two different physical machines: the sandbox machine to execute malware samples and the monitoring machine to capture the generated traffic.

In Figure 3.4, the monitoring machine stands between the sandbox machine and the Internet to capture their communicating traffic. To carry out its job, the monitoring machine has two physical network adapters which are joined together using Bridged Network Connection functionality of Windows 10. The traffic is captured and visualized using Wireshark¹¹.

To ensure rapid experiment setup and free of noise traffic, the sandbox machine

¹¹<https://www.wireshark.org>

Figure 3.4: Experiment setup.

must satisfy the following two minimal requirements: it must be able to quickly revert to a clean state after being infected by a malware and it must not generate any background traffic, which would interfere with the traffic generated by the malware.

For the first requirement, Deep Freeze¹² is used to restore the system to a clean state after being infected by a malware. Deep Freeze is a program which allows protecting operating system by redirecting information being written to hard drive, giving the users the impression of making changes to the system. The changes are discarded once the system restarted. Malware samples are stored in a thawed space to make sure that they are not cleared out every restart.

For the second requirement, a freshly installed sandbox machine is left running for two days and its generated traffic is analyzed to determine the services which generate background traffic. All irrelevant background services are then turned off to make sure that the captured traffic is solely generated by the malware. In addition, we decide to use Windows XP as the environment to execute malware samples since we notice that Windows XP generally generates less background traffic than Windows 7.

3.3.2.2 Malware Execution

Experimented malware samples are stored in a thawed space in the sandbox machine and executed one at a time to capture their generated traffic. The traffic is plotted in real time using IO Graph functionality of Wireshark. In IO Graph, users are allowed to aggregate network packets by second, minute, ten minute, or hour and plot the aggregated traffic in terms of number of packets, byte or custom measure.

Once executed, if the malware doesn't generate any traffic during the first thirty

¹²<http://www.faronics.com/de/products/deep-freeze/>

minutes, it will be replaced by another sample of the same malware family. This is due to the fact that some Trojan samples appear to not generate any traffic during the first 30 minutes even though they have been executed on a physical machine, not a virtual one. In this case, we would have to look at alternative sources for a different sample of the same malware family.

For each malware, the experiment stops when one of the following two conditions occurs. The first condition is when a visually clear periodic traffic is observed. This is the case when the malware does generate periodic traffic. The second condition is when no visually clear periodic traffic can be observed during a long period (three days). The second condition is mutual exclusive to the first condition and occurs when the malware is unlikely to generate periodic traffic.

3.3.2.3 Periodicity Measure

To provide an objective measurement for the periodicity of network traffic, we devise a periodicity measuring technique, based on Fourier transform. This periodicity measure would serve as a supplement to the subjective visual recognition of periodic traffic performed by experimenter. Fourier transform¹³ is a mathematical operation (4.2) which decomposes arbitrary input signal $\{s\}_{i=1}^N$ into elementary periodic signals. The result of this operation $\{\mathcal{F}s\}_{i=1}^N$ is called the frequency spectrum of $\{s\}_{i=1}^N$. A demonstration of Fourier transform is shown in Figure 3.5.

$$(\mathcal{F}s)_n = \sum_{m=1}^N s_m e^{-2i\pi \frac{(n-1)(m-1)}{N}} \quad (3.1)$$

In Figure 3.5, two plots on the first row (A and B) show a simulated random signal on the left and on the right, its frequency spectrum. A simulated periodic signal and its frequency spectrum are shown in Figure 3.5C and Figure 3.5D, respectively. It can be seen from Figure 3.5 that spectral energy of a periodic signal would center on a few dominant frequencies while spectral energy of a random signal would spread across the whole frequency range. Therefore, one way to analytically differentiate between

¹³https://en.wikipedia.org/wiki/Discrete_Fourier_transform

Figure 3.5: A demonstration of Fourier transform.

periodic signal and random signal is to compare the energy percentage occupied by the few dominant frequencies. Based on this idea, we propose the following algorithm (Algorithm 3.3.2.3) to measure periodicity of a signal.

Algorithm 1 Compute Periodicity Measure

Input: signal s

Output: periodicity measure of signal s

```

freqSpec = FFT(s)
energyDist = sort(|freqSpec|)
domFreqs = length(s)/10
periodicity =  $\frac{\text{sum}(\text{energyDist}(1:\text{domFreqs}))}{\text{sum}(\text{energyDist})}$ 
return periodicity
    
```

In Algorithm 1, the frequency spectrum of input signal is first computed in line 1 using a fast implementation of Fourier transform (FFT stands for Fast Fourier Transform). In line 2, the energy distribution is constructed from the frequency spectrum and sorted in decreasing order. We then choose to use the energy percentage occupied by the top 10% strongest frequencies (line 3) as the measure for signal periodicity. This periodicity measure is computed in line 4. The highest possible value of this measure is 1 for sinusoidal signals and 0.1 for signal which has a flat energy distribution. Using this method, the periodicity measure for the periodic signal in Figure 3.5C is 0.9059, which is much higher than 0.3058 – the periodicity

Figure 3.6: Periodicity measures of all malware families.

measure of the random signal in Figure 3.5A.

3.4 Discussion

3.4.1 Malware Behavior

The most notable finding is that almost all malware samples (30/31) are found to generate periodic traffic, which can be identified numerically (Figure 3.6) and visually (Figure 3.7A and Figure 3.7B). The one exception is of Trojan Rustock [87], which has been observed over the duration of 80 hours and found to show highly random traffic (Figure 3.7C).

Periodicity measures of network traffic generated by all experimented malware families are shown in Figure 3.6. It is interesting to note that Trojan Rustock, which generates visually random traffic (Figure 3.7C), also has the lowest periodicity measure of 0.3652, which is comparable to the periodicity measure of 0.3058 of the simulated random signal in Figure 3.5C. On the other hand, the other malware families have relatively high periodicity measures with the mean of 0.6992 (dash line in Figure 3.6). Therefore, regarding identification of periodic traffic, the result due to the proposed objective periodicity measure highly agrees with the result due to visual identification performed by experimenter.

In addition, the periodicity measures of network traffic generated by malware families other than Trojan Rustock also have a relative low variance of 0.0038. The reason

for this observation is due to the fact that most of these malware families generate visually resembling traffic, of which the pulse train traffic generated by RAT Babylon as shown in Figure 3.7A is a good representative. The highest periodicity measure is of the traffic generated by Trojan Tinba¹⁴ whose traffic resembles a sinusoidal signal (Figure 3.7B).

Figure 3.7: Network traffic of Babylon, Tinba and Rustock.

Table 3.1 shows the collected information on networking behavior of all experimented malware families. In Table 3.1, there are three different columns specifying three different kinds of period: reconnaissance period, connection period and keep-alive period. These three periods are respectively associated to three consecutive phases in malware activities and will be analyzed in subsection 3.4.2 of Behavior Analysis. The longest observed period is four hours in the traffic generated by a sample of Trojan Simda¹⁵ and the shortest observed period is 3 seconds in the traffic generated by a sample of Trojan njRat¹⁶.

Moreover, we find that malware tend to use shorter period in the first phase when they try to look for a remote server to connect and longer period when they want to maintain the already established connections. This happens with RATs and Old Crack Trojans, for which we are able to obtain their complete toolkits and set up

¹⁴<https://securityintelligence.com/tinba-worlds-smallest-malware-has-big-bag-of-nasty-tricks/>

¹⁵<https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/simda>

¹⁶<https://phishme.com/the-return-of-njrat>

fake remote servers. The reason for this observation may be due to the fact that, in attacker's perspective, a connected malware is more valuable than an unconnected one, hence they deliberately try to use longer check-in period to evade detection.

Finally, it is interesting to note that almost all samples of Old Cracked Trojans (9/10) fails to find a remote server to connect. This is because of the fact that all of these Trojans are commonly found two years ago only, hence all the involved command and control servers have been taken down. On the other hand, a large portion of Recent Trojans (8/10) are successful in connecting to their remote server – for example, Ramnit¹⁷, Tinba or Sality¹⁸.

3.4.2 Behavior Analysis

In general, the overall networking behavior of a malware, from the time it infects a victim machine, can be roughly divided into three phases: the first phase to check for Internet connectivity, the second phase to look for a remote server to connect and the third phase to maintain established connection.

In the first phase, malware tries to determine whether the infected machine is connected to the Internet (Trojan Cridex¹⁹ or Trojan Gootkit²⁰). Performing DNS queries and trying to connect to popular websites such as Google.com and Microsoft.com are the most common choices. This reconnaissance phase helps to not reveal the remote server until Internet connectivity is confirmed. However, some malware families may skip this phase such as Trojan Hesperbot²¹ or Trojan Ramnit and directly look for the designated remote servers to connect.

In the second phase, malware tries to look for an active remote server, controlled by attacker, to connect. Malware often actively resolves many different domain names to increase the chance of finding an active server. Some malware families are found

¹⁷<https://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Win32%2FRamnit>

¹⁸<https://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Win32%2FSality>

¹⁹https://www.symantec.com/security_response/writeup.jsp?docid=2012-012103-0840-99

²⁰https://www.symantec.com/security_response/writeup.jsp?docid=2010-051118-0604-99

²¹<http://www.welivesecurity.com/2013/09/04/hesperbot-a-new-advanced-banking-trojan-in-the-wild/>

Table 3.1: Periodicity measures and periods of all experimented malware families.

Class	Malware Family	Recons Period	Pe-riod	Connection Period	Keep-alive Period	Periodicity Measure	Server Connection
RAT	Darkcomet	N/A		21 s	21 s	0.7061	successful
	xRat	N/A		7 s	25 s	0.6843	successful
	njRat	N/A		3 s	15 s	0.7531	successful
	Sub-7	N/A		10 s	20 s	0.7533	successful
	Novalite	N/A		21 s	20 s	0.6435	successful
	Babylon	N/A		6 s	20 s	0.7132	successful
Cracked Old Trojan	Citadel	N/A		25 mins	4 mins	0.6451	successful
	Zeus	N/A		5 mins	20 mins	0.7748	successful
	Andromeda	N/A		1 min 10 s	9 mins	0.6343	successful
	Neutrino	N/A		16 s	4 min 10 s	0.6752	successful
	Athena	N/A		1 min 7 s	1 min 37 s	0.5835	successful
Uncracked Old Trojan	Cridex	14 mins 56 s		N/A	N/A	0.6795	unsuccessful
	Gootkit	16 mins 13 s		N/A	N/A	0.7098	unsuccessful
	Kelihos	N/A		N/A	3 mins 55 s	0.7331	successful
	NgrBot	N/A		17 s	N/A	0.7512	unsuccessful
	SmokeLoader	N/A		17 mins	N/A	0.7671	unsuccessful
	Rustock	N/A		N/A	N/A	0.3652	unsuccessful
	Kuluoz	N/A		21	N/A	0.5953	unsuccessful
	Conficker	20 mins		N/A	N/A	0.6439	unsuccessful
	Simda	N/A		15 mins	N/A	0.7102	unsuccessful
	Sinowal	N/A		45 s	N/A	0.6735	unsuccessful
Recent Trojan	Silly	15 mins 12 s		N/A	N/A	0.6983	unsuccessful
	Virut	5 mins 33 s		N/A	N/A	0.6992	unsuccessful
	Dyre	N/A		N/A	6 mins	0.6951	successful
	Hesperbot	N/A		N/A	1 min 5 s	0.7798	successful
	Gameover	N/A		N/A	13 mins	0.6745	successful
	Ramnit	N/A		N/A	3 mins 34 s	0.7689	successful
	Tinba	N/A		N/A	24 mins 45 s	0.8998	successful
	Sality	N/A		N/A	4 hours	0.7423	successful
	Gozi	N/A		N/A	5 mins 47 s	0.6992	successful
	Shylock	N/A		N/A	4 mins	0.6961	successful

to use domain names, which are randomly and automatically generated by a domain name generation (DGA) algorithm, to further increase the chance. These DGA-enabled malware families includes Gameover Zeus and Dyre²². The second phase is the noisiest phase and usually ends when the malware is able to find an active server.

In the third phase, malware tries to maintain an active connection with connected server. This is the phase when attacker actually carries out criminal actions such as extracting confidential information or encrypting victim's data, with the help of the malware. The network traffic due to these criminal activities is usually random and unpredictable. However, when the malware is left alone, its behavior often becomes regular. In addition, malware families such as RAT Darkcomet and RAT xRat are

²²<http://blogs.cisco.com/security/talos/threat-spotlight-dyre>

found to check in the connected server less often than the previous two phases to evade detection

These three phases of malware behavior are only for the general cases and some malware families are found to skip one or two phases. In addition, in all three phases, we found that the malware shows highly periodic networking behavior with possibly different periods during different phases. Under certain conditions, this property sets the network traffic generated by malware apart from user-generated traffic and provides a valuable chance for intrusion detection.

3.5 Analyzing Traffic using Wavelet Transform

To exploit this prevalent periodic behavior of malware for intrusion detection, we need to address the following two problems: frequency localization and time localization of periodic signals. Frequency localization refers to the problem of identifying the fundamental frequency of the hidden periodic signals. In the context of intrusion detection, frequency localization involves the determination of how often the malware regularly connect to attacker's server. On the other hand, time localization addresses the problem of determining the temporal occurrence of the periodic signals. In the context of intrusion detection, time localization concerns with the identification of the precise points in time that the malware infects and disinfects a network.

The first problem of frequency localization has been well studied by previous studies [8, 19, 88] with the use of Fourier Transform. On the contrary, the problem of time localization is largely left open due to an inherent weakness of this transform: the frequency spectrum produced by the application of Fourier Transform contains no information regarding the temporal occurrences of the reported periodic signals. In this section, we explore of the use of Continuous Wavelet Transform to address both problems of frequency localization and time localization.

3.5.1 Frequency Analysis

3.5.1.1 Signal and Signal Correlation

We consider a signal (or time series) f as a series of indexed complex numbers: $f = (f_1, f_2, \dots, f_N)$. N is the length of signal f . The correlation between two signals is defined as follow.

Definition 3.1 *Let's f and g be two arbitrary signals of the same length N . The correlation between f and g is also a signal of the same length as of f and g , denoted as $(f : g)$, and defined by the following formula:*

$$(f : g)_n = f_1 g_n + f_2 g_{n+1} + \dots + f_N g_{n+N-1} \quad (3.2)$$

In formula (3.2), we use the wrapping convention that the signal will wrap around if the index becomes larger than N . For example, g_{N+1} will be wrapped around and becomes g_1 or g_{N+2} becomes g_2 .

3.5.1.2 Discrete Fourier Transform

Definition 3.2 *Fourier transform [89] (discrete form) is a mathematical operation which decomposes arbitrary input signals into elementary (and periodic) signals.*

$$f_m = \frac{1}{N} \sum_{n=-N/2}^{N/2-1} (Ff)_n e^{i2\pi \frac{(n-1)(m-1)}{N}} \quad (3.3)$$

In formula (3.3), the exponential components are periodic complex signals with varying frequencies. The Fourier coefficients $(Ff)_n$ in formula (3.3) are obtained by using the following formula:

$$(Ff)_n = \sum_{m=1}^N f_m e^{-i2\pi \frac{(n-1)(m-1)}{N}}$$

3.5.1.3 Continuous Wavelet Transform

Consider a function, called mother wavelet function (or shortly as wavelet), defined as follow:

Figure 3.8: Mexican Hat Wavelets.

$$\psi(x) = 2\pi w^{-1/2}(1 - 2\pi(x/w)^2)e^{-\pi(x/w)^2} \quad (3.4)$$

In formula (3.4), parameter w is known as the width parameter, which controls how wide the graph of $\psi(x)$ spreads over the x -axis. The meaning of w can be visually illustrated by the three graphs of $\psi(x)$, which are associated with three different values of w ($\frac{1}{16}$, $\frac{1}{8}$ and $\frac{1}{4}$) shown in Figure 3.8, respectively from the top to the bottom. This particular wavelet is called Mexican Hat wavelet because of the shape of its graph.

For some w , let's denote a scaled version of the corresponding $\psi(x)$ with respect to the scale factor $s = 2^{-k/M}$, in which $k = 0, 1, 2, \dots, I \cdot M$ as:

$$\psi_s(x) = \frac{1}{\sqrt{s}}\psi\left(\frac{x}{s}\right)$$

With a choice of positive integers I and M , we will be able to obtain the total of $I \cdot M + 1$ different scale factors s , hence that many scaled versions of $\psi(x)$. I is usually referred to as the number of octaves, M the number of voices per octave, and $\psi_s(x)$ the scaled mother wavelet function.

Let's sample each scaled wavelet, $\psi_s(x)$, at N discrete sampling points $\{t_i\}_{i=1}^N$, with N is the length of an input signal f . We will obtain a signal g_s , defined as:

$$g_s = (\psi_s(t_1), \psi_s(t_2), \dots, \psi_s(t_N))$$

Therefore, with a choice of the mother wavelet function $\psi_s(x)$ (including the width parameter w), and a choice of the scale factor s (including the number of octaves I and the number of voices per octave M), we will be able to obtain a collection G_s of $I \cdot M + 1$ reference signals g_s 's. The reference signals g_s 's can be thought of as discretized scaled versions of the mother wavelet function $\psi(x)$. The Continuous Wavelet Transform is defined as follow:

Definition 3.3 *Given a collection G_s of reference signals, the Continuous Wavelet Transform (CWT) of an input signal f is defined as the collection of signals obtained by correlating f with each signal of the collection G_s [89]. The wavelet $\psi(x)$ which is used in the construction of collection G_s is called the reference function of CWT.*

3.5.1.4 An Illustration of Continuous Wavelet Transform

We will use periodic sinusoidal signals to illustrate the working of CWT. Figure 3.9A shows the plot of the following signal (after being discretized), which will be then referred to as signal A:

$$\begin{aligned} & \sin(40\pi x)e^{-100\pi(x-0.2)^2} + \\ & [\sin(40\pi x) + 2\cos(160\pi x)]e^{-50\pi(x-0.5)^2} + \\ & 2\sin(160\pi x)e^{-100\pi(x-0.8)^2} \end{aligned}$$

The plot in Figure 3.9A shows three visually distinct parts, corresponding to the three terms making up the formula of signal A. The first part on the left corresponds to the term $\sin(40\pi x)e^{-100\pi(x-0.2)^2}$, which is the multiplication of a sine component of frequency 20 Hz and a damping factor. The damping factor is to localize the sine component onto a small region around $x = 0.2$. Similarly, the second part at the middle contains the superposition of a sine component and a cosine component of

Figure 3.9: Wavelet Transform.

frequencies 20 Hz and 80 Hz, respectively. The second part centers on $x = 0.5$. The third part on the right contains a sine component of frequency 80 Hz centering on $x = 0.8$.

Figure 3.9B shows the CWT of signal A, called the scalogram. The vertical axis of the scalogram represents the reciprocal of scale factor s and its horizontal axis represents the time index (varying from 1 to N , in which N is the length of input signal A). Each line of the scalogram corresponds to a scale factor s and plots the correlation between signal A and the reference signal g_s . This resulting signal is plotted on a single line using a jet color scale which ranges from the most bluish (the smallest value) to the most reddish (the largest value).

In the scalogram of signal A (depicted in Figure 3.9B), we can see that there are four visually distinct regions. These four regions of the scalogram reveals two insights regarding signal A that cannot be easily seen in its time domain plot (Figure 3.9A). Firstly, projecting these four regions onto the vertical axis, we will be able to obtain two visually distinct reddish lines. The lower line indicates the frequency of the lower two regions, which is 20 Hz; and the upper line indicates the frequency of the upper two regions, which is 80 Hz. Therefore, the scalogram of signal A has correctly

revealed the two frequencies, 20 Hz and 80 Hz contained in signal B. This capability of frequency localization is common to both Continuous Wavelet Transform and Fourier Transform.

Secondly, projecting these four regions onto the horizontal axis, we will be able to obtain three visually distinct reddish lines, which are located directly under the three distinct parts making up the plot of signal A, $x = 0.2$, $x = 0.5$ and $x = 0.8$. The meaning of these three lines is that the scalogram of signal A has correctly identified that the components of frequency 20 Hz roughly occurs at $x = 0.2$ and $x = 0.5$ and the components of frequency 80 Hz roughly occurs at $x = 0.5$ and $x = 0.8$. Therefore, the scalogram of an input signal not only tell the presence of a periodic patterns hidden in that signal but also the temporal occurrences of the detected patterns. This capability of time localization is exclusive to Continuous Wavelet Transform marking its advantage over Fourier Transform in detecting periodic signals generated by malware.

In summary, the scalogram of a signal, produced by Continuous Wavelet Transform, not only reveals which sinusoidal components making up that signal—the property of frequency localization—but also where those frequency components occur in time—the property of time localization. This characteristic of Continuous Wavelet Transform is its advantage over Fourier Transform and makes it an ideal tool for detecting periodic signals generated by malware.

