Titre: Title:	Better Understanding Malware through a Deep Analysis of the Infection Chain
Auteur: Author:	François Labrèche
Date:	2021
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Labrèche, F. (2021). Better Understanding Malware through a Deep Analysis of the Infection Chain [Thèse de doctorat, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/6638/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/6638/
Directeurs de recherche: Advisors:	Gianluca Stringhini, & Jose Manuel Fernandez
Programme: Program:	Génie informatique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Better Understanding Malware through a Deep Analysis of the Infection Chain

FRANÇOIS LABRÈCHE
Département de génie informatique et génie logiciel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie informatique

Mai 2021

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Better Understanding Malware through a Deep Analysis of the Infection Chain

présentée par **François LABRÈCHE**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

François-Raymond BOYER, président
José FERNANDEZ, membre et directeur de recherche
Gianluca STRINGHINI, membre et codirecteur de recherche
Benoit DUPONT, membre
Juan CABALLERO, membre externe

ACKNOWLEDGEMENTS

I would like to thank Eve, for supporting me and being by my side during the writing of this thesis.

To all my lab colleagues, these years would have been considerably duller without you, and it has been a pleasure to share this time with you. I would like to thank my supervisor José Fernandez, for his insight and support, and my co-supervisor Gianluca Stringhini, for his support and for giving me great opportunities. I would like to thank the people at ESET for giving a hand in analyzing binaries, namely Joan, Thomas, Matthieu, Alexis and Jean-Ian. I would also like to thank Simon Guigui, who did considerable work on the experiment cluster of the lab, and made some of the experiments in this thesis possible.

I would like to thank the people I met during my brief stay at University College London, and the lovely adventures we had in London. I thank Enrico Mariconti for his help in our project exploring the targeted victim profiles of malicious downloaders. I would also like to thank everyone involved in the POISED project: Shirin Nilizadeh, Alireza Sedighian, Ali Zand, José Fernandez, Christopher Kruegel, Gianluca Stringhini and Giovanni Vigna.

For their financial support throughout the years, I would like to thank Fonds de recherche du Québec - Nature et technologies (FRQNT), Fondation de Polytechnique, ESET, and my supervisors.

RÉSUMÉ

De nos jours, l'Internet fait partie de la vie courante d'une grande majorité de personnes, et de plus en plus d'entreprises l'utilisent à travers leurs activités quotidiennes. De plus, de nombreuses institutions importantes, telles que les hôpitaux, se modernisent et implémentent des systèmes se connectant à l'Internet. Les logiciels malveillants ont donc maintenant un large éventail de victimes potentielles, et ont la possibilité de causer des dommages considérables dans plusieurs entreprises. Bien que les firmes d'antivirus développent des signatures ainsi que des heuristiques de détection pour les logiciels malveillants, les cybercriminels améliorent constamment leurs logiciels afin de contourner les logiciels de sécurité. Ces logiciels malveillants représentent donc toujours, à ce jour, une grande menace.

Dans cette thèse, nous analysons la chaîne complète d'infection d'exploitation par logiciel malveillant à travers ses différentes étapes, dans le but de mieux comprendre l'écosystème des acteurs malveillants et leurs opérations.

Premièrement, nous mettons l'accent sur l'attraction de victimes en établissant des communautés d'intérêts dans les réseaux sociaux et en classifiant les messages malveillants selon leurs sujets et leur chemin de diffusion à travers ces communautés. Les travaux antérieurs ont mis l'accent sur la construction de modèles de détection utilisant des caractéristiques issues du contenu de messages ou des caractéristiques du compte, ce qui peut être contourné par les attaquants en modifiant leurs messages malveillants en conséquence. Nous présentons une nouvelle approche détectant les spams sur les réseaux sociaux en modélisant la façon dont un message circule à travers des communautés. Nous soutenons que la façon dont un message légitime circule à travers les réseaux sociaux est plus difficile à simuler pour les attaquants.

Deuxièmement, nous extrayons les redirections web se produisant lorsqu'un utilisateur accède à un lien malveillant, puis classifions la page web malveillante fournissant la charge malveillante à l'utilisateur, c'est-à-dire le kit d'exploitation, selon ces chaînes de redirection. Les travaux antérieurs ont construit des modèles de détection de binaires de logiciel malveillant et de kits d'exploitation en utilisant les caractéristiques de leur contenu et leur comportement. Nous présentons une nouvelle approche pour l'identification de famille de kits d'exploitation en utilisant seulement des caractéristiques extraites de la chaîne de redirection menant à celle-ci. Avec cette approche, nous fournissons de nouvelles informations sur quelles familles de kits d'exploitation suivent un modèle identifiable dans ses chaînes de redirection précédentes, et lesquelles semblent plutôt suivre le modèle commercial d'exploit en tant que service.

Troisièmement, suite aux redirections vers la page malveillante, un logiciel de type téléchargeur

est fréquemment installé, ayant comme seul but de télécharger des logiciels malveillants additionnels. Dans ce contexte, notre but est d'établir quel profil utilisateur est visé par les cybercriminels, en identifiant le lien entre les caractéristiques de l'utilisateur infecté et le logiciel malveillant téléchargé par le téléchargeur. Nous construisons un système de tests automatisé afin d'exécuter et analyser des téléchargeurs malveillants, utilisant des machines virtuelles avec des caractéristiques variables. Ce dernier nous a permis d'établir quelle caractéristique d'une machine a un impact sur le comportement d'une famille de téléchargeur, c.-à-d., quelle famille et quel type de logiciel malveillant est envoyé au téléchargeur quand celui-ci est exécuté sur une machine virtuelle différente. Ainsi, nous cherchons à fournir de nouvelles informations sur le comportement et le fonctionnement interne de la vente de machines infectées.

Nos résultats montrent tout d'abord que les utilisateurs de réseaux sociaux forment des communautés centrées autour de sujets de discussion spécifiques, et que ces communautés peuvent être identifiées, à travers une combinaison de traitement automatique du langage naturel et de méthodes de partitionnement de graphes. Celles-ci ont été utilisées avec succès dans un classificateur afin de prédire les messages malveillants, en tirant parti des chemins que les messages empruntent à travers ces communautés. Notre modèle, entraîné sur un ensemble de données de 1.3M de messages collectés sur le réseau social Twitter, obtient une haute précision et un haut rappel, et est efficace dans l'identification de messages de spam.

Deuxièmement, notre analyse de chaînes de redirection web menant aux kits d'exploitation fournit de nouvelles informations sur les campagnes associées à divers kits d'exploitation au fil du temps. Nous montrons que certaines familles de kits d'exploitation peuvent être identifiées avec haute précision selon leurs chaînes de redirection web. Nous observons également que certaines autres familles ne peuvent être identifiées que lorsque la date est prise en compte, laissant entendre que le kit d'exploitation est utilisé dans une seule campagne dans le temps.

Finalement, nous construisons un système de tests automatique pour les binaires malveillants, où la localité, le système d'exploitation, la session du navigateur, la disposition de touches du clavier et la langue d'affichage sont configurables, et ce dans le but d'identifier les caractéristiques clés influençant le comportement du binaire testé. En utilisant ce système, nous présentons une expérimentation sur des téléchargeurs malveillants pour une période de 12 mois, où les caractéristiques de la machine exécutant le téléchargeur sont liées à la charge malveillante téléchargée. En utilisant des analyzes de variance et la détection de points de rupture sur des séries temporelles d'infections sur nos machines, nous identifions plusieurs caractéristiques de profils de machine liés à des familles de charges malveillantes spécifiques.

ABSTRACT

Nowadays, the Internet is part of a large majority of people's daily life, and more and more businesses use it in their daily activities. Moreover, many important institutions, such as hospitals, are modernizing themselves and implementing systems that connect to the Internet. Thus, malicious software, i.e., malware, now has a large pool of potential victims, and has the potential to do considerable damage in many businesses. Although antivirus companies develop malware signatures and detection heuristics to address this issue, cybercriminals constantly update their malware in order to evade security software. Thus, to this day, malware still presents a large threat.

In this thesis, we analyze the entire malware exploitation infection chain through its different steps in order to better understand the ecosystem of malicious actors and their operations.

First, we focus on the attraction of victims by establishing communities of interests in social networks and classifying malicious messages according to their topics and diffusion paths through these communities. Previous research focused on building detection models using features derived from message content or account caracteristics, which can be circumvented by attackers by modifying their malicious messages accordingly. We present a novel approach which detects spam messages on social networks by modeling the way in which a message travels through communities. We argue that the way a legitimate message spreads through online social networks is harder to simulate for attackers.

Second, we extract the web redirections occurring once a user accesses a malicious link, and we then classify the malicious web page sending the malicious payload to the user, namely the exploit kit, according to these redirection chains. Previous research built detection models for malware binaries and exploit kits using their content features and their behavior. We present a novel approach at identifying an exploit kit family by using solely features extracted from the redirection chain leading to it. With this approach, we provide insights into which exploit kit families follow an identifiable pattern in its prior web redirections, and which ones appear to employ the Exploit-as-a-Service business model.

Third, following the redirections to the malicious webpage, a *downloader* software is often installed, with the sole purpose of downloading additional malware. In this context, we establish which user profile is targeted by cybercriminals, by identifying the link between the infected user's characteristics and the malware downloaded by the downloader. For that purpose, we build an automated testing framework to run and analyze malicious downloaders, using virtual machines with varying characteristics. It helped us identify which feature of a

machine impacts the behavior of a family of downloader, i.e., what family and type of malware is sent to the downloader when run on a different virtual machine. Thus, we provide valuable new insights into the behavior and inner workings of the sale of infected machines.

Our results first show that users form communities on social networks centered around specific topics of discussion, and that these can be identified, through a combination of natural language processing and graph clustering methods. These have been employed successfully in a classifier to predict malicious messages, by leveraging the path that these messages take through them. Our model, trained on a dataset of 1.3M messages collected from the Twitter social network, obtains high precision and recall and is effective at identifying spam messages.

Second, our analysis of web redirection chains leading to exploit kits provides some insights into the campaigns associated with various exploit kits through time. We show that some exploit kit families can be identified with high accuracy according to their web redirection chains. We also observe that other families can only be identified when considering the time, hinting at the fact that the exploit kit is used in a single campaign in time.

Finally, we build an automated testing framework for malicious binaries, where the location, the operating system, the browser session, the keyboard layout and the display language are configurable in order to identify the key features affecting the behavior of the tested binary. Using this framework, we present a 12-month period of malicious downloader experimentation, where the characteristics of the machine running the malicious downloader is linked to the downloaded payload. Using variance analyses and changepoint detection on time series of our machine infections, we identify multiple features of the machine profiles linked to specific malicious payload families.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

API	Application program interface
AMI	Adjusted Mutual Information
ARI	Adjusted Rand Index
AIS	Artifical Immune Systems
AV	Antivirus
BaaS	Botnet-as-a-Service
CaaS	Cybercrime-as-a-Service
C&C	Command and control
CFG	Control Flow Graph
CSS	Cascading Style Sheets
DaaS	Deception-as-a-Service
DGA	Domain Generating Algorithm
DNS	Domain Name System
EaaS	Exploit-as-a-Service
EPaaS	Exploit-Package-as-a-Service
FFSN	Fast-Flux Service Network
FN	False negative
FP	False positive
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDS	Intrusion detection system
IP	Internet Protocol
ISP	Internet Service Providers
k-NN	k-nearest neighbors algorithm
LDA	Latent Dirichlet allocation
MAC	Media Access Control
NLP	Natural Language Processing
OCR	Optical Character Recognition
OpCode	Operation Code
OS	Operating System
PE	Portable Executable
PPI	Pay-per-install
PUP	Potentially Unwanted Programs

P2P	Peer-to-peer
RaaS	Ransomware-as-a-Service
RAT	Remote Access Trojan
ROC	Receiver Operating Characteristic
SEO	Search Engine Optimization
SMO	Sequential Minimal Optimization
SMOTE	Synthetic Minority Over-sampling Technique
SSH	Secure Shell
SVM	Support Vector Machines
TaaS	Traffic-as-a-Service
TCP	Transmission Control Protocol
TDS	Traffic direction system
TFIDF	Term Frequency-Inverse Document Frequency
TLS	Transport Layer Security
TN	True negative
TP	True positive
TRaaS	Traffic-Redirection-as-a-Service
URL	Uniform Resource Locator
VM	Virtual Machine
VPN	Virtual Private Network
WINE	Worldwide Intelligence Network Environment

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

1.1 Context

Computers have nowadays become a major aspect of businesses throughout various industries around the globe. In recent years, the Internet has seen its users increase dramatically. As of this writing, there are around 4.6 billion Internet users worldwide [1]. Thus, with more computers controlling more aspects of businesses and frequently being connected to the Internet, there is an increasingly higher number of potential victims for cybercriminals to attack. These attacks take many forms and have different goals when infiltrating a victim's machine. Although not all cyber-attacks monetize their attacks, malicious software, i.e., malware, generally employ a number of different monetization schemes. These include the theft and reselling of information, the use of infected machines to send unsolicited messages, i.e., *spam*, attacking a target and producing a denial of service, extorting the user by encrypting their files and asking for a ransom, installing a fake antivirus service and demanding a payment, or simply using the infected machine to distribute other malicious files or host malicious content [2]. Malware has also been associated with worst patient outcomes in hospitals where an attack occurred in the previous months [3].

Malicious software is a global problem, affecting users from all around the globe each year. Kaspersky, a large antivirus software company, reports tens of thousands of infected machines each day [4]. Research has been made on specific aspects of malicious infections, as well as specific types of malware. However, protecting against malware remains a challenge, as more industries start to modernize their systems and incorporate in their activities an increasing number of computer software connected to the Internet, while criminals constantly update their malicious software to bypass current detection techniques.

In Canada alone, 71% of businesses reported having their operations affected by cyber security incidents in 2019 [5], with only 41% having mandatory cybersecurity training for all employees. In particular, the banking institutions, universities and governments are still the target of a high number of cyber attacks. In 2018, the average cost of investigating an attack and deploying a successful remediation, for Canadian organizations, was 9,250,000$ [6]. Globally, the most expensive type of cyber attack for organizations remain malware attacks [6], which have increased 11% in cost from 2017 to 2018.

1.2 Definitions and Concepts

Malware is defined as any software or code that is designed to cause damage or be harmful to computers, servers, users or computer networks. Malware can take many forms and affect the machine and the user in a number of ways. The most common variants of malicious software are :

Definition 1 Viruses *are malware that copy themselves on the computer and other processes, and spread while causing harm.*

Definition 2 Worms *are malicious software that aim at spreading to a maximum of other computers, often harming the computer or user while doing so.*

Definition 3 Trojans*, or trojan horses, infect the computer without the user's knowledge, by hiding or masking its malicious purpose. It often extracts confidential information from the machine once it's infected. A trojan downloader is a variant which downloads one or more additional malware once it has infected the machine.*

Definition 4 Ransomware *is a type of malware that, once inside a machine, encrypts files and asks the user for a ransom in exchange for the decryption key.*

Definition 5 *A* rootkit *is a malware which hides itself from the machine and the user, and gives its operator a root access to the infected machine.*

Definition 6 Spyware *consists in any malware which monitors the user's activity and collects information, and then sends the collected data back to its operator.*

Definition 7 Adware*, or Potentially Unwanted Programs (PUP), is unwanted software which inserts advertisements on the user's computer screen and into other software, generating ad revenue for the operator.*

Definition 8 Cryptominers *are malicious software which uses the machine's resources to mine cryptocurrencies without the user's consent.*

Definition 9 Exploit kits *are software that run on malicious websites and fingerprint the visiting users' machines in order to serve them exploits targeting vulnerable software on their machines. An exploit kit will generally exploit a user in order to infect them with any of the above malware variants.*

One of the most common ways of infecting users is through *drive-by downloads*, which consists in making a user unknowingly download a malicious software. This software subsequently executes itself without the user's consent. The drive-by download infection scheme typically involves multiple steps. An infection will first need to attract victims. This can be achieved by serving malicious advertisements or malicious code on a website, or by sending *spam* messages, containing links to malicious content. In this work, this will be referred to as the *Attraction* step of the infection.

Once the victim comes into contact with malicious content, they are typically redirected through a multitude of servers, be it advertising markets, a Traffic direction system (TDS), i.e., a server that automatically redirects users based on selected features, or a network of compromised machines. This step of the infection will be referred to as the *Redirection* step.

The redirection path typically ends at a malicious website running an *exploit kit*. Following the fingerprinting of the machine by the exploit kit, the malicious website exploits the victim according to their machine's specifications, and infects them with malicious software. This step will be referred to as the *Infection* step.

The infection potentially only consists in a *downloader* software, also known as *dropper*, i.e., a software whose sole purpose is to download additional malicious software. Finally, once the malicious software is installed on the machine without the user's consent, it is used in the *monetization* step. In this step, the cybercriminals monetize the infection, e.g., by sending spam messages, stealing information, utilizing it for denial of service attacks or extorting the user. Figure 1.1 illustrates these different steps in an infection. It is worth noting that the different steps of the malware infection chain can be managed either by the same or a different set of actors.

The criminals monetizing computer infections can notably use what is called Pay-Per-Install (PPI) services to acquire victims. These services revolve around infecting users and then selling the access to the infected machine to make a profit. Research by Caballero *et al.* [7] has shown that actors from many malware families use these services to buy infected machines.

The Attraction step can consist in malicious advertisements, compromised legitimate websites, malicious or spam messages, and any other platform permitting the distribution of content to users. In this work, we concentrate on infections where the Attraction step involved the spread of malicious messages or links through social networks by malicious actors. Social networks are websites permitting users to connect and share with each other through various means. They have risen in popularity in the previous years, with now 3.6 billion users in 2021 [8], compared to 0.97 billion users in 2010. Given their goal of connecting many users together for them to interact and share messages, they represent an ideal platform to

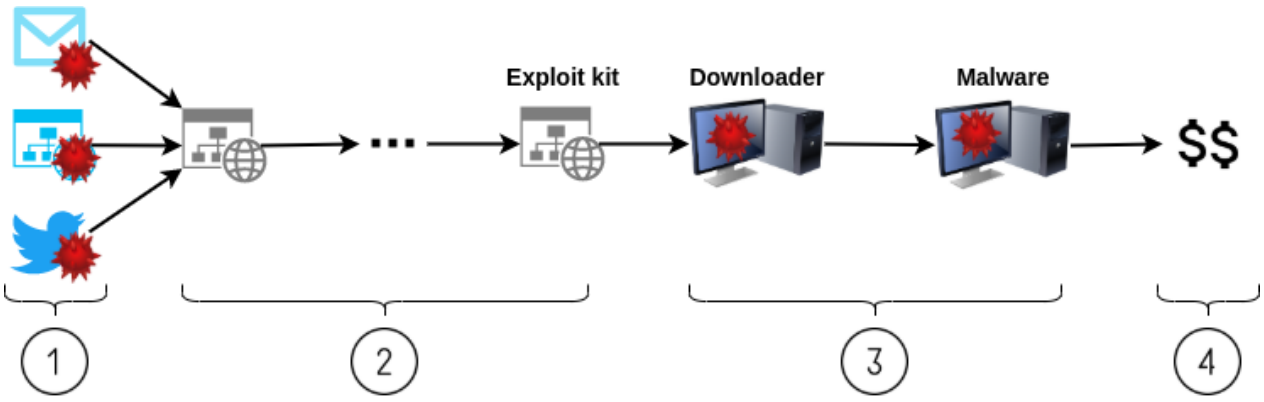


Figure 1.1 An illustration of the infection chain. 1) Initially, in the Attraction step, the victim user accesses a malicious link through an email, a website, or social networks. 2) They are then, in the Redirection step, redirected through a number of intermediate websites, terminating on a page hosting a malicious exploit kit. This exploit kit fingerprints the victim's machine, and exploits a vulnerable component in order to send to the user a malicious downloader. This malicious downloader then downloads one or more malicious payloads, which infect the machine. This consists in the 3) Infection step. Finally, in the 4) Monetization step, the payload(s) now present on the machine exploit the user to their benefit, be it by extorting him, stealing valuable data or sending spam and malicious payloads to other machines, among others

send unsolicited messages to many users, and are as such used by malicious actors in various ways. On Twitter, a popular social network where users can send short messages of up to 280 characters, these messages consist in *tweets*, and users can connect to other users, either by *following* them or being *followed*, i.e., being *friends*. Twitter also features *Hashtags*, which consist in a method of specifying a tag or a topic to a message. In social networks, attackers might use fake accounts, called *Sybil* accounts or bots, as well as use compromised accounts of legitimate users to send malicious messages and infect users. Our work will aim at detecting malicious messages in social networks regardless of the attack vector, with our implementation being made to work with Twitter.