3.5.2 Experiments with Wavelet Transform

We will study—in two experiments—the capability of Continuous Wavelet Transform to localize the frequency as well as the temporal occurrences of the periodic signal generated by Darkcomet²³. Once infected, Darkcomet is known to regularly initiate connections to the configured remote server. As a result, the generated traffic is highly periodic with the intervals between consecutive connection initiations following the pattern: 3 second, 6 second, 12 second and then 3 second (Figure 3.1A). The funda-

²³<http://hack1291.blogspot.de/2011/10/darkcomet-rat-v42.html>

mental period of the generated signal is $3 + 6 + 12 = 21$ seconds and its fundamental frequency $\frac{1}{21} = 0.04762$ Hz.

In the first experiment, we will simulate the situation that a clean and idle computer is infected by a malware during the duration of 10 minutes. The captured traffic, therefore, contains only malicious traffic and the background traffic generated by operating system. The second experiment is identical to the first experiment except that the involved computer is actively used by a user. The captured traffic, hence, additionally contains the browsing traffic, which then plays the role of noise to challenge the analyzing power of certain chosen wavelet function. The two traffic captures gathered from the two experiments are used to construct two signals, signal B and signal C, which are to be analyzed using Continuous Wavelet Transform. Both signals are constructed in the same way: summing the sizes of the captured TCP SYN packets every half a second interval. Both experiments lasted 30 minutes.

3.5.2.1 First Experiment - Malicious Traffic without Noise

In this experiment, a computer (a fresh installation of Window 7) was connected to the Internet and left idle during the first ten minute; the computer was then infected by DarkComet for the middle ten minutes; in the last ten minutes, the computers was disinfected and left idle again. The signal obtained from this traffic capture will then be referred to as signal B. We will analyze signal B using three different frequency transforms: Discrete Fourier Transform, Continuous Wavelet Transform using Mexican Hat wavelet and Continuous Wavelet Transform using Morlet wavelet [90]. These transforms will be shortly referred to as DFT, MHT and MT, respectively.

We can see from the time domain plot of signal B (Figure 3.10A) that there is a highly periodic signal of period 21s taking place during the middle ten minutes. This periodic pattern is correctly reported by its DFT plot (Figure 3.10B): the fundamental peak in the DFT plot is located at frequency 0.04678 Hz, corresponding to the fundamental period of 21s of signal B. However, the DFT plot fails to report the precise moments that this periodic signal starts and stops.

Figure 3.10: DarkComet Traffic, without Noise - Signal B. A) Time domain plot. B) DFT plot. C) CWT plot using Mexican Hat wavelet. D) CWT plot using Morlet wavelet.

On the contrary, this required localization in time of the identified periodic pattern is correctly reported by the MHT plot of signal B (Figure 3.10C). The starting time and stopping time are marked by a region located at the bottom of the MHT plot, which spans the middle 10 minutes. Therefore, we can conclude that the MHT is more advantageous than the DFT in term of time localization. However, the MHT plot does not perfectly identify the involved periodic signal as the resultant representing region

Figure 3.11: DarkComet Traffic with Noise Injected - Signal C. A) Time domain plot. B) DFT plot. C) CWT plot using Mexican Hat wavelet. D) CWT plot using Morlet wavelet.

is highly unconnected.

The MT plot of signal B (Figure 3.10D) performs even better than the MHT plot at localizing the periodic signal in frequency and in time. There is a single connected region in the MT plot which spans precisely the middle 10 minutes centering on the scale of 41, which corresponds to the frequency of 0.04678 Hz. Therefore, we can conclude that the Morlet wavelet is more suitable than the Mexican Hat wavelet in

detecting DarkComet-generated signals.

However, we can still see an imperfection in the MH plot as there is still an unconnected region, which locates directly below the connected one. This imperfection makes Morlet wavelet a good choice of wavelet but not the optimal choice for analyzing this specific periodic signal generated by Darkcomet. In the second experiment when user traffic is included, these regions will completely disappear due to the effect of added noise.

3.5.2.2 Second Experiment - Malicious Traffic with Noise Injected

The second experiment is identical to the first experiment except that the experimented computer is actively used by an ordinary user throughout the whole 30 minutes of experimentation. The time series obtained from the captured network traffic will then be referred to as signal C, which can be thought of as a noise injected version of signal B. From the top to the bottom, Figure 3.11 shows the time domain plot of signal C, its DFT plot, MHT plot and MT plot, respectively.

The first observation that we can notice from the DFT plot of signal C (Figure 3.11B) is that the representational peak of the hidden periodic signal (Signal B) is not visible anymore. This observation signifies the fact that the noise, generated by the user, completely dominates and conceals the periodic traffic, generated by Darkcomet. Similarly, there are also no visible regions representing signal B visible from the MHT plot (Figure 3.11C) and the MT plot (Figure 3.11D). Therefore, we can conclude that, even with a good choice (but not the optimal choice) of the wavelet function, high level of noise can completely contaminate the representative peak of the hidden periodic signal in frequency spectrum and the representative region in the scalograms.

We have tried four wavelet functions available in Matlab 2015a—namely, Mexican Hat wavelet, Morlet wavelet, Paul wavelet and Bump wavelet [90]—and we find that the Morlet wavelet gives the best analyzing power. However, as we have already seen in the first experiment, Morlet wavelet is still not the optimal wavelet for the detection

of periodic signal generated by the DarkComet.

3.5.3 Discussions

In each of the two experiments described previously, we have used two different wavelets as the reference functions for CWT: the Mexican Hat wavelet and the Morlet wavelet. In the first experiment, the Morlet wavelet clearly works better in the task of identifying the periodic traffic generated by RAT Darkcomet. This result is reasonable as the Morlet wavelet was specifically designed to detect periodic sinusoidal signal whereas the Mexican Hat wavelet to detect anomalies in seismic data [91]. This comparison brings us to the first insight.

The first insight is about the features that indicate a good wavelet. The answer depends on the specific periodic pattern that we would like to detect. As suggested by the first experiment, to detect a periodic pattern, we need to design a wavelet in such a way that its corresponding CWT produces a single connected region in the scalogram, representing each pattern. By this standard, the Morlet wavelet is a good but not the optimal wavelet to detect the traffic pattern generated by Darkcomet due to the fact that the involved periodic traffic results in two different regions in the MT plot: a connected region and a highly unconnected one.

In the second experiment, the high level of noise has completely buried the periodic pattern generated by Darkcomet even though we have already utilized a fairly good wavelet, the Morlet wavelet. This signifies the fact that other than the specific wavelet that we choose, the level of noise also greatly contributes to the effectiveness of CWT in the detection of periodic pattern hidden in input signal. This brings us to the second insight. The effectiveness of CWT in detecting periodic pattern is determined mainly by two factors: the chosen wavelet and the level of noise.

Due to the second insight, the failure to detect the periodic traffic generated by DarkComet in the second experiment can be explained by the high level of noise and/or the wrong choice of wavelet. The wrong of choice of wavelet is justifiable because we have known from the first experiment that the Morlet wavelet is good but

not the optimal wavelet in detecting such periodic pattern. The high level of noise is also justifiable because of the fact that we cannot identify any peak in the DFT plot of signal C (Figure 3.11B). Therefore, the first challenge of the approach to detect intrusion based periodic behavior of malware is the design of the optimal wavelet for identifying hidden periodic patterns in high level of noise in practice.

The second challenge of this approach is its practicality with regard to operational intrusion detection. According to another study by Bartlett et al. [92], there is actually a wide range of standard applications which also exhibit periodic networking behavior such as Windows Update Service or Web Counters, which are very popular services in operational settings. Therefore, the detection of periodic signals hidden in network traffic with the aim to detect the presence of malware is bound to generate many false positive alerts, which exacerbates the alert fatigue problem faced by cyber analysts. To address this issue, our future work includes the design of a visual analytics solution to couple with a CWT-based detection method allowing quick verification of detection result.

3.6 On Periodicity-Based Intrusion Detection

3.6.1 Opportunities

3.6.1.1 Supporting Observations

There are two observations from the experiments that support periodicity-based intrusion detection. These two observations reflect the differences in short term and long term between network traffic generated by malware and network traffic generated by typical user.

Firstly, almost all experimented malware families are found to be consistently generating regular traffic patterns (30/31), with one exception of Trojan Rustock). For comparison, Figure 3.12. shows the network traffic generated by a typical user recorded in the duration of 3 days. From Figure 3.7 and Figure 3.12, we can see that regularity is the feature which discriminates malware-generated traffic from user-generated traffic (in one day).

Figure 3.12: Browsing traffic generated by a typical user in 3 days.

Secondly, in long term, network traffic of typical user would also become periodic with the rough period of 1 day or 24 hours (Figure 3.12). This behavior is common for a typical user because one tends to repeat our browsing pattern every day. However, there's still an important difference: for malware-generated traffic, the period is considerably less than 24 hours (the maximum observed period is 4 hours – Table 3.1) and for user-generated traffic, the period is roughly 24 hours.

In summary, there are two evidential observations that support periodicity-based intrusion detection. The first is the observation that malware-generated traffic is usually more periodic than user-generated traffic in short term (less than 1 day). And in long term (more than 1 day), both types of traffic are observed to repeat but user-generated traffic has a period much higher than the period of malware-generated traffic, as observed in the experiments.

3.6.1.2 Possible Approaches

The purpose of intrusion detection is to identify infected internal host or malicious external host so as to take countermeasure actions. Correspondingly, there are two approaches towards periodicity-based intrusion detection.

In the first approach, the purpose is to identify infected internal host. In the perspective of periodicity-based intrusion detection, the detection can be carried out by identifying periodic traffic pattern associated with that host. Any internal host

which generates periodic patterns in network traffic is considered as infected (and false positives are the consequence). The overall traffic is, thereby, aggregated based on local IP addresses (internal host) and the information on remote IP addresses (external host) is discarded. As the result, each internal host is associated with an aggregated traffic, which contains both user-generated traffic as noise and malware-generated traffic as signal. Therefore, in this approach, the problem of intrusion detection is reduced to a signal detection problem and the challenge is to design a robust detection algorithm which can work well under high noise in practice.

In a typical network, as the number of external hosts usually outnumbers the number of internal hosts, the advantage of the first approach is the less number of time series to process. However, the disadvantage of this approach is that malicious periodic traffic is usually completely buried in noisy traffic generated by normal users, making detection very difficult or almost impossible. Nonetheless, this method is capable of detecting malware traffic during night time or idle hours.

In the second approach, the purpose is to identify malicious external host and the detection can be carried out by identifying periodic traffic patterns associated with that host. Any external host which is associated with periodic network traffic is considered as malicious (false positives are a consequence of this assumption). The overall traffic is, thereby, aggregated based on remote IP addresses (external hosts), discarding information on local IP addresses (internal hosts). The expectation here is that network traffic associated with malicious host would numerically and/or visually appear to be more periodic than the traffic associated with benign hosts (or servers). Therefore, in this case, the intrusion detection problem is reduced to a signal identification problem. As the number of external hosts usually outnumbers the number of internal hosts in a typical network and the challenge here is to design periodicity measuring algorithm to cope with the speed of real-time traffic.

Unlike the first approach in which malware-generated traffic is mixed with user-generated traffic, the most notable advantage of the second approach is that there is a separation between them when the overall traffic is aggregated on external hosts.

Table 3.2: Periods of some good applications.

Application	Observed Period
WeatherEye	30 - 120 mins
MacOS Dashboard	30 - 120 mins
Gnome Clock Applet	30 - 120 mins
RSS News Feeds	30 - 60 mins
Web Counter	5 - 30 mins
Peer-to-Peer	20 - 30 mins

However, the disadvantage of this approach is that it would fail if the malware generate many random domain names during the second phase when it looks for an active server to connect.

In summary, there are generally two approaches towards periodicity-based intrusion detection: aggregating overall traffic based on internal hosts and aggregating overall traffic based on external hosts. Both of these approaches have the same two challenges that are discussed in subsection 3.6.2.

3.6.2 Challenges

There are generally two challenges in periodicity-based intrusion detection: false positive alerts and robustness. These two challenges are involved in both approaches towards periodicity-based intrusion detection.

3.6.2.1 False Positive Alerts

In a normal network, there are generally many good services that potentially leave periodic patterns in network traffic such as Operating System Automatic Update Services and RSS News Feeders. Table 3.2 shows an estimate on the repeating periods of network traffic generated by these services [9]. These periods are comparable to the periods exhibited by malware and can easily confuse any periodicity detection algorithms. Therefore, there must be some additional mechanism to further verify the detection result. Two possible approaches are verification of external hosts and cross-checking of the content of network packets by leveraging deep packet inspection.

The other source of false positive alerts comes from the regularity in the network traffic generated by typical users. This issue arises because of the fact that the daily network usage of a user typically repeats after roughly 24 hours (Figure 3.12). However, as the periods exhibited by automatic programs (malwares as well as benign applications) are usually less than 24 hours (the maximum observed period is 4 hours), the periodic traffic patterns generated by typical user can be safely eliminated by limiting the period of interesting periodic signal to less than 24 hours.

3.6.2.2 Robustness

In the first approach, network traffic is aggregated based on internal hosts and the problem of intrusion detection is reduced to the problem of detecting periodic signal buried in random noise. However, in a normal network, the energy of the noise is usually prohibitively higher than the energy of interesting periodic signals, making it very hard (almost impossible) to carry out detection robustly.

The second approach avoids this difficulty by aggregating traffic based on external hosts. By doing this, malware-generated traffic is separated from user-generated traffic and expected to have higher periodicity measure. However, this approach also has robustness difficulty because it can be evaded easily in practice by malware which uses domain name generation algorithm.

In summary, there are generally two main challenges in periodicity-based intrusion detection. The first challenge of false positive alerts can be overcome by, for example, further cross-checking of content of network packet. However, the second challenge of robustness is an inherent weakness of periodicity-based intrusion detection and may not be easy to tackle in practice.

3.7 Summary

In this chapter, we have thoroughly studied the networking behavior of two large classes of malwares commonly found in the wild: RATs and Trojans. As the data shows, 30 out of 31 malware families are found to generate periodic network traffic.

Therefore, regarding the first question set out in Section 3.2, it is very likely (96.77%) that a malware captured in the wild would leave periodic traces in network traffic. Under certain conditions, this periodic behavior makes the malwares stand out from typical users and can be exploited to develop intrusion detection algorithms. We would call this approach periodicity-based intrusion detection.

Regarding the second question, there are generally two different approaches towards periodicity-based intrusion detection. In the first approach, network traffic is aggregated based on internal hosts and the problem of intrusion detection is formulated as a signal detection problem. The advantage of this approach is the less number of time series' to process. However, the challenge of this approach is that detection algorithms would have to be robust enough to cope well with high level of noise in practice. In the second approach, network traffic is aggregated based on external hosts and the problem of intrusion detection is reduced to a signal identification problem. The advantage of this approach is the separation of periodic signal generated by malwares from random signals generated by normal users. However, the challenge of this approach is that the algorithm to measure the periodicity of captured network traffic should be fast enough to cope with the speed of real-time traffic.

Continuous Wavelet Transform is promising in tackling both frequency localization and time localization issues involved in this problem. Although, this technique does work in the simplest case of noise free but in the case of heavy noise injected, it failed to detect the periodic signals. There are two closely related reasons for this failure: the non-optimal choice of Morlet wavelet for detecting the kind of periodic patterns generated by DarkComet and the high level of injected noise.

4

Detecting Periodically-behaved Malware

This chapter presents the second part of the **Contribution 1** of the thesis. It addresses the problem of detecting the presence of malware that leave periodic traces in network traffic. This characteristic behavior of malware was found to be surprisingly prevalent in Chapter 3. To this end, we propose a visual analytics solution that supports both automatic detection and manual inspection of periodic signals hidden in network traffic. The detected periodic signals are visually verified in an overview using a circular graph and two stacked histograms as well as in detail using deep packet inspection. Our approach offers the capability to detect complex periodic patterns, but avoids the unverifiability issue often encountered in related work. The periodicity assumption imposed on malware behavior is a relatively weak assumption, but initial evaluations with a simulated scenario as well as a publicly available network capture demonstrate its applicability.

4.1 Introduction

According to the study by Bartlett et al. [92], there is actually a wide range of standard applications that also exhibit periodic networking behavior such as Windows Update

Service or Web Counters, which are very popular services in operational settings. Therefore, the detection of periodic signals in network traffic with the aim to detect the presence of malware is bound to generate many false positive alerts, which exacerbates the alert fatigue problem faced by cyber analysts. Motivated by this situation, in this chapter, we address two problems: How to robustly detect periodic signals hidden in network traffic, and How to help analyst quickly verify detection results.

The former problem has been addressed in the work of Hubballi et al. [7] and Chen et al. [8]. However, this previous work is either not able to detect complicated periodic signals often generated by malware or has an issue with the verification of detected periodic signals for ground truth. Therefore, we adopt a visual analytics approach that involves a novel automatic algorithm to detect complex periodic signals and the usage of interactive visualization for quick verification of detection result.

The contribution of this chapter includes:

- A novel method, based on Fourier transform, to identify periodic signals hidden in network traffic (Section 4.4). This method is the core of the periodicity detector, which is designed to detect complex periodic patterns and avoids the unverifiability issue left open by previous work.
- An interactive visualization (Section 4.5), whose main component is a visualization consisting of a circular graph and two stacked histograms to help analysts quickly verify the results returned by the periodicity detector. These visualizations are connected via a brushing mechanism to connect overview and details of the detection results.

Analysts often want to explore the root cause of alerts in addition to the notification of some indicator of compromise, so as to take appropriate countermeasures. This final step of the detection result verification is often omitted in previous works but very essential in operational settings. Furthermore, our visualization also enables the analyst to inspect hidden periodic signals manually without being guided by the periodicity detector (Section 4.5.2). In both cases, the final result is either the falsifi-

cation or the confirmation of suspected periodic signals, which in turn help to adjust the parameters as well as the whitelist of the periodicity detector.

In Section 4.2, we describe the different phases in the post-infection stage of a malware which potentially leave periodic traces in network traffic and then formalize the problem of periodicity-based intrusion detection using a time series binning technique. The system architecture is described in Section 4.3. We evaluate the applicability of the system in two different scenarios (Section 4.6). The first scenario is a simulated one, in which we set up a small network with three computers to assess the implemented features of the system. In the second scenario, we evaluate the system with the ISCX Botnet dataset [93]. In Section 4.7, we then discuss advantages and disadvantages of our detection method highlighted by the evaluated scenarios. We summarize the chapter in Section 4.8.

4.2 Problem Setting

As in our study [6], we suggest to divide the post-infection stage of malware into three phases. In the first phase, a malware would try to determine whether the infected machine is connected to the Internet. This reconnaissance phase helps to not reveal the address of the remote server until Internet connectivity is confirmed. In the second phase, the malware tries to look for an active remote server to connect. Some malware samples were found to use domain name generation algorithms (DGA), such as the recent malware trojan Dyre¹, to further increase the chance. In the third phase, the malware would try to maintain active connections with connected servers. This is the phase when the attacker actually carries out criminal actions such as extracting confidential information or encrypting a victim's data with the help of the implanted malware.

In all these three phases, malware samples are found to leave periodic traces in network traffic [18]. To exploit this characteristic behavior of malware, we formulate a periodic signal identification problem, in which each internal and external IP address

¹<http://blogs.cisco.com/security/talos/threat-spotlight-dyre>

Figure 4.1: System architecture.

is individually assessed for periodicity (Section 4.2.1). As discussed in Section 4.2.2, the problem of detection result verification (or alert verification), omitted in [7, 8, 92], is imperative in operational settings and deserves serious consideration.

4.2.1 Periodicity-based Intrusion Detection

There is a wide range of periodic behavior exhibited by malware. In order to maximize the chance of detecting their presence, we propose to assess the periodicity of network traffic associated with each individual internal IP address (to target the second phase of DGA-enabled malware) and each external IP address (to target the first phase and the third phase). To this end, we divide the network traffic associated with each entity (internal IP address or external IP address) into a series of communication sessions (also known as netflows), each characterized by a 5-tuple: source IP address, source port, destination IP address, destination port and protocol.

Each netflow is associated with two timestamps: start time and end time. These two timestamps mark the start and the end of the corresponding communication session. Other than these two temporal parameters, a netflow is also associated with two other value parameters: the number of packets that the netflow contains and the size of the netflow in number of bytes. These four parameters (start time, end time, number of packets and netflow size) form the basis of the computation of periodicity

and the visualization of the netflows.

The netflows associated with each entity are grouped together based on any one of the two temporal parameters using a time bin, resulting in a time series for each entity:

$$f = (f_0, f_1, \dots, f_{N-1})$$

The value of each bin, f_n , is the sum of the chosen value parameter of the netflows residing in the same bin. The choice of the temporal parameter, the value parameter and the size of the time bin can all be specified by the analyst. After this time binning step, the problem becomes the measurement of the constructed time series f (later referred to as signal) for periodicity, which is addressed by Algorithm 2.

4.2.2 Detection Result Verification

Operational intrusion detection is only half of the problem. Equally important is the interpretation of the detected results. As argued by Robin et al. [34], this semantic gap is very different from other fields and essential to operational intrusion detection. This gap arises because of the fact that all activities detected by an intrusion detection algorithm are only suspicious, not the actually malicious activities. Therefore, in intrusion detection, the presence of an analyst is crucial for judging within the context of the alerts and verifying detection results.

In our case of periodicity-based intrusion detection, this situation is very obvious. For example, it may be detected that some internal host is acting periodically. This periodicity may equally likely be caused by a good operating system service or by malware. Therefore, the problem here is to effectively bring up the context surrounding each alert, thereby helping the analyst to quickly verify detection results. To this end, we propose an interactive visualization, which successively reveals the three levels of details of the time series associated with each alert. The research on the techniques for visualization of time series is well established [94], and in our work, we use a circular graph (Section 4.5.1) and two stacked histograms to provide the first two levels

of detail (Section 4.5.2). In the third and lowest level, the alerts are verified against the actual content of the netflows, provided by the help of deep packet inspection (Section 4.5.3).

4.3 System Architecture

Our system architecture is shown in Figure 4.1. Network traffic is firstly forwarded to a netflow aggregator (SplitCap²), which assembles the netflows by grouping network packets based on five attributes: source IP address, destination IP address, source port, destination port, and protocol. The metadata of the netflows is then forwarded to the periodicity detector and an Elastic Search server. The raw data, however, is forwarded to a file system for long term storage.

The job of the periodicity detector (Section 4.4) is to identify periodic signals based on Algorithm 2. For a good tradeoff between the ability to detect a wide range of periodic signals and the required computing power, we set the default time bin to 5 seconds. The periodicity detector is also associated with a whitelist of IP addresses previously confirmed to be benign. These IP addresses will be ignored by the periodicity detector in computing periodicity measure. The time bin and the whitelist can be dynamically modified by the analyst during the process of exploring periodic signals. The detected periodic signals are finally pushed to the Alert View for verification by the analyst (Section 4.5.1).

We use Elastic Search³ as the backend for the visualization of the time-series associated with each alert because of its support for fast interactive analysis. The Time Series View (Section 4.5.2) allows the analyst to try out new time bins as well as different temporal and value parameters. While the same functionality is supported by the I/O Graph feature of Wireshark⁴, this feature is very slow as Wireshark is designed to rescan the whole network capture whenever a new filter is formulated.

The raw packet data contain the ground truth of the generated alerts and can grow

²<https://www.netresec.com/?page=SplitCap>

³<https://www.elastic.co>

⁴<https://www.wireshark.org>

Figure 4.2: A demonstration of Fourier transform showing how the energy of different signals is redistributed differently.

very big in a busy network. It is therefore stored in a file system and is only retrieved on demand. If the analyst wants to inspect the content of a certain netflow, the raw data of that netflow will be retrieved dynamically from the file system for deep packet inspection (implemented using Wireshark). The high level content is presented in an understandable way for the human analyst in the Content View (Section 4.5.3).

4.4 Periodicity

4.4.1 Frequency Analysis

4.4.1.1 Preliminary

We define signal f as a discrete series of indexed complex numbers $(f_0, f_1, \dots, f_{N-1})$. A periodic signal is a signal which repeats itself after some index. The energy of signal f is defined as the sum of the squared modulus of its components:

$$E_f = \sum_{n=0}^{N-1} |f_n|^2$$

4.4.1.2 Fourier Analysis

Fourier transform [95] (discrete form) is a mathematical operation which decomposes arbitrary input signals into elementary (and periodic) signals.

$$f_n = \frac{1}{N} \sum_{k=0}^{N-1} (\mathcal{F}f)_k e^{2i\pi \frac{nk}{N}} \quad (4.1)$$

In formula (4.1), the exponential components are complex periodic signals with varying frequencies. The Fourier coefficients $(\mathcal{F}f)_k$ in formula (4.1) are related to the signal via the following equation:

$$(\mathcal{F}f)_k = \sum_{n=0}^{N-1} f_n e^{-2i\pi \frac{nk}{N}}$$

Parseval's theorem of energy conservation [95] states that:

$$\sum_{n=0}^{N-1} f_n^2 = \frac{1}{N} \sum_{k=0}^{N-1} (\mathcal{F}f)_k^2$$

The meaning of this theorem is that the total energy of the original signal f is preserved (up to the scaling factor $1/N$) but redistributed by the coefficients of the transformed signal $\mathcal{F}f$. We show in Figure 4.2 that signals of different characteristics have their energy redistributed differently. In Figure 4.2, the plots of the coefficients $|(\mathcal{F}f)_k|$, called frequency spectrum, of two different input signals are shown: a periodic signal and a random signal. Two plots on the first row (Figure 4.2A and Figure 4.2B) show a simulated random signal on the left and its corresponding frequency spectrum on the right. A simulated periodic signal and its frequency spectrum are shown in Figure 4.2C and Figure 4.2D, respectively.

We can see that, in the time domain, the energies of both random signal (Figure 4.2A) and periodic signal (Figure 4.2C) are evenly distributed along the time dimension, making it very hard to analytically distinguish between them. However, in their frequency spectrums, the energy of a random signal would evenly spread across the whole frequency range, while the energy of a periodic signal tends to concentrate on a few dominant frequencies. Therefore, one way to analytically differentiate be-

Figure 4.3: Visualization model.

tween a periodic signal and a non-periodic signal is to study how their energies are distributed in the frequency spectrums. Based on this idea, we propose Algorithm 2 to measure the periodicity of a signal.

4.4.2 Periodicity Measure

We propose to use the percentage of energy occupied by the top 10% most dominant frequencies in the frequency spectrum of a signal as the measure for its periodicity (see Formula (4.2)). Due to the different ways that the signal energy is redistributed in the frequency spectrum, this measure should be high for periodic signals and low for random signals.

$$Periodicity\ Measure = \frac{E_{top\ 10\% \ frequencies}}{E_f} \quad (4.2)$$

This method for computing signal periodicity is implemented in Algorithm 2. The frequency spectrum of the input signal is first computed in line 1 using a fast implementation of Fourier transform (FFT stands for Fast Fourier Transform). In line 2, the energy distribution is constructed from the frequency spectrum and sorted in decreasing order. We then choose to use the energy percentage occupied by the top 10% strongest frequencies (line 3) as a measure for signal periodicity. This periodicity measure is computed in line 4. The highest possible value of this measure is 1 for sinusoidal signals and 0.1 for a signal which has a flat energy distribution. Using this method, the periodicity measure for the periodic signal in Figure 4.2C is 0.9059,

which is much higher than 0.3058 – the periodicity measure of the random signal in Figure 4.2A. In Section 4.6, we will use the mean (0.6059) of these two measurements as the threshold for deciding whether a signal is periodic or random.

Algorithm 2 Compute Periodicity Measure

Input: signal s **Output:** periodicity measure of signal s

- 1: $freqSpec = FFT(s)$
 - 2: $energyDist = sort(|freqSpec|)$
 - 3: $domFreqs = length(s)/10$
 - 4: $periodicity = \frac{sum(energyDist(1:domFreqs))}{sum(energyDist)}$
 - 5: return $periodicity$
-

4.4.3 Characterization of Periodic Signals

To reduce the number of alerts being generated, we cluster generated alerts into groups of similar alerts using a density-based clustering algorithm [96]. For this purpose, each periodic signal is characterized by three features: fundamental period, periodicity measure, and signal energy. The fundamental period of a periodic signal is estimated from the highest peak in its frequency spectrum. The energy of a signal is restricted to the signal components within a fundamental period.

As shown in our evaluation (Section 4.6), this clustering step turned out to not only reduce the number of alerts being presented to the analyst, but also happens to discover the different periodic services operating in a network. This discovery is reasonable as the same service, even running on different computers, tends to generate the same network traffic pattern. As a result, each cluster of alerts contains the alerts being generated by that specific service. With this capability, the analyst only has to ideally check one representational alert in a cluster in order to verify whether the whole cluster is actually malicious or not.

4.5 Visualization

The objective of the visualization is to allow the analyst to quickly assess and interpret the generated alerts. The desired workflow to verify the alerts is shown in Figure 4.3.

This workflow consists of three stages. In the first stage, the analyst would like to get alerted on suspicious periodic signals detected by the periodicity detector. In the second stage, the analyst would like to verify the time series associated with a certain alert. In the third stage, the analyst would like to examine the content of a certain netflow. The analyst may interact with the visualization in the second phase (overview verification) to adjust the parameters of the periodicity detector (for example, changing the time bin) or in the third phase (detail verification) to amend the whitelist of IP addresses.

To support this workflow, our design of the visualization correspondingly consists of three consecutive panels shown in Figure 4.4. The left panel (panel A) presents the alerts on periodic signals automatically detected by the periodicity detector. These alerts serve as starting points for further analysis by the analyst. The center panel (panel B) is an interactive visualization that shows the time series associated with the chosen alerts. In panel B, the analyst is allowed to try out different time bin size or various choices of temporal parameters and value parameters. The found optimal time bin can be used to replace the currently used time bin of the periodicity detector. The right panel (panel C) shows the high level content of the network flow selected by the analyst in the center section.

4.5.1 The Left Panel - Alert View (Panel A)

The left panel displays the alerts returned by the periodicity detector. These alerts provide the analyst with starting points for further analysis of periodic signals hidden in network traffic. To group together similar alerts, which are likely to be generated by the same service, we use the density-based clustering algorithm [96]. The clustering is based on three features: fundamental period, periodicity measure, and signal energy. The alerts belonging to the same group are coded and visualized with the same color (Figure 4.5A).

The purpose of the clustering step is to save analysis time as it is inefficient to check every alert generated by the same service. The clustering result enables the analyst

Figure 4.4: Visualization Dashboard. Panel A shows the alerts on periodic signals. Panel B shows the time series associated with a particular alert. Panel C shows the content of a particular netflow.

to check only one representational alert and generalize the result to other alerts in the same cluster. If the analyst chooses to inspect one particular alert visualized on the left panel, the associated time series will be accordingly visualized on the second panel (Section 4.5.2). The center panel deals with metadata of the netflows and allows verifying the overview of the alerts whereas the right panel works with the actual content of relevant netflows and allows verifying the details (Section 4.5.3).

In addition, to allow temporal comparison between different alerts, we use a circular graph to visualize the time series associated with the selected alerts (Figure 4.5B). Circular graphs are often used to visualize periodic time series, highlighting their periodicity characteristic. In the work of Müller et al. [97], the authors proposed to use a variant of the circular graph, called spiral graph, to accentuate the periodic patterns (such as sunrise, sunset or cloudy periods) visible in the time series of sunshine intensity. In our work, we visualize many time series on the same graph. Each time series is quantized using a color coding scale. The time scale is mapped to the angle of the circular graph. The outer rings in the circular graph are associated to

the most recent alerts. Figure 4.5B shows the six alerts generated by three different services, Windows Updates, RSS Reader and a sample of trojan Dyre. The circular graph clearly shows the similarly looking time-series generated by the same service, which the clustering plot in Figure 4.5A fails to highlight.

4.5.2 The Center Panel - Time Series View (Panel B)

The center panel is a dynamic visualization of the time series associated with the alert chosen in the left panel. The center panel also allows the analyst to perform exploratory analysis on the netflows to manually discover periodic patterns. The center panel consists of three main components: a search textbox, a context histogram and a focus histogram. These components work closely together to support three tightly-coupled functionalities: filtering out netflows, visualizing the filtered netflows and inspecting particular netflow.

4.5.2.1 Netflow Filtering

The interesting netflows can be filtered by specifying the following five features in the search textbox: source IP address, source port, destination IP address, destination port and the protocol. These are the features which fully characterize a netflow. However, due the flexibility offered by an Elastic Search⁵ backend, the analysts are not restricted to these features. They are also allowed to perform filtering using other features such as the timestamp of the netflows or their size. The filtered netflows automatically populate the context histogram for the overview visualization.

4.5.2.2 Stacked Histogram of Netflows

To construct a histogram of the filtered netflows, the analyst has to specify two parameters: the type of the timestamp used for the binning process and the value parameter to aggregate the netflows.

Timestamp selection. For each netflow, there are two types of timestamps that mark its occurrences: the arrival time of the last packet and the arrival time of the

⁵<https://www.elastic.co>

Figure 4.5: Alert processing. A) The clustering result of six alerts generated by three programs: a sample of trojan Dyre, RSS Feed Reader and Windows Update Service, based on period and periodicity estimates. B) Circular graph of the time series associated with these alerts - outermost two rings (a sample of trojan Dyre), innermost ring (RSS Feed Reader) and the rest three rings (Windows Update Service). The time unit is in minutes.

first packet. The analyst is allowed to choose either one of these two timestamps for the time binning process.

Value parameter selection. The chosen time stamp type is used to group close netflows together but the heights of the bars in the histogram are determined by the value parameter to be selected. The analyst is allowed to choose from various value parameters such as the number of packets, netflow size, or average packet size. The scale of the histogram can also be switched between logarithm and linearity to deal with situations showing a large difference between binned netflow groups.

4.5.2.3 Drill-down Analysis

As it is often difficult to visualize a large number of netflows using just one histogram, we decide to use two histograms connected via a brushing mechanism (Figure 4.6B and Figure 4.6C). The upper histogram shows the overview of the time series, associated with the selected alert, whereas the lower histogram shows the focus view. The context histogram visualizes all filtered netflows in which the analyst can select and brush a section of interest. The focus histogram visualizes the netflows in the selected section (Figure 4.6C).

The bin size of the context histogram is preset to 5 seconds, while the bin size of

the focus histogram is adjustable. The research on the optimal number of bins to best analyze a time series is well established. For our prototype, we provide the analyst with a list of automatically computed “optimal” numbers of bins using the following formulas: Sturges’ formula, Rice Rule, Doane’s formula, Scott’s normal reference rule, and Freedman-Diaconis’ choice⁶. On the other hand, the analyst is also allowed to adjust the bin size manually in order to adapt to particular time series.

In each bin of the histogram, the netflows are individually visualized as vertical bars, which are stacked on one another in the order of the timestamps of the netflows. The bars, whose corresponding netflows have the same external IP address, external port, internal IP address and protocol, are encoded by the same color (Figure 4.6). The internal port is omitted here because it tends to change every time the application makes a new connection. When the analyst clicks on a bar, the deep packet inspection result of that netflow is dynamically computed and displayed on the right panel for the analyst’s inspection.

4.5.3 The Right Panel - Detail on Demand (Panel C)

The previous two panels, panel A and panel B, only deal with the metadata of the netflows and help to discover periodic signals hidden in network traffic. The right panel, on the other hand, works with the actual content of the netflow. This functionality is possible thanks to the deep packet inspection capability provided by Wireshark. This operation is only carried out on demand by the analyst to reduce the computation load of the whole system. Basically, it combines the payloads of the relevant raw packets and constructs the high level content understandably to human analyst.

This functionality is critical in operational settings because the analyst requires contextual understanding about the network in order to correctly verify the truthfulness of the alerts. For example, as described in Section 4.6, a cluster of periodic signals of period 5 minutes is discovered in a network traffic by the periodicity detector. Upon inspection of netflow content, it turns out that the relevant traffic is

⁶<https://en.wikipedia.org/w/index.php?title=Histogram>

Figure 4.6: Histogram of netflows. A) The overall random traffic. B) The traffic generated by a sample of trojan Dyre. C) A section of interest of Figure B, selected via the brushing mechanism.

actually generated by a benign Chrome RSS Feeder extension. However, it is also discovered that another cluster of periodic signals is present in the same traffic but generated by trojan Dyre of period 25 minutes. In this case, the high level content of the netflows is required by the analyst to verify whether some periodic signal is benign or malicious.

4.6 Evaluation

We performed a qualitative evaluation of the two main capabilities of the system: the ability of the periodicity detector to correctly generate alerts on periodic patterns of any complexity and the convenience provided by the visualization (Panels A, B and C) to help assess and evaluate the alerts. The system is evaluated through two scenarios. The first scenario is a simulated scenario to demonstrate the two main capabilities of the system. The second scenario is a real scenario to assess the usefulness of the system in practice.

4.6.1 Simulated Scenario

In this scenario, we set up a small network consisting of three Windows-based computers, A, B and C. We infected computer A and computer C with trojan Dyre⁷. On computer A and computer B, we extended the installed Chrome web browsers with a plug-in called RSS Feed Reader. We then turned on Windows Update Service on computer B and computer C while turning off this service on computer A. As trojan Dyre, RSS Feed Reader and Windows Update Service are known to leave periodic traces in network traffic, we expect that the system should be able to detect the periodic signals generated by them.

The experiment lasted 8 hours and the captured network traffic was mixed with the realistic background user traffic provided by the DARPA 1999 dataset [98]. As we can see, the context histogram of the overall network traffic (Figure 4.6A) looks highly random and does not exhibit any interesting patterns. However, the periodicity

⁷<http://blogs.cisco.com/security/talos/threat-spotlight-dyre>

Figure 4.7: The content of 4 netflows generated by trojan Dyre.

detection algorithm (Algorithm 2) is able to detect three clusters of periodic signals (Panel A of Figure 4.4). This result validates the first capability of the system. We are going to further investigate these three clusters of periodic signals using the visualization.

Among the three notified alert clusters, the 1st cluster is the most interesting one. This cluster contains two different alerts and its associated context histogram (Figure 4.6B) shows a rather complicated periodic pattern generated by an unknown service. As the purpose of the context histogram is to visualize the overview of the alert, the details of the associated netflows are missing. By brushing a particular region, we can investigate the relevant netflows in detail using the focus histogram (Figure 4.6C).

By using Panel C to inspect the bars on the focus histogram, we can see that the contents of the represented netflows are almost identical and only differ on the queried domain names (Figure 4.7). This observation is indicative of the beaconing activity of a DGA-based malware. In fact, all these domain names are generated due to the periodic beaconing activity of trojan Dyre, which is trying to query automatically

generated domains for connections. The analyst can take action by isolating the infected machine and performing deep scanning.

For the 2nd and the 3rd alert clusters, the analyst carries out the same steps to evaluate the alerts. Unlike the 1st alert cluster, by inspecting the content, the netflows associated with the 2nd and the 3rd alert clusters are found to be generated due to periodic http requests sent by the Chrome RSS Feed Readers (Figure 4.8A) and Windows Update Service (Figure 4.8B). These two services resolve domain names windows.microsoft.com and newsimg.bbc.co.com, which can be added to the white list of the periodicity detector to avoid generating the same false positive alerts.

4.6.2 Real Scenario

In this scenario, we are going to analyze the botnet dataset provided by the University of New Brunswick [93]. This dataset is the combination of a few other botnet datasets and contains botnet traffic generated by 16 different malware families such as Zeus, Rbot or ZeroAccess. We use the test set of this dataset to evaluate the system as the test set contains the traffic of all 16 included botnets, and hence, is more suitable to evaluate the detection capability of the system. The test set consists of 177,278 netflows and involves 27,891 unique IP addresses. The analysis result is that the system is able to generate 33 different alert clusters associated with different internal and external IP addresses.

Following the same steps as in the simulated scenario, we are able to find out that 26 of 33 generated alert clusters are actually generated by benign applications such as Windows Update Service, browser refreshers, and others. The remaining seven alert clusters are generated by unknown services. By cross-checking with the ground truth of the test set, all these seven alert clusters are verified to be generated by different malware families. Table 4.1 shows the duration and periodicity measures of all 16 malware samples. The true positive rate of the system is 21.21% and 9 of 16 malware samples were not detected by the system.

Figure 4.8: Netflows by Windows Update Service (A) and RSS Feed Reader (B).

4.7 Discussions

In this section, we discuss the effectiveness of the periodicity detector as well as the interactive visualization used to verify detection result. We also discuss the limitations of the approach.

While the system works perfectly well in the simulated scenario, there are a few problems only encountered in the real scenario. The true positive rate of the system is 21.21%, which is a relatively low rate. This is due to the fact that there exists a lot of benign and periodically acting applications contained in the dataset. As the result, the system would generate many false positive alerts. This is the reason why we need a good visualization to couple with the periodicity detection algorithm in the first place. This issue can be effectively dealt with by the interactive visualization.

In addition, there are 9 of 16 malware samples which go undetected. Close inspection of these cases reveals the reason for this result: the network traffic generated by these failed cases are too short for the periodic patterns to emerge. For example, two samples of the Zeus malware are only recorded for 30 seconds. On the other hand, the malware samples which are recorded for long enough time, such as Weaset (2,839 minutes) or Virut (977 minutes), all generate periodic traffic. Therefore, this method

Table 4.1: Periodicity of malware samples.