Depending on the source of the malicious attack, the Redirection step will be defined by various HTTP redirections through intermediary websites, starting with the initial website, originating from the previous attraction step, and ending with possibly an exploit kit, and the final infection webpage. These redirections happen automatically over a short period of time, with each redirection taking at most a couple of seconds to happen. These are executed without any user interaction, and possibly without the knowledge of the user. In this work, machine learning techniques will be employed to classify exploit kits according to the redirection chain. Machine learning techniques are used by many researchers in various

fields of study, and their use is growing considerably in computer security. This work will make use of classification, which is a form of machine learning that aims at creating a model that maps the input data to specific categories through a mathematical function, using features identified in the dataset.

Finally, the Infection step will be the download of malware without the user's knowledge or consent. In this work, infections through downloader software will be observed, in the hopes of establishing the impact of the user's profile and the machine's configuration on the payload, i.e., the actual infection, downloaded by the downloader. In this work, we consider a malicious downloader as any software compiled to run on one or more operating systems that include a functionality which downloads a malicious executable binary payload from a Command and Control (C&C) server. A C&C server is a malicious server managed by an attacker that controls one or more distant malware on infected machines. When an infection includes the use of a downloader, the C&C will control the downloader on the various machines, i.e., send commands and files to it, according to the will of one or more cybercriminals. This cybercriminal might notably make use of PPI networks to target victim machines.

Our approaches presented in Chapter 4 and Chapter 5 will employ machine learning in order to build models capable of identifying a specific malicious item in the infection chain. Our approaches will make use of both supervised and unsupervised machine learning. Supervised machine learning consists in the use of an algorithm to build a mathematical model from a number of features, in order to predict a target value, while unsupervised machine learning aims at identifying patterns and grouping the data based on its features.

In order to assess the effectiveness of our supervised machine learning classifiers, we will measure their predictions based on multiple metrics. A True positive (TP) is a data point correctly classified as positive, while a False positive (FP) is a data point incorrectly classified as positive. A True negative (TN) is a data point correctly classified as negative, and finally a False negative (FN) is a data point incorrectly classified as negative. The total number of data points with a positive label is identified as P, while the total number of data with a negative label is identified as N. Our metrics are the following :

Definition 10 *The* accuracy *consists in the number of correctly identified instances, i.e.,* $(TP + TN)/(P + N)$.

Definition 11 *The* TP Rate*, or* recall*, is the number of positive instances correctly classified, i.e.,* TP/P.

Definition 12 *The* FP Rate *is the error rate of the model, i.e.,* $(FP)/(FP + TN)$.

Definition 13 *The* precision *consists in the number of correctly identified positive instances, i.e.,* $TP/(TP + FP)$.

Definition 14 *The* F1-score*, or* F-measure*, is the harmonic mean of the precision and the recall metrics, i.e.,* $2 * precision * recall/(precision + recall)$.

Definition 15 *The* Receiver Operating Characteristic (ROC) *curve shows the increase in the TP rate compared to the FP rate. The area under the curve consists in the accuracy. This curve is useful for comparing different models.*

1.3 Elements of the Problem

In this work, we aim at better understanding the malware ecosystem and the different actors involved in an infection, through an analysis of the various steps in the infection.

We first inspect the attraction step of the infection chain. Criminals profit from attracting a number of victims through social networks, among other techniques. The detection of these actors remains a challenge, due to the fact that attackers adapt themselves according to the improvements made in the detection techniques and evolving technologies. These detection techniques either detect malicious messages or malicious user accounts. Malicious users can also consist in automatically generated fake accounts, i.e., *bot* or *Sybil* accounts, or compromised legitimate accounts. Many of the current approaches detect malicious content by its characteristics, i.e., by analyzing malicious messages or accounts and their relations between themselves, and employing a machine learning approach. They create a detection model based on these features and train it on a specific dataset, or on datasets obtained from security vendors or previous research. While some of these approaches have a high detection accuracy on their respective datasets, attackers can further modify their approach to mimic legitimate accounts and evade detection. It is possible for an attacker to evade a detection model built on features of malicious accounts or messages by modifying its account features or message features used by the model. If attackers do not have access to the model, they can try modifying their account features at random until the model does not identify it as malicious. In the past ten years, there has been a rise in adversarial machine learning [9–11] applied to the evasion of detection models, where an attacker *poisons* part of the training or testing dataset used by a detection model in order to appear legitimate, or uses machine learning to optimize its attacks against the built model.

This results in an arms race between defenders and attackers. Recent research [12,13] showed how attackers can successfully evade both Sybil-based defenses and account-based ones.

Once a user has visited a malicious link, in the majority of cases, a series of Web redirections occur. These can be the result of passing through ad brokers and ad exchanges, in the case of an infection by a malicious advertisement, or it can be the result of passing through other networks of infected machines in the case of a malicious actor trying to hide its attacking domain. Attackers also use TDS, which consist in servers that direct the incoming traffic according to the need of the attacker, e.g., to send the victims to a malicious website based on their location or their system's configuration. Redirections can render the detection of the infection by antivirus software and detection tools more difficult, since it gives criminals more flexibility by having to only update part of the malicious infrastructure to evade the blacklist

of a particular domain or the detection of malicious code running on one website of the chain. However, modifying the entire redirection chain is a complex task and takes more effort from the criminal, while also possibly not being feasible for them. Thus, a detection system focusing on the infection's redirections and the characteristics of the malicious infrastructure might prove more resilient than current approaches identifying malicious websites with signatures, reputation scores or domain name blacklists. The Web redirection chain, unless captured directly on the machine through the browser or the software making the request, cannot be directly extracted from network captures. To use the redirection chain as a detection feature, one must first extract all requests from a network capture and then recreate the redirection by analyzing the various request headers and content, and their respective response content. Thus, redirection chains are not used in the classification of exploit kits, and are rarely used in the detection of malicious software, due to the complexity in reconstructing them.

Once the user is infected as a result of visiting a website, the first malicious software installed on their machine is often a *downloader*. This software is used to install one or more additional malicious files. It can also do reconnaissance work on the machine, and analyze the characteristics of the user and the machine to send to its C&C server, which will then dispatch the malicious files. Other names given to these types of files in the literature are *trojan downloaders* or *droppers*. Malware binaries frequently hide and obfuscate their code, and employ various techniques to hinder debugging of their software by researchers. More so, the distant C&C servers communicating with these malicious downloaders are also not accessible, unless the physical servers have been seized by law enforcement and provided for analysis. Thus, the code sending commands to the malware cannot be analyzed. Although researchers have analyzed particular samples of malicious downloaders, it is generally not established how much of an impact the user and machine characteristics have on how cybercriminals target victims through the use of malicious downloaders. Thus, establishing if there exists a link between the user profile and the malicious files downloaded by the downloader can help in better understanding the market of malicious installs.

1.4 Research Objectives

This work's main research objective consists in better understanding malware and the actors involved in malicious infections, by establishing the entire chain of malicious infections. This research objective can be separated into three research avenues, each one aiming at better understanding one part of the infection chain.

Firstly, we investigate the attraction step by identifying how malicious messages attract victims through social networks. This first avenue's research objective is to detect malicious

messages in social networks through their paths of diffusion in communities of interest. It is divided into the following sub objectives :

1. Detect communities of interests in social networks.
2. Employ machine learning to detect spam messages according to their paths of interest.

Secondly, once victims have accessed a malicious URL in the attraction step, we aim at identifying where the Web redirection leads, i.e., to what exploit kit the victim is sent. The second avenue's research objective consists in classifying exploit kits according to their Web redirection chain. It is divided into the following sub objectives:

1. Develop a method to recreate the redirection chain of a malware infection from a network trace.
2. Use machine learning to classify exploit kit families according to their Web redirection chain and establish the most important features of malicious Web redirection chains for different exploit kit families.

Thirdly, once the redirection step has occurred, the user downloads a malicious file in the infection step. In this work, we aim at establishing what malware is sent to the victim when this infection occurs. The third avenue's research objective is to establish if there exists a correlation between the profile of a user and the malicious payload of a downloader. It is divided into the following sub objectives:

1. Establish an automated testing infrastructure using customized virtual machines to capture downloader activity.
2. Establish if a correlation between the user profile and the final malicious payload exists.

1.5 Thesis Outline

We will first present, in Chapter 2, a general overview of the malware infection ecosystem, its different components and how they interact with each other. Secondly, Chapter 3 will describe the organization of our various chapters, including our submitted paper. Following this, Chapter 4 will present POISED, our approach in detecting spam through their path of diffusion in communities of interest. Chapter 5 will then describe our approach in classifying exploit kit families according to their Web redirection chain. Chapter 6 will follow by presenting our third project : building an automated downloader testing framework, and using it to establish the link between a user profile and the payload downloaded by a downloader. This chapter will present our submitted paper : *Shedding Light on the Targeted Victim Profiles of Malicious Downloaders*. Finally, a general discussion of our various findings will be shown in Chapter 7, detailing the links between our various chapters. We will conclude with a brief summary in Chapter 8, along with the limitations and future works.

CHAPTER 2 LITERATURE REVIEW

In this work, we explore different elements of the malware infection chain in order to build frameworks which help in better understanding how this ecosystem works. Over the years, the malware ecosystem has evolved to become significantly more complex, now including a large number of different entities which can interact with one another to execute sophisticated attacks. With the rise of Cybercrime-as-a-Service (CaaS), malicious actors now have much more flexibility when performing attacks, and can simply purchase the necessary missing pieces of their attack. Malware in itself has also become increasingly more complex over the years, where the size of malware and the estimated effort into making them have grown exponentially by one order of magnitude every decade for the last 30 years [14]. The code found in malware samples has also increased in quality, now having code quality and maintainability similar to regular software [14].

While we frequently associate a typical cybercriminal to someone directly causing harm to a user, for example by stealing information or through extortion, a number of actors are involved in the malware ecosystem. Some are in direct contact with the victim, while others operate in the background, only interacting with other cybercriminals. One of the most known cases of the latter is a malicious actor lending infected machines he controls, i.e., a botnet, to other cybercriminals. The botnet might be used to redirect traffic in other attacks, or to send spam emails, where the users of the infected machines might not be impacted by the operation.

Many actors can now be involved in a single attack, possibly having little contact between themselves. The structure of cybercrime groups and organizations is still unclear, and it is unknown if true organized crime dominates this ecosystem [15, 16], and if so, in what proportion. The trust relations between cybercriminals and criminals from other fields can vary, or be nonexistent.

Previous research has explored different components of this vast ecosystem, and identified key aspects of malware campaigns and their value chain. In the following sections, we will first give a general overview of the malware ecosystem, followed by a more in-depth look at specific parts of this ecosystem: the attraction, the redirection, exploit kits and the monetization. Finally, we will present a subset of currently used malicious attacks and how they fit into this ecosystem.

2.1 Ecosystem Overview

As mentioned in Chapter 1, a malicious infection goes through several steps in order to infect a victim. These steps fit into the malware infection ecosystem, which also includes a monetization step where cybercriminals aim at making profits from infections, such as selling information and credit card numbers, extorting users, or laundering money. The different elements of this ecosystem are not controlled by a single actor, but by a number of different actors and groups of actors. Figure 2.1 details the ecosystem and its different steps and actors.

The infection chain starts with the user visiting a malicious link received through the attraction step: by receiving a spam email, a spam message on a social network, a malicious advertisement or simply by visiting a malicious website. This step will be controlled by a malicious actor, which we will refer to as the *attraction campaign manager*. The detection of spam messages in social networks will be covered in Chapter 4.

Once the user visits a link, a number of redirections occur, through TDS, botnets, advertisement markets, or legitimate websites. The actor(s) controlling these redirection infrastructures will be referred to as *redirection provider*. While this redirection is part of the attack, the redirection service might be legitimate, and used maliciously by a cybercriminal. Chapter 5 will explore these redirections up to the exploit kit, controlled by the *Exploit Kit Operator*. This malicious software fingerprints the user's machine in order to redirect them to a final landing page, which will infect their machine according to the vulnerabilities present.

The user ends his infection cycle on a landing page, which sends the malware or malicious downloader to their machine. In the case of a malicious downloader, the cybercriminal controlling it will sell the access to the user's machine to other malicious parties and install additional malware. Different users will receive different payloads through downloaders, which will be covered in Chapter 6.

The actors involved in each step can be completely independent and working together on an attack through an offering of their services, in order to gain a profit. Different groups and actors fit into a single attack through CaaS. The needed expertise or tools in a malicious attack can be purchased as a service [17–21]. Among others, the following services are being offered through dark markets [21]:

- Exploit-as-a-Service (EaaS): Development of an exploit according to the attacker's needs
- Exploit-Package-as-a-Service (EPaaS): Packaging of exploits into an exploit kit

- Deception-as-a-Service (DaaS): Generation of fake websites, emails or software to lure victims
- Payload-as-a-Service (PaaS): Providing malicious payloads
- Botnet-as-a-Service (BaaS): Providing the means to build a botnet
- Traffic-Redirection-as-a-Service (TRaaS): Redirecting traffic to a website hosting an exploit kit
- Traffic-as-a-Service (TaaS): Generating traffic for a given target

In the malware infection chain identified in Figure 2.1, DaaS can be used in the attraction step by the attraction campaign manager to lure unsuspecting victims, while TRaaS and TaaS correspond to the redirection step. EaaS and EPaaS are used by malicious actors to build the exploit kit that follows the redirections. BaaS will provide attackers with the means to build a botnet. A botnet consists in a tool which can be used to carry out other steps of the attack, such as redirecting traffic, hosting exploit kits or sending spam. PaaS represents any payload provided to an attacker, which might include an exploit kit, but can also be part of payloads given through the malicious landing page. Additionally, a number of other services exist to provide support services for website hosting, payload obfuscation, payload protection from Antivirus (AV) software and money laundering. These fit into the monetization step of the ecosystem, and will not be covered in this work.

In this chapter, we will give an overview of the overall malware ecosystem: its presence in dark markets, how actors interact with one another, and the value chain associated with different malicious operations. The following chapters will delve into specific steps of the infection chain in detail.

2.2 Attraction

The attraction of victims is done through many media: spam emails, spam messages on social networks, bought or hijacked advertisements, and malicious websites.

2.2.1 Evolution of Spam Detection

Already in 2003, Hinde *et al.* [22] highlight the nuisance caused by spam, as the storage cost of email servers increases, employee productivity diminishes, and the bandwidth of networks is clogged by spam emails. They highlight legislation which has been put in place to combat

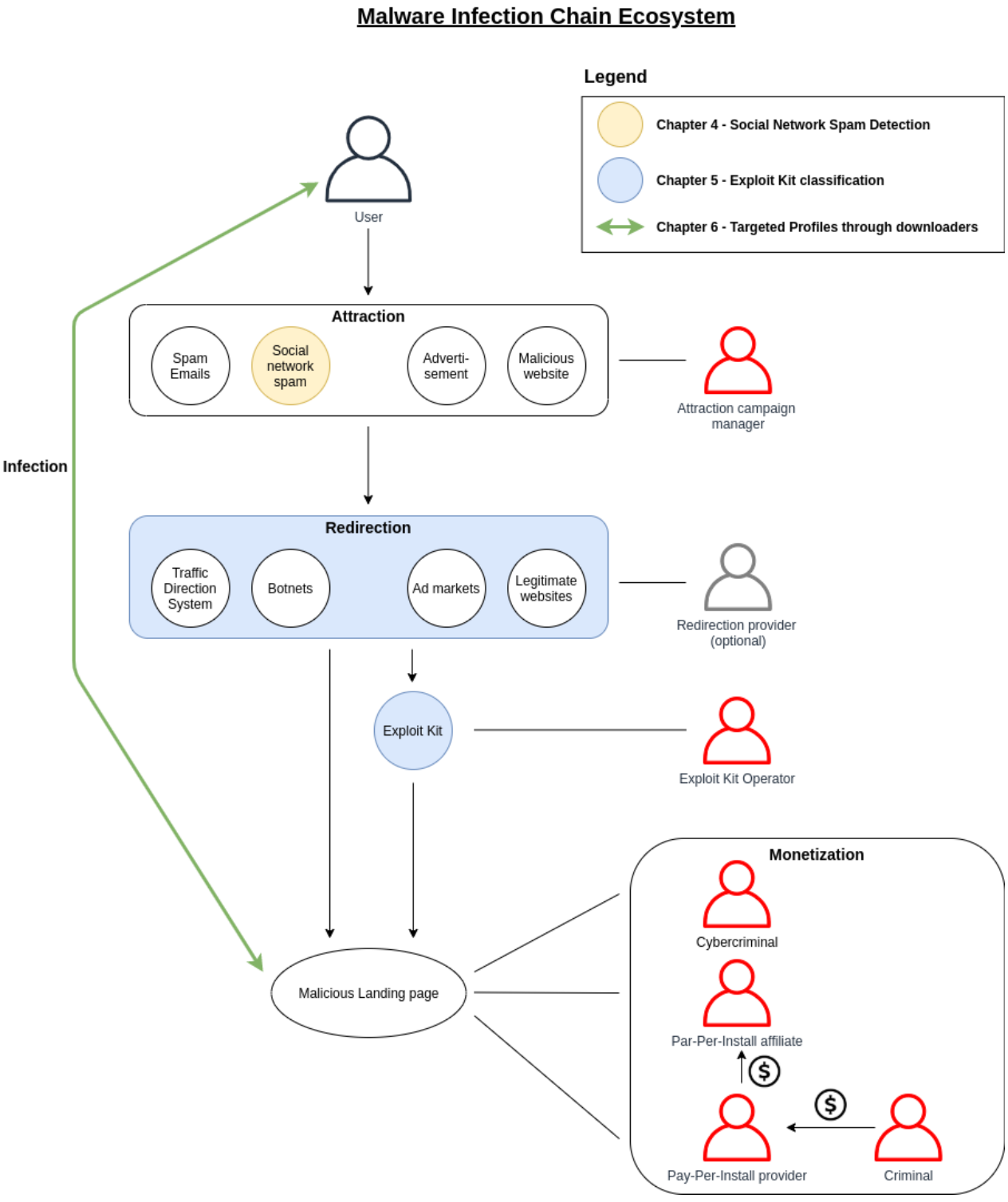


Figure 2.1 An illustration of the malware infection chain ecosystem. A user goes through an attraction phase, followed by a series of redirections. The victim then optionally travels to a website hosting an exploit kit, and finally hits the landing page, which infects them with malware. Through the monetization step, cybercriminals profit from the infection, possibly through the use of a downloader which installs additional malicious software from other cybercriminals. The legend identifies where our upcoming chapters fit in this ecosystem

spam, as well as litigation. Unfortunately, spam in 2003 was already increasing exponentially despite these measures. They also describe the arms race existing between spammers and researchers building solutions to combat them.