IP Address	Label	Duration	Periodicity
74.78.117.238	Weasel	2839 mins	0.7532
147.32.84.130	Murlo	1162 mins	0.6752
147.32.84.160	Virut	977 mins	0.7231
147.32.84.180	Neris	201 mins	0.6980
147.32.84.150	Menti	120 mins	0.6303
147.32.84.170	RBot	36 mins	0.6520
172.29.0.109	Osx_trojan	25 mins	0.5421
172.16.253.130	TBot	21 mins	0.2340
192.168.248.165	Zeroaccess 2	16 mins	0.6732
147.32.84.140	Sogou	10 mins	0.4309
10.37.130.4	Smoke	7 mins	0.2130
192.168.3.35	Zeus 1	3 mins	0.3340
10.0.2.15	IRCbot	2 mins	0.4351
172.16.253.132	Zeroaccess 1	2 mins	0.3970
192.168.3.25	Zeus 2	30 secs	0.3467
192.168.3.65	Zeus 3	30 secs	0.2875

of periodicity-based intrusion detection is only effective if the malware is present in the network long enough for the periodic patterns to show up.

The purpose of network intrusion detection is to check millions of network packets for malicious intent. Towards this goal, the periodicity detector reduces the working load to a handful of alerts on periodically behaved IP addresses, which are directly visualized on Panel A. In Panel B, some of the alerts would be further discarded upon the inspection of the time-series associated with each alert. The periodicity detector, Panel A and Panel B only deal with metadata of the netflows and it is sometimes not obvious to tell whether an indicator of compromise is actually malicious. This issue is tackled by Panel C, which shows the content of the netflows - the concrete evidences of intrusion.

There are two ways by which this system can be evaded by attackers in practice. The first way is the introduction of randomness to the interval that the infected machine is supposed to connect to the remote server. However, in this case, this evasion technique would introduce more complexity and unpredictability to the mechanism that the attacker manages the infected machines. It is common in intrusion detection

research that the introduction of new detection methods is not to solve the problem of intrusion detection completely but to make it harder and harder for attackers to succeed. This phenomenon is referred to as the pyramid of pain in operational settings⁸. The second way is the use of encryption. If the attacker decides to use encryption, Panel C would fail to show the content of interesting netflows. However, the system is still useful in this case as the reported suspicious entity would become even more suspicious because of the encryption and should be closely monitored for additional evidence of compromise.

The performance overhead of the method lies in the computation of the Discrete Fourier Transform and the clustering algorithm used to cluster the alerts. The complexity of Discrete Fourier Transform using split-radix algorithm⁹ is $m \log(m)$ in which m is the length of the signal. The complexity of the DBSCAN is n^2 in which n is the number of alerts triggered by the system.

4.8 Summary

We have described the details and evaluation of a system consisting of an automatic periodicity detection algorithm and an interactive visualization to address the problem of periodicity-based intrusion detection. The detection of hidden periodic signals is carried out by a Fourier transform-based periodicity detector. The detected periodic signals are verified against the content of relevant netflows with the help of deep packet inspection capability, which combines separate packets together to form the high level content, understandable to human analysts. The verification is enabled by an interactive dashboard with several panels to show the netflows on different levels of detail. Furthermore, the dashboard also allows analysts to perform exploratory analysis on network traffic to manually look for hidden periodic signals.

The basic assumption of this work is the detectable periodic behavior exhibited by malware in network traffic. As shown in Section 6.4.2, although this assumption may not hold if the observation time is too short for the periodic patterns to emerge,

⁸<http://detect-respond.blogspot.de/2013/03/the-pyramid-ofpain.html>

⁹https://en.wikipedia.org/wiki/Split-radix_FFT_algorithm

the periodic patterns are clearly visible in the cases that the observation time is long enough. As for robustness, the system can be overcome by malware that randomizes the connection interval or uses encryption to hide the malicious content. However, in both cases the system does help to raise the bar for intrusion attempts by attackers.

5

Predicting the Risk Level of Executables

Nowadays, the number of new malware samples discovered every day is in millions, which undermines the effectiveness of the traditional signature-based approach towards malware detection. To address this problem, machine learning methods have become an attractive and almost imperative solution. In most of the previous work, the application of machine learning to this problem is batch learning. Due to its fixed setting during the learning phase, batch learning often results in low detection accuracy when encountered zero-day samples with obfuscated appearance or unseen behavior. Therefore, in this chapter, we propose a new online algorithm, called FTRL-DP, to address the problem of malware detection under concept drift when the behavior of malware changes over time. We argue that online learning is more appropriate in this case than batch learning since online algorithms are inherently resilient to the shift in the underlying distribution. The experimental results show that online learning outperforms batch learning in all settings, either with or without retrainings. This work is the **Contribution 2** of the thesis.

5.1 Introduction

VirusTotal.com is an online service which analyzes files and urls for malicious content such as virus, worm and trojan by leveraging on an array of 52 commercial antivirus solutions for the detection of malicious signatures. On record, VirusTotal receives and analyzes nearly 2 million files every day. However, only a fraction of this amount (15%) can be identified as malicious by at least one antivirus solution. Given the fact that it is fairly easy nowadays to obfuscate a malware executable [5], it is rather reasonable to believe that a sheer number of the unknown files are actually obfuscated malware samples. In principle, the rest of the unknown cases should be manually reverse engineered to invent new signatures, but this is infeasible due to the large number of files to be analyzed. Therefore, looking for an automated way to address this problem is imperative and has attracted a lot of research effort, especially in the direction of using machine learning which has gained a lot of successes in various domains of pattern recognition such as face analysis [99] and sentiment analysis [100].

In a recent paper, Saxe et al. [10] train a 3-layer neural network to distinguish between malicious and benign executables. In the first experiment, the author randomly splits the whole malware collection into train set and test set. The trained network can achieve a relatively high detection accuracy of 95.2% at 0.1% false positive rate. It is noted that the first experiment disregards the release time of the executables, which is an important dimension due to the adversarial nature of malware detection practice since malware authors are known to regularly change their tactics in order to stay ahead of the game [101]. In the second experiment, the author uses a timestamp to divide the whole malware collection into train set and test set. The results obtained show that the detection accuracy drops to 67.7% at the same false positive rate. We hypothesize that the reasons for this result are two-fold: the change in behavior of malware over time and the poor adaptation of neural network trained under batch mode to the behavioral changes of malware. In addition, another recent study also reports similar findings when the release time of the executables is taken into

account [77].

The working mechanism of batch learning is the assumption that, the samples are independently and identically drawn from the same distribution (iid assumption). This assumption may be true in domains such as face recognition and sentiment analysis where the underlying concept of interest hardly changes over time. However, in various domains of computer security such as spam detection and malware detection, this assumption may not hold [54] due to the inherently adversarial nature of cyber attackers, who may constantly change their strategy so as to maximize the gains. To address this problem of concept drift, we believe online learning is a more appropriate solution than batch learning. The reason is that, online algorithms are derived from a theoretical framework [75] which does not impose the iid assumption on the data, and hence can work well under concept drift or adversarial settings [102]. Motivated by this knowledge, we propose the Follow-the-Regularized-Leader with Decaying Proximal (FTRL-DP) algorithm – a variant of the proximal Follow-the-Regularized-Leader (FTRL-Proximal) algorithm [76] – to address the problem of malware detection.

To be specific, the contributions of this chapter are as follow:

- A new online algorithm (FTRL-DP) to address the problem of concept drift in Windows malware detection. Our main claim is that online learning is superior to batch learning in the task of malware detection. This claim is substantiated in Section 5.6 by analyzing the accuracy as well as the running time of FTRL-DP, FTRL-Proximal and Logistic Regression (LR). The choices of the algorithms are clarified in Section 5.3.
- An extensive data collection of more than 100k malware samples using the state-of-the-art open source malware analyzer, Cuckoo sandbox [14], for the evaluation. The collected data comprises many types such as system calls, file operations, and others, from which we were able to extract 482 features. The experiment setup for data collection and the feature extraction process are described in Section 5.4.

For LR, the samples in one month constitutes the test set and the samples in a number of preceding months are used to form the train set; for FTRL-DP, a sample is used for training right after it is tested. The detailed procedure is presented in Section 5.5. We formulate the problem of malware detection as a regression problem in Section 5.2. We summarize the chapter in Section 5.7.

5.2 Problem Setting

We have framed the problem of malware detection as the regression problem of predicting the risk level of an executable. We rely on the labels provided by all 52 antivirus solutions and do not follow the labels provided by any single one. The reason for this choice is that antivirus solutions are known to report inconsistent labels [103]. In addition, we also do not use two different thresholds to separate the executables into two classes, malicious and benign, as in [10] since it would discard the hard cases where it is difficult to determine the nature of the executables, which may be of high value in practice.

Specifically, given 52 antivirus solutions, we address the problem of predicting the percentage of solutions that would flag an executable file as malicious. Formally, this is a regression problem of predicting the output $y \in [0, 1]$ based on the input $x \in R^n$ which is the set of 482 hand-crafted features extracted from the reports provided by the Cuckoo sandbox (Section 5.4). The semantic of the output defined in this way can be thought of as the risk level of an executable. We augment the input with a constant feature which always has the value 1 to simulate the effect of a bias. In total, we have 483 features for each malware sample.

5.3 Methodology

The basic idea of the application of machine learning to malware detection is the learning of recurrent patterns in malware behavior exhibited in the past so as to make predictions about the samples at present. This methodology is usually referred to as supervised learning, in which the purpose is to learn a function which can correctly

map an input x to an output y . We are particularly interested in two ways that this mapping function can be learned: batch learning and online learning. In batch learning, the mapping function is constructed once using the bulk of training data, which is assumed to be available all at once. On the other hand, in online learning, the mapping function is updated frequently based on the most recent training samples.

To allow a fair comparison between batch learning and online learning, we use the models of the same linear form, represented by a weight vector w , in both cases. The sigmoid function ($\frac{1}{1+e^{-z}}$) is then used to map the dot product (biased by the introduction of a constant feature) between the weight vector and the input, $w^\top x$, to the $[0,1]$ interval of possible risk levels. Additionally, in both cases, we optimize the same objective function, which is the sum of logistic loss (log loss – the summation term in Equation 5.1). In the batch learning setting, the sum of log loss is optimized in a batch manner in which each training example is visited multiple times in minimizing the objective function. The resultant algorithm is usually referred to as Logistic Regression with log loss (Section 5.3.1). On the other hand, in the online setting, we optimize the sum of log loss in an online manner, in which each training sample is only seen once. The resultant algorithm is the proposed FTRL-DP algorithm (Section 5.3.2).

5.3.1 Batch Learning – LR

Given a set of n training examples $\{(x_i, y_i)\}_{i=1}^n$, Logistic Regression with log loss corresponds to the following optimization problem:

$$\underset{w}{\operatorname{argmin}} \left\{ - \sum_{i=1}^n (y_i \log(p_i) + (1 - y_i) \log(1 - p_i)) + \lambda_1 \|w\|_1 + \frac{1}{2} \lambda_2 \|w\|_2^2 \right\} \quad (5.1)$$

in which $p_i = \operatorname{sigmoid}(w^\top x_i)$

The objective function of Logistic Regression (Equation 5.1) is a convex function with respect to w as it is the sum of three convex terms. The first term is the sum of log losses associated with all training samples (within a time window). The last two terms are the L1–norm regularizer and the L2–norm regularizer. The L1 regularizer

is a non-smoothed function used to introduce sparsity into the solution weight w . On the other hand, the L2 regularizer is a smooth function used to favor low variance models that have small weight.

The fundamental assumption of batch learning is that, the distribution of the train set must be similar to the distribution of the test set (iid assumption). This is to ensure that the weight learned from the train set is applicable to the test set. As a consequence, this method is quite successful in stationary domains such as object recognition or sentiment analysis, but may fail in malware detection due to the change in behavior of malware over time [54]. This is the main motivation of this chapter to explore the framework of adaptive online learning [75] for deriving better algorithms to address this problem of concept drift.

5.3.2 Online Learning – FTRL-DP

5.3.2.1 Online Convex Optimization.

The general framework of online convex optimization can be formulated as follows [104]. We need to design an algorithm that can make a series of optimal predictions, each at one time step. At time step t , the algorithm makes a prediction, which is a weight vector w_t . A convex loss function $l_t(w)$ is then exposed to the algorithm after the prediction. Finally, the algorithm suffers a loss of $l_t(w_t)$ at the end of time step t (Algorithm 3). The algorithm should be able to learn from the losses in the past so as to make better and better decisions over time.

Algorithm 3 Online Algorithm

```
1: for  $t = 1, 2, \dots$  do  
2:   Make a prediction  $w_t$   
3:   Receive the loss function  $l_t(w)$   
4:   Suffer the loss  $l_t(w_t)$ 
```

The objective of online convex optimization is to minimize the regret with respect to the best classifier in hindsight (Equation 5.2). The meaning of Equation 5.2 is that we would like to minimize the total loss incurred up to time t with respect to the supposed loss incurred by the best possible prediction in hindsight, w^* .

$$\mathbf{Regret}_t = \sum_{s=1}^t l_s(w_s) - \sum_{s=1}^t l_s(w^*) \quad (5.2)$$

Since the future loss functions are unknown, the best guess or the greedy approach to achieve the objective of minimizing the regret is to use the prediction that incurs the least total loss on all past rounds. This approach is called Follow-the-Leader (FTL), in which the leader is the best prediction that incurs the least total loss with respect to all the past loss functions. In some cases, this simple formulation may result in algorithms with undesirable properties such as rapid change in the prediction [104], which lead to overall high regret. To fix this problem, some regularization function is usually added to regularize the prediction. The second approach is called Follow-the-Regularized-Leader (FTRL), which is formalized in Equation 5.3.

$$w_{t+1} = \underset{w}{\operatorname{argmin}} \left\{ \sum_{i=1}^t l_i(w) + r(w) \right\} \quad (5.3)$$

It is notable to see that the FTRL framework is formulated in a rather general sense and performs learning without relying on the iid assumption. This property makes it more suitable to adversarial settings or settings in which the concept drift problem is present.

5.3.2.2 The Proposed FTRL-DP Algorithm.

In the context of FTRL-DP, an online classification or an online regression problem can be cast as an online convex optimization problem as follows. At time t , the algorithm receives input x_t and makes prediction w_t . The true value y_t is then revealed to the algorithm after the prediction. The loss function $l_t(w)$ associated with time t is defined in terms of x_t and y_t (Equation 5.4). Finally, the cost incurred at the end of time t is $l_t(w_t)$. The underlying optimization problem of FTRL-DP is shown in Equation 5.5.

$$l_t(w) = -y_t \log(p) - (1 - y_t) \log(1 - p) \quad (5.4)$$

in which $p = \operatorname{sigmoid}(w^\top x_t)$

Compared with Equation 5.3, Equation 5.5 has the actual loss function $l_t(w)$

replaced by its linear approximation at w_t , which is $l_t(w_t) + \nabla l_t(w_t)^\top (w - w_t) = g_t^\top w + l_t(w_t) - g_t^\top w_t$ (in which $g_t = \nabla l_t$). The constant term $(l_t(w_t) - g_t^\top w_t)$ is omitted in the final equation without affecting the optimization problem. This approximation is to allow the derivation of a closed-form solution to the optimization problem at each time step, which is not possible with the original problem in Equation 5.3.

$$w_{t+1} = \underset{w}{\operatorname{argmin}} \left\{ g_{1:t}^\top w + \lambda_1 \|w\|_1 + \frac{1}{2} \lambda_2 \|w\|_2^2 + \frac{1}{2} \lambda_p \sum_{s=1}^t \sigma_{t,s} \|w - w_s\|_2^2 \right\} \quad (5.5)$$

in which $g_{1:t}^\top = \sum_{i=1}^t g_i^\top$

FTRL-DP utilizes 3 different regularizers to serve 3 different purposes. The first two regularizers of L1-norm and L2-norm serve the same purpose as in the case of Logistic Regression introduced in Section 5.3.1. The third regularization function is the proximal term used to ensure that the current solution does not deviate too much from past solutions with more influence given to most recent ones by using an exponential decaying function ($\sigma_{t,s} = \gamma^{t-s}$ with $1 > \gamma > 0$). This is our main difference from the original FTRL-Proximal algorithm [76] and its recently proposed variant [24]. The replacement of the per coordinate learning rate schedule by the decaying function proves to improve the prediction accuracy in the face of concept drift (discussed in Section 5.6).

The optimization problem in Equation 5.5 can be efficiently solved in closed form (Equation 5.6). Please refer to Section 6.3 in Chapter 6 for an in-depth analysis and extension.

$$w_{t+1,i} = \begin{cases} 0 & \text{if } \|z_{t,i}\|_1 \leq \lambda_1 \\ -\frac{z_{t,i} - \lambda_1 \operatorname{sign}(z_{t,i})}{\lambda_2 + \lambda_p \frac{1-\gamma^t}{1-\gamma}} & \text{otherwise.} \end{cases} \quad (5.6)$$

in which $z_t = g_{1:t} - \lambda_p \sum_{s=1}^t \sigma_{t,s} w_s$

In summary, we aim to compare between the performance of batch learning and online learning on the problem of malware detection. To make all things equal, we

Figure 5.1: Distribution of executables through time.

use the models of the same linear form and optimize the same log loss function, which lead to the LR algorithm in the batch learning case and the FTRL-DP algorithm in the online learning case. For LR, only the samples within a certain time window contribute to the objective function (Equation 5.1). On the other hand, the losses associated with all previous samples equally contribute to the objective function of FTRL-DP (Equation 5.5). This difference is critical as it leads to the gains in the performance of FTRL-DP over LR, which is discussed in Section 5.6.

5.4 Data Collection

5.4.1 Malware Collection

We used more than 1 million files collected in the duration from Nov/2010 to Jul/2014 by VirusShare.com for the experiments. VirusShare is an online malware analyzing service that allows Internet users to scan arbitrary files against an array of 52 antivirus solutions (the scan results are actually provided by VirusTotal). In this study, we are only interested in executable files and able to separate out more than 100k executables from the 1 million files downloaded.

Figure 5.1 shows the distribution of the executables with respect to executables' compile time. The horizontal axis of Figure 5.1 shows the months during the 4 years from Nov/2010 until Jul/2014, which is the period of most concentration of executables and chosen for the study. The vertical axis of Figure 5.1 indicates the number

of executables compiled during the corresponding month. Figure 5.2, instead, shows the risk level distribution of the executables. The horizontal axis indicates the maliciousness measure and the vertical axis the number of corresponding executables.

5.4.2 Malware Execution

Due to the large number of samples to be executed, we have to make use of some kind of the parallelism to cut down the time required to conduct the experiments. For this requirement, we use the Cuckoo sandbox, which is an open source malware analysis sandbox. Cuckoo sandbox supports both physical and virtual sandbox machines such as VirtualBox, VMWare and KVM. Since it is infeasible to execute this large number of samples on physical environments, we decided to execute them on an array of VirtualBox virtual machines.

We make use of the facility provided by DeterLab [105] as the testbed for the execution of the executables. DeterLab is a flexible online experimental lab for computer security, which provides researchers with a host of physical machines to carry out experiments. In our setup, we use 25 physical machines with each physical machine running 5 virtual machines for executing the executables. Each executable is allowed to run for 1 minute. The experiment ran for more than 20 days and collected the behavioral data of roughly 100k executables.

5.4.3 Feature Extraction

There are basically two different ways that we can extract the features that characterize an executable depending on whether the executable is executed or not. In the first method, the executable is not executed and features are extracted directly from the file. This method is called static analysis. The advantage of this approach is the relatively short amount of time that it requires to extract the features but its disadvantage being that it is vulnerable to obfuscation techniques [5].

On the other hand, dynamic analysis is usually robust to obfuscation techniques. In this method, the executables are actually executed in a protected environment to collect generated artifacts such as network traffic, dropped files, and created registries.

Figure 5.2: Distribution of risk level.

Dynamic features can then be extracted from the generated artifacts for further study such as classification and regression. The downside of dynamic analysis is the tedious task of executing the malware and the timespan required to do so. However, this disadvantage can be effectively overcome by paralleling the experimental process, and analysis sandboxes such as Cuckoo supports the required features.

In this chapter, we mostly consider dynamic features of the following 4 categories for regression: file system category, registry category, system call category, and the category of other miscellaneous features.

We do not focus on engineering novel features to increase the detection accuracy but perform the comparative analysis between batch learning and online learning. The features that we used were mostly influenced by the recent work of Korkmaz et al [12] who also used Cuckoo sandbox as the framework for data collection. All these features can be easily derived from the reports by the Cuckoo sandbox and will be described in details as follows.

5.4.3.1 API Call Category

API (Application Programming Interface) calls are the functions provided by the operating system to grant application programs the access to basic functionality such as disk read and process control. Although these calls may ease the process of manipulating the resources of the machine, it also provides hackers with a lot of opportunities

to obtain confidential information. For this category, we consider the invoking frequencies of the API calls as a set of features. In addition, we also extract as features the frequencies that the API files are linked. The total number of features in this category is 353 and the complete set of API calls as well as the set of API files are available at <https://git.io/vDywd>.

5.4.3.2 Registry Category.

In Windows environments, the registry is a hierarchal database that holds the global configuration of operating system. Ordinary programs often use the registry to store information such as program location and program settings. Therefore, the registry system is like a gold mine of information for malicious programs, which may refer to it for information such as the location of the local browsers or the version of the host operating system. Malicious program may also add keys to the registry so as to be able to survive multiple system restarts. We extract the following 4 registry related features: the number of registries being written, opened, read and deleted.

5.4.3.3 File System Category.

File system is the organization of the data that an operating system manages. It includes two basic components: file and directory. File system-related features are an important set of features to consider since malware has to deal with the file system in one way or another in order to cause harm to the system or to steal confidential information. We consider the following file-related features: the number of files being opened, written, in existence, moved, read, deleted, failed and copied. In addition, we also consider the following 3 directory related features: the number of directories being enumerated, created and removed. In total, we were able to extract 11 features in this category.

5.4.3.4 Miscellaneous Category.

In addition to out-of-the-box functionalities, Cuckoo sandbox is further enhanced by a collection of signatures contributed by the public community. These signatures can

identify certain characteristics of the analyzed binary such as the execution delay time or the ability to detect virtual environment. All these characteristics are good indicators for the high risk level of an executable but may just be false positives. We consider the binary features of whether the community signatures are triggered or not. In addition, we also consider 3 other features that may be relevant to the behavior characterization: the number of mutex created, the number of processes started and the depth of the process tree. The total number of features being extracted in this category is 118.

In summary, we are able to extract 482 features that spans 4 different categories: API calls, registry system, file system and miscellaneous features.

5.5 Evaluation

By the analysis in Section 5.3, Logistic Regression and the family of FTRL algorithms share a similar mathematical formulation. There's a slight difference in the examples to be included in the objective functions. In Logistic Regression, only the examples within a certain time window contribute to the objective function, which is solved via Statistical Gradient Descent. On the other hand, FTRL algorithms take into account all previous learning examples. Linear approximation of the cross entropy function is required to allow the derivation of closed form solution.

In this section, we additionally evaluate Support Vector Machine (SVM) [106] and two state-of-the-art FTRL algorithms: FTRL-Proximal [76] and Time-decaying Adaptive Prediction algorithms [24]. LR and SVM are two representations of batch learning paradigm while FTRL-DP, FTRL-Proximal and TDAP are three representations of online learning paradigm.

5.5.1 Experiment with LR and SVM

We evaluate LR and SVM in four different settings: once, multi-once, monthly, and multi-monthly. In the once setting, the samples appeared in the first month of the whole dataset are used to form the train set and the rest of the samples are used to

form the test set. The multi-once setting is similar to the once setting except that the samples in the first 6 months are used to form the train set instead. It should be noted that retraining is not involved in the first two settings.

On the other hand, the other two settings do involve retraining, which is a crude mechanism to address the change in behavior of malware over time. Since it is infeasible to carry out retraining upon the arrival of every new sample, we perform retraining on a monthly basis. Due to the characteristic of our dataset, we find that the monthly basis is a good balance to ensure that we have enough samples for the train set and the training time is not too long (the monthly average number of samples is 2.4k). In the monthly setting, we use the samples released in a month to form the test set and the samples released in the immediately preceding month to form the train set. The multi-monthly setting is similar to the monthly setting except that we use the samples in the preceding 6 months to form the train set instead.

For a quick evaluation, we make use of the LR and SVM implementation, provided by the TensorFlow library [107] to train and test the LR and the SVM regressors. TensorFlow is a framework that leverages high performing GPUs for training large scale neural networks. We use two regularizations of L1-norm and L2-norm to regulate the weights. 20% of each train set is dedicated for validation and the maximum number of epochs that we use is 100. We stop the training early if the validation loss does not get improved in 3 consecutive epochs.

5.5.2 Experiment with FTRL algorithms

We use the standard procedure to evaluate FTRL-Proximal, TDAP, and FTRL-DP (jointly referred to as FTRL algorithms). Each new sample is tested on the current model giving rise to an error, which is then used to make modification to the current model right after. This evaluation is usually referred to as the mistake-bound model.

Due to their simplicity, FTRL-Proximal, TDAP, and FTRL-DP can be implemented in not more than 40 lines of python code each. The implementation makes heavy use of the numpy library, which is mostly written in C++. As TensorFlow also

has C++ code under the hood, we believe that the running time comparison between the two cases is sensible. Evaluated on the same computer, it actually turns out that the running time of FTRL algorithms is much lower than that of LR. We use the same amounts of three regularizations for the three FTRL algorithms. For TDAP and FTRL-DP, we report the best possible setting for hyperparameter γ .

The computer used for all the experiments has 16GB RAM and operates with a 1.2GHz hexa-core CPU. The running times of all experiments are shown in Table 5.1. The mean cumulative absolute errors are reported in Figure 5.3.

5.6 Discussion

5.6.1 Prediction Accuracy

We use the mean cumulative absolute error (MCAE) to compare the performance between FTRL-Proximal, TDAP, and FTRL-DP and different batch settings of LR and SVM, which are reported in Figure 5.3. The MCAE is defined in Equation 5.7, in which y_t is the actual risk level of an executable and p_t the risk level predicted by the algorithms. In Figure 5.3, the horizontal line shows the cumulative number of samples and the vertical line the MCAE. There are four notable observations that we can see from Figure 5.3.

$$\frac{1}{n} \sum_{t=1}^n |y_t - p_t| \quad (5.7)$$

Firstly, the more data that we train the batch model (LR and SVM) on, the better performance we can achieve. This observation is evidenced by the fact that, in most of the time, the error line of the multi-once setting stays below the error line of the once setting, and the error line of the multi-monthly setting stays below the error line of the monthly setting. A possible explanation for this observation is that the further we go back in time to obtain more data to train the model on, the less variance the model becomes, which results in the robustness to noise, and consequently, higher prediction accuracy.

Secondly, the retraining procedure does help to improve prediction accuracy. It

Figure 5.3: Mean cumulative absolute errors of of FTRL-DP, FTRL-Proximal and different settings of LR.

is evidenced by the fact that the monthly setting outperforms the once setting, and similarly the multi-monthly setting outperforms the multi-once setting. This observation is a supporting evidence for the phenomenon of evolving malware behavior. As a consequence, the most recent samples would be more relevant to the current samples, and training on most recent samples would result in a more accurate prediction model.

From the first two observations, we can conclude that the further we go back in time to obtain more samples and the more recent the samples are, the better the trained model would perform. This conclusion can be exploited to improve prediction accuracy by going further and further back in time and retraining the model more often. However, this approach would become impractical at some point when the training time required to frequently update an accurate model via periodic retrainings would become too long to be practical. It turns out that this issue can be elegantly addressed by the FTRL algorithms, which produces much higher prediction accuracy at considerable lower running time.

Thirdly, FTRL algorithms (worst MCAE of 0.123) are shown to outperform the LR and SVM in all settings (best MCAE of 0.152). The error lines corresponding to the performance of FTRL algorithms consistently stays below error lines of LR and SVM. The gain in the prediction accuracy of FTRL algorithms can be explained by the contribution of all previous samples to its objective function. In different batch

Experiment	Running Time
LR Multi-monthly	44m 31s
LR Monthly	14m 14s
LR Multi-once	55s
LR Once	42s
SVM Multi-monthly	50m 35s
SVM Monthly	16m 12s
SVM Multi-once	1min 2s
SVM Once	50s
FTRL-Proximal	28s
TDAP	28s
FTRL-DP	26s

Table 5.1: Running time of FTRL-DP, FTRL-Proximal and different settings of LR.

settings of SVM and LR, only the losses associated with the samples within a certain time window contribute to the respective objective functions.

Finally, the fourth observation is that TDAP (MCAE of 0.117) and FTRL-DP (MCAE of 0.116) outperforms FTRL-Proximal (MCAE of 0.123). The gain in performance of TDAP and FTRL-DP over FTRL-Proximal can be explained by their ability to cope with concept drift via the mechanism to discount the contribution of previous learning examples. This mechanism makes use of an exponential decaying function to favor the most recent solutions over older ones. The effective result is that the most recent samples would contribute more to the current solution thereby alleviating the problem of concept drift.

5.6.2 Running Time

In terms of running time (training time and testing time combined), FTRL algorithms are clearly advantageous over LR and SVM. From Table 5.1, we can see that the running times of FTRL algorithms are much lower, especially compared to the settings with retraining involved (monthly and multi-monthly). The reason for this result is that FTRL algorithms only needs to see each sample once to update the current weight vector whereas in the case of LR and SVM, it requires multiple passes over each sample to ensure convergence to the optimal solution.

5.7 Summary

The evolving nature of malware over time makes the malware detection problem more difficult. According to previous studies, batch learning based methods often perform poorly when encountered zero-days samples. Our research is motivated to fill in this gap by proposing FTRL-DP – a variant of the FTRL-Proximal algorithm – to address this problem. We evaluated two learning paradigms using an extensive dataset generated by more than 100k malware samples executed on Cuckoo sandbox. The experimental results show that FTRL algorithms (worst MCAE of 0.123) outperforms LR and SVM in the typical setting of batch learning as well as the settings with retrainings involved (best MCAE of 0.152). The gain in performance of FTRL algorithms over different batch settings of LR can be accounted for by its objective function taking into account the contribution of all previous samples. Furthermore, the improvement of FTRL-DP over FTRL-Proximal can be explained by the usage of an adaptive mechanism that regularizes the weight by favoring recent samples over older ones. In addition, FTRL algorithms are also more advantageous in terms of running time.

6

Learning under Concept Drift with FTRL

Concept drift is the problem that the statistical properties of the data generating process change over time. Recently, Xiao et al. [24] proposed the Time Decaying Adaptive Prediction (TDAP) algorithm to address the problem of concept drift. TDAP is designed to account for the effect of drifting concepts by discounting the contribution of previous learning examples using an exponentially decaying factor. The drawback of TDAP is that the rate of its decaying factor is required to be manually tuned. To address this drawback, we propose a new adaptive online algorithm, called Follow-the-Regularized-Leader with Adaptive Decaying Proximal (FTRL-ADP). There are two novelties in our approach. First, we derive a rule to automatically update the decaying rate, based on a rigorous theoretical analysis. Second, we use a concept drift detector to identify major drifts and reset the update rule accordingly. Comparative experiments with 14 datasets and 6 other online algorithms show that FTRL-ADP is most advantageous in noisy environments with real drifts. This work is the **Contribution 3** of the thesis.

6.1 Introduction

The degradation of batch models in nonstationary settings originates from the fact that the formulation of batch learning does not factor in the issue of concept drift. The underlying assumption of batch learning is that the distribution of the data generating process does not change over time (the stationary assumption [108]). This assumption may be true in established application domains such as face recognition and digit recognition. In highly dynamic applications such as intrusion detection and fraud detection, this assumption may be challenged. On the contrary, online learning, specifically the framework of Follow the Regularized Leader (FTRL [109]), is formulated without basing on the Stationary Assumption. This property makes FTRL a more suitable tool to address nonstationary applications. This framework has been successfully applied to highly dynamic domains such as click through prediction [76] and video-on-demand prediction [24].

The most notable algorithm derived from the FTRL framework is the FTRL-Proximal algorithm [76]. In this algorithm, all learning examples contribute equally to the loss function resulting in the slow adaptation to concept drift. To enhance responsiveness to concept drift, Xiao et al. [24] introduces an exponentially decaying factor to augment the per-coordinate learning rate schedule proposed in FTRL-Proximal. The basic idea in this approach is the usage of a decaying factor to discount the contribution of previous examples. Although interesting and effective, this approach has two drawbacks.

The first drawback is that the proposed decaying rate is fixed and requires manual tuning. This process of manual tuning arises since different problems have different drifting dynamics, hence different rates of decaying are necessary. This tuning process is time-consuming and requires expertise. The second drawback is that a fixed decaying rate may not be optimal in the settings with complex dynamics (e.g. interleaved abrupt drifts and gradual drifts). In these settings, the decaying rate is expected to change from time to time to optimally adapt to the change in the drifting dynamics.

In this chapter, we aim to address these two drawbacks by proposing a new adaptive online algorithm, called Follow-the-Regularized-Leader with Adaptive Decaying Proximal. We introduce two novel contributions in this algorithm:

- We derive a formula to adjust the decaying rate based on a rigorous theoretical analysis (Section 6.3.2). This formula is theoretically justified to ensure that the algorithm has a sublinear regret bound.
- We anticipate major drifts to reset the learning using a drift detector to monitor the performance of the algorithm (Section 6.3.3). We base our idea on the Drift Detection Method introduced by Gama et al. [55].

The experimental result shows that FTRL-ADP achieves state-of-the-art performance in noisy datasets with real drifts owing to these two novelties.

The rest of the chapter is structured as follows. In Section 6.2, we present the framework of FTRL and point out 3 necessary refinements to this framework to address the problem of concept drift. These refinements are elaborated in Section 6.3 and lead to the proposed FTRL-ADP algorithm. We evaluate FTRL-ADP in Section 6.4. Specifically, we present the controlled experiments with 8 synthetic datasets in Section 6.4.1; and in Section 6.4.2, we use 6 real-world datasets to evaluate the performance of FTRL-ADP. We conclude the chapter in Section 6.5.

6.2 Problem Setting

In this section, we revise the online learning framework of Follow the Regularized Leader, formulated in the language of online convex optimization. We then present 3 necessary refinements to this framework. The refinements are designed to allow real-time online prediction and quick adaptation to concept drift.

The framework of online convex optimization can be formulated as follows [109]. We need to design an algorithm that makes a series of optimal predictions, each at one time step. At time step t , the algorithm makes a prediction, which is the weight vector w_t . A convex loss function $l_t(w)$ is then exposed to the algorithm. Finally, the

algorithm suffers loss $l_t(w_t)$ at the end of time step t (Algorithm 4). The algorithm should be able to learn from the losses in the past to make better predictions over time.

Algorithm 4 Online Algorithm

```

1: for  $t = 1, 2, \dots$  do
2:   Make a prediction  $w_t$ 
3:   Receive the lost function  $l_t(w)$ 
4:   Suffer the lost  $l_t(w_t)$ 
    
```

The optimality of a series of prediction is conditioned on the minimization of the regret with respect to the best classifier in hindsight (Equation 6.1). The meaning of Equation 6.1 is that we would like to minimize the total loss incurred up to time t with respect to the supposed loss incurred by the best possible prediction in hindsight w^* .

$$\mathbf{Regret}_t = \sum_{s=1}^t l_s(w_s) - \sum_{s=1}^t l_s(w^*) \quad (6.1)$$

Since the future loss functions are unknown, the greedy approach to achieve the objective of minimizing the regret is to leverage on the prediction that incurs the least total loss on all past rounds. This approach is called Follow-the-Leader (FTL), in which the leader is the best prediction with respect to all past loss functions. In some cases, this simple formulation may result in algorithms with undesirable properties such as rapid changes in the predictions [109], which lead to overall high regret. To fix this problem, some regularization function is usually added to stabilize the prediction. The second approach is called Follow-the-Regularized-Leader (FTRL), which is formulated in Equation 6.2 (the version presented here is the proximal version [75], which can be conveniently modified to handle the problem of concept drift).

$$w_{t+1} = \underset{w}{\operatorname{argmin}} \left\{ \sum_{s=1}^t l_s(w) + \sum_{s=1}^t \sigma_{t,s} \|w - w_s\|_2^2 \right\} \quad (6.2)$$

We next point out 3 refinements to this framework necessary to allow for real-time prediction under concept drift.

1st Refinement. In general, Equation 6.2 does not have a closed-form solution and we need to run some sort of gradient descent at each time step. This operation is computationally expensive and not suitable for real-time applications. This issue can be addressed by approximating the loss function using its linearization at w_t . Consequently, the optimization problem can be solved in closed-form and enjoy an efficient recursive structure. The recursive algorithm is constant in time and memory, which makes it suitable for real-time applications. This refinement is elaborated in Section 6.3.1.

2nd Refinement. The proximal terms in Equation 6.2 regularize the current solution by enforcing losses on deviations from past solutions. If $\sigma_{t,s}$ is a constant, all past solutions have equal regularizing effect. However, in the face of concept drift, we argue that it is more appropriate for the current solution to be biased towards the most recent past solutions, to counter the drifting effect. This idea can be realized by using an exponentially decaying factor $\sigma_{t,s} = \gamma^{t-s}$ (with $0 < \gamma < 1$) to increasingly discount the proximal terms. This refinement is illustrated in Section 6.3.2.

3rd Refinement. Different applications may have different drifting dynamics and require different decaying rates. Therefore, we propose a data-driven mechanism to adaptively adjust the decaying rate based on two novel ideas: sublinear regret analysis and drift detection. The latter idea is adapted from the method introduced by Gama et al. in [55]. This drift detection method has been shown to achieve the best performance across different tasks in [110]. This refinement is elaborated in Section 6.3.3.

6.3 Methodology

6.3.1 Linearization of Loss Function

We cast an online classification problem as an online convex optimization problem as follows. At time t , the algorithm makes prediction w_t and receives input x_t . The true

value y_t is then revealed to the algorithm. The loss function $l_t(w)$ associated with time t is defined in terms of x_t and y_t (Equation 6.3). Finally, the cost incurred at the end of time t is $l_t(w_t)$. The final optimization problem is shown in Equation 6.5. The solution to this problem is the Follow the Regularized Leader with Decaying Proximal (FTRL-DP) algorithm.

$$l_t(w) = -y_t \log(p) - (1 - y_t) \log(1 - p) \quad (6.3)$$

in which $p = \text{sigmoid}(w^\top x_t)$

Compared to Equation 6.2, Equation 6.5 has the actual loss function $l_t(w)$ replaced by its linear approximation at w_t , which is:

$$l_t(w) \approx l_t(w_t) + \nabla l_t(w_t)^\top (w - w_t) = g_t^\top w + l_t(w_t) - g_t^\top w_t \quad (6.4)$$

in which $g_t = \nabla l_t(w_t)$

The constant term $(l_t(w_t) - g_t^\top w_t)$ is omitted in Equation 6.5. Overall, this linear approximation is to enable the derivation of a closed-form solution at each time step, which is not possible with the original problem in Equation 6.2.

$$w_{t+1} = \underset{w}{\operatorname{argmin}} \left\{ g_{1:t}^\top w + \lambda_1 \|w\|_1 + \frac{1}{2} \lambda_2 \|w\|_2^2 + \frac{1}{2} \lambda_p \sum_{s=1}^t \sigma_{t,s} \|w - w_s\|_2^2 \right\} \quad (6.5)$$

in which $g_{1:t}^\top = \sum_{i=1}^t g_i^\top$ and $\sigma_{t,s} = \gamma^{t-s}$

FTRL-DP utilizes 3 different regularizers to serve 3 different purposes. The 1st regularizer is the L_1 -norm used to enforce sparsity on the solution. The 2nd regularizer is the L_2 -norm which serves to favor low variance solutions with small weights. The 3rd regularizer is the proximal terms used to stabilize the solution by not allowing it to deviate too much from past solutions. The solution to the objective function of FTRL-DP is stated in Theorem 1.

Theorem 1 *The optimization problem in Equation 6.5 can be solved in the following closed form:*

$$w_{t+1,i} = \begin{cases} 0 & \text{if } \|z_{t,i}\|_1 \leq \lambda_1 \\ -\frac{z_{t,i} - \lambda_1 \operatorname{sign}(z_{t,i})}{\lambda_2 + \lambda_p \frac{1 - \gamma^t}{1 - \gamma}} & \text{otherwise.} \end{cases} \quad (6.6)$$

$$\text{in which } z_t = g_{1:t} - \lambda_p \sum_{s=1}^t \gamma^{t-s} w_s$$

The proof of Theorem 1 is provided in A.1. Theorem 1 provides a formula to compute w_t at each time step. This formula has a recursive structure and can be efficiently realized by Algorithm 5.

In the next section (Section 6.3.2), we prove that the regret of FTRL-DP can be sublinearly bound via a proper choice of the decaying rate. This result leads to the proposed adaptive formula to update the decaying rate γ .

6.3.2 Decaying Proximal Regularization

The effect of the exponentially decaying proximal, $\sum_{s=1}^t \gamma^{t-s} \|w - w_s\|_2^2$ is to bias the current solution towards the most recent past solutions. This refinement is the novelty of our work compared to the FTRL-Proximal algorithm proposed by McMahan et al. [76]. McMahan et al. proposed the following per-coordinate learning schedule to regulate the proximal terms:

$$\sigma_{s,i} = \left(\sqrt{\sum_{j=0}^s (g_{j,i}^2)} - \sqrt{\sum_{j=0}^{s-1} (g_{j,i}^2)} \right) = \frac{g_{s,i}^2}{\sqrt{\sum_{j=0}^s (g_{j,i}^2)} + \sqrt{\sum_{j=0}^{s-1} (g_{j,i}^2)}}$$

The drawback of this schedule is that $\sigma_{s,i}$ tends to decrease with time, given roughly the same gradient $g_{s,i}$, since the denominator of $\sigma_{s,i}$ monotonically increases. Consequently, the current solution is based towards the least recent solutions, instead of the most recent ones. This consequence is unfavorable as it enhances the effect of concept drift rather than counterbalances it.