While the impact of spam is not always known or quantified, Kanich *et al.* [23] conducted an experiment in 2008 attempting to measure how much a single botnet can benefit from spam. They infiltrated the *Storm* botnet, and calculated the number of spam emails delivered, the number of spam emails that had their URL visited, and finally the number of sales derived from these. While the number of sales relative to the number of emails sent was low, they estimated at around $7,000 the daily revenue of the entire botnet. In terms of traffic, the three campaigns they analyzed comprised 469 million emails, sent over a period of 37 days. Apart from revenue originating from botnets, McCoy *et al.* [24] studied pharmaceutical affiliate programs in 2012, where spammers sign up to distribute various products and benefit from any sales resulting from it. This work looked at $170 million in sales from pharmaceutical businesses, where a significant portion of the sales are of repeat customers. While affiliates have a low revenue of around 20% of the sales made, the high number of sales show the importance of this phenomenon, and the motivation behind spammers.

As with other types of malicious activity online, spam often originates from infected machines. Internet Service Providers (ISP) employ various blacklists of hosts, such as the Composite Black List[1], to combat spam and other malicious activity. While this method is successful in slowing down the delivery rate of spam messages [23], it does not stop it entirely. Further protections against spam include two main approaches : signature-based detection mechanism and machine learning-based detection frameworks.

Signature-based Approaches

Early work in detecting spam originated from email filtering services [25, 26], which were employed by users to filter out unwanted emails. Following basic keyword filtering services, more advanced methods were published to identify spam. A well-known signature-based approach, Spam Assassin [27], uses various means of identifying spam, from blacklists to Bayesian filtering. As early as 2006, Pu *et al.* [28] observed the evolution of spam through this tool. They noticed certain spam techniques dying off, such as the presence of Username:password URL in emails, which were eradicated with specific rules, and various HTML-based spam obfuscation techniques, which were also defeated by Spam Assassin rules. While various types of spam emails were defeated, other types began appearing, such as messages using illegal characters in their *subject* headers. While spam messages with blacklisted URL increased

[1] https://www.abuseat.org/

during their experiments, the number of spam messages containing malicious URL did not decline, hinting at the fact that URL are too useful for spammers to remove from emails. This work highlights the arms race happening between software using detection rules and spammers, where each side evolves by countering the other.

One early method to detect spam was developed by Xie *et al.* [29]. They built AutoRE, a framework that generates signatures from spam emails. In particular, they operated a large-scale study of botnets in order to establish trends of spam email. Their approach focused on the aggregate aspect of spam emails: they construct signatures using regular expressions. With their approach, they evaluated the relation between spam campaigns, botnets, and groups of similar spam emails.

Machine Learning-based Approaches

With spammers obfuscating their messages to evade spam protections, signatures dropped in efficiency and machine learning-based approaches rose to tackle the problem of detecting spam.

Previous research [30, 31] studied the use of naive Bayes for spam filtering, using the email text as a bag of words, combined with phrasal features, i.e., particular wordings associated with spam emails, and non-textual features from the email headers. Many approaches have tried identifying spam messages using different sets of features [30, 31]:

- Features extracted from the URL
- Applying Term Frequency-Inverse Document Frequency (TFIDF) to the text to build features
- Employing N-grams, which consist in a series of N words or bytes, as found in the text
- Using Optical Character Recognition (OCR) algorithms in order to identify text in an image
- Extracting color saturation and color heterogeneity from images

Classifiers were also used on the links contained in web content in order to establish if the link is spam [32].

2.2.2 Spam Value Chain

While detecting spam is in itself a challenge, a number of previous work explored the spam value chain, in order to identify mitigation strategies based on the value chain, outside of the

actual spam message.

Levchenko *et al.* [33] crawled spam messages from spam feeds, URL feeds and botnets, and identified that the network infrastructure, in terms of web hosting, Domain Name System (DNS) hosting and domain name registering, is in majority not shared among affiliate programs. These programs use spammers as affiliates to advertize their product. While the authors of the paper only analyzed spam messages selling pharmaceuticals, replicas, and software, this distribution of the infrastructure through the entire value chain might also be shared by spam messages selling malicious software. The spam value chain includes a number of actors, that frequently operate in different countries, to manage the domain registrar, affiliate programs, DNS servers, proxy servers and final merchant bank. Considering this, the main weakness of this value chain identified in their work was the payment processing.

Stringhini *et al.* [34] inferred the relation between email harvesters, botmasters and spammers, by setting up servers leaking email addresses and then fingerprinting incoming connections. Since they also controlled the email servers, they observed incoming connections which were using the previously stolen credentials. They grouped the connections into spam campaigns, and identified 63 campaigns originating from 9 harvesters. These campaigns included the sale of counterfeit goods, Search Engine Optimization (SEO), financial scams, and phishing. Interestingly, spammers did not rent more than one botnet. Two harvesters resold the email lists, and some email lists were used directly by the harvester.

As can be observed, spam messages are closely linked to botnets and redirection services.

2.2.3 Dark Market Listings

Attraction services are offered in underground networks and dark markets. In these, cybercriminals list services related to the attraction phase, as part of the CaaS ecosystem. The attraction of victims requires an access to victims, such as a list of emails. These are sold on dark markets, along with personalized phishing email services [19, 20]. Attracting victims is one of core activities of botnets, and as such, these are offered for that very purpose in a number of dark markets [21, 35–37]. Some cybercriminals simply advertise malware spreading services [37]. In terms of social network spam messages, offerings include the purchase of compromised or fake accounts [20] in order to reach potential victims.

2.3 Redirection

The redirection step of the malicious infection chain occurs after the user follows the initial URL entry point. In the case of spam emails or social network spam, users will be redirected

in part through free hosting services and URL shorteners, but in a larger part through compromised websites [33]. Possible redirection services also include proxy servers [20].

A number of redirection services are offered through dark market offerings, as TRaaS [21]. As with the attraction step, botnets can also serve as a redirection medium to send victims to an exploit kit or a landing page, i.e., the final malicious webpage serving the malware. As mentioned previously, botnets are at the core of the CaaS, providing a number of different services, in the attraction step, the redirection step and at later stages. The life cycle of a botnet consists in 1) The creation of a botnet, 2) The recruitment of nodes by infecting unsuspecting users, 3) The advertising on dark markets and 4) The execution of attacks, either directly or through support activities [35].

When infecting users through advertisements, commonly named *malvertisement*, attackers will substitute a legitimate advertisement link with one linking to malware [38–40]. The user will thus travel through advertisement markets up until the final webpage, which will infect their machine.

2.4 Exploit Kits

The redirection step of the infection either ends at a landing page, or at an exploit kit. Exploit kits are used by malicious actors to fingerprint the machine of a victim and deliver a suitable payload. Exploit kits are sold in underground markets [21, 36], where an attacker buying it will host it on one of their servers so that users visiting the website will get infected with their malicious payload. The packaging of exploits into an exploit kit is also available as EPaaS on dark markets.

Although exploit kits are frequently only malicious code analyzing the user's machine, they are a key point in the malicious infection chain, often consisting in the first malicious software the user comes in contact with.

2.5 Monetization

After visiting the webpage hosting the exploit kit, the user will be redirected to the landing page which will infect them with a malware. This can be part of a malicious campaign operated by one or a group of cybercriminals, where these will infect the user to eventually make a profit. The malware installed on the user's machine also frequently consists in a downloader, or dropper, which will only install additional malware, either instantly or at a later time, according to the needs of the attacker managing the downloader. These are

sold through PPI services, where a malicious actor, i.e., a PPI *affiliate*, has a downloader installed on one or more victim machines. This affiliate will be paid through a PPI provider by other criminals needing a vector to install their malware. Downloader software also install additional downloader software [7], hinting at the fact that PPI services sell their services between themselves as well.

Another important aspect of the PPI ecosystem is the fact that it is often mixed with other software distribution, such as PUP software [41], which adds a layer of complexity to researchers investigating the inner workings of PPI services.

Many additional services are present in the monetization step, such as money laundering services, hacker recruiting services, and resilient hosting, more commonly known as bullet-proof hosting [21, 36].

2.6 Use Cases

In order to better highlight how real world attacks fit into this ecosystem, this section describes a subset of currently popular attack scenarios [42] and how they operate inside the malware infection ecosystem. It is worth noting that this is a nonexhaustive list of malware attacks.

2.6.1 Ransomware-as-a-Service

Ransomware attacks have been particularly prevalent among government organizations since 2013 [43], and have seen a surge in recent years. A ransomware is a malware which, once it gains access to a machine, encrypts partially or completely the user's files. The malware then demands a ransom, i.e., a payment, in exchange for the decryption key. This malware is particularly effective, since a correctly encrypted system will be unrecoverable without the decryption key, and users without a backup of their personal files will need to pay to avoid losing them.

Following the rise of CaaS, Ransomware-as-a-Service (RaaS) started to appear in dark markets [44]. Ransomware can now be bought, complete with support services and optionally botnet services. This fits into the *monetization* step of the ecosystem. A criminal will buy the ransomware through RaaS, and must then find a way to deliver the malware to victims, through an *attraction* campaign. This can be done through PPI services, where the ransomware is simply installed directly on infected machines [44]. Otherwise, if the criminal opts to infect users through a bought spam campaign, they must host their exploit kit and ransomware, or use the services of EaaS and EPaaS. The spam campaign will attract victims

who will travel through a series of *redirections* and end up on the website hosting the *exploit kit*. This exploit kit will in turn retrieve the ransomware and install it through a vulnerable component of the victim's machine. The victim user, now infected with a ransomware, will see their files encrypted, and will pay the criminal to regain access to their files. A popular method of payment is to make the user pay in bitcoin, to limit the traceability of the funds.

2.6.2 Financial Malware

Banks and financial organizations are attractive targets for cybercriminals, given the sensitive data they possess. Malicious actors target these entities through their users and employees [45].

Van *et al.* [45] identified a general value chain of mass financial malware:

1. The development of the malware through EaaS
2. The purchase of installs through PPI services. These operate the attraction step of the malicious infection and act as attraction campaign manager
3. The banking account takeover through PaaS
4. The cashing out in the monetization step

They also highlighted two other targeting schemes of cybercriminals, where the attraction step differs. First, they identified malicious actors who use Remote Access Trojan (RAT) software to infect users of small and medium-sized businesses targeted through any number of attraction methods mentioned previously. Second, they observed other actors using RAT to target banks directly, by infecting a bank employee through, for example, spam emails bought in an underground market.

2.6.3 Cryptomining

With the rise of cryptocurrencies, malicious actors also adapted, and created cryptomining malware. This type of malicious software, once inside a victim's machine, uses the system's resources to mine cryptocurrencies for the malicious actor. This type of malware can go undetected by the user since it operates in the background, and does not cause immediate harm to the user.

Huang *et al.* [46] explored a number of cryptomining malware family operations. Cryptomining was, in 2014, a profitable avenue for cybercriminals: through the five botnets analyzed,

they observed a daily US payout of 200 to 900 USD. In this scenario, malicious actors interested in finding victims to install their cryptomining software buy the services of botnets and use the infected machines to mine cryptocurrency for them. The botnet operators sell the access through BaaS. Since the botnet machines are already infected, the attraction step is skipped.

Cryptomining can be done with the user's consent as well, to become an alternative revenue stream for content creators [47, 48]. However, many cybercriminals gain access to legitimate websites or software through an exploit, and then inject malicious cryptomining scripts into them to take advantage of their userbase through browser-based cryptojacking [47, 48].

2.7 Summary

The malware ecosystem is a complex entity now composed of a large number of services, offering specific tools or assets to carry out parts of an attack. With the rise of CaaS, criminals aiming to carry out an attack can now purchase the services of whichever expertise they lack. Thus, many different actors can take part in a single attack, with limited contact between themselves. In addition to increasing the complexity in mapping an entire attack and the actors involved, this results in a larger number of attacks.

As mentioned previously, some of the most popular malware attacks all fit differently into this ecosystem, and can employ the services of a variety of supporting services, from the creation of fake accounts on social networks, to the purchase of a botnet, to the development of an exploit.

In the following chapters, we will explore in detail different sections of this ecosystem, from the attraction of victims through social networks, to the redirection of victims through intermediate websites, to the actual infection and its targeting of user characteristics. Through this detailed analysis, we aim at uncovering some of the inner workings of this complex ecosystem and providing a clearer picture of the malware infection ecosystem.

CHAPTER 3 ORGANISATION

The main research objective of this work consists in establishing the entire chain of malicious infections. This work will be split into three main chapters, addressing various steps of the infection chain, as stated in Chapter 1.

In this work, we first focus on the attraction of victims through spam, particularly social network spam. Email has been slowly overtaken by social network messages as the ideal medium for spam, following the heavy filtering of emails done by email providers. Already in 2013, social spam was rapidly gaining in popularity with a growth of 355% [49]. In order to send spam messages on social networks, attackers often create fake accounts. This particularly affects large social networks. According to Facebook's transparency report [50], 1.7 billion fake accounts were removed between January 2020 and March 2020. While spam is present on most communication medium, such as blogs, bookmarking websites, online reviews, location searches and online comments, social networks offer the biggest available platform to send spam to large numbers of users [51].

Chapter 4 presents our approach at identifying communities of interest in social networks, and using this knowledge, we build a probabilistic model capable of detecting spam messages by the path they take through these communities. Social networks consist in an important infection vector for cybercriminals, where spam messages are employed to attract victims to malicious URLs. Through our work, we aim at improving the general knowledge of how users are connected in social networks, and improving protections against malware by building a novel approach at detecting spam messages directly in the attraction step.

When spam messages are successful, victims visit malicious URLs and are sent through a series of redirections, ending on a website hosting an exploit kit. This exploit kit then infects the user's machine by identifying its vulnerable components. Chapter 5 aims at improving on previous studies analyzing this second step of an infection, by building a framework which automatically reconstructs web redirection chains from a network capture file, and then employing a classifier to identify exploit kit families using only features from the previously reconstructed redirection chains. This work improves on previous methodological approaches employing web redirection chains, as well as improving on previous works classifying exploit kit families, by using the structure of their redirections as input to the model. Given that redirections can be the result of botnets, server proxies, advertisements and TDS, and that the traffic is not necessarily all from the same redirection provider, we aim at establishing if a structure of redirections can be linked to an exploit kit family, or if on the contrary, each

exploit kit operator uses their own selection of infrastructure.

Finally, Chapter 6 follows the infection chain one step further, where an exploit kit has already infected a user's machine with a malicious downloader software. Since this software is often the one installing the final malware on the user's machine, we specifically target these downloaders in order to better understand how they choose their victims. Thus, we build a framework which automates the process of running malicious downloaders in customized sandboxes, and execute samples of two known families in it for a year-long period. With this experiment, we aim at better understanding the targeting choice of cybercriminals and the inner workings of PPI services, in order to help security specialists better protect users against malware.

In a nutshell, our work aims at building a better detection framework for spam messages in the attraction step. It achieves this by building communities of interest. Then, exploit kit families are classified according to the redirections in the redirection step, by reconstructing web redirections from network packet capture data. Finally, in the context of the infection step, we build a downloader testing framework and use it to run malicious downloader executables and capture their behavior, in order to better understand how malicious actors target their victims.

CHAPTER 4 IDENTIFYING SPAM MESSAGES BY THEIR PATHS OF DIFFUSION IN COMMUNITIES OF INTEREST

4.1 Introduction

Spam messages have always been a nuisance to Internet users, and as such have been the topic of a number of research works. While detection methods improve over time, cybercriminals evolve their attack schemes to adapt and find new victims. In recent years, one such adaptation has been the steadily increasing use of social networks as a medium to infect users [52]. These consist in effective infection vectors to attract victims, due to the fact that they are now among the most popular websites, according to Alexa's top websites ranking[1], and thus provide a large potential user base to attack. Also, attackers who compromise an existing user or community can gain the trust associated with them and abuse it [53, 54].

In social networks, cybercriminals employ different tactics to reach the maximum number of potential victims. Spam messages can be spread by bots, sybil communities, or compromised accounts. Previous work detecting spam messages and accounts focused on detecting Sybil communities [55, 56], bots [57–59], compromised accounts [53], or using a combination of these techniques [60, 61].

While these approaches respectively obtained good accuracy in detecting spam accounts or messages, recent research has highlighted how both Sybil-based defenses [13] and account-based detections [62] can be evaded by attackers. This is due to detection methods employing features of accounts or messages as a mechanism to identify spam, rather than using the way in which malicious messages spread in social networks. Thus, in this work, we aim at identifying spam messages by the way they propagate through online communities.

In social networks, users form structural communities, i.e., they connect to other users within a same community. These communities can be identified by observing the connections in the underlying user connection graph. Structural communities consist of clusters of users which are more strongly connected to each other than to other users. In life, people connect to others who share similar thoughts, values, interests, activities, etc. This principle is called the homophily principle [63]. In this work, we postulate that the homophily principle applies to online communities on social networks as well. First, we aim at establishing that users form *communities of interest*, i.e., structural communities which share similar topics of interest. Thus, our first hypothesis consists in the following :

[1] https://www.alexa.com/topsites

Hypothesis 1: In social networks, structural communities have strongly connected users who also share topics of interest, which are distinct from other communities.

The network topology of a social network dictates the dissemination path of messages and information [64, 65]. Notably, information spreads faster in a networked community [66]. However, it will also travel to other communities that share its topic of interest.

We hypothesize that, since messages are shared among communities with specific topics of interest, legitimate messages will have a distinct path of dissemination among communities of interest which will differ from that of spam messages. For example, if two communities A and B share messages with the topics of interest i and j, a new message with i as its topic has a high probability of traveling among these communities, whereas a new message of a topic of interest l will be seen as suspicious, given that its topic has not been observed in these communities. Our second hypothesis can be viewed as :

Hypothesis 2 : Benign messages travel through predictable paths of interest among communities of interest, whereas malicious spam messages' dissemination path differ from what is expected.

In this work, we aim at testing our hypotheses on the Twitter social network. Twitter is one of the largest social networks, and its users connect to each other by following the other's tweets and reacting. The posts, named *tweets*, are made by users and are public as well, which allows us to gather them in a dataset. These characteristics are needed in order to build our communities of interest and identify spam. To fulfill our second hypothesis, we will build a framework named POISED, which identifies spam tweets by their diffusion path in communities of interest. In this work, we will use *messages*, *posts* and *tweets* interchangeably.

In a nutshell, our approach first identifies structural communities in a graph made from Twitter users and their connections, and then applies topic modeling to their posts to identify topics of discussion. It then maps the topics to the structural communities to form *communities of interest*. Following this, it identifies clusters of similar messages and tracks their propagation through these communities to build a probabilistic model of probable message paths. Finally, a classifier is built from the probabilistic paths of messages to predict if a message will be spam or not.

Our contributions are the following :

1. Through our experiments on a subset of the Twitter graph, we show that structural communities align with topics of discussion of their users.

2. We build POISED, a framework capable of detecting Twitter spam messages by their

paths of diffusion through communities of interest, using a dataset of 202 Twitter neighborhoods consisting of a total of close to 1,300,000 tweets and 64,000 users.

4.2 Background

4.2.1 Threat Model

Spam messages are frequently posted on a large scale, e.g., in the context of a large-scale malicious campaign [67, 68]. Thus, in our work, we consider messages to be part of such campaigns, and generated by templates, since spam messages are most commonly generated this way [69, 70]. We consider messages to be generated in groups of similar messages. Spam messages can be generated through the creation of sybil accounts [56] or bot accounts [71], or by using compromised accounts [53].

Previous work has focused on specific malicious content in spam messages [72, 73], or on a type of malicious account, such as compromised accounts [56, 74]. These related work target specific types of malicious messages or accounts, and thus, limit the scope of their approach. In POISED, we consider every type of spam message in our threat model, and that it could originate from any type of malicious account.

4.2.2 Communities and Parties of Interest

In social networks, users can connect and reach out to other users. The interactions between users happen around similar topics of interest [75–77]. It has been shown that users connect to other users who share similar beliefs and values, through the homophily principle [63]. While users tend to connect according to this principle, there can also be disconnected groups of people sharing the same values. In social networks, similar users will not necessarily connect to each other and form one large cluster : rather, they will form a multitude of distinct clusters of various sizes, where each cluster will regroup users with similar topics of interest. A similar topic can be shared by two or more disconnected distant clusters.