The decaying rate γ in Equation 6.5 controls the stability-plasticity balance of FTRL-DP. Stability refers to the ability to perform robustly in noisy environments. Plasticity refers to the ability of the algorithm to quickly adapt to drifting concepts. We can see that these two requirements are irreconcilable as they require the decaying rate to change in two opposite directions. A larger value of decay rate γ means that the model is more robust to noise but less responsive to drifting concepts. On the

contrary, for smaller γ , the model can quickly respond to concept drift at the cost of sensitivity to noise.

In practice, we need to manually search for the optimal decay rate to suit the specific application. This process is time-consuming. In Theorem 2, we prove a result that sheds some light on how to choose a good decaying rate.

Theorem 2 *Suppose that $\|w_t\|_2 \leq R$ and $\|g_t\|_2 \leq G$. With $\lambda_1 = \lambda_2 = 0$ and $\lambda_p = 1$, we have the following regret bound for FTRL-DP:*

$$\text{Regret}(w^*) \leq 2R^2 \frac{1 - \gamma^T}{1 - \gamma} + \frac{G^2}{2} \frac{1 - \gamma}{\gamma^T} \sum_{t=1}^T \frac{\gamma^t}{1 - \gamma^t} \quad (6.7)$$

We prove that a proper choice of γ can lead to a sublinear growth of the expression on the right-hand side of 6.7. Specifically, if we choose $\gamma = 1 - \frac{\ln T}{2T}$, we have the following sublinear regret bound:

$$\text{Regret}(w^*) \leq 4R^2 \frac{T}{\ln T} + \frac{G^2}{2} \frac{(1 + \ln T)}{(1 - \frac{\ln T}{2T})^T} \quad (6.8)$$

The proof to Theorem 2 is provided in A.2. Theorem 2 means that if we are allowed to change the decaying rate of FTRL-DP at each time step, the specific choice of $\gamma = 1 - \frac{\ln T}{2T}$ will ensure that the average regret of FTRL-DP (dividing both sides of 6.8 by T) diminishes with time.

Algorithm 5 Follow the Regularized Leader with Decaying Proximal

- 1: Input: $\lambda_1, \lambda_2, \lambda_p, \gamma$
 - 2: Initialize $v_t = h_t = z_t = w_t = 0, r_t = 1$
 - 3: **for** $t = 1, 2, \dots$ **do**
 /* Make prediction */
 4: Make prediction $w_{t,i} = \begin{cases} 0 & \text{if } \|z_{t,i}\|_1 \leq \lambda_1 \\ -\frac{z_{t,i} - \lambda_1 \text{sign}(z_{t,i})}{\lambda_2 + \lambda_p r_t} & \text{otherwise.} \end{cases}$
 5: Receive x_t, y_t /* and suffer the loss $l_t(w_t)$ 6.3*/
 /* Update parameters */
 6: $p_t = \frac{1}{1 + e^{-w_t^\top x_t}}$
 7: $g_t = (p_t - y_t)x_t$
 8: $v_t = v_{t-1} + g_t$
 9: $h_t = \gamma h_{t-1} + w_t$
 10: $z_t = v_t - \gamma_p h_t$
 11: $r_t = \gamma r_{t-1} + 1$
-

Algorithm 5 is initialized with 5 parameters (v_t , h_t , z_t , w_t , and r_t). These parameters are buffered and iteratively updated after each time step. In each time step, the algorithm firstly makes a weight prediction w_t (Line 4). It then uses the new weight w_t and the feature vector x_t (Line 5) to compute the probability p_t of class 1 (Line 6). The difference between the predicted probability and the true label is used to adjust the rest 4 buffered parameters (v_t in Line 8, h_t in Line 9, z_t in Line 10, and r_t in Line 11). It is noted that caching the intermediate computation results in v_t , h_t , z_t , and w_t allows the computation cost to remain constant in each step.

Therefore, in the next section (Section 6.3.3), we propose an update rule to adjust the decaying rate of FTRL-DP based on this theorem ($\gamma = 1 - \frac{\ln T}{2T}$). In addition, we also consider the effect of drifting speed and baseline noise level on the decaying rate.

6.3.3 Adaptive Decaying Rate

We propose Algorithm 6 as an adaptive mechanism to adjust the decaying rate. In Line 6 of Algorithm 6, the decaying rate increases following the rule derived in Theorem 2. This update rule to ensure sublinear regret bound is only meaningful in stationary settings. The rest of the algorithm are to address nonstationary settings by resetting the update rule on detected drifts.

In nonstationary settings, the optimal decaying rate depends on two factors: the drifting speed and the noise level. The relationship between the drifting speed and the decaying rate is reversely proportional. Namely, if the drifting speed is high, the decaying rate should be low so that the algorithm quickly forgets past solutions. On the other hand, the baseline noise level and the decaying rate are proportionally related. In a noisy environment, the decaying rate should be high for the learner to stay conservative to prediction errors. This analysis highlights the opposite effect of the drifting speed and the baseline noise level on the decaying rate, which is usually referred to as the stability-plasticity dilemma encountered in adaptive learning systems [111].

Algorithm 6 Adaptive Decaying Rate

```

1: Input:  $p_t, y_t$ 
2: Initialize  $p_{min} = s_{min} = 1, count = error = count_{warn} = error_{warn} = 0$ 
3: for  $t = 1, 2, \dots$  do
4:    $count = count + 1$ 
5:    $error = error + \mathbb{1}_{\|p_t - y_t\|_1 > 0.5}$ 
6:    $\gamma = 1 - \frac{\ln(count)}{2 \times count}$ 
7:   if  $count > 30$  then
8:      $p_i = \frac{error}{count}$ 
9:      $s_i = \sqrt{\frac{p_i(1-p_i)}{count}}$ 
10:     $sum = p_i + s_i$ 
11:     $\gamma = 0.99\gamma + 0.01p_i$ 
12:    if  $sum < p_{min} + s_{min}$  then
13:       $p_{min} = p_i$ 
14:       $s_{min} = s_i$ 
15:    if  $sum < p_{min} + 2s_{min}$  then /* Trigger warning stage */
16:       $count_{warn} = 0$ 
17:       $error_{warn} = 0$ 
18:    else
19:       $count_{warn} = count_{warn} + 1$ 
20:       $error_{warn} = error_{warn} + \mathbb{1}_{\|p_t - y_t\|_1 > 0.5}$ 
21:      if  $sum > p_{min} + 3s_{min}$  then /* Trigger critical stage */
22:         $count = count_{warn}$ 
23:         $error = error_{warn}$ 
24:         $p_{min} = 1$ 
25:         $s_{min} = 1$ 
    
```

6.3.3.1 Drifting Speed

The basic idea is to reset the update rule (thereby decreasing the decaying rate) when the learner starts to degrade. To this end, we keep track of the current error rate, p_i , which is computed in Line 8 of Algorithm 6. We then consider each prediction as a Bernoulli trial with mean p_i and standard deviation $s_i = \sqrt{\frac{p_i(1-p_i)}{count}}$ (Line 9).

According to statistical learning theory [108], the variance of the error rate must decrease when the number of examples increases. Therefore, if the error rate deviates significantly from the current rate, a major drift may have happened. In this case, we are supposed to decrease the decay rate to speed up the forgetting process. This idea was first introduced by Gama et al. [55].

To be more sensitive to low drifting speed, the pair (p_{min}, s_{min}) with minimum sum $p_{min} + s_{min}$ (Line 12 to Line 14) is used as the reference distribution, instead of

the most recent pair (p_i, s_i) . At any point, if $p_i + s_i > p_{min} + 2s_{min}$, a concept drift is anticipated and the warning stage is triggered (Line 15). The factor of 2 is chosen to ensure 95% confidence of drift. If $p_i + s_i > p_{min} + 3s_{min}$, a drift is detected and the learning is reset (Line 21 to Line 25). The factor of 3 is chosen to ensure 99% confidence of drift.

6.3.3.2 Noise Level

The basic idea is to increase the decaying rate when the noise level increases and vice versa. To realize this idea, we establish a simple linear relationship between the decaying rate and error rate p_i (Line 11). The contribution amount of the error rate to the decaying rate ($0.01 = 1\%$) is empirically determined and found to work well across all datasets.

6.3.4 Space and Time Complexity of FTRL-ADP

Both Algorithm 5 and Algorithm 6 have constant space and time complexities. Algorithm 5 is a recursive procedure to compute the latest solution weight by leveraging on the proceeding one. This is the result of the linearization trick to avoid running gradient descent at each time step in solving Equation 6.2. On the other hand, Algorithm 6 is a recursive procedure to estimate the current error rate. Combining Algorithm 5 and Algorithm 6 results in Follow the Regularized Leader with Adaptive Decaying Rate algorithm, which therefore also enjoy constant space and time complexities.

6.4 Experiment

We use 8 synthetic and 6 real-world datasets to evaluate the performance of FTRL-ADP. Table 6.2 summarizes the properties of the datasets.

In the experiments with 8 synthetic datasets, we evaluate 3 different versions of FTRL-ADP (FTRL-P with fixed $\gamma = 1$, FTRL-DP with fixed $\gamma < 1$, and FTRL-ADP with adaptive γ) against TDAP [24] and FTRL-Proximal [76]. All these algorithms are derived from the FTRL framework.

Dataset	Number of Examples	Number of Features	Duration	Drift Type
Stationary	20k	2	N/A	Stationary
Dynamic	70k	2	N/A	Mixed Drift
SEA [65]	40k	3	N/A	Abrupt Drift
Hyperplane [113]	40k	10	N/A	Gradual Drift
Noisy Stationary	20k	2	N/A	Stationary
Noisy Dynamic	70k	2	N/A	Mixed Drift
Noisy SEA [65]	40k	3	N/A	Abrupt Drift
Noisy Hyperplane [113]	40k	10	N/A	Gradual Drift
Spam Dataset [114]	9.324	39.916	Unknown	Unknown
URL Dataset [115]	616.000	3.231.960	30 days	Unknown
Airline Dataset [116]	539.383	612	20 years	Unknown
Weather Dataset [117]	18.159	8	50 years	Unknown
Electricity Dataset [118]	27,549	60	2.5 years	Unknown
Intrusion Dataset [119]	494,021	122	7 weeks	Unknown

Table 6.1: 14 Evaluated datasets.

In the experiments with 6 real-world datasets, we compare FTRL-ADP to 6 state-of-the-art online algorithms: Confident Weighted Learning [22], PA I, PA II [21], ALMA [112], TDAP [24], and FTRL-Proximal [76]. These algorithms are single learners and comparable to FTRL-ADP.

We compare the performances of the algorithms using average cumulative error (or prequential error [120]). We use 0.5 as the threshold to binarize the probabilistic predictions when applicable.

6.4.1 Synthetic Experiments

The objective of this experiment is to study how the optimal decaying rate depends on the drifting speed and the noise level.

For a fair comparison, we use the same amount of regularizations ($L_1 = 1$, $L_2 = 1$ and $L_p = 1$) in the 5 algorithms, namely: FTRL-P ($\gamma = 1$), FTRL-DP ($\gamma < 1$), FTRL-ADP (adaptive γ), TDAP, and FTRL-Proximal. The decaying rates of FTRL-DP and TDAP are set to the same value of 0.9.

6.4.1.1 Datasets

Other than abrupt and gradual, concept drifts can also be classified as real and virtual [102]. In a classification problem, the objective is to determine the posterior probability $P(y|x) = \frac{P(x|y)P(y)}{P(x)}$. In this context, real drift refers to the shift in $P(y|x)$.

Virtual drift refers to the change in the data generating process $P(x)$. Virtual drift may or may not co-occur with real drift.

We use two publicly available synthetic datasets to evaluate the case of real drift: SEA dataset [65] and Hyperplane dataset [113]. The SEA dataset contains abrupt real drifts. The drifts in the Hyperplane dataset is also real but the drifting speed is gradual. To study more complex drifting dynamics, we generate a new dataset that contains both abrupt and gradual virtual drifts. In addition, we also generate a stationary dataset to serve as the baseline for evaluation.

SEA dataset This dataset simulates abrupt real drifts. It involves three features, which are the predictors of two classes; only the first two features are correlated with the labels. The generated points are separated into two classes using the threshold f on the sum of the first two features. For a given point (f_1, f_2, f_3) , if $f_1 + f_2 < f$, it is labelled as class 0 and class 1 otherwise. Consequently, the decision line only translates in the feature space, which is different from the rotating decision line in the Hyperplane dataset to be introduced next. We use the framework of Massive Online Analysis (MOA [121]) to generate a dataset of 40k examples with abrupt concept changes at 10k and 20k and 30k using the thresholds 7, 13, 6, and 12 in succession.

Hyperplane dataset This dataset resembles the SEA dataset except that the decision line can also rotate in the feature space. For a given point $(f_1, f_2, \dots, f_n)$, if $\sum_{i=1}^n f_i w_i < w_0$, it is labelled as class 0 and class 1 otherwise. The drifting dynamics is controlled by varying the weights following the law $w_i = w_i + \sigma d$, in which σ is the probability that the drifting direction is reversed and d is the amount of drift. Using MOA, we generate a dataset of 40k examples with 10 features, $\sigma = 10\%$, and $d = 30\%$.

The above two datasets are commonly used to evaluate the capability to handle concept drifts of online algorithms. However, they only represent the simple scenarios of a single type of drift. We propose a new dataset that simulates a scenario with more complicated drifting dynamics.

Dynamic dataset In this dataset, we interleave abrupt drifts with gradual drift. The data points are generated by two moving Gaussian distributions to simulate real concept drifts. The speed of drifting is controlled by how fast the means of the Gaussians move in the feature space. Using this method, we generate a dataset of 70k examples with the follow drifting sequence: stationary, gradual, abrupt, gradual, abrupt, gradual, and abrupt in succession.

Stationary dataset We also use the same method as in the Dynamic dataset to generate a stationary dataset of 20k examples. However, to ensure stationarity, the positions of the two Gaussian means are not moving in this dataset.

For each of the 4 datasets above, we generate two versions: the first version with a low level of noise and the second version with a high level of noise.

6.4.1.2 Discussion

We use the same parameters to generate each dataset 10 times. Figure 6.1 shows the evolution of the prediction errors (average score of 10 iterations) of all algorithms on all synthetic datasets. Table 6.3 summarizes the errors using means and standard deviations. The bold numbers indicate the best performers.

We can see that FTRL-P and FTRL-Proximal outperform FTRL-ADP, FTRL-DP, and TDAP in the two Stationary datasets (Figure 6.1). The gaps in the prediction errors are more announced in the noisy Stationary dataset. This result validates our claim that high level of noise requires the learner to be conservative and less responsive to prediction errors. In this case, the stability of FTRL-P and FTRL-Proximal, owing to the omission of the decaying factor (or $\gamma = 1$), explains their best performance.

At the same time, we can see that FTRL-DP and TDAP become too paranoid with prediction errors as it tries to correctly classify every example by quickly forgetting old weights ($\gamma < 1$). Overall, they make the most number of mistakes due to this short-sightedness. On the contrary, FTRL-ADP performs comparable to FTRL-P and FTRL-Proximal while avoiding the shorted-sightedness of FTRL-DP of TDAP, due to its adaptive mechanism. This mechanism increases the decaying rate of FTRL-ADP

Figure 6.1: Error Rates of FTRL-P, FTRL-Proximal, FTRL-DP, TDAP and FTRL-ADP on 8 Synthetic Datasets.

to asymptotically reach 1 ($\gamma = 1 - \frac{\ln T}{2T}$) to best perform in the stationary case. There are no concept drifts detected in the stationary case according to Figure 6.2.

The addition of noise degrades the performance of all learners in all cases. The error rates of FTRL-P, FTRL-Proximal, FTRL-DP, TDAP, and FTRL-ADP increase by 15.7%, 15.5%, 19.24%, 21.3%, and 16.1% in the noisy Stationary dataset. FTRL-ADP are more robust than FTRL-P and FTRL-DP but less robust than FTRL-P and FTRL-Proximal. This result can be explained by the conservativeness when $\gamma = 1$

Figure 6.2: Evolution of decaying rates of FTRL-ADP on 8 synthetic datasets.

and the responsiveness when $\gamma < 1$.

On the contrary, FTRL-P and FTRL-Proximal perform the worst in the Dynamic dataset. After the onsets of the drifts (localized by the plunges in the decaying rate (Figure 6.2)), the performances of all the learners decrease. However, the error rates of FTRL-Proximal and FTRL-P sharply increase while the error rates of TDAP, FTRL-DP, and FTRL-ADP quickly plateau. For example, after the first drift at example 1300th in the Dynamic dataset, the error rates of FTRL-Proximal and FTRL-P steeply increase from around 0.08 to 0.17 while the error rates of TDAP, FTRL-DP, and

FTRL-ADP settle at around 0.10. This result supports our claim that conservative learners (FTRL-P and FTRL-Proximal) do not perform well when drifting concepts are present.

Here, we can see the opposite effects of drifting speed and noise level on the conservativeness of the algorithm: they require the conservativeness to change in two opposite directions. This is a manifestation of the Stability-Plasticity dilemma usually encountered in adaptive learning systems [111].

In the SEA and Hyperplane datasets, the performances of the algorithms are mixed. FTRL-DP, TDAP, and FTRL-ADP usually outperform FTRL-P and FTRL-Proximal in the cases with less noise and vice versa in the noisy cases. It is interesting to note that the FTRL-P and FTRL-Proximal are rather robust to virtual drifts (contained in the SEA and Hyperplane datasets). Their error rates do not sharply increase at the onsets of the drifts like in the Dynamic datasets, which contain real drifts. On the other hand, FTRL-DP, TDAP, and FTRL-ADP are robust to both types of drifts.

In addition, the performances of FTRL-DP, TDAP, and FTRL-ADP are comparable in the 3 nonstationary datasets with less noise. When the noises are injected into these 3 datasets, FTRL-ADP becomes significantly outperforming FTRL-DP and TDAP. This result again validates our claim that high level of noise penalizes responsive learners the most. Compared to FTRL-DP and TDAP, FTRL-ADP is more robust to noise due to its decaying rate being adjusted according to the level of noise ($\gamma = 0.99\gamma + 0.01p_i$).

Therefore, we can conclude that the optimal decaying rate depends on the drifting speed and the noise level. Higher drifting speed requires the learner to be more responsive, which can be achieved by decreasing the decaying rate. On the other hand, noisy cases require the learner to be conservative by increasing the decaying rate. These two factors are considered when designing the adaptive scheme (Algorithm 6) to adjust the decaying rate (specifically, Line 21 to consider drifting speed and Line 11 to consider noise level).

Dataset	FTRL-P	FTRL-DP	FTRL-ADP	FTRL-Proximal	TDAP
Stationary	8.05±0.38	8.72±0.46	8.25±0.78	8.08±0.96	9.49±0.93
Dynamic	15.54±3.91	10.14±0.8	10.02±1.06	13.25±2.68	10.41±0.67
Sea	8.4±1.9	3.93±2.77	3.25±2.14	7.54±3.34	5.58±4.19
Hyperplane	13.36±4.53	4.62±3.71	9.35±2.74	15.15±6.48	10.72±4.23
Noisy Stationary	23.6±0.78	27.88±0.84	24.19±0.9	23.54±0.94	30.71±0.7
Noisy Dynamic	26.78±2.1	26.25±0.54	23.41±0.71	26.42±1.87	29.36±0.77
Noisy Sea	26.51±3.16	28.51±2.49	26.78±1.88	25.1±2.27	26.64±2.15
Noisy Hyperplane	35.98±2.01	36.96±0.71	34.24±1.49	36.03±2.4	38.43±0.83

Table 6.2: Means and Standard Deviations of Error Rates of FTRL-P, FTRL-Proximal, FTRL-DP, TDAP and FTRL-ADP on 8 Synthetic Datasets.

The experimental results show that this scheme enables FTRL-ADP to perform better than or comparably well to FTRL-DP and TDAP in all 8 datasets with the additional advantage of not having to tune the decaying rate γ .

6.4.2 Real-world Experiments

The objective of this experiment is to evaluate FTRL-ADP against 6 state-of-the-art online algorithms: Confident Weighted Learning [22], PA I, PA II [21], ALMA [112], TDAP [24], and FTRL-Proximal [76]. We also include the evaluation of DDM-enabled Hoeffding Tree [70] as the baseline drift-enabled classifier for comparison.

We use the same amount of L_1 , L_2 and L_p regularizations in FTRL-ADP, TDAP, and FTRL-Proximal. The decaying rate γ of TDAP and parameter α of ALMA is searched in $\{0.5, 0.55, \dots, 0.95\}$. The parameters C of PA-I and PA-II and η of CWL used are the best performing values in $\{10^{-3}, 10^{-2}, \dots, 10^1\}$.

6.4.2.1 Datasets

We evaluate the algorithms on the following datasets: Spam [114], URL [115], Airline [116], Weather [117], Electricity [118], and Intrusion [119]. These datasets cover a wide range of applications showing the prevalence of concept drift in real-world applications.

Spam dataset. This dataset contains spam emails collected using the SpamAssassin data collection framework. The bag of words model is used to represent each spam as

Figure 6.3: Error Rates of FTRL-ADP, TDAP, PA I, PA II, ALMA, FTRL-Proximal and CW on 6 Real-world Datasets.

a vector. Concept drift contained in the dataset has been studied in detail in [114].

URL dataset. This dataset is collected in an effort aimed to create a machine learning based firewall to replace the commonly used blacklists. The dataset contains anonymized and labeled URLs collected over the period of 120 days. Due to the adversary nature of the domain of computer security, concept drifts are anticipated in this dataset and have been studied in detail in [115].

Airline dataset. This dataset is motivated by the need to predict flight delays. It contains more than half a million examples collected over the period of more than 20 years. The raw dataset contains many categorical features and we use one hot encoding to convert them to numerical features.

Weather dataset. This dataset is motivated by the need to predict rain in weather forecasting. It is a collection of weather records over more than 50 years curated by the National Oceanic and Atmospheric Administration, USA.

Electricity dataset. This dataset records the changes in electricity consumption in New South Wales (Australia). The duration of the dataset is from 7 May 1996 to 5 December 1998. A record contains 6 attributes: day of week, time of day, demand in New South Wales, price in Victoria, demand in Victoria, and electricity transfer between the states. The label is either UP for the increase in the consumption of electricity or DOWN vice versa.

Intrusion dataset. This dataset records the network connections made at the gateway of a simulated military network environment. The network traffic contains various type of attacks in the midst of normal traffic. The dataset contains 41 features including both categorical features, such as the protocol type, and numerical features, such as the number of failed logins. The label of a connection is either attack or normal.

6.4.2.2 Discussion

Like the synthetic experiment, the cumulative prediction errors are used as the performance metric. Figure 6.3 shows the error rates associated with the best parameters incurred by all algorithms on all datasets. Table 6.3 shows the final error rates. The bold numbers in Table 6.3 indicate the best performers. Figure 6.4 shows the evolution of the decaying rate reported by FTRL-ADP.

From Figure 6.3, we can see that FTRL-ADP and TDAP tend to be more robust to concept drifts than FTRL-Proximal. For example, in the case of the Spam dataset, the prediction error of the FTRL-Proximal consistently rises after the onset of the drift (occurs at around example 1000th based on the evolution of decaying rate in Figure 6.4). The error rates of FTRL-ADP and TDAP also increase but plateau quickly. We can notice the same observation in the Airline dataset and the Intrusion dataset. The performance of FTRL-Proximal significantly increases from example

Figure 6.4: Evolution of decaying rates of FTRL-ADP on 6 real-world datasets.

Dataset	FTRL-ADP	FTRL-Proximal	TDAP	PAI	PAII	CW	ALMA	DDM-HT
Spam	13.24	19.28	14.71	11.06	11.09	11.5	11.79	20.28
Airline	36.3	38.78	37.3	40.73	40.7	40.87	40.84	35.1
Url	33.91	35.84	34.37	36.94	36.97	37.56	37.56	35.74
Weather	28.32	30.13	26.0	29.08	28.97	26.25	27.24	26.71
Electricity	15.23	23.16	21.63	15.82	15.76	15.71	15.84	15.88
Intrusion	0.15	1.70	0.48	2.11	2.14	1.77	1.82	0.80

Table 6.3: Error Rates of FTRL-ADP, TDAP, PA I, PA II, ALMA, FTRL-Proximal and CW on 6 Real-world Datasets.

200kth (Airline dataset) and 10kth (Intrusion dataset) while the performance of TDAP and FTRL-ADP also increase but at a much lower rate.

TDAP and FTRL-ADP perform comparably well in real-world datasets. This is the result of fine tuning the decaying rate of TDAP by running the experiment multiple times to select the best decaying rate. On the contrary, the decaying rate of FTRL-ADP is adaptive and doesn't require tuning.

PA I, PA II, CW, and ALMA outperform other algorithms in the SPAM dataset. They are the most adaptive to concept drift in this case with better performance after the drift. However, these 4 learners are often over-sensitive to noise; this characteristic

Figure 6.5: Error Rates of FTRL-ADP, TDAP, PA I, PA II, ALMA, FTRL-Proximal and CW on the two SEA Datasets.

may explain their poor performance in the other 3 datasets. We can demonstrate this characteristic in a simulated scenario. For example, Figure 6.5 shows the cumulative error rates of all algorithms on the two SEA datasets. We can see that CW performs almost perfectly (99.35% accuracy) and significantly outperforms other algorithms in the clean SEA dataset. However, its performance degrades substantially when the noise is injected.

6.5 Summary

In this chapter, we have introduced the idea of adaptive decaying rate to the FTRL framework. Adaptive decaying rate is a well-known idea in the theory of adaptive filter [122]. We have tried to derive the rule to update the decaying rate following the approaches in the adaptive filter theory, but the derived rule turned out to change the decaying rate wildly leading to an unstable algorithm. Therefore, we turn to analyzing the regret of the FTRL framework for the derivation of the update rule. The result is a simple update rule ($\gamma = 1 - \frac{\ln T}{2T}$). This rule is backed by Theorem 2, which ensures that the resulting regret bound is sublinear. The derived adaptive decaying rate is although theoretically justified but still not good enough in practice when we test it on real datasets. We need to borrow another idea from the concept drift detection research – the concept drift detectors. The result is a concept drift enabled online algorithm called FTRL-ADP. Experiments with 14 datasets and 6 other online

algorithms show that FTRL-ADP is most advantageous in noisy environments with real concept drifts.

7

Conclusion and Future Work

7.1 Conclusion

We have presented our 3 key contributions through 6 chapters. Here we summarize and connect the contributions based on the timeline of discovery.

In first-hand experiments with a few malware toolkits available in the wild, we identified a surprising observation: all the malware samples generated by these toolkits leave periodic signals in network traffic. We went on to carry out a scientific experiment to confirm this observation. To this end, we searched over hacker forums and collected 31 malware toolkits often used by script kiddies and experienced attackers alike. We found out that 30/31 of them actually generated periodic signals. The experiment was documented in Chapter 3 and is the first part of the **Contribution 1** of the thesis.

In Chapter 4, we aimed to address the challenges in detecting malware based on their periodic behavior, which we had posed in Chapter 3. Since time domain signals are difficult to analyze, we relied on Discrete Fourier Transform to decompose time domain signals into elementary periodic signals. We then proposed a novel periodicity measure based on Fourier transform to gauge the regularity of network traffic. In addition, the other challenge is that not all periodic signals are generated by malware

alone. To address this challenge, we proposed a unified visual analytics solution to address both the detection and the verification of periodic signals. This is the second part of the **Contribution 1** of the thesis.

Chapter 5 presents the **Contribution 2** of the thesis in which we focused on another behavioral characteristic of malware: the evolution in behavior of malware over time. This characteristic has been confirmed in previous studies by other researchers. Due to the lack of publicly available datasets on the dynamic behavior of malware, we had carried out an extensive experiment to collect the behavioral data generated by 100K malware samples. From our experiments, we compiled a dataset that recorded 482 behavioral features of the samples. The dataset was donated to the UCI machine learning repository to promote research on behavioral malware detection. In addition, we also proposed a new online algorithm to predict the risk level of malware based on the curated 482 behavioral features.

We found out that the problem of concept drift is ubiquitous and appears in many other security problems such as spam filtering and non-security ones such as weather prediction. Therefore, we extended the algorithm proposed in Chapter 5 and applied the improved algorithm to other settings with concept drifts. The experimental results showed that the new algorithm achieved state-of-the-art performance in noisy settings with real concept drifts. This development was documented in Chapter 6. This is the **Contribution 3** of the thesis.

7.2 Future Work

The research carried out for writing this thesis opens up many new interesting research questions. Following we will discuss three notable directions for future research:

The visualization techniques employed in Chapter 4 are also applicable to other time-series based alerts, such as network anomalies. Typically, the alerts generated by a time-series based anomaly detector are very hard to interpret as they contain no context information with respect to the detected anomaly. In this case, a stacked histogram can be used to visualize the associated time series enabling the analyst to

cross-check the detected anomaly with the content of individual netflow. Furthermore, this visualization technique can also be used to explore suspicious entities such as the most active internal IP addresses or external IP addresses. Therefore, a possible extension includes the integration of various detectors and visualizations into the system, allowing the system to detect a wider set of malicious signals.

In Chapter 5, we rely on the features extracted from the Cuckoo sandbox reports to represent the behavior of the malware samples. Some valuable networking behaviors of the malware such as the amount and the periodicity of the networking traffic are not included in these reports. In addition, the periodicity feature has been shown to be highly characteristic of malicious behavior in Chapter 3 and should contribute significantly to the risk prediction accuracy. Therefore, one interesting direction to explore in the future is to include more networking behavior of the malware such as the two features introduced previously.

In Chapter 6, we have discussed the mechanism to decay the proximal terms as a way to address the problem of concept drift. A possible extension of this mechanism is to decay the actual loss associated with previous learning examples. However, it is not clear as to how we should design the decaying mechanism to ensure at least sublinear regret. The first candidate for the decaying function is the exponential decaying factor used in FTRL-ADP. However, any decreasing function can play the role of the decaying function. In addition, a closed-form solution must be ensured, otherwise, the resulting algorithm would become too computationally expensive.

Another possible research direction includes the theoretical analysis of the ability to track concept drift of FTRL-ADP. We are now only able to visually demonstrate the tracking ability of the algorithm, which is a good starting point for a rigorous analysis. The objective is to develop a mathematical analysis to prove the convergence property of FTRL-ADP in the case of concept drift. For example, we can analyze the adaptive regret analysis starting with some simple drifting dynamics like the linear relationship and then extend the work to the general dynamics.

Lastly, the problem of concept drift is present in any domains with time dimension.

In this thesis, we have already applied the algorithm to weather and electricity predictions and the results are promising. It would be interesting to apply the proposed algorithm to other temporal domains such as audio and video signal processing. In audio signal processing, we have the problem of voice activity detection [123], which aims to detect the presence of human speech in an audio recording. As the environment condition, such as the noise, changes with time, this serves as a good use case with concept drift. In video signal processing, we have the problem of detecting objects in changing environment for tracking robots [124]. Like the problem of voice activity detection, the surrounding environment of a video capture is expected to change dramatically from frame to frame hence this also serves as a good application domain for the proposed algorithm. On the other hand, one challenge of video and audio signal processing is the hierarchical structure of the features, which would be hard for a linear method like the proposed method to work well. Therefore, it may be necessary to develop a non-linear version of this method for it to be competitive to other state-of-the-art methods such as deep learning models.

References

- [1] Signature based network intrusion detection system and method. 00136 U.S. Classification 726/23, 713/153, 726/22, 713/154, 713/188, 709/223, 726/24, 713/166, 713/165, 713/164, 370/392; International Classification G06F11/00; Cooperative Classification H04L63/0263, H04L63/1416; European Classification H04L63/14A1, H04L63/02B6.
- [2] Igor Santos, Felix Brezo, Javier Nieves, Yoseba K. Peña, Borja Sanz, Carlos Laorden, and Pablo G. Bringas. *Idea: Opcode-Sequence-Based Malware Detection*, pages 35–43. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
- [3] Deepak Venugopal and Guoning Hu. Efficient signature based malware detection on mobile devices. *Mobile Information Systems*, 4(1):33–49, 2008.
- [4] Martin Roesch and others. Snort: Lightweight intrusion detection for networks. In *LISA*, 1999.
- [5] Ilsun You and Kangbin Yim. Malware obfuscation techniques: A brief survey. pages 297–300. IEEE, 2010.
- [6] Huynh Ngoc Anh, Do Hoang Giang, and Keong Ng Wee. On periodic behavior of malware: Experiments, opportunities and challenges. *IEEE International Conference on Malicious and Unwanted Software*, pages 1–8, 2016.
- [7] Neminath Hubballi and Deepanshu Goyal. FlowSummary: Summarizing network flows for communication periodicity detection. In *Pattern Recognition and Machine Intelligence*, pages 695–700. Springer, 2013.

- [8] Li Ming Chen, Shun-Wen Hsiao, Meng Chang Chen, and Wanjiun Liao. Slow-paced persistent network attacks analysis and detection using spectrum analysis. *IEEE Systems Journal*, pages 1–12, 2014.
- [9] Genevieve Bartlett, John Heidemann, and Christos Papadopoulos. Low-rate, flow-level periodicity detection. *Computer Communications Workshops (INFOCOM WKSHPS)*, pages 804–809, 2011.
- [10] J. Saxe and K. Berlin. Deep neural network based malware detection using two dimensional binary program features. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 11–20, October 2015.
- [11] Abdelaziz Mohaisen and Omar Alrawi. Unveiling zeus: automated classification of malware samples. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 829–832. ACM, 2013.
- [12] Yakup Korkmaz. *Automated detection and classification of malware used in targeted attacks via machine learning*. PhD thesis, Bilkent University, 2015.
- [13] Annamalai Narayanan, Liu Yang, Lihui Chen, and Liu Jinliang. Adaptive and scalable android malware detection through online learning. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 2484–2491. IEEE, 2016.
- [14] C Guarnieri, M Schloesser, J Bremer, and A Tanasi. Cuckoo sandbox-open source automated malware analysis. *Black Hat USA*, 2013.
- [15] N. Anh Huynh, W. Keong Ng, A. Ulmer, and J. Kohlhammer. Uncovering periodic network signals of cyber attacks. In *2016 IEEE Symposium on Visualization for Cyber Security (VizSec)*, pages 1–8, Oct 2016.
- [16] Huynh Ngoc Anh, Wee Keong Ng, and Kanishka Ariyapala. A new adaptive learning algorithm and its application to online malware detection. In *20th International Conference on Discovery Science, DS 2017*, pages 18–32, 2017.

- [17] Ngoc Anh Huynh, Wee Keong Ng, and Kanishka Ariyapala. Learning under concept drift with follow the regularized leader and adaptive decaying proximal. *Expert Systems with Applications*, 96:49–63, 2017.
- [18] Sebastián García, Vojtěch Uhlíř, and Martin Rehak. Identifying and modeling botnet c&c behaviors. In *Proceedings of the 1st International Workshop on Agents and CyberSecurity*, ACySE '14, pages 1:1–1:8, New York, NY, USA, 2014.
- [19] Basil AsSadhan, José MF Moura, and David Lapsley. Periodic behavior in botnet command and control channels traffic. In *Global Telecommunications Conference*, pages 1–6. IEEE, 2009.
- [20] John C. Duchi, Shai Shalev-Shwartz, Yoram Singer, and Ambuj Tewari. Composite objective mirror descent. In *COLT*, pages 14–26, 2010.
- [21] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.
- [22] Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-weighted linear classification. In *Proceedings of the 25th international conference on Machine learning*, pages 264–271. ACM, 2008.
- [23] Stephen P. Boyd and Lieven Vandenberghe. *Follow-the-Regularized-Leader and Mirror Descent*. Cambridge University Press, 2010.
- [24] Yinyan Tan, Zhe Fan, Guilin Li, Fangshan Wang, Zhengbing Li, Shikai Liu, Qiuling Pan, Eric P Xing, and Qirong Ho. Scalable time-decaying adaptive prediction algorithm. In *KDD*, pages 617–626, 2016.
- [25] Lin Xiao. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11:2543–2596, 2010.

- [26] Pieter De Boer and Martin Pels. Host based intrusion detection system. *International Journal of Science and Research*, 4(4):559–561, 2015.
- [27] G. Vigna and R. A. Kemmerer. NetSTAT: a network-based intrusion detection approach. In *Proceedings 14th Annual Computer Security Applications Conference (Cat. No.98EX217)*, pages 25–34, 1998.
- [28] Sandeep Yadav, Ashwath Kumar Krishna Reddy, A. L. Reddy, and Supranamaya Ranjan. Detecting algorithmically generated malicious domain names. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010.
- [29] Miranda Mowbray and Josiah Hagen. Finding domain-generation algorithms by looking at length distribution. In *Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on*, pages 395–400. IEEE, 2014.
- [30] Steven Cheung, Rick Crawford, Mark Dilger, Jeremy Frank, Jim Hoagland, Karl Levitt, Jeff Rowe, Stuart Staniford-Chen, Raymond Yip, and Dan Zerkle. The design of GrIDS: A graph-based intrusion detection system. *UC Davis Computer Science Department Technical Report CSE-99-2*, 1999.
- [31] Christopher Kruegel, Darren Mutz, William Robertson, and Fredrik Valeur. Bayesian event classification for intrusion detection. In *19th Annual Computer Security Applications Conference Proceedings*, pages 14–23. IEEE, 2003.
- [32] Blake Anderson, Daniel Quist, Joshua Neil, Curtis Storlie, and Terran Lane. Graph-based malware detection using dynamic analysis. *Journal in Computer Virology*, 7(4):247–258, 2011.
- [33] Shaojun Zhang and Shanshan Song. A Novel Attack Graph Posterior Inference Model Based on Bayesian Network. *Journal of Information Security*, 02(01):8–27, 2011.

- [34] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 305–316. IEEE, 2010.
- [35] Yong Qiao, Yue-xiang Yang, Jie He, Chuan Tang, and Ying-zhi Zeng. Detecting p2p bots by mining the regional periodicity. *Journal of Zhejiang University SCIENCE C*, 2013.
- [36] John R Goodall. Introduction to visualization for computer security. In *IEEE Symposium on Visualization for Cyber Security (VizSec)*. 2008.
- [37] P. A. Legg. Visualizing the insider threat: challenges and tools for identifying malicious user activity. In *Visualization for Computer Security (VizSec)*, pages 1–7, 2015.
- [38] D. M. Best, R. P. Hafen, B. K. Olsen, and W. A. Pike. Atypical behavior identification in large-scale network traffic. In *IEEE Symposium on Large Data Analysis and Visualization*, pages 15–22, 2011.
- [39] K. Webga and A. Lu. Discovery of rating fraud with real-time streaming visual analytics. In *IEEE Symposium on Visualization for Cyber Security (VizSec)*, pages 1–8, 2015.
- [40] Lili Bai, Jianmin Pang, Yichi Zhang, Wen Fu, and Jiafeng Zhu. Detecting malicious behavior using critical api-calling graph matching. In *Information Science and Engineering (ICISE), 2009 1st International Conference on*, pages 1716–1719. IEEE, 2009.
- [41] Hao Zhang, Danfeng (Daphne) Yao, Naren Ramakrishnan, and Zhibin Zhang. Causality reasoning about network events for detecting stealthy malware activities. *Computers & Security*, 58:180–198, 2016.
- [42] Diane Oyen, Blake Anderson, and Christine M Anderson-Cook. Bayesian net-

- p>works with prior knowledge for malware phylogenetics. In
- AAAI Workshop: Artificial Intelligence for Cyber Security*
- , 2016.
- [43] A. Makandar and A. Patrot. Malware analysis and classification using Artificial Neural Network. In *2015 International Conference on Trends in Automation, Communications and Computing Technology (I-TACT-15)*, volume 01, pages 1–6, 2015.
 - [44] Ashish Saini, Ekta Gandotra, Divya Bansal, and Sanjeev Sofat. Classification of PE Files using Static Analysis. pages 429–433. ACM Press, 2014.
 - [45] C. Liangboonprakong and O. Sornil. Classification of malware families based on N-grams sequential pattern features. In *2013 IEEE 8th Conference on Industrial Electronics and Applications (ICIEA)*, pages 777–782, June 2013.
 - [46] Monire Norouzi, Alireza Souri, and Majid Samad Zamini. A Data Mining Classification Approach for Behavioral Malware Detection. *Journal of Computer Networks and Communications*, 2016:1–9, 2016.
 - [47] R. Canzanese, S. Mancoridis, and M. Kam. Run-time classification of malicious processes using system call analysis. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 21–28, 2015.
 - [48] Guozhu Meng, Yinxing Xue, Zhengzi Xu, Yang Liu, Jie Zhang, and Annamalai Narayanan. Semantic modelling of android malware for effective malware comprehension, detection, and classification. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pages 306–317. ACM, 2016.
 - [49] Annamalai Narayanan, Guozhu Meng, Liu Yang, Jinliang Liu, and Lihui Chen. Contextual weisfeiler-lehman graph kernel for malware detection. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 4701–4708. IEEE, 2016.