Previous work has established communities of interest by using the connections between users as a proxy to their shared interest. They identified users in an online social network as nodes in a graph, and considered the detection of communities of users as a graph problem [78]. They thus employed graph community algorithms to find clusters of highly connected nodes, and infered that the users representing these nodes are part of a community of interest when in a cluster. In our work, we establish communities of interest by constructing graph communities from a social network, and employing topic modeling on messages shared by users

in the various communities. With this, we aim to demonstrate that structural communities of users do indeed share topics of interest.

In short, a community of interest consists of a more densely connected group of users who share topics of interest. An example of such communities is shown in Figure 4.1. These communities are built from Twitter data, where nodes represent users, scaled according to their degree, and edges represent the *following* relationship between users, which consists in Twitter's way of connecting between users. These communities are built from a cluster of 2000 users and the topics are established from their tweets. Each community has its own set of topics of discussion: $Comm1$ showcases discussions about Hollywood celebrities, friends, and families. $Comm2$'s users are more interested in YouTube users and bloggers. Finally, $Comm3$ is home to discussions related to soccer, inspirational quotes, and specific TV shows.

Previous research has established how information propagates through social networks, e.g., by predicting viral memes [64]. The homophily principle [63] and the topology of networks [64,65] impact how the information propagates in online communities. In this work, we hypothesize that malicious messages spread differently from benign messages in communities of interest. We build a framework from this hypothesis to identify spam messages, named POISED (Parties Of Interest Semantic Extraction and Discovery), which is implemented on the Twitter social network.

4.3 Related Work

Our approach aims at detecting spam messages in social networks by identifying their propagation through communities of interest. We combine message propagation and community detection in social networks to the detection of spam messages to build our novel approach. In this section, we present recent research in detecting malicious messages and malicious accounts online. We then show previous work which examined the propagation of messages in social networks.

4.3.1 Malicious Messages Detection

Yardi *et al.* [79] identified spammers attaching themselves to trending topics on Twitter, and extracted their features and behavior. Most notably, spam accounts did not appear to be newer than legitimate accounts. Thomas *et al.* [73] developed Monarch, a framework which crawls URLs found in tweets to identify if the message is spam. They showed that spam on Twitter differs from email spam on a number of points. Another approach specifically examining the maliciousness of URLs on Twitter is WarningBird, by Lee and Kim [72]. They

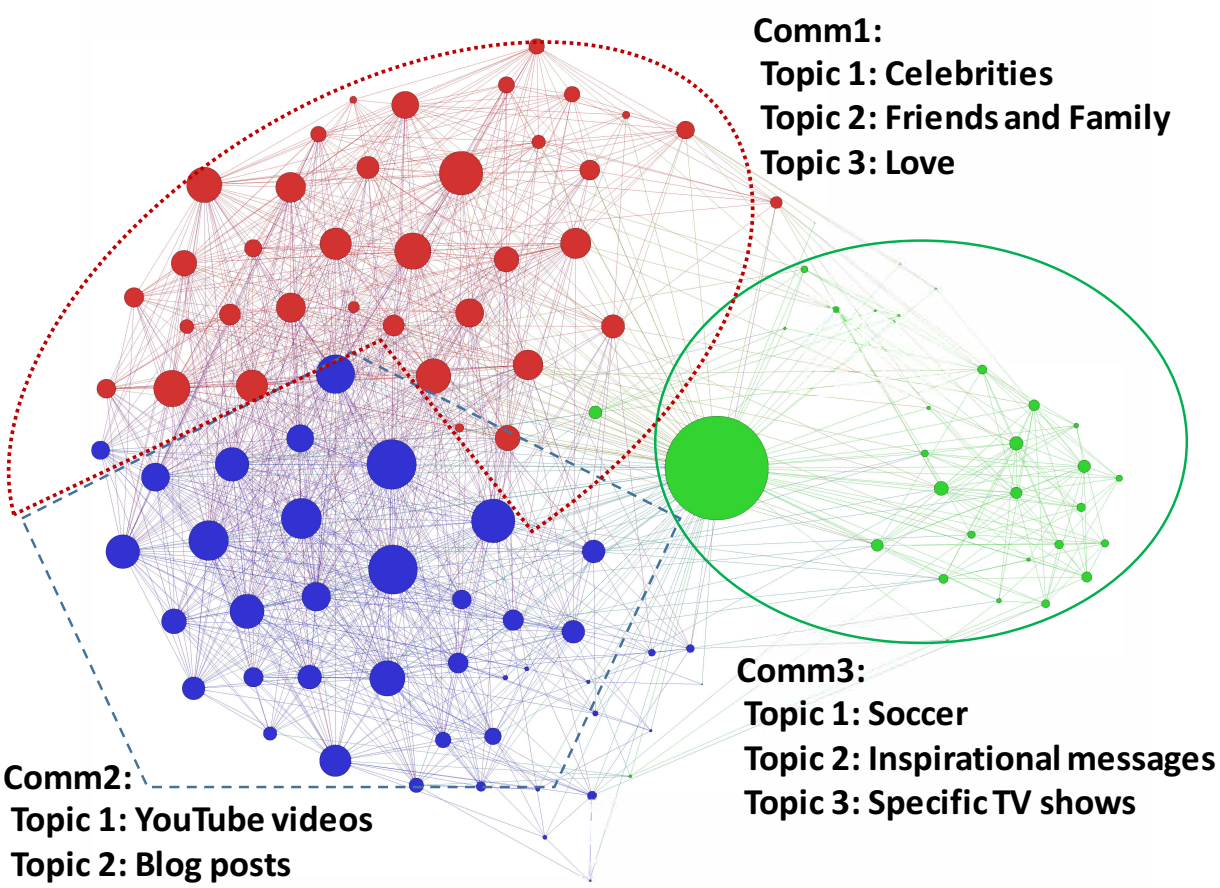


Figure 4.1 An example of three Twitter communities, each discussing a specific set of topics

built a classification model which uses the web redirections happening after visiting a URL in a spam message. Gao *et al.* [69] analyzed large-scale spam campaigns on Facebook, and built an offline approach that clusters spam messages in campaigns by using their URLs. In their results, they identified that most of the spam accounts are not new accounts, but compromised accounts. Another offline approach was built by Grier *et al.* [52], where they explore malicious URLs in Twitter spam messages and identify spam campaigns and their features. Xu *et al.* [80] built an early detection worm detection framework, which identifies the specific behavior of worms on online social networks in order to quickly identify and stop them.

While these previous studies present effective approaches at detecting spam messages, they explore specific types of spam messages, or work in specific contexts, such as only identifying malicious URLs, or operating an offline analysis of spam campaigns. Our approach aims at detecting all types of spam messages, regardless of their content, by examining the way they propagate through the network instead of their features.

4.3.2 Malicious Accounts Detection

Malicious accounts post spam messages online, and can be fake or bot accounts, or compromised legitimate accounts.

Benevenuto *et al.* [81] used supervised machine learning to model the features of spam accounts and built a detection model. Gao *et al.* [69] studied *wall* messages between Facebook users to detect and characterize spam campaigns. The authors found that more than 70% of malicious posts advertise phishing websites. Cao *et al.* [82] built SybilRank, a tool which leverages the graph structure in social networks in order to identify sybil accounts. Cai *et al.* [83] also established a framework which detects sybil accounts, by employing a machine learning approach that leverages the fact that these attacks are made by creating many false identities. Thus, these identities form communities that differ from the norm and can be detected. Danezis *et al.* [56] adopted a Bayesian Inference approach in order to detect fake accounts. Wang *et al.* [55] analyzed the click patterns of users in order to cluster them and identify clusters of sybil accounts. Ferrara *et al.* [57] analyzed the features and behavior of bots in online social networks and showed how they differ from legitimate accounts.

Liu *et al.* [84] employed a topic model approach to identify topics of discussion of accounts. They then used a machine learning approach to detect spam accounts from benign accounts. Stringhini *et al.* [58] used honey profiles on social networks in order to attract spam accounts and study their behavior; in turn building a detection model out of them. One of the ways malicious accounts build their reputation is with *Link Farming*, which consists in a group of

unrelated spam accounts all following each other for the purpose of appearing legitimate and boosting their ranking in search engines. Ghosh *et al.* [85] investigated this phenomenon, and through an analysis of 40,000 spam accounts, identified that the majority of farmed links in their dataset come from a small number of influential legitimate users. Viswanath *et al.* [60] applied Principal Component Analysis to spam account features in order to differentiate them from legitimate accounts.

Many tools have been proposed to detect malicious accounts. Integro [86] adopts a novel approach and identifies victim users in order to track malicious accounts. COMPA [74] is another system which aims at specifically detecting compromised accounts by observing their behavior over time. Normal users tend to be consistent in their behavior over time, whereas malicious accounts will show anomalous behavior. SynchroTrap [61] is a tool which identifies spam accounts by looking at the behavior of newly created accounts. Finally, EvilCohort [87] identifies communities of social network accounts operated by botnets by investigating common sets of IP addresses accessing the accounts.

Contrary to these works, our approach aims at identifying spam messages regardless of their source, and do not rely on specific features of malicious or compromised accounts.

4.3.3 Message Propagation in Social Networks

While our approach aims at detecting spam messages, we leverage the way these propagate through the network to achieve our goal. There have been a number of recent works examining the spread of messages online.

First, Ye and Wu [88] measured the propagation of breaking news in Twitter through an extensive analysis. They studied how news spread through time and how they correlated with each other. Weng *et al.* [64] built an approach which predicts viral memes. In short, the number of communities a meme spreads through in its early stages determines its popularity. Nematzadeh *et al.* [65] investigated the spread of messages through communities, which is more closely related to our approach. They demonstrated that strong communities help spread information globally, by having a strong local spreading. Tan *et al.* [89] studied the effect of the wording of a message, by analyzing the spread of differently worded messages promoting the same URL. They identified the best wording to adopt in order to spread more effectively. Finally, Mezzour *et al.* [90] showed that the diffusion of messages initiated by hacked accounts differ from the diffusion of messages created by legitimate accounts.

In our approach, we inspire ourselves from these previous works in order to identify the communities of interest a message travels through, and use this information to differentiate spam messages from benign messages.

4.4 Methodology

Our approach involves a number of steps in order to obtain our final prediction model. Firstly, we gather network and message data from Twitter to build our dataset. We then apply community detection algorithms on the graphs constructed from our dataset, and extract topics of discussion from users through a topic modeling approach. Following this, we cluster similar messages, and finally build our probabilistic model and apply classification on it to detect spam messages.

4.4.1 Data Gathering

We employ the Twitter streaming API to obtain user messages. This API returns a random number of active users' timelines. We download these user *timelines*, which consist in lists of users' *tweets* i.e., messages. On Twitter, users connect through an action called *following*, where one user can follow another user, and one user can be followed by another user. This relation is thus not necessarily reciprocated. We formally define users and their connections as :

Definition 16 *A* social network graph $G\langle V, E\rangle$ *is a set of vertices* V *representing the users in the network and a set of edges* $E \subseteq \{(u, v) : u, v \in V\}$ *representing the set of connections between users.*

As mentioned previously, relations are not necessarily reciprocated; thus, G is a directed graph. This graph can be converted to an undirected graph if only reciprocated *following* connections are kept as edges.

4.4.2 Community Detection

Before building *communities of interest*, we extract structural communities from the social network graph. In graph theory, a community consists of a set of vertices which are more densely connected to each other. In this work, we use existing community detection algorithms to build our structural communities. Our intuition is that these structural communities will be linked to external interests, such as an interest in certain activities, an interest in

politics, an interest in computer sciences, etc. These communities are not completely distinct from one another, and in reality overlap with each other [91, 92]. Structural communities, however, are defined as disjoint sets of nodes that do not overlap. In our work, we employ structural communities as they apply well to our dataset. These non-overlapping communities provide enough insight to build our communities of interest. A structural community can be defined as :

Definition 17 *A structural community C inside a graph G is a disjoint partition of vertices in V, namely $C = \{C_1, \ldots, C_h\}$, where $C_i \subseteq V$, $C_i \cap C_j = \varnothing$ if $i \neq j$. Nodes in a community C_i have a higher probability to be connected to each other than to nodes in other communities.*

4.4.3 Topic Detection

In Natural Language Processing (NLP), topic modeling is a type of clustering which aims at establishing the underlying topics of a corpus of text. It is an unsupervised machine learning approach, which groups documents together by identifying their topics. It has been successfully applied to messages from social networks through a number of variants such as Latent Dirichlet allocation (LDA) [93] and Topic Mapping [94]. A number of previous work have employed LDA with success [84, 95–97], so we chose LDA to identify topics of discussion in messages sent by users in our dataset.

LDA takes as input a list of documents, and identifies a set of topics for the entire corpus. Each document has a weight associated with each of these topics, denoting the importance of each topic for this particular document. For documents of small length, such as tweets, topic detection is shown to underperform [98]. Hong *et al.* show that grouping small documents together improves the performance of LDA. Thus, we follow their approach and regroup our tweets from a same user into a number of documents.

For a user u, a set of documents D_u, namely $D_u = \{d_1, d_2, .., d_k\}$, is generated by splitting the user's timeline into k groups with l messages. Since the length of a document impacts the topic modeling algorithm [98], we use a fixed l number of messages, with a varying k number of groups of messages. To assess the impact of the size of l, we evaluated our approach with $l = \{1, 5, 10, 20, 50, all\}$, and established that the length of the documents do not have a significant impact on the overall results. We chose $l = 20$ as the value for our documents building.

Thus, a user u's topics of interest T_u are generated by combining every topic obtained from running LDA on the user's list of documents D_u. A community C's topics of interest are in turn the combination of its users' topics of interest. Formally :

Definition 18 *A community C's topics of interest T_c are the union of its users' topics of interest, namely $T_c = \{T_1 \cup T_2 \cup ..T_n\}$, for n users, where a user $u \subseteq C$.*

POISED builds a *Community of Interest* as a set of topics of interest of users representing nodes inside a structural community.

4.4.4 Clustering Similar Messages

As mentioned previously, spam messages have been shown to be most often generated by templates [69,70]. In order to consider this, we group messages by their similarity. There are many ways to group messages by their similarity, using a number of similarity measures [99–101]. One such method, *n-grams*, consists in using series of n continuous words in the texts to compare their similarity. This technique has been used successfully in previous spam detection frameworks targeting Twitter messages [53, 102]. Specifically, these previous work used *four-gram*.

In our approach, we opt for the same similarity measure, and apply four-gram to our list of tweets, regardless of the user, in order to obtain all the groups of similar messages in our dataset. Upon a manual inspection of 60 random clusters of varying size of similar messages found in our dataset, we observed that all groups were indeed made up of the same text, i.e., all messages were correctly grouped. Thus, for the remainder of our work, messages were grouped according to a *four-gram* analysis. For any message of less than four words, all words were used for the *n-gram* analysis.

4.4.5 Probabilistic Model

To construct our probabilistic model, we must first define the concept of *Parties of Interest*. Groups of similar messages, as identified by *four-gram*, appear in one or more communities. We define as *Parties of Interest* messages from these groups that appear in communities. By identifying in which community of interest a message from a cluster is posted, we can, over time, predict if one message appearing in a community will be posted in another community.

For each group of similar messages, we calculate how many times a message has been observed in a community with a certain topic of interest. For example, let us have three communities C_1, C_2, C_3, where C_1 has topics $\{T_1, T_2\}$, C_2 has topics $\{T_1, T_3\}$ and C_3 has topics $\{T_1, T_4\}$. We have one group of four similar messages posted by four different users, three of which post in C_1 and one in C_3. We can then compute the probabilities of each message appearing in a community with a specific topic. First, we establish the set of all topics as $\{T_1, T_2, T_3, T_4\}$.

Then, we count the number of times a message from the group has been posted in a community with a topic T_i, normalized by the message counts of the union of topics from that group. In our example, we end up with the probabilities $\{\frac{4}{8}, \frac{3}{8}, 0, \frac{1}{8}\}$, where all four messages appear in communities with T_1 as their topic, three messages appear in communities with T_2 as their topic, no message is posted in a community with the topic T_3, and only one message is posted in a community with T_4 as its topic.

Thus, our approach tracks the diffusion of messages through parties of interest and computes a probabilistic model for every group of similar messages. The overall table of probability for our approach is built with the union of every probability vector of groups of similar messages, i.e., $\{prob_g1, prob_g2, ..., prob_gh\}$, where h is the number of clusters of similar messages. This constitutes the input for our classification model, which aims at detecting spam messages from benign messages.

4.4.6 Classification

We can use the probability models established previously as features for a classifier, where the target is a binary label, and identifies if a message is spam or not. A supervised machine learning model is used to correctly label a message as spam. Thus, a ground-truth dataset is made from our data, by labeling a subset of messages as either spam or benign. This, in turn, is used to train the classifier to predict spam messages.

4.4.7 Summary

In short, our approach first collects data from Twitter, and builds structural communities from user connections. It then applies topic modeling to messages, i.e., tweets, posted by users in these communities. Groups of similar messages are then identified, from which probabilistic models of parties of interest are built. Finally, these probabilistic models are used by our classifier to detect spam messages. Our approach is summarized in Figure 4.2.

4.5 Evaluation Setup

Our approach is applicable to any social network permitting users to post messages and to connect to each other. We opted to implement POISED on Twitter, given that it is one of the most popular social networks, and had, as of December 2015, over 320 million active users [103].

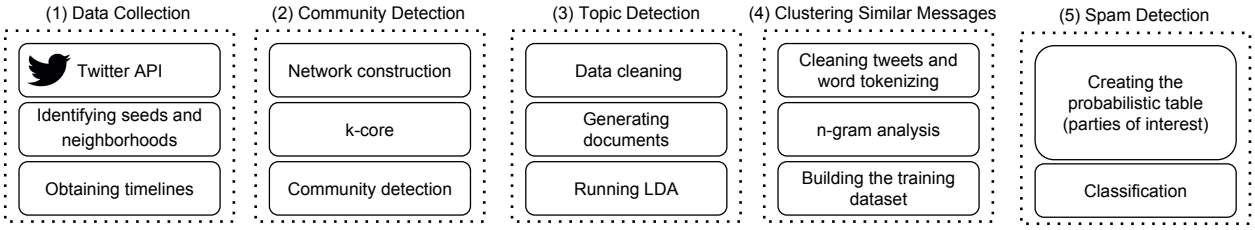


Figure 4.2 POISED constructs a probabilistic model based on the diffusion of messages throughout communities of interest. Then, it employs supervised machine learning to classify messages as spam or benign

4.5.1 Dataset

Our dataset consists of random users and their posts, i.e., *timelines*, from Twitter. It was collected in December 2015, through the Twitter API. We used the streaming API, which provides a random stream of currently active Twitter users. We collected 300 users through this API, and crawled all of their followers and friends as well. We consider these 300 users as *seeds*, where the seed and its related users are a *neighborhood*. We only collected seeds with a maximum of 2000 friends and followers, so the crawling could be manageable on our server. We collected users who follow them, i.e., their followers, and users that they follow, i.e., their friends. We only collected users who specified *English* as their language in their profile settings, since our chosen text processing of messages can only be applied on one single language. Future work could explore combining languages or further separating our communities by language. With this approach, we collected around 52,000,000 tweets posted by around 159,000 users between December 2006 and December 2015. Figure 4.3 shows the timeline length for users in our dataset, where timelines have an average of 332 tweets and a median of 177 tweets.

We then further filtered our dataset. Of every user collected from the API, we only kept the 300 most recent tweets in order to limit the bias imposed by older accounts having a high number of posts. Following this filtering, the average age of our crawled users' oldest tweet is nine months, while the average age of their most recent tweet is six months.

4.5.2 Graph Construction

Once our dataset is obtained, we must then build a graph representation of our users. We build both a directed and undirected graph of our user connections. In the directed graph, a user consists of a node, and edges are made from the *following* connections, where the direction of the edge is the direction of the follow. For example, if user A follows user B,

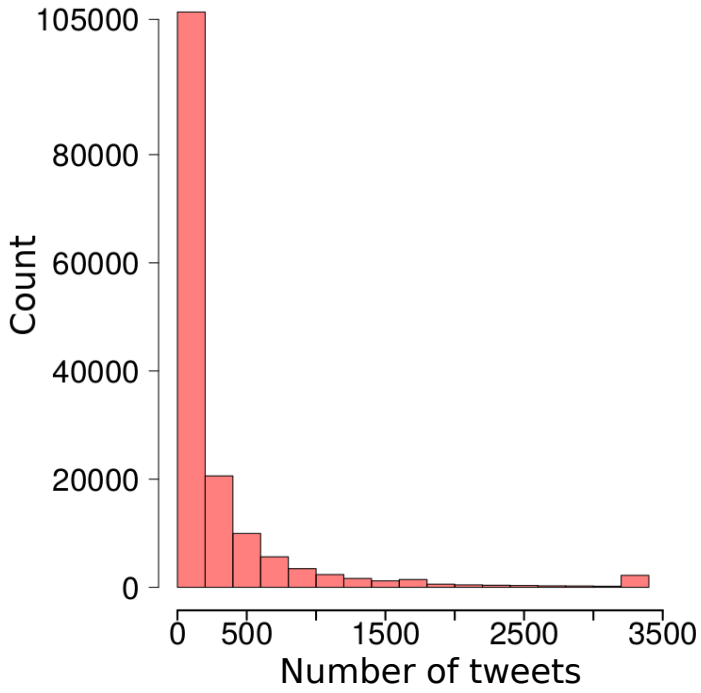


Figure 4.3 The size of timelines in our dataset

then the edge e connecting them goes from node A to node B. From that directed graph, we build a second undirected graph by keeping only edges where both nodes follow each other, i.e., user A follows user B and user B follows user A. All other edges are discarded. Thus, the obtained undirected graph displays a lower number of edges.

In our networks, some users have only one connection to another user. These users eventually create communities of only one or two nodes, which cannot be used in our evaluation. We filter these users out by employing k-core [104], which outputs the subgraph of a graph, where every node has a minimum degree of k. In our graph, k-core is used with $k = 2$, in order to remove those lone nodes. After building a graph for every seed, we obtain an average neighborhood size of 271 and a median neighborhood size of 178. Figure 4.4 shows the neighborhood sizes for our 300 neighborhoods.

Finally, after the graph construction and the previous filtering of users, the dataset used to build our communities of interest and validate our first hypothesis consists of 15,751,198 English tweets posted by 82,275 users in 300 neighborhoods.

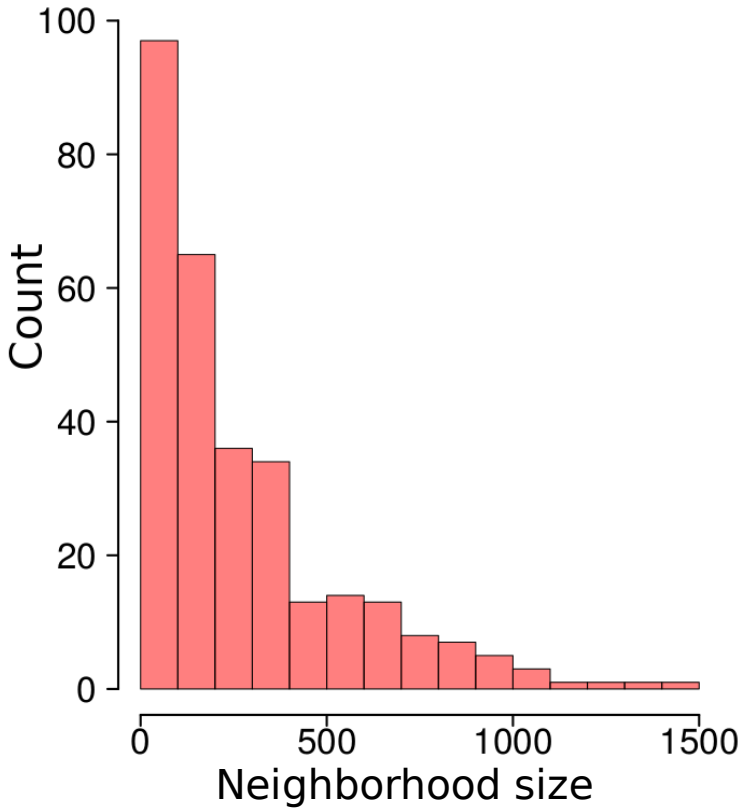


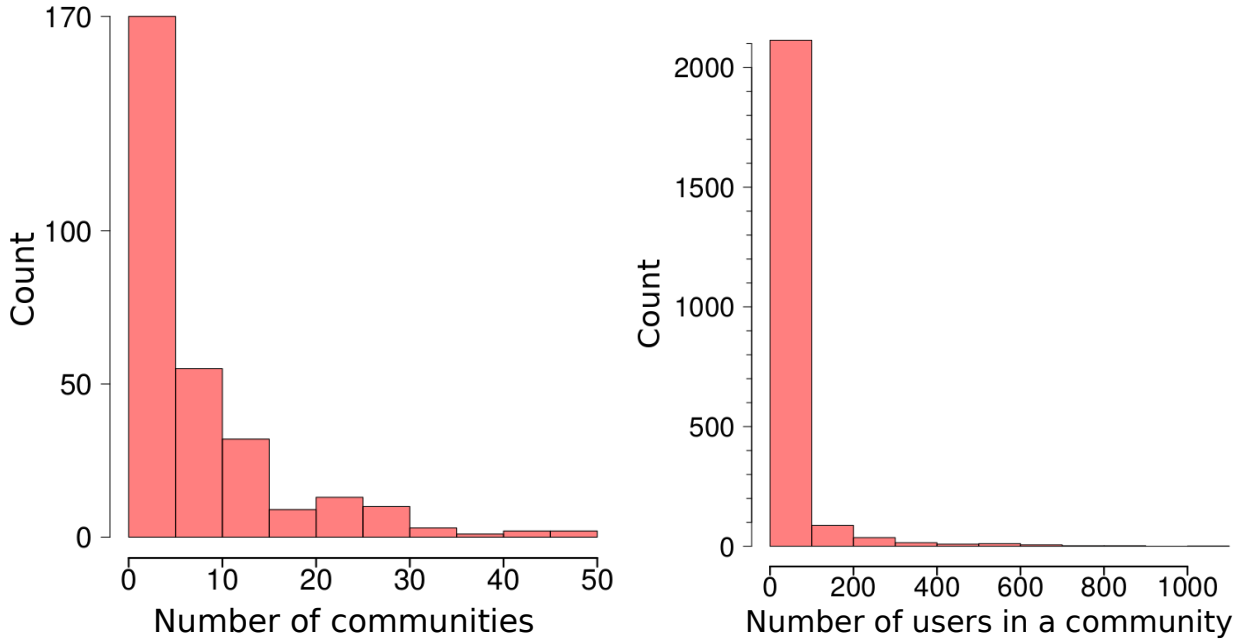
Figure 4.4 The size of neighborhoods in our dataset

4.5.3 Structural Community Detection

To extract communities from our social network graph, we employed the most commonly used and widely accepted community detection algorithms. On our dataset, we tested Infomap [105], Spinglass [106], Walktrap [107], Leading eigenvector [108], Fastgreedy [109] and Multilevel [110]. We compared the performance of these different algorithms using clustering metrics throughout our experiments, and obtained the best scores with Infomap. This algorithm has been shown to perform well in previous benchmark works of community detection algorithms [111, 112]. As mentioned in the previous section, we construct both a directed and undirected version of our graphs. However, only Infomap, Spinglass and Walktrap can be applied to undirected graphs, and thus, only these three algorithms are used when detecting structural communities in our undirected graphs. In Infomap, a community in a graph consists in a subgraph minimizing the average number of bits required per step, where a step consists of a compression of the path of random walkers.

When applied to our 300 neighborhoods, Infomap detected a total of 2,283 communities. On

average, a neighborhood contains 7.5 communities and the median number of communities is 4. Figure 4.5 displays the number of communities and the number of nodes per community observed in our dataset. The average community size is 30 while the median is 6.



(a) The number of communities per neighborhood.

(b) The community size.

Figure 4.5 Histograms for our structural communities

While for simplicity, we only used non-overlapping community detection algorithms, there also exists overlapping community detection algorithms. Using these would more closely mimic the structure of social networks where a single user would typically be a member of different networked communities, thus of different communities of interests. Nonetheless, our approach was effective with the use of non-overlapping communities.

4.5.4 Topic Detection

We employ LDA to find the underlying topics of a set of tweets. This topic modeling approach takes a list of documents as an input, and outputs a set of topics. Our documents have a fixed l number of tweets from community users, and are randomly shuffled. We tested different values of l but did not find a significant difference. Nonetheless, we chose 20 as the value for l given that it gave slightly better results, as will be shown later. As mentioned previously,

since the length of timelines vary from user to user, we only use the 300 most recent tweets per user. When creating documents, we filter the tweets when aggregating them in order to improve the topic modeling. Firstly, we remove URLs and any non-printable character, and then remove *stop words*: these are common short words in the English language, which do not bring any insight into a topic of a text.

For the implementation of LDA, we opted to use Machine Learning for Language Toolkit (MALLET) [113]. When building the model through MALLET, we obtain as output a set of topics, and, for each document, a weight associated to every one of the topics. We then set the document topic to the one with the highest weight. When using MALLET, we must specify the number of topics to cluster to. We execute our experiments with different numbers of topics to assess the impact of this feature. In the end, our clustering metrics were at their highest when using 500 as the number of topics. We run MALLET with 200 iterations, in order to obtain a more robust model, at the cost of a lengthier execution time.

4.6 Evaluation: Communities of Interest

In this section, we aim at testing our first hypothesis, by establishing if members of structural communities also share topics of interest, thus identifying *communities of interest*.

4.6.1 Metrics

In order to validate our mapping between topics of interest and communities identified inside our neighborhoods' graph, we employ a number of clustering metrics: *completeness*, *homogeneity*, and *V-measure*, proposed by Rosenberg and Hirschberg [114], and Adjusted Rand Index (ARI) [115, 116] and Adjusted Mutual Information (AMI) [117].

First, completeness aims at testing with what degree documents inside a community are assigned to a single topic. A higher score will imply that communities in general discuss close to a single topic. Secondly, homogeneity identifies whether a topic is seen in only one community. A higher score will thus indicate that topics of discussion tend to not be shared by multiple communities. Finally, V-measure consists of the harmonic mean between completeness and homogeneity. These three metrics have been used in previous work to validate natural language processing models [114, 118]. They output a value between $[0, 1]$, where 0 is the lowest score and 1 is a perfect clustering.

In addition to these measures, we used two criteria to provide an adjustment for chance groupings and for the positive effect of communities with a high number of documents. The ARI measures the similarity between communities and topic classes, where it measures the

number of documents in agreement versus the number of documents in disagreement. The AMI is a measure of similarity and dependence between communities and topic classes, and its computation is based on the mutual information measure. Each of these metrics is similar to the V-measure: in order to satisfy the ARI and AMI criteria, a topic must have been observed in a single community and a community must contain documents with a single topic. A score of 1 signifies that communities and topic classes are in perfect agreement. ARI and AMI can produce values between $[-1, 1]$. These measures provide additional information over the completeness, homogeneity and v-measure, wherein a *positive value* will indicate that the communities are dependent on the topics.

With the use of these metrics, we can observe how closely our topics of discussion match our structural communities.

4.6.2 Topics of Interest of Users

Before evaluating our topic modeling on communities, we first verify how a single user's tweets cluster into topics. We evaluate how the various documents of a user cluster together, in order to establish if individual users are consistent in their topics of interest, or if their discussions are too wide and varied to be correctly identified by our topic modeling. A community's topics of interest consists in the set of its users' topics. Thus, we must first confirm that users do have specific topics of discussion.

Figure 4.6 presents our results in applying our topic modeling to users in neighborhoods. Although we test individual users' topics, we average their results by the size of their neighborhood, in order to identify if the number of friends a user has impacts the identification of their topics of discussion.

As can be observed, the results are consistent across the different sizes of neighborhoods. More importantly, the completeness, homogeneity and V-measure are all high, at around 0.85, indicating an effective clustering. A high completeness indicates that users tend to discuss a low number of topics, and a high homogeneity indicates that they generally have distinct topics. Particularly, the high completeness shows us that user topics can be identified, and that users tend to have a low number of topics. With this information, community topics can be evaluated in order to establish if communities have topics which are distinct from one another. The positive AMI and ARI scores further confirm our results.

Another parameter of our experiment is the number of tweets per document. As previously mentioned in section 4.4.3, we group l number of tweets in documents to identify topics. Figure 4.7 shows the average accuracy of our measures when applied to individual users' in-

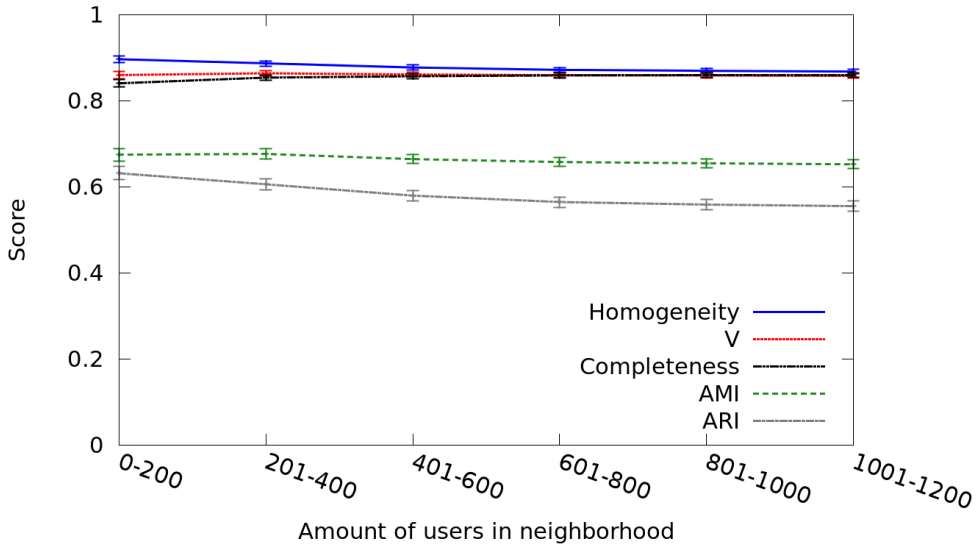


Figure 4.6 Average metric scores for the users according to the size of neighborhoods

terests, ranging from 5 tweets per document to 50 tweets per document. Since this evaluation is on the documents of a single user, testing with all tweets in a single document would not have provided any useful information.

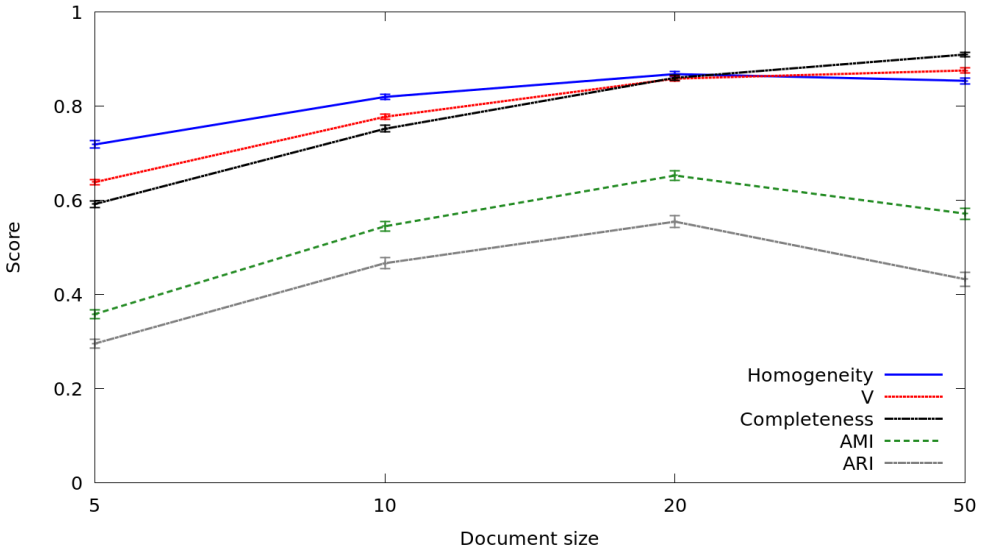


Figure 4.7 Average metric scores for users interests according to document size

As can be observed, our metrics are at their highest when the number of tweets per document is at 20.

4.6.3 Baseline: Null Model

Null models have been previously used as a baseline for validating a model [119]. Notably, null models have been used in recent work to validate network structures [119–122] as well as community structures [123]. We employ a null model to establish a baseline of comparison for our communities of interest, using the previously mentioned clustering metrics. If we see no improvement in our approach when compared to the null model, then we can assume that our communities of interest cannot be differentiated from a random cluster of documents. If our approach shows better clustering, then we can validate our first hypothesis that users inside a structural community discuss specific topics of interest.

We generate the null model by randomly shuffling our documents into groups to act as communities for the baseline model. The documents are shuffled so that their initial distribution through structural communities remain the same, and the number of communities does not change. Thus, we can directly compare communities from our approach to communities in the null model.

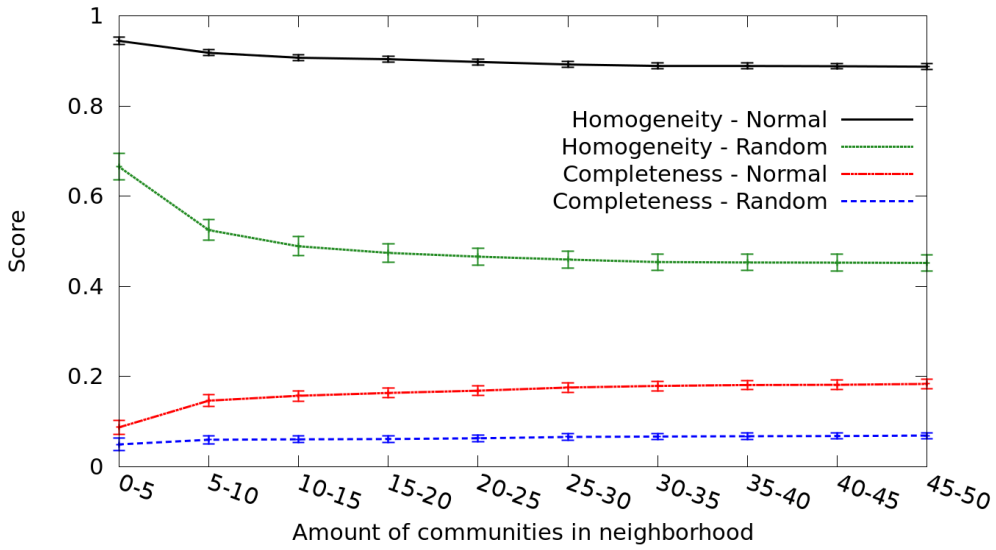


Figure 4.8 Homogeneity and completeness highly decrease for the null model confirming that Twitter users form communities of interest

Our results can be observed in Figure 4.8. To assess the baseline, only homogeneity and completeness are shown, since they provide enough information to establish if our approach performs better. The scores from our actual communities built in our approach are denoted by *normal*, whereas the scores from the null model are shown as *random*.

We can observe a large drop in the scores for the baseline model, which obtains, on average,

a score of 0.063 for completeness and 0.49 for homogeneity. Our communities, on the other side, achieve results of 0.16 for completeness and 0.90 for homogeneity. In addition, we ran a *Z-test* to compare each metric from the baseline to our approach. For both of these metrics, we obtained a statistically significant difference with p-values lower than 0.0001.

The higher completeness and homogeneity scores confirm our hypothesis, in that communities discuss certain topics, which are different from other communities' topics. Particularly, the high homogeneity score indicates that topics are effectively mostly seen in one community, thus, communities discuss topics that are distinguishable from other communities. The low completeness score indicates that, while communities have distinct topics, they do not discuss a single topic, but multiple ones.

4.6.4 Communities of Interest Clustering Evaluation

To establish topics of discussion with LDA, we must specify the number of topics we wish to identify through the documents clustering. The number of topics can impact our mapping with structural communities. A low number of topics will yield more general topics, and might combine some topics and thus incorrectly identify two communities as discussing a same topic. A higher number of topics will provide more specific topics, but might decrease the clustering effectiveness. We test multiple values for the number of topics, from 100 to 1000, to establish which one best identifies our communities of interest. The average results are illustrated in Figure 4.9. As shown, a topic count of 500 provides the best scores.

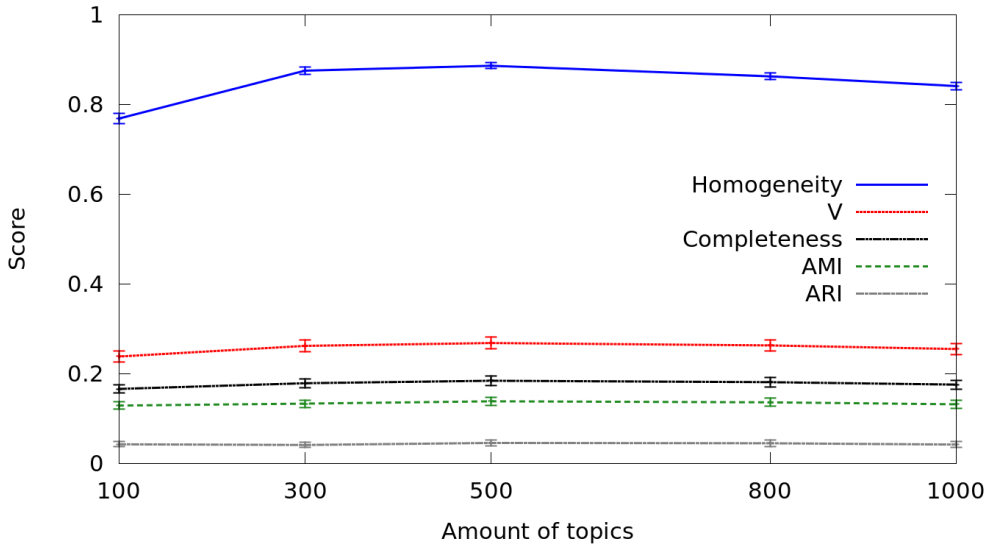


Figure 4.9 Our clustering metrics with different numbers of topics. The approach performs best with 500 topics

The number of topics does not greatly change our results. Most of the metrics showed an almost identical score, with only the homogeneity being higher for 500 topics, with a value of 0.87.

Another parameter of our experiment is the number of tweets per document. As previously mentioned in section 4.4.3, we group l number of tweets in documents to identify topics. We tested multiple values, from 5 tweets per document to all tweets in a single document. Figure 4.10 shows the scores of our metrics, where the best scores on average are observed with 20 tweets per document.

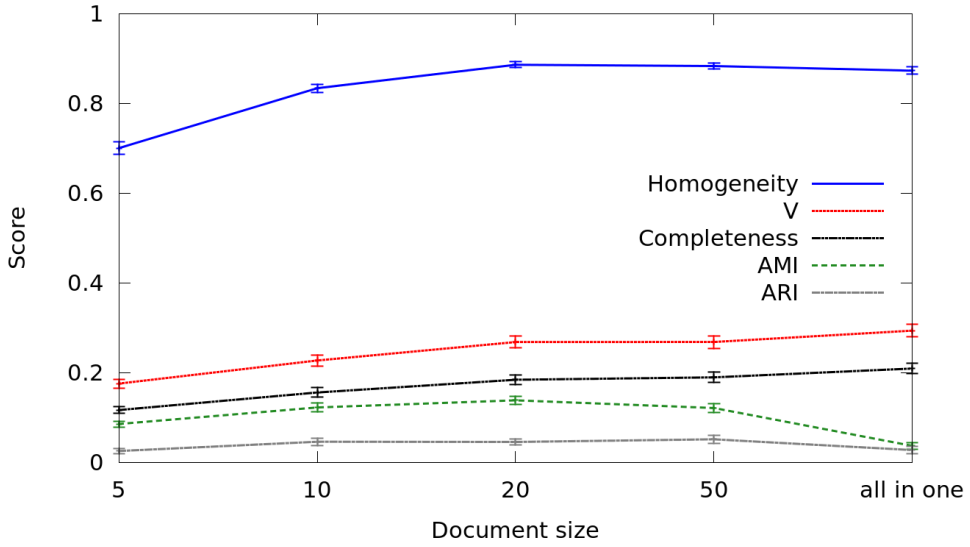


Figure 4.10 Our clustering metrics with different document sizes. The approach performs best with 20 tweets per document

Finally, the last factor studied in the construction of our communities of interest is the structural community detection algorithm. We tested our approach with every algorithm previously mentioned in section 4.5.3, with both the directed and undirected version of our graph. For this, the LDA topic detection results remain the same, but the structure of communities change. Figure 4.11 presents our results averaged per neighborhood. As can be observed, the algorithms obtain similar results, with a high homogeneity in every case, with 0.8 at its lowest to 0.89 at its highest, for Infomap. This demonstrates that communities have relatively distinct topics from one another. The low completeness is a sign that while communities have distinct topics, they each talk about more than one single topic. On average, the completeness is around 0.25, and Walktrap obtains the highest score with 0.31. Since we aim at having distinguishable communities of interest, we prioritize a high homogeneity over a high completeness, and thus, choose Infomap as the algorithm. Our experiments on

undirected graphs obtained similar scores to their directed counterparts. Since a directed graph provides more information and additional edges, we opted to use directed graphs with Infomap for the remainder of our approach.

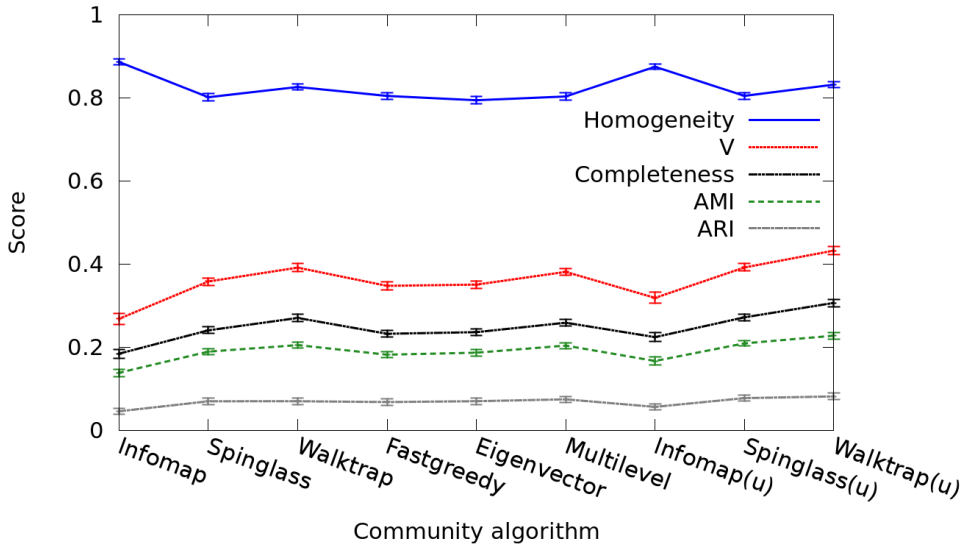


Figure 4.11 The communities and their topics of interest show high homogeneity but relatively low completeness. Algorithms applied on undirected graphs are specified with (u)

4.7 Evaluation: Spam Detection

In this section, we establish if spam messages can be identified by their paths of diffusion through communities of interest, in order to confirm our second hypothesis.

4.7.1 Clustering Similar Messages

Before building our probabilistic model, we first grouped similar messages using four-gram analysis on tweets found in our neighborhoods, after removing stop words and extracting URLs as single words. From our 300 neighborhoods, we obtained 1,219,991 groups of similar messages, ranging in size from 2 to 94,382.

4.7.2 Labeling the Dataset

In order to build our probabilistic models, and then train a classifier to detect spam messages, we must obtain a labeled dataset identifying if a message is spam or benign.

Since the number of clusters of similar messages is too large to manually label, we used a subset of the largest 5000 groups as our labeled dataset. Our intuition is that spam campaigns frequently use templates and operate at a large-scale, so using the top 5000 groups as a labeled dataset might provide a larger sample of spam messages. The size of our subset's groups ranges from 68 to 94,382 messages. These groups were then independently labeled by 14 security researchers, i.e., *coders*, following the methodology from previous work [124]. Since establishing if a tweet is spam or not frequently involves visiting URLs in the text, all researchers labeling our dataset were advised on the risks of potentially malicious URLs in messages. They used a safe environment to visit them, such as a fully patched and up-to-date VM, recreated between each visit. Each group of similar messages was labeled by three researchers, and the majority vote was used as a final label.

After a preliminary look at our tweets, we defined five categories for our labels, based on what we observed :

Spam : A message which asks the user to do something nefarious to themselves or to someone else, such as buying a suspicious product, voting in a certain way, visiting a URL, etc. The maliciousness of any URL is assessed by the coders when they visit it. A tweet with a URL linking to malware, adware, phishing, scams or other suspicious content fits into this category.

App-generated : A message which is automatically posted on the timeline of a user by an app. Many apps, such as Twittascope, regularly post identical messages on behalf of users. This category includes messages such as weather reports from IFTTT, or health and progress reports from health-tracking apps such as Fitbit.

Quote : A message which only consists in a famous or popular quote. Many users post inspiring quotes to their timelines, and since these quotes are posted as is, they form large clusters of similar messages.

Normal : A benign message, which appears in a group of similar messages because it is popular and trending, and many users are discussing it. News articles or blog posts posted as-is appear in this category, along with viral messages, such as memes.

Unknown : A message which cannot be labeled with enough certainty by coders.

Examples of manually labeled messages for each of our categories are provided in table 4.1. Potentially unsafe URLs were removed and replaced by [_URL].

By labeling 5000 groups of similar messages, we obtained labels for 1,277,833 tweets. Table 4.2 shows the result of our labeling of top clusters of similar messages. The size of each category is shown, as well as the number of tweets. As can be observed, our clusters of normal messages and spam messages are similar in their numbers, respectively at 44% and 42% of all groups,

Table 4.1 Examples of messages for each category

Spam	"Fellas need a mix 2 get ur lady in the mood heres a mix to help u succeed. [_URL]" "@X Listen To My Lul Song XXX Hot! Download And Share [_URL]" "Wow! another great item; available on eBay [_URL]" "WOW! No Cost Traffic For Your Website Home Based Business Blog Click Here Now Please #retweet [_URL]"
App-generated	"4 tweeps unfollowed (goodbye!) me in the past week. Thank you [_URL]." "New week; new tweets; new stats. 2 followers; 3 unfollowers. Via good old [_URL]." "August 28; 201X #Fitbit activity: 11XXX steps taken; 5.XX miles walked/ran; and 2XXX calories burned." "Your key planet Venus is now moving through your 12th House of... More for Libra [_URL]"
Quote	"Either you run the day or the day runs you. - Jim Rohn" "You're only as good as the people you hire. - Ray Kroc" "Do you want to know who you are? Don't ask. Act! Action will delineate and define you. - Thomas Jefferson" "Problems are only opportunities in work clothes. - Henry J. Kaiser"
Normal	"If you could ask a business consultant any question; what would you ask?" "Brendan Rodgers: Liverpool boss has no plans to leave club [_URL]" "RT @X: Thank you XX for last night. Hope you all enjoyed the show; you've always been lovely to us." "RT @X: Puppy caught eating paper decides killing the witness is the only way out [_URL]

although normal messages regroup a larger number of tweets. An interesting fact is that the app-generated messages, while only consisting of 7.5% of all clusters, include 33% of all tweets.

We can transfer the labels to users by identifying all the labels of a user's tweets. Table 4.2 shows the number of users in each category. A total of 66,788 users were identified this way. An interesting fact is that the number of users in the normal category, at 55,473, consists in the large majority of users. While the spam and normal categories have similar numbers of groups, the number of users in the spam category is considerably smaller, at 13,179. This indicates that users labeled as spam are responsible for more clusters of similar messages and more message posts per single user. Some users share multiple categories, where some of their tweets are labeled with one category and others with a different one. Some users are in both the normal and spam category. This might be the result of bots or fake accounts emulating normal accounts by posting benign tweets, or it could be the result of compromised accounts

that started posting spam messages, or it could simply be the result of a mislabeling of some messages.

Since our approach is run on neighborhoods, some of them had to be removed due to having too few benign or spam clusters. Neighborhoods were removed when having fewer than 10 benign clusters or less than 10 spam clusters. The *Final Labeled Dataset* section in Table 4.2 shows the metrics for our filtered dataset.

Table 4.2 Statistics for the manually labeled datasets

	Spam	**App**	**Quote**	**Normal**	**Unknown**
Labeled Dataset (300 neighborhoods)					
Nb. of groups	2,110 (42.2%)	376 (7.5%)	335 (6.7%)	2,178 (43.6%)	1 (0%)
Nb. of tweets	344,540 (27%)	416,099 (32.5%)	34,138 (2.7%)	482,975 (37.8%)	81 (0%)
Nb. of users	13,179	9,740	3,819	55,473	1
Final Labeled Dataset (202 neighborhoods)					
Nb. of groups	854 (30%)	274 (9%)	260 (9%)	1508 (52%)	
Nb. of tweets	168,181 (17%)	408,395 (40%)	32,607 (3%)	408,340 (40%)	
Nb. of users	12,504	9,614	3,815	55,219	

4.7.3 Parties of Interest

Next, our approach identifies *Partie of Interest*, i.e., messages that are posted in specific communities of interest. As mentioned in Section 4.2, we build a probabilistic table of the probabilities of messages from clusters of similar messages appearing in each community of interest. These tables are built using our labeled dataset of groups of messages.

As an example, for a neighborhood containing three communities, five topics of interest, and ten groups of similar messages, the probabilistic table will contain ten rows and five columns. The entry at row i and column j consists in the probability that messages in group i are being observed in communities with topic j.

Since our dataset consists of neighborhoods gathered from Twitter, our approach is applied at a local level, i.e., we identify spam messages and compute our probabilistic table for each neighborhood. Thus, our approach can be applied to larger datasets, or possibly the entirety of Twitter, simply by using additional machines and running our approach in parallel on additional neighborhoods. The entire process is scalable and can be run in parallel for every neighborhood chosen.

4.7.4 Spam Classification

In the end, following the various stages of our approach, the dataset used to classify spam messages, i.e., to test our second hypothesis, consists of 202 neighborhoods, 2,896 clusters of similar messages regrouping 1,017,523 tweets and 81,152 users.

Our final detection model incorporates all of our previous steps, and applies supervised machine learning in order to predict if a group of messages is spam or not. The features used in our training are the list of topics identified by LDA in a neighborhood. Observations are the parties of interest from the probabilistic table, i.e., probabilities that the group of messages has been observed in communities interested in each topic from our features. Finally, another feature is added to capture if messages tend to be posted by many users or a small minority. It consists in the number of users in the group divided by the total number of messages in the group.

The target classes which the model is trained to predict are the labels identified previously. Our model works with a binary target, so we combine our labels into two classes, *spam* and *benign*, following different combinations :

Combination 1 : $Positive = \{spam\}$, $Negative = \{normal, quote, app - generated\}$

Combination 2 : $Positive = \{spam\}$, $Negative = \{normal\}$

Combination 3 : $Positive = \{spam, app - generated\}$, $Negative = \{normal, quote\}$

This choice of combination can be specified in our approach depending on what type of spam messages we wish to detect.

Before training our model, we apply an oversampling method in order to balance our dataset, since the number of observations in both target classes are unbalanced. We apply the SMOTE [125] technique, which creates synthetic samples of the minority class in order to balance both classes.

To evaluate the results of our classification, we use a number of metrics, notably the accuracy, F1-score, precision and recall. We use k-fold cross-validation, in order to limit possible biases in our dataset and limit overfit. This technique consists in randomly splitting the dataset into k subsamples, where all but one subsample is used as a training set, while the remaining fold is used as the testing set. This process is repeated k times, using a new subsample as the test set for each iteration. The results are then averaged to obtain the final metric scores. We chose $k = 10$ as the value for our cross-validation, since it is the most commonly used value in previous work, and has been shown to be effective [126–128].

We build a model and classify on each of the 202 neighborhoods separately. There are on

average around 410 observations in each of these neighborhoods, while the minimum and maximum number of observations are 30 and 1,009 respectively. We test multiple algorithms for our model : Naives Bayes, SVM and Random forests.

All three algorithms provide similar results. Table 4.3 shows our results for SVM and the scores of the various metrics, for all three combinations of labels. As can be observed, our approach is effective at identifying all three combinations of spam messages, with high accuracy, F1-score, precision and recall. With combination 3, we obtain the highest values for our metrics, with an accuracy of 0.90, a precision of 0.91 and a recall of 0.93. It is worth noting that the majority of our observations are labeled as either spam or normal, which might explain why the different combinations do not greatly impact the effectiveness of the model.

Thus, following our results, we can validate our second hypothesis that normal and spam messages diffuse through distinguishable paths of interest, and that we can detect spam messages based on this.

Table 4.3 The performance of POISED on our various combinations of labels. The +/- values indicate the standard error

Label Combination	**Accuracy**	**F1-score**	**Precision**	**Recall**
Combination 1	0.89 (+/- 0.02)	0.90 (+/- 0.02)	0.85 (+/- 0.02)	0.95 (+/- 0.02)
Combination 2	0.90 (+/- 0.02)	0.91 (+/- 0.02)	0.87 (+/- 0.02)	0.95 (+/- 0.02)
Combination 3	0.90 (+/- 0.02)	0.90 (+/- 0.02)	0.91 (+/- 0.02)	0.93 (+/- 0.02)

4.7.5 Spam Accounts Detection

As previously mentioned, we identify spam accounts indirectly by considering a user posting spam messages as a spam account. For every user, we grouped their messages and computed the percentage of them being spam messages. As can be observed in Figure 4.12, the majority of users post either only spam messages or only benign messages.

However, in order to assess if the users posting both spam messages and benign messages are spam accounts, we establish a percentage threshold, where users posting a higher percentage of spam messages than the threshold are considered as spam. The threshold can be set depending on the needs of whoever uses our approach, i.e., if they want to adopt a stricter spam filtering or not. For our analysis, we set the threshold at 40%. With this, our dataset contains 15,055 spam accounts and 49,675 benign users, which correspond respectively to 23% and 77% of our total users.

While this method of spam account identification is a simple approach, it should be accurate

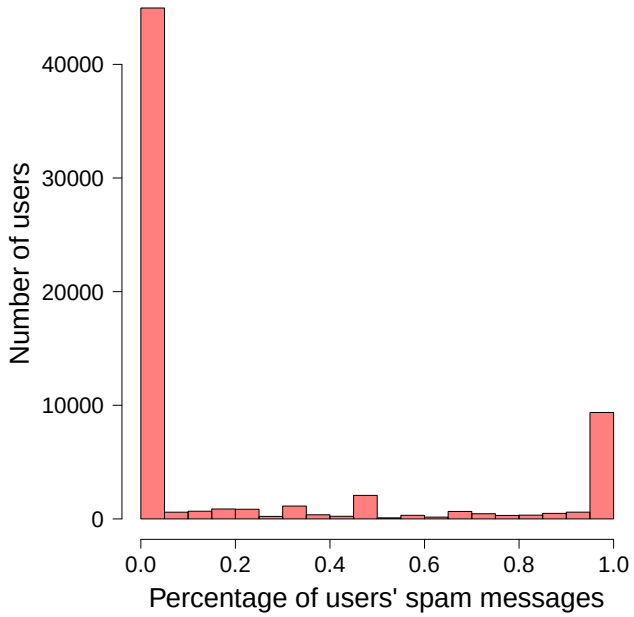


Figure 4.12 Percentage of spam messages posted by individual users

for most cases. Specifically for the case where an account might have been compromised, the average date and time of spam messages could provide an additional indicator to identify if the account posting spam was previously compromised, or if it might be currently compromised.

4.8 Discussion

4.8.1 Live Implementation

We presented our approach, POISED, which identifies spam with high precision and recall. Our experiments were made offline, on data gathered from Twitter. In this section, we identify the complexity of the various steps in our approach, and identify the requirements for running it as an online model in a live implementation.

First, our community detection algorithm, Infomap, has a complexity of $O(N)$, where N is the number of nodes in the network graph built from user connections. Infomap performs well on large graphs containing millions of nodes [112, 129]. Secondly, our topic modeling algorithm, LDA, has a complexity of $O(DTV)$, where D is the number of documents in the text corpus, T is the number of topics specified, i.e., 500 in our experiment, and V is the number of words in the vocabulary, built from our tweets. LDA ran relatively well on our large corpus, computing our topics in nine hours when run from a single machine. Even so, there have been a number of works examining how to improve the efficiency of LDA [130–134],

which could be integrated into our approach. Thirdly, our clusters of similar messages are identified using four-gram analysis, which has a complexity of $O(M^2S^2)$, where M is the number of messages and S is the maximum size of a message, i.e., 144 characters for Twitter, at the time of our experiments. Running four-gram analysis on our dataset took an hour on a single machine. There are a number of more efficient text comparison methods, which could improve the efficiency of our approach [135–137]. Finally, the SVM classifier is light on resources and runs quickly on a single machine.

Our approach is applied locally, in neighborhoods, and can thus scale up well with the addition of machines that run in parallel. The daily average of posted tweets on Twitter is 500 million tweets. Considering that our approach was tested on a million tweets, we can assume it could be run on all new daily tweets by using roughly 250 machines in parallel. Additionally, the topic detection algorithm can be run offline and less frequently to improve the performance.

In order to run our approach in a live setting on Twitter, it would need to have a labeled dataset of tweets to build the probabilistic model and train the classifier. Fortunately, many social networks, including Twitter, have a functionality which permits users to report messages as spam, which could be used as the ground truth for POISED.

4.8.2 Limitations

Our approach was evaluated on a subset of data from Twitter. We retrieved a random sample of 300 active users and crawled their neighborhoods in order to obtain a large dataset. While our initial dataset is relatively large, we further filtered the data by only keeping the 300 most recent tweets of every user, eliminating small communities, and removing small clusters of similar messages. Nonetheless, our final dataset consists in 202 neighborhoods, including 81,152 users and 1,017,523 tweets, which we consider sufficient in order to test our hypotheses.

Our approach is applied to groups of similar messages, similarly to previous works [53, 102]. While most spam campaigns use templates, an attacker aware of our text comparison algorithm, four-gram, could build spam messages to evade detection. To solve this, a more sophisticated text similarity analysis could be implemented into our approach [53].

Finally, another limitation of our approach consists in the identification of communities of interest, which requires a minimum number of users in communities and a minimum number of messages to work. An attacker could evade our detection by having a low number of connections to other users, or a low number of posted messages. However, this goes against the primary goal of spammers, which is to send a maximum number of malicious messages

to a maximum number of users.

4.8.3 Related Work Published Following POISED

This work was part of a collaborative project, which has been published [138] and presented at the ACM Conference on Computer and Communications Security (CCS) conference in 2017. Since the publication of the experiments described in this chapter, more research have tackled the problem of spam detection in social networks, some directly building upon our approach.

Alghamdi *et al.* [139] published a short paper on a framework detecting spam messages on Twitter with a machine learning approach, working with the focus of users on the social network. This was established with the use of multiple features related to users' topics of discussion, built with LDA. Another research by Andriotis *et al.* [140] focused on the detection of spammers on Twitter, using LDA to build features based on the topics of tweets and combining them to other content-based features to identify spam. They experimented with many combinations of features to identify how to best detect spam content and obtained detection rates (F1-score) of 95%.

Echeverria *et al.* [12] tackled the issue of spam detection by trying to establish if a classification approach to detecting Twitter bots is generalizable. For this, they employed a classifier with features derived from the users and the tweets, and built their model on a set of bot classes. Their classes are the datasets from which the bots originate, accumulated through different means and by various actors. When classifying, they aimed at establishing how accurately a bot class can be detected when trained on other classes, i.e., if a model trained on one set of datasets generalizes to another dataset. Thus, for each class, they trained their model on all the other classes and tested on the target bot class. Their results show that a model accuracy drops when trying to classify an unseen class, hinting at the fact that spam classifiers have trouble adapting to new bots.

Mulamba *et al.* [141] built a framework to detect fake accounts on social networks, namely sybil accounts. Their approach revolves around the use of structural features of graphs constructed from social network accounts. The features used are centered around the k-core of graphs, the degrees of nodes and indicators of centrality. Their framework implements various classifiers that use the previous features, and they are tested on datasets from Twitter and Facebook where they obtain high accuracy. Wang *et al.* [142] detected fake accounts on social networks, also by utilizing the graph structure of the accounts. They removed the labeling step of the classification used by previous approaches by employing state-of-the-art detectors on randomly assigned labels and applying a newly establishing heuristic, named the

Homophily-Entropy Aggregator. Their heuristic searches for groups of accounts with similar labels through various metrics in order to use these in the classification. It has been shown that fake account detectors perform well when the number of fake accounts is higher. Thus, they built a classifier which does not require manual labeling. They tested their approach on Twitter data, against other state-of-the-art methods, and obtained higher detection rates of fake accounts, particularly when using only a small labeled dataset as a starting point for the training.

Xiao *et al.* [143] built a framework to establish if a cluster of new accounts is a cluster of fake accounts as soon as they register on the social network. Most of the previous approaches explored account characteristics or behavior after its creation, in order to establish if the profile is fake. This research was able to obtain a dataset consisting of the registration information of accounts. From this, they built clusters of new accounts based on their registration IP addresses and registration dates, and then built a classifier to predict if a cluster of new accounts is a cluster of fake accounts. Their accuracy for detecting malicious accounts is 98%. Their system was deployed at LinkedIn and performed well on live data. Another similar framework was built by Yuan *et al.* [144], where they establish if an account is spam based on its registration pattern. Similar to [141, 142], they built a graph with the registration features of accounts : they employed both synchronization based features and anomaly based features. Since sybil accounts cluster together in the graph, they used community detection algorithms in order to identify sybil clusters. They tested and deployed their system on WeChat, a large Chinese social network, and obtained a high sybil detection rate.

Finally, Tuttle *et al.* [145] built a similar framework as our published approach, and applied the concept to detecting individual spam messages instead of clusters of similar messages. They built communities of interest following our methodology, and created a message dissemination vector to be used in a classifier identifying spam from benign messages. This vector represents the number of times a message has been seen in a community of interest. While they limited their spam messages to three specific categories, they obtained high accuracy in the detection of spam messages.

4.8.4 Future Work

Our communities of interest are built using structural community detection algorithms which separate nodes into distinct communities. While our framework was effective with this approach, we could implement an overlapping community detection algorithm, which permits users to be part of more than one community.

We applied our approach to 202 neighborhoods, where the seed and its friends and followers were used as data. An interesting future experiment could be to test our approach on larger networks, e.g., by also using friends and followers of the seed's connections in the data. Our neighborhoods would then include 2nd-degree connections to the seed as well as their direct connections. However, it is worth noting that this would impact the performance of our approach, and it would require more resources to run.

Another future work could be to combine our approach with other spam detection models that are based on identifying spam by using other features, such as user account characteristics. We could build a combined framework which detects a larger set of spam messages.

Finally, while our approach can be applied to any social network, it was implemented and tested on Twitter. It would be interesting to test POISED on other social networks, such as Facebook, or Instagram, and observe how its results change.

4.9 Conclusion

In this work, we built POISED, a novel spam detection framework which uses the way messages spread through communities of interest to identify spam messages, and thus, identify spammers. Contrary to previous works, our approach does not rely on user features or message features, which can be evaded, and instead observes the way spam messages travel through communities to identify them. We first constructed communities of interest by applying topic modeling to messages in structural communities, and confirmed that communities have distinct topics of discussion which can be identified. Then, we built probabilistic models of clusters of similar messages spreading through these communities of interest. These were in turn used in a classifier to identify spam messages from benign messages, and we obtained high recall and precision when evaluating the model.

Through an experiment on a dataset of close to 1,300,000 tweets and 64,000 users from Twitter, we showed that the path of diffusion of spam messages through communities of interest differ from normal messages.

CHAPTER 5 CLASSIFYING EXPLOIT KITS BY THEIR WEB REDIRECTION CHAINS

5.1 Introduction

In the process of an infection, once a victim clicks a malicious link, they will, in many cases, be automatically redirected through a series of websites, until finally landing on a website hosting an exploit kit. This exploit kit will fingerprint the user's machine, e.g., by establishing its browser version, identifying which browser plugins are present and detecting which version of common software such as Java, Flash and Silverlight are installed. This information is then optionally encoded or encrypted and sent back to the malicious actor's server, which will in turn send back to the client a webpage exploiting one or many vulnerabilities present in one or more of the previously fingerprinted items.

This exploitation will complete the infection of the victim's machine by installing a malicious software.

Previous research has focused on detecting malicious files and malicious websites, and creating signatures or behavioral patterns identifying them. One approach, by Zhang *et al.* [146], also employed a different avenue, by utilizing HTTP request errors as an identifying factor of malicious activity, given that malicious software often make more requests to web domains that have been taken down, or to domains that might not exist. Although this work uses network data to identify malicious activity, it does not use the web redirections occurring in malicious infections. Other research, such as Stringhini *et al.* [147] and Mekky *et al.* [148], use features from the network and malicious infrastructure in their malware detection system, but do not focus on classifying exploit kits solely on the prior network redirections.

With the emergence of Exploit-as-a-Service (EaaS) [149–151], exploit kits are now offered to cybercriminals as a service. Malicious actors performing a cyberattack can simply buy the services of the exploit kit in underground markets and integrate it to their own operation. Although the use of exploit kits has declined in the past years [152], these remain a threat to online users as long as vulnerabilities exist in web browsers and plugins.

In this work, our approach aims at using the underlying infrastructure and systems implicated in an infection by classifying an exploit kit according to the different websites a user has to go through to arrive to it, and how they redirect to one another.

In the context of EaaS, we wish to establish if the path a user travels when redirected to an exploit kit changes completely from one campaign to another, or if patterns can be identified

and used to predict the exploit kit family. In short, since exploit kits are sold by themselves, can they still be clustered according to the chain of redirections leading to them?

Our hypothesis consists in:

Hypothesis: *Exploit kits can be clustered according to the web redirection chains leading to them.*

In a nutshell, our approach aims at 1) reconstructing from network traces the web redirections leading to a malicious file, and 2) classifying exploit kits according to their web redirections, to better understand the exploit kit ecosystem and malicious actors.

The main contributions of our work are the following :

1. We build a framework capable of successfully recreating web redirection chains by extracting requests from a network capture and establishing the links between them.
2. We build a classifier capable of predicting exploit kit families using solely features derived from their web redirection chains.

5.2 Related Work

The presence of web redirections in malware and exploit kits is a phenomenon that has been observed by researchers, notably by Cova *et al.* [153] in their analysis of drive-by-download attacks through JavaScript code. Web redirections occur in web attacks for a multitude of reasons, for example to hide the final malicious webpage, to fingerprint the browser before the attack, to direct traffic, to evade detection by security software, or because the attack happens through advertising and goes through advertisement market networks. Although web redirections can be legitimate, studying their behavior in a malicious context can help in preventing and detecting attacks on the web. In the following section, we present research that studied redirection chains in the context of malicious infections.

5.2.1 Redirection Chains as a Detection Method

Web redirection chains can be used as a tool or as a feature among others to detect malicious websites.

Stringhini *et al.* [147] studied web redirection chains as a way of identifying malicious webpages. They created SpiderWeb, a tool that detects malicious webpages by establishing a redirection graph. This graph regroups similar web redirections and a number of features to be used by a classifier. Their data consists in web requests captured from a browser plugin,

taken from February 1, 2012, to March 31, 2012. A graph is established from these, and then aggregated according to similarity metrics. Following this, 28 characteristics are extracted from redirect chains, and supervised learning is employed to model malicious chains. The ground truth consists in chains identified as either malicious or benign by an antivirus software, and they manually verified the labels. Redirection chains smaller than 4 in length or bigger than 8 were not considered for the study. Their classifier is based on SVM and trained with SMO, and uses 10-fold cross-validation. Its implementation is provided by the Weka machine-learning tool [154]. Following their classification, the most prominent features were the *referrer country ratio*, which indicates a ratio of the geographic diversity of pages leading to the redirection chain, the *Graph has hub 80%*, which refers to the redirection chain containing a hub that is present in 80% of all chains, and finally the *Final page domain is an IP*, which indicates that the final domain is an IP address.

Lu *et al.* [155] observed web redirection chains in the context of search poisoning detection. A search poisoning consists in the manipulation of search engines with the goal of redirecting a maximum of users towards a certain page, frequently malicious, without that page being relevant to the initial search query. They captured the web redirections when visiting a search engine result, and established their features to use in a classifier for automatic detection. They implemented a browser plugin that captures requests originating from a search engine, along with the following web redirection chain. Similarly to Stringhini *et al.* [147], features were then extracted from the redirection chains and used in a J48 classifier in Weka [154], which consists in an implementation of the C4.5 algorithm [156], a supervised learning algorithm using decision trees. The labeling of the 2344 redirection chains was done manually by researchers.

Lee *et al.* [157] built a system, named WarningBird, that detects malicious messages on Twitter. This framework's data consists in Twitter messages containing a URL, which was visited so that the following web redirection would be captured. Their system only supports static redirections, which consists in HTTP 30x redirections. These redirections happen when the server sends to the client an HTTP response code indicating a redirection, with the location of the new resource. Thus, they do not support JavaScript redirections. They used a two-month period for the training of their classifier. They then employed supervised learning with features extracted from the accounts posting the tweets. They labeled the tweets according to the status of the account, i.e., whether it is suspended or not. They both tested logistic regression and support vector classification. They obtained their best results with logistic regression.

Li *et al.* [158] analyzed the insertion of a malicious redirection to known legitimate JavaScript

libraries. They crawled the 25,000 most popular websites, according to Alexa[1], to gather a list of common JavaScript libraries. Microsoft provided the dataset of malicious libraries for the ground truth. They employed multiple similarity algorithms, starting with hashes of parts of scripts, and then using Google Diff on similar pages that include small differences. If the added code appeared to be a redirection, they considered it as malicious. Their tool identified 68.07% of scripts from the set provided by Microsoft, with a false positive rate of 0.0073%. A noteworthy discovery through these experiments was that a high proportion of code is reused in malicious pages, and that 54% of their malicious JavaScript files used a type of obfuscation.

Another approach to using redirection chains as a method to detect malicious webpages was presented by Mekky *et al.* [148]. They obtained network traces of HTTP traffic from an ISP for a 24-hour period in August 2011 and April 2012, which resulted in 101,000,000 HTTP requests. They established the redirection chains of requests by creating trees of user browsing. The trees start with the original URL and then add a child for every request made from this site. These children in turn have a child added for every request made from it, and so forth. To establish if a request has been made from a site, they analyzed the content for the target URL. Various characteristics were extracted from these redirection chains, and a decision-tree algorithm implementation in Weka [154] was used for their classifier. The labeling was done through the identification of malicious pages with a commercial IDS.

The most important features of malicious redirection chains obtained from their classification are :

- The number of different domains in HTTP 3xx redirections
- The number of consecutive short edges
- The redirection duration
- The length of the redirection
- The number of consecutive HTTP 3xx redirections
- The number of different domains
- The number of consecutive HTTP 3xx redirections from different domains
- The number of HTTP 3xx redirections

[1] http://www.alexa.com/topsites

Nelms *et al.* [159] observed a sample of drive-by download malware from an academic network and established *web paths*, i.e., the HTTP requests leading to the download of malware on the victim machine. They developed a system called WebWitness, which aims at classifying the malware based on its web path, i.e., identifying if it is a social engineering, drive-by download or update/dropper software. To establish the web path leading to malware, they employed a similar technique to Mekky *et al.* [148]. Rather than establishing trees of user browsing, they made transactions graphs with weights of the previous requests leading to malware. Their system uses a random forests algorithm for classification. They also detected malicious code injection in legitimate pages using their classifier.

Akiyama *et al.* [160] established a honeypot in order to attract malicious requests over a period of four years. They purposefully left credentials accessible in their target websites, and analyzed the web redirection injected onto their 776 websites. The majority of redirections, consisting in 71% of all redirections, were done with the addition of JavaScript code inside an HTML page, often obfuscated. Attackers had various goals when inserting redirections, from doing click-fraud to executing a drive-by download. Particularly, they observed a shift from redirecting to malicious downloads in the beginning to making click-fraud towards the end. They noticed the use of domain-flux by web-based DGAs, IP-flux by FFSNs, control of redirection by TDSs, and target profiling by tracking services.

Kwon *et al.* [161] developed a system to detect malicious URLs by using their redirection graph. Their motivation was the observation that the malicious redirection infrastructure is interconnected. They extracted features from the redirection chains of URLs, according to 3 categories: features established from the way resources are shared in malicious infrastructure, features established from the heterogeneity of the infrastructure, and finally features built from the need in flexibility of spammers. Due to the complexity of labeling each URL from their dataset, a semi-supervised classification was used, where a small number of URLs were labeled. Their dataset consists in 15M tweets from Twitter from 2014. They crawled the redirection chains for each URL, where 99% of chains' length were less than 6. An account was labeled as malicious if its page was suspended. The most important features emerging from their classification were :

1. Total geo-distance (in kilometers) of hops in redirection chains
2. Mean domain count of the redirection chain connected component landing URLs
3. Mean domain name count per URL in redirection chains
4. Edge density of the redirection chain connected component

Hans *et al.* [162] extracted redirection features from visiting 2383 malicious URLs from PhishTank, a website focused on identifying phishing websites. The data was labeled semi-manually, and a neural network algorithm was employed to model the malicious URLs according to their features. They used MATLAB for their implementation, which uses back-propagation, with 2 hidden layers. The features used in their algorithm consist in:

1. Number of different domains
2. HTTP status code
3. Meta refresh
4. Number of script generated redirections
5. Number of redirection hops
6. Use of iframes

Burgess *et al.* [163] built *REdiREKT*, a framework which extracts malicious redirections from exploit kit network traffic. Their approach constructs web redirection chains from network packet captures, similarly to other approaches [148]. Their datasest is from network captures around exploit kits, although some of their redirection chains are missing the initial host. They identify a number of features to build a machine learning model. Contrary to our approach, they include a number of features generated from URLs. Their approach successfully extracts malicious domains from their redirection chains, and they additionally present statistics on the proportion of different types of redirection for various exploit kit families. The referrer is the most common method of redirection, however JavaScript code is used in higher proportion for malicious redirections.

5.2.2 Research With Other Uses of Redirection Chains

Other works have also used web redirection chains to analyze the infrastructure of malicious actors or to build a classifier modeled on a broader set of features.

Li *et al.* [164] used redirection chains to establish a graph of malicious infrastructure, composed of *Hostname-Ip clusters*. They visited URLs with a browser and noted the requests made from an initial malicious URL. Malicious nodes were identified in the graph by their interconnection and the small number of connections from external legitimate sites, using the PageRank algorithm [165]. A large percentage of malicious nodes, at 31.98%, were systems hosting TDSs.

Research by Zhang *et al.* [166] consisted in analyzing HTTP network traces to establish the central servers of distribution of drive-by downloads. They automatically generated regular expressions based on these central servers, regardless of their position in the HTTP trace. They built a tool named *ARROW*, which establishes signatures for drive-by downloads.

Huang *et al.* [167] explored redirection chains briefly in their analysis of *spam* infrastructure, but only as one characteristic among others of spam.

Cova *et al.* [153] developed a malicious JavaScript detection tool named *JSand*. They extracted features from malicious webpages and built a classifier to detect malicious JavaScript code. These features included the number of times a browser is redirected to a different URI upon visiting the page. However, their dataset mostly consisted of the end point of an infection, thus, the redirection feature was not impactful in their model.

Chen *et al.* [168] built a detection model of hidden malicious redirections, using URL and Cascading Style Sheets (CSS) features of webpages. Their approoach obtained a high accuracy and performs well over time.

5.2.3 Exploit Kit Detection

Exploit kits consist in one aspect of malicious infections, and various research has been conducted on the detection and analysis of these.

Taylor *et al.* [169] approached the detection of exploit kits by transforming it into a subtree similarity search problem. First, they built a malware index from malicious network traces of honeypots, where they constructed malicious trees from the HTTP interactions of exploit kits. Web sessions were recreated by establishing redirections between sessions, using the referer HTTP header field and 302 redirect status code. They employed content-based indexing, e.g., the URL and HTTP header features, as well as structural indexing. Secondly, they monitored traffic on the edge of an enterprise network, and extracted the web sessions, using the previous methodology, in order to compare them to their malware index. They compared the effectiveness of their system to Snort, a popular IDS [170], and it obtained a similar true positive ratio, but their system obtained a significantly lower false positive ratio.

Stock *et al.* [171] developed a system called Kizzle, which creates detection signature for exploit kits. It splits into tokens the unpacked JavaScript code of exploit kits and applies similarity algorithms to create signatures to automatically detect the exploit kit. They trained their classifier on a total of four exploit kit families.

Kotov *et al.* [172] successfully deployed 33 exploit kits in order to analyze their functioning. They established how exploit kits fingerprint their victim's system and browser, and what

characteristics they observe. Through their analysis, they showed that exploit kits use the same general method. First, they detect the user agent, then proceed to do IP blocking, user agent validation, potential exploits detection, and finally exploits obfuscation. The control panel of exploit kits was analyzed, along with its various statistics and settings. Finally, they demonstrated that each exploit kit has a distinct code base.

De Maio *et al.* [150] also analyzed over 50 exploit kits. They detailed similar aspects as Kotov *et al.* [172], e.g., the fingerprinting of a victim's machine, and the deployment of exploits. A new aspect shown by this study was that many of the exploit kits share common code, hinting at the fact that they are operated by the same entity or forked from the existing exploit kit. They implemented an automatic exploit kit analysis tool, named *PExy*, that can detect all the paths to malicious output.

Eshete *et al.* [151] implemented a classifier to detect malicious URLs hosting exploit kits. They analyzed 38 exploit kit families, and extracted multiple features from exploit kits' attack-centric and self-defense behaviors. They identified multiple exploit kits using complex redirection schemes. These redirections are part of the exploit kit, and do not take into account the whole malicious infrastructure leading to the malicious webpage. In total, they extracted 30 features, and then built a classifier that detects malicious webpages following these features. Weka [154] was used to implement multiple algorithms: J48 Decision Tree, Random Tree, Random Forest, Bayes Network, and Logistic Regression. They used 10-fold cross-validation, and obtained perfect detection on their dataset, with multiple algorithms. When testing on a new set, they observed the best results with the J48 Decision Tree.

While some research has focused on detecting exploit kits specifically, others aimed at identifying general malicious files used by exploit kits. Šrndić*et al.* [173] implemented Hidost, a malware detection system utilizing the file content and logical structure to build a machine learning model. It is based on a malicious PDF detection system by Šrndić*et al.* [174]. They built a classifier from the extracted features of PDF and SWF(Flash) files, and obtained a high accuracy. PDF and SWF files are frequently used as the attack vector by exploit kits.

Grier *et al.* [149] investigated how exploit kits serve drive-by downloads in order to infect users. While they did not build a detection framework for exploit kits, they uncovered valuable information regarding the ecosystem of EaaS. They observed that over 47% of their malicious URLs led to an exploit kit, and identified that the nine most popular exploit packs account for 92% of exploits. In these experiments, 32 malware families were identified as installed by exploit kits. Finally, they also established the popularity of various malware families by the number of DNS lookups recorded, and showed that exploit kits are short-lived, receiving traffic for only 2.5 hours on average.

Kim *et al.* [175] approached the problem of detecting exploit kits by employing a string similarity matching algorithm, notably in the hopes of identifying isomorphic variants. Their features are probabilistic, size-based, and distance-based text features. Firstly, they clustered the exploit kits, and then extracted text features and converted them to numerical values. These were used in various machine learning algorithms. They employed a similar classification technique as Webwinnow, i.e., using Weka [154] with similar algorithms. They obtained their best results with the Random Forests algorithm.

While many researchers have been studying various methods of detecting exploit kits, software has also been developed to tackle the problem. Dell'Aera [176] implemented *Thug*, a honey client designed to mimic browser interactions in order to detect malicious pages and exploit kits. They employed vulnerability modules in order to detect malicious code.

Kim *et al.* [177] implemented a malicious webpage detection system where they inject a DLL into Internet Explorer, which in turn hooks onto various Windows API calls. These malicious webpages include exploit kits, but their system also detected various malicious webpages after an infection occurred. They identified malicious webpages and exploit kits by employing a blacklist and a whitelist of process names, and identifying suspicious API function calls. They compared their system with previous research, including Thug [176], and obtained better performance in both the accuracy and analysis time.

5.2.4 Summary

Previous research utilized redirection chains as part of multiple features used to build a detection model for malware. Specifically for the detection of exploit kits, redirection chains have only been analyzed in the context of the exploit kit's execution. However, our approach aims at analyzing the entire redirection chain leading to an exploit kit, in order to capture the whole infrastructure and the different actors involved in infecting a user with malware. With this information, we aim at classifying the various exploit kit families. Although some research analyzed malicious infrastructure, none observed the complete redirection chains leading to an exploit kit.

5.3 Methodology

In this section, we gather our dataset, and then present our approach in reconstructing web redirection chains, followed by the construction of our classification model.

5.3.1 Data Collection

The data consists in network traces of a browser accessing malicious sites. The data was retrieved from Threatglass [178], a former online platform that published malicious network traces gathered from virtual machines visiting suspicious links. We crawled an initial 8863 traces from this website, but further filtered this dataset, as will be described in the 5.3.2 section. We wrote a script to download the files from the website. Our dataset spans from May 2012 to June 2014. The network traces consist in PCAP files, i.e., packet capture files, which list all network packets intercepted while the capture was in effect. It includes every type of network activity, from DNS traffic to SMTP requests, to HTTP requests. Any encrypted traffic, such as HTTPS requests using the TLS protocol or connections made through the SSH protocol remain encrypted in our files, since the traffic is not generated by machines we control. From our 8863 traces, 2627 included at least one TLS connection. Upon further inspection, the main redirection chains leading to exploit kits in our data did not use HTTPS, and thus did not impact our analysis. Our approach can also be used with HTTPS, provided connections can be captured with a proxy, such as in many enterprise networks.

5.3.2 Redirection Trees

In this section, we will explain our methodology of extracting HTTP content from network traces, and building a tree of HTTP web redirections.

Formally, a web redirection chain can be described as :

Definition 19 *A* redirection chain *is a series of HTTP responses redirections, such that* $R = \{R_1, \ldots, R_n\}$*, where* R_1 *is the initial page, and the user travels through* $n-1$ *redirections until landing at the final landing page* R_n*.*

Web servers have a multitude of methods to automatically redirect a user to another page. The most common ways of redirecting a client requesting a webpage are :

30x redirect: The server sends to the client an HTTP response code indicating a redirection. The HTTP header will typically include the new location of the resource the client is trying to access. The most commonly used redirect code is a 302 status code, and status codes in the 300-400 range will also indicate a type of redirection.

meta http: The HTML code of the server's response includes a *meta http-equiv="refresh"* tag, which will be interpreted by the client's browser as a redirection to the page specified in the tag.

iframe: The HTML code of the server's response includes an *iframe* tag, which results in the browser requesting, within the current page, the URL specified in the tag. This type of redirection is stealthier, since the original page remains the main visible page. Although less frequently used, the *frame* tag also opens another webpage, alongside other pages. We will employ the term iframe to designate both types of frames for the remainder of the paper.

JavaScript: The server's response page includes JavaScript code that will be executed when the client loads the page, and will contain a redirection to another webpage. The JavaScript code can also be included in a remote script loaded and executed on load. Such code can use various functions, such as simply changing the value of the *document.location* variable for the intended redirection page.

Application: The server responds with an application, e.g., written in the Java or Flash, which will include a redirection in its behavior.

An example of a redirection chain from our dataset is presented in Figure 5.1.

HTTP is a stateless protocol, which implies that each web request and response pair is independent from each other. This provides a challenge when constructing a series of requests and responses from a single user browsing a website, as the order of requests cannot be extracted directly. Moreover, a typical visit to a website will not only involve loading the page's content, but also loading various resources, images, and advertisements. These can also in turn load their own resources and images. In the case where multiple redirections happen when browsing the web, this loading process will occur at each step of the redirections. Considering this, recreating the correct path of redirections leading to a certain final webpage amidst the various resources and requests requires a method of identifying redirections between webpages. Thus, our first step is to create a tool which builds the correct redirection from a network packet file.

Our redirection builder tool works as follows. First, the HTTP requests and responses are extracted chronologically from the network traces, using TcpTrace [179], a software that parses network traces and extracts every HTTP request and response according to the timestamp field of TCP packets. We then decompress the Gzip content of each HTTP request, and only keep GET and POST requests, with their responses. Gzip is a compression format used to compress HTTP requests body. It is specified in the *Content-Encoding* HTTP header of the requests extracted from our network traces. Finally, similar to Mekky *et al.* [148], we then construct a *redirection tree* from the HTTP requests and responses.

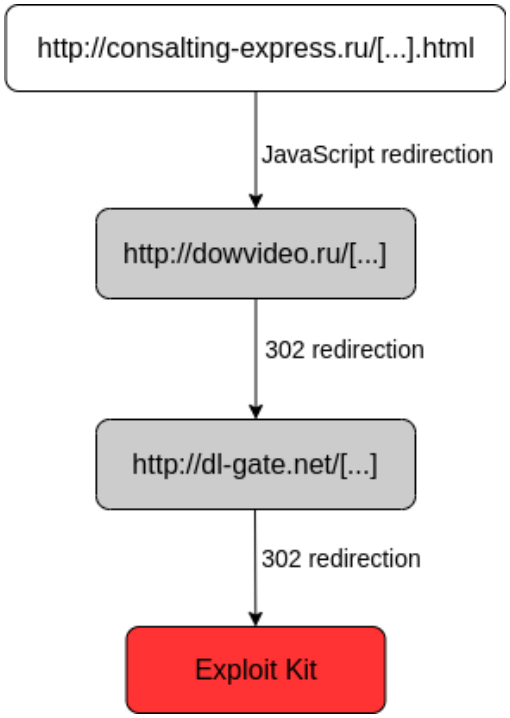


Figure 5.1 An example of a web redirection chain. The initial node is the URL visited by the user. The gray nodes depict potentially malicious nodes, while the red node is the page hosting the exploit kit. Edges depict the redirection method between websites

Formally, a redirection tree can be described as :

Definition 20 *A* redirection tree T *consists in a tree where the root node R, a starting HTTP request, has j children $T_i = [T_1, T_2, ..., T_j]$, where T_i consists in an HTTP request which has been redirected from R. T_i will have a subtree for each of its redirections, and will consist in a leaf when there is no additional redirection.*

For each HTTP request and response, we visit previously treated requests and responses, and identify each one containing a redirection, by employing a combination of string search and regular expressions through the source code and the HTTP headers. If a redirection is found through our search with the URL of our current request, the node is added to the redirection's branch as a child. If this isn't successful, we then look for the 'referer' HTTP header of the current request, which specifies the origin, i.e., the parent, of the current page, when present. The current node is added as a child to branches ending with the 'referer' URL. Finally, if the parent of the page was not found, we search in the previous responses' source code for the URL once again, but without its *query*, i.e., without the URL content following the ? symbol. While this is less precise, i.e., it might result in more false positives, we opt for this only if every other case failed to find any match, and we only consider this successful if a single match was found. Our algorithm is described in Algorithm 1.

Once this process is finished, we are left with a tree of all possible redirection paths that occurred from the root node. An example of a simple redirection tree taken from our dataset is presented in Figure 5.2, where the URLs have simply been identified by letters.

Third, we develop a heuristic to identify the malicious redirection chain among the various paths of the web redirection tree. This will be used to extract redirection chains from each redirection tree built previously, so that we are left with a series of final redirection chains. The different variables established that affect positively our selection of a path are:

1. The length of the path
2. The presence of an executable in the path
3. The amount of different website domains and redirection types included in the path

These criteria were chosen based on the fact that the desired path will span the entire tree depth or close to its entirety, and will typically end with an executable after going through a multitude of different hosts, whereas the loading of resources will be from the same host, and advertisements redirection paths will be shorter. It is worth noting that a large bonus is applied when an executable is found since a redirection chain including one is almost

```
Result: A tree of all the possible HTTP redirections
for Every network capture file do
    Decompress packets;
    root = first GET request;
    Create a tree with root;
    for Every HTTP request/response do
        if '30?' is in the response header status code then
            redirection = '30x';
        else if The URL ends with '.js' then
            redirection = 'jsfile';
        else if '<script>' is in the response body then
            redirection = 'script';
        else if '<iframe>' is in the response body then
            redirection = 'iframe';
        else if '<meta http-equiv' is in the response body then
            redirection = 'metahttp';
        end
        for Every leaf in the tree do
            if (leaf redirection is '30x' and leaf 30x URL == request URL)
            or (leaf redirection is '.js' and request URL is in leaf source code)
            or (leaf redirection is '<script>' and request URL is in script source code)
            or (leaf redirection is '<iframe>' and iframe URL is equal to the request
             URL)
            or (leaf redirection is '<meta http-equiv' and meta http-equiv URL is equal
             to request URL) then
                Duplicate the leaf's branch and add the request as a child with the
                 detected redirection;
            else if request URL truncated at '?' is in leaf source code then
                Duplicate the leaf's branch and add the request as a child with
                 unknown redirection as a tag;
        end
    end
end
```

Algorithm 1: Redirection Tree Creation

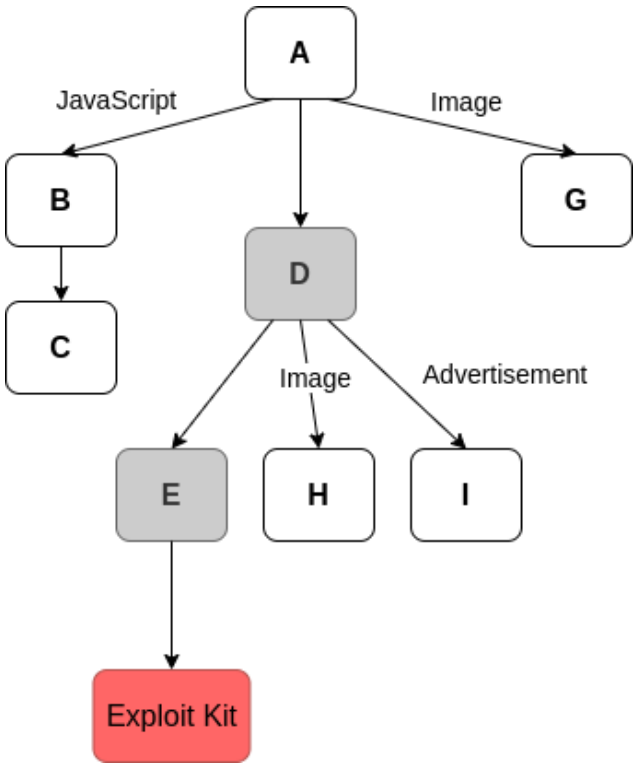


Figure 5.2 An example of a web redirection tree. The redirection chain $R = \{A, D, E, ExploitKit\}$ denotes the redirections towards the exploit kit

always malicious, given that the network traces are of the visit of a single URL with no user interaction.

Formally, our malicious branch identification heuristic can be described as :

Definition 21 *A heuristic calculating a score $T_h + T_r + B$ where T_h consists in the number of different domains found in the redirection tree T, T_r consists in the number of different redirection types found in the redirection tree T and B is a bonus value of 10 added if an executable is found in the chain. If no executable is found, B is equal to 0.*

To verify our redirection builder tool, as well as our heuristic, we manually constructed the redirection chains of 20 network traces from our dataset. We extracted HTTP requests and responses and looked through the headers and source code to construct the malicious redirection chains by hand. Our tool was able to correctly identify all malicious chains, with 2 of them having the correct path but missing only the final link due to obfuscated code in the previous response.

Thus, one limitation of our method is the impossibility of finding a redirection if it is only present in obfuscated JavaScript code. Obfuscation consists in a method where code is transformed as to appear difficult to read and interpret by humans, as well as to hide data, e.g., URLs. Fortunately, from our manual analysis of redirection chains, high obfuscation typically occurred late in the redirection chain, on the website running the actual exploit kit, or one step before. Thus, in these cases, our approach will still identify the redirection chain, but with the final node missing. It is worth noting that in the case where the 'referer' header field is present in the server response following the page containing the obfuscated code, the redirection can be established from it without needing to look at the obfuscated code.

5.3.3 Classifier

Malicious redirection chains have been extracted from our dataset using the tool described in the previous section. We then build a classifier to classify exploit kits according to these redirection chains using supervised machine learning. Classification consists in the application of an algorithm to map input values to a labeled output result, based on a set of features. Since we are using supervised machine learning, inputs must be initially labeled with the corresponding values we wish to correctly predict. The classification will then use part of the data, with their labels, to build a model that uses the data's features to correctly predict their target label. The model will then be tested on the remaining unused data, to predict the labels correctly, and calculate its performance. In our case, the data consist in redirection chains, and the labels are the exploit kit family associated with them.

To build our classifier, we first label our redirection chains with the exploit kit employed by the attacker. To obtain this, we first tested each network trace with regular expressions of known exploit kit URL patterns, provided by our antivirus partner. These are constructed from exploit kit samples scanned by the antivirus software, and they are constructed to precisely detect a given exploit kit, with minimum false positives. While this provides us with labels for multiple redirection chains, it does not help identifying exploit kits deviating from previous samples. Thus, to obtain additional labels, we then run Snort, a popular IDS by Roesch *et al.* [170], on every network capture in our dataset. This IDS includes many detection rules for multiple exploit kits and provides us with labels for additional files in our dataset. Some of the network captures had the same initial URL, and thus, the resulting redirection chain and exploit kit were also identical to previously tested samples with this URL. We consider these network captures testing a previously seen URL as duplicates, and thus remove them from our dataset. In total, after removing duplicates and any redirection chain not labeled with an exploit kit, we are left with 971 instances, from our initial 8863 network captures.

Following this, we establish 32 features from the redirection chains to be used in the classifier. Our features can be separated into three categories:

1. **Statistical**: These represent the various statistical features and measures extracted from the redirection chains. They consist in the length of the chain, the number of distinct websites, the number of IP addresses directly accessed in a redirection URL, and the number of each type of redirection.

2. **Contextual**: These represent various contextual features. They consist in the date when the network capture occurred, the first and second redirection types, and the last and second to last redirections types.

3. **Redirections**: These represent information regarding the actual redirections occurring in the chain. These features consist in the number of consecutive redirections of the same type, for each redirection type, and the number of each specific pair of redirections, i.e., a website using one type of redirection following another website using a different type of redirection.

Table