- [50] Sanjeev Das, Yang Liu, Wei Zhang, and Mahintham Chandramohan. Semantics-based online malware detection: Towards efficient real-time protection against malware. *IEEE transactions on information forensics and security*, 11(2):289–302, 2016.
- [51] S. Nari and A. A. Ghorbani. Automated malware classification based on network behavior. In *2013 International Conference on Computing, Networking and Communications (ICNC)*, pages 642–647, January 2013.
- [52] M. Zubair Rafique, Ping Chen, Christophe Huygens, and Wouter Joosen. Evolutionary algorithms for classification of malware families through different network behaviors. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 1167–1174. ACM, 2014.
- [53] Aziz Mohaisen, Omar Alrawi, and Manar Mohaisen. AMAL: High-fidelity, behavior-based automated malware analysis and classification. *Computers & Security*, 2015.
- [54] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Are your training datasets yet relevant? In *International Symposium on Engineering Secure Software and Systems*, volume 8978, pages 51–67. Springer, 2015.
- [55] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Brazilian Symposium on Artificial Intelligence*, pages 286–295. Springer, 2004.
- [56] S. Scardapane, D. Comminiello, M. Scarpiniti, and A. Uncini. Online sequential extreme learning machine with kernels. *IEEE Transactions on Neural Networks and Learning Systems*, 26(9):2214–2220, 2015.
- [57] Xinying Wang and Min Han. Improved extreme learning machine for multi-variate time series online sequential prediction. *Engineering Applications of Artificial Intelligence*, 40:28–36, 2015.

- [58] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, 2015.
- [59] Manuel Baena-Garcia, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, Ricard Gavaldà, and Rafael Morales-Bueno. Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams*, volume 6, pages 77–86, 2006.
- [60] E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1):100–115, 1954.
- [61] M. Masud, J. Gao, L. Khan, J. Han, and B. M. Thuraisingham. Classification and novel class detection in concept-drifting data streams under time constraints. *IEEE Transactions on Knowledge and Data Engineering*, 23(6):859–874, 2011.
- [62] Tegjyot Singh Sethi, Mehmed Kantardzic, and Hanquing Hu. A grid density based framework for classifying streaming data in the presence of concept drift. *Journal of Intelligent Information Systems*, 46(1):179–211, 2016.
- [63] J. Lee and F. Magoulès. Detection of concept drift for learning from stream data. In *2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems*, pages 241–245, 2012.
- [64] G. Ditzler and R. Polikar. Hellinger distance based drift detection for nonstationary environments. In *2011 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE)*, pages 41–48, 2011.
- [65] W. Nick Street and YongSeog Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 377–382. ACM, 2001.

- [66] Adam Pocock, Paraskevas Yiapanis, Jeremy Singer, Mikel Luján, and Gavin Brown. Online non-stationary boosting. In *International Workshop on Multiple Classifier Systems*, pages 205–214. Springer, 2010.
- [67] Nikunj Chandrakant Oza. Online ensemble learning. *University of California, Berkeley*, 2001.
- [68] J. Zico Kolter and Marcus A. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research*, 8:2755–2790, 2007.
- [69] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80. ACM, 2000.
- [70] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106. ACM, 2001.
- [71] Jun-seok Lim, Seokjin Lee, and Hee-Suk Pang. Low complexity adaptive forgetting factor for online sequential extreme learning machine (OS-ELM) for application to nonstationary system estimations. *Neural Computing and Applications*, 22(3):569–576, 2013.
- [72] Dong Liu, YouXi Wu, and He Jiang. Fp-elm: An online sequential learning algorithm for dealing with concept drift. *Neurocomputing*, 207:322–334, 2016.
- [73] N.G. Pavlidis, D.K. Tasoulis, N.M. Adams, and D.J. Hand. Lambda-perception - an adaptive classifier for data streams. *Pattern Recognition*, 44(1):78–96, 2011.
- [74] Jialei Wang, Peilin Zhao, and Steven CH Hoi. Exact soft confidence-weighted learning. In *Proceedings of the 29th International Conference on International Conference on Machine Learning*, pages 107–114. Omnipress, 2012.

- [75] H Brendan McMahan. A survey of algorithms and analysis for adaptive online learning. *The Journal of Machine Learning Research*, 18(1):3117–3166, 2017.
- [76] H. Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, and others. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1222–1230. ACM, 2013.
- [77] Dmitri Bekerman, Bracha Shapira, Lior Rokach, and Ariel Bar. Unknown malware detection using network traffic classification. In *Communications and Network Security (CNS), 2015 IEEE Conference on*, pages 134–142. IEEE, 2015.
- [78] Esparza Jose Miguel. Travelling to the far side of andromeda, 2015.
- [79] Nicolas Falliere and Eric Chien. Zeus: King of the bots. *Symantec Security Response*, 2009.
- [80] Frederic Giroire, Jaideep Chandrashekar, Nina Taft, Eve Schooler, and Dina Papagiannaki. Exploiting temporal persistence to detect covert botnet channels. In *Recent Advances in Intrusion Detection*, 2009.
- [81] Pallav Khandhar. Banking botnets persist despite takedowns, 2015.
- [82] Symantec. Internet security threat report, 2015.
- [83] Tom Liston and Ed Skoudis. *On the cutting edge: Thwarting virtual machine detection*. 2006.
- [84] Ralph Langner. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security & Privacy*, 2011.
- [85] Boldizsár Bencsáth, Gábor Pék, Levente Buttyán, and Márk Félegyházi. Duqu: Analysis, detection, and lessons learned, 2012.

REFERENCES

- [86] S. Shin, G. Gu, N. Reddy, and C. P. Lee. A large-scale empirical study of conficker. *IEEE Transactions on Information Forensics and Security*, 7(2):676–690, 2012.
- [87] Ken Chiang and Levi Lloyd. A case study of the rustock rootkit and spam bot. In *Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, HotBots’07, pages 10–10, Berkeley, CA, USA, 2007.
- [88] Mian Zhou and Sheau-Dong Lang. Mining frequency content of network traffic for intrusion detection. In *Proceedings of the IASTED international conference on communication, network, and information security*, pages 101–107, 2003.
- [89] James S. Walker. *A Primer on Wavelets and Their Scientific Applications, Second Edition*. CRC Press, January 2008.
- [90] Christopher Torrence and Gilbert P Compo. A practical guide to wavelet analysis. *Bulletin of the American Meteorological society*, 1998.
- [91] Rongfeng Zhang et al. Spectral decomposition with extracted seismic wavelet. In *2013 SEG Annual Meeting*. Society of Exploration Geophysicists, 2013.
- [92] Genevieve Bartlett, John Heidemann, and Christos Papadopoulos. Using low-rate flow periodicities for anomaly detection: Extended. Technical report, USC/Information Sciences Institute, 2009.
- [93] E. Biglar Beigi, H. Hadian Jazi, N. Stakhanova, and A. A. Ghorbani. Towards effective feature selection in machine learning-based botnet detection approaches. In *IEEE Conference on Communications and Network Security (CNS)*, pages 247–255, 2014.
- [94] W. Aigner, S. Miksch, W. Muller, H. Schumann, and C. Tominski. Visual Methods for Analyzing Time-Oriented Data. *IEEE Transactions on Visualization and Computer Graphics*, 14(1):47–60, 2008.

- [95] Julius O Smith. Mathematics of the discrete fourier transform with audio applications, second edition. W3K Publishing, <http://books.w3k.org/>. ISBN 978-0-9745607-4-8. 2007.
- [96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.
- [97] Marc Weber, Marc Alexa, and Wolfgang Müller. Visualizing time-series on spirals. *IEEE Symposium on Information Visualization*, 2001.
- [98] Richard Lippmann, Joshua W Haines, David J Fried, Jonathan Korba, and Kumar Das. The 1999 darpa off-line intrusion detection evaluation. *Computer networks*, pages 579–595, 2000.
- [99] Roberto Valenti, Nicu Sebe, Theo Gevers, and Ira Cohen. Machine learning techniques for face analysis. In *Machine Learning Techniques for Multimedia*, pages 159–187. Springer, 2008.
- [100] Ronen Feldman. Techniques and applications for sentiment analysis. *Communications of the ACM*, (4):82–89, 2013.
- [101] D. Iliopoulos, C. Adami, and P. Szor. Darwin inside the machines: Malware evolution and the consequences for computer security. *arXiv:1111.2503 [cs, q-bio]*, 2011.
- [102] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4):44, 2014.
- [103] Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Brad Miller, Vaishaal Shankar, Rekha Bachwani, Anthony D Joseph, and J Doug Tygar. Better malware ground truth: Techniques for weighting anti-virus vendor labels. In *Pro-*

- ceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, pages 45–56. ACM, 2015.
- [104] Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2011.
- [105] Jelena Mirkovic and Terry Benzel. Deterlab testbed for cybersecurity research and education. *Journal of Computing Sciences in Colleges*, 28(4):163–163, 2013.
- [106] Steve R Gunn et al. Support vector machines for classification and regression. 1998.
- [107] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, and others. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.
- [108] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.
- [109] Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2011.
- [110] Paulo M. Gonçalves, Silas G.T. de Carvalho Santos, Roberto S.M. Barros, and Davi C.L. Vieira. A comparative study on concept drift detectors. *Expert Systems with Applications*, 41(18):8144–8156, 2014.
- [111] Stephen Grossberg. Adaptive resonance theory: How a brain learns to consciously attend, learn, and recognize a changing world. *Neural Networks*, 37:1–47, 2013.
- [112] Claudio Gentile. A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, 2:213–242, 2001.

- [113] Wei Fan. Systematic data selection to mine concept-drifting data streams. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 128–137. ACM, 2004.
- [114] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. Tracking recurring contexts using ensemble classifiers: an application to email filtering. *Knowledge and Information Systems*, 22(3):371–391, 2010.
- [115] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Identifying suspicious URLs: an application of large-scale online learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 681–688. ACM, 2009.
- [116] Hadley Wickham. Asa 2009 data expo. *Journal of Computational and Graphical Statistics*, 20(2):281–283, 2011.
- [117] G. Ditzler and R. Polikar. Incremental learning of concept drift from streaming imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 25(10):2283–2301, 2013.
- [118] Michael Harries and New South Wales. Splice-2 comparative evaluation: Electricity pricing. *Technical report, The University of South Wales*, 1999.
- [119] Richard P Lippmann, David J Fried, Isaac Graf, Joshua W Haines, Kristopher R Kendall, David McClung, Dan Weber, Seth E Webster, Dan Wyschogrod, Robert K Cunningham, et al. Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*, volume 2, pages 12–26. IEEE, 2000.
- [120] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. On evaluating stream learning algorithms. *Machine Learning*, 90(3):317–346, 2013.

REFERENCES

- [121] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *Journal of Machine Learning Research*, 11:1601–1604, 2010.
- [122] Jose CM Bermudez. Adaptive filtering-theory and applications. 2011.
- [123] Sameeraj Meduri and Rufus Ananth. A survey and evaluation of voice activity detection algorithms: speech processing module. 2012.
- [124] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *Acm computing surveys (CSUR)*, 38(4):13, 2006.

Appendix

A.1 Proof of Theorem 1

Proof: To remind the optimization objective of FTRL-DP:

$$\begin{aligned}
 w_{t+1} &= \underset{w}{\operatorname{argmin}} g_{1:t}^\top w + \lambda_1 \|w\|_1 + \frac{1}{2} \lambda_2 \|w\|_2^2 + \frac{1}{2} \lambda_p \sum_{s=1}^t \gamma^{t-s} \|w - w_s\|_2^2 \\
 w_{t+1} &= \underset{w}{\operatorname{argmin}} \left(g_{1:t}^\top - \lambda_p \sum_{s=1}^t \gamma^{t-s} w_s^\top \right) w + \lambda_1 \|w\|_1 + \frac{1}{2} \left(\lambda_2 + \lambda_p \frac{1 - \gamma^t}{1 - \gamma} \right) \|w\|_2^2 \\
 &\quad + \frac{1}{2} \lambda_p \sum_{s=1}^t \gamma^{t-s} \|w_s\|_2^2
 \end{aligned}$$

Omitting the constant term $\frac{1}{2} \lambda_p \sum_{s=1}^t \gamma^{t-s} \|w_s\|_2^2$, we have:

$$w_{t+1} = \underset{w}{\operatorname{argmin}} z_t^\top w + \lambda_1 \|w\|_1 + \frac{1}{2} (\lambda_2 + \lambda_p r_t) \|w\|_2^2 \quad (\text{A.1})$$

In Equation A.1, $z_t = g_{1:t}^\top - \lambda_p \sum_{s=1}^t \gamma^{t-s} w_s^\top$ and $r_t = \frac{1 - \gamma^t}{1 - \gamma}$. Each component of w contribute independently to the objective function of A.1 hence can be solve separately:

$$w_{t+1,i} = \underset{w_i}{\operatorname{argmin}} z_{t,i} w_i + \lambda_1 \|w_i\|_1 + \frac{1}{2} (\lambda_2 + \lambda_p r_t) \|w_i\|_2^2 \quad (\text{A.2})$$

Note that w_i in A.2 refers to the i^{th} component of w . Let $f(w_i) = z_{t,i} w_i + \lambda_1 \|w_i\|_1 + \frac{1}{2} (\lambda_2 + \lambda_p r_t) \|w_i\|_2^2$. There are two cases:

- If $\|z_{t,i}\|_1 \leq \lambda_1$, we have:

$$f(w_i) \geq -\|z_{t,i}w_i\|_1 + \lambda_1\|w_i\|_1 + \frac{1}{2}(\lambda_2 + \lambda_p r_t)\|w_i\|_2^2$$

$$f(w_i) \geq -\lambda_1\|w_i\|_1 + \lambda_1\|w_i\|_1 + \frac{1}{2}(\lambda_2 + \lambda_p r_t)\|w_i\|_2^2 = \frac{1}{2}(\lambda_2 + \lambda_p r_t)\|w_i\|_2^2 \geq 0$$

$f(w_i)$ achieves the minimum at $w_i = 0$

- If $\|z_{t,i}\|_1 \geq \lambda_1$, $z_{t,i}$ and w_i must have opposite signs at the minimum of $f(w_i)$ as otherwise w_i can always have sign flipped to further reduce $f_i(w_i)$. Therefore, it is equivalent to solving:

$$w_{t+1,i} = \underset{w_i}{\operatorname{argmin}} z_{t,i}w_i - \operatorname{sign}(z_{t,i})\lambda_1\|w_i\|_1 + \frac{1}{2}(\lambda_2 + \lambda_p r_t)\|w_i\|_2^2$$

which achieves minimum at zero gradient or $w_i = -\frac{z_{t,i} - \operatorname{sign}(z_{t,i})\lambda_1}{\lambda_2 + \lambda_p r_t}$

This concludes the proof.

A.2 Proof of Theorem 2

Proof:

As $r_{1:t}(w) = \frac{1}{2} \sum_{s=1}^t \sigma_s \|w - w_s\|_2^2$ is 1-strongly convex (Lemma 3) w.r.t. norm $\|w\|_{(t)} = \sqrt{\sigma_{1:t}}\|w\|_2$, whose dual is $\|w\|_{(t),*} = \frac{1}{\sqrt{\sigma_{1:t}}}\|w\|_2$, applying Theorem 2 in [75], we have:

$$\operatorname{Regret}(w^*) \leq \frac{1}{2\eta_T}(2R)^2 + \frac{1}{2} \sum_{t=1}^T \eta_t \|g_t\|^2$$

In which, $\eta_t = \frac{1}{\sigma_{1:t}} = \frac{1}{\sum_{s=1}^t \gamma^{T-s}} = \frac{(1-\gamma)\gamma^t}{\gamma^T(1-\gamma^t)}$, therefore:

$$\operatorname{Regret}(w^*) \leq 2R^2 \frac{1-\gamma^T}{1-\gamma} + \frac{G^2}{2} \frac{1}{\gamma^T} \sum_{t=1}^T \frac{(1-\gamma)\gamma^t}{(1-\gamma^t)}$$

If we choose $\gamma = 1 - \frac{\ln T}{2T}$, we have:

$$\operatorname{Regret}(w^*) \leq 4R^2 \frac{T}{\ln T} + \frac{G^2}{2} \frac{1}{\gamma^T} \sum_{t=1}^T \frac{\gamma^t}{\sum_{i=0}^{t-1} \gamma^i}$$

As $\frac{\gamma^t}{\sum_{i=0}^{t-1} \gamma^i} \leq \frac{1}{t}$ for $\gamma < 1$, we have:

$$\operatorname{Regret}(w^*) \leq 4R^2 \frac{T}{\ln T} + \frac{G^2}{2} \frac{1}{\gamma^T} \sum_{t=1}^T \frac{1}{t}$$

As $\sum_{t=1}^T \frac{1}{t} \leq 1 + \ln T$ (Lemma 4), we have:

$$\text{Regret}(w^*) \leq 4R^2 \frac{T}{\ln T} + \frac{G^2}{2} \frac{(1 + \ln T)}{(1 - \frac{\ln T}{2T})^T}$$

Therefore:

$$\lim_{T \rightarrow \infty} \frac{\text{Regret}(w^*)}{T} \leq \lim_{T \rightarrow \infty} 4R^2 \frac{1}{\ln T} + \frac{G^2}{2} \frac{(1 + \ln T)}{\sqrt{T}} \frac{1}{\sqrt{T}(1 - \frac{\ln T}{2T})^T} \quad (\text{A.3})$$

As $\lim_{T \rightarrow \infty} \sqrt{T}(1 - \frac{\ln T}{2T})^T = 1$ (Lemma 5), the right hand side of A.3 vanishes when T goes to infinity. Therefore:

$$\lim_{T \rightarrow \infty} \frac{\text{Regret}(w^*)}{T} \leq 0$$

This concludes the proof.

Lemma 3 *Prove that $r_{1:t}(w) = \frac{1}{2} \sum_{s=1}^t \sigma_s \|w - w_s\|_2^2$ is 1-strongly convex w.r.t. norm $\|w\|_{(t)} = \sqrt{\sigma_{1:t}} \|w\|_2$, given $\sigma_1, \sigma_2, \dots, \sigma_t > 0$ and $w_1, w_2, \dots, w_t$.*

Proof: With arbitrary $x, y \in \mathbb{R}^n$, we have:

$$\begin{aligned} r_{1:t}(y) &\geq r_{1:t}(x) + \nabla r_{1:t}(x)(y - x) + \frac{1}{2} \|y - x\|_{(t)}^2 \\ \Leftrightarrow r_{1:t}(y) - r_{1:t}(x) &\geq \nabla r_{1:t}(x)(y - x) + \frac{1}{2} \|y - x\|_{(t)}^2 \\ \Leftrightarrow \frac{1}{2} \sum_{s=1}^t \sigma_s (y + x - 2w_s)^\top (y - x) &\geq \sum_{s=1}^t \sigma_s (x - w_s)^\top (y - x) + \frac{1}{2} \|y - x\|_{(t)}^2 \\ \Leftrightarrow \frac{1}{2} \sum_{s=1}^t \sigma_s (y - x)^\top (y - x) &\geq \frac{1}{2} \sigma_{1:t} \|y - x\|_2^2 \end{aligned}$$

The last inequality is actually an equality. This concludes the proof.

Lemma 4 *Prove that:*

$$\sum_{t=1}^T \frac{1}{t} \leq 1 + \ln T \quad (\text{A.4})$$

Proof: We have:

$$\ln T = \int_{x=1}^T \frac{dx}{x} = \sum_{k=1}^{T-1} \int_{x=k}^{k+1} \frac{dx}{x} \geq \sum_{k=1}^{T-1} \int_{x=k}^{k+1} \frac{dx}{k+1} = \sum_{k=1}^{T-1} \frac{1}{k+1}$$

Therefore:

$$1 + \ln T \geq \sum_{t=1}^T \frac{1}{t}$$

This concludes the proof.

Lemma 5 *Prove that:*

$$\lim_{T \rightarrow \infty} \sqrt{T} \left(1 - \frac{\ln T}{2T}\right)^T = 1 \quad (\text{A.5})$$

Proof:

Change T to $\frac{1}{x}$, and take the log of the left-hand side of A.5, we have:

$$\lim_{T \rightarrow \infty} \ln \left[\sqrt{T} \left(1 - \frac{\ln T}{2T}\right)^T \right] = \lim_{x \rightarrow 0} \frac{\ln \left(1 + \frac{x \ln x}{2}\right) - \frac{1}{2}x \ln x}{x}$$

Applying L'Hôpital's rule, we have:

$$\lim_{T \rightarrow \infty} \ln \left[\sqrt{T} \left(1 - \frac{\ln T}{2T}\right)^T \right] = \lim_{x \rightarrow 0} -\frac{x \ln x + x(\ln x)^2}{2(2 + x \ln x)}$$

As $\lim_{x \rightarrow 0} x(\ln x)^n = 0$ for all integer n , we have:

$$\lim_{x \rightarrow 0} -\frac{x \ln x + x(\ln x)^2}{2(2 + x \ln x)} = 0$$

Therefore:

$$\lim_{T \rightarrow \infty} \sqrt{T} \left(1 - \frac{\ln T}{2T}\right)^T = 1$$

This concludes the proof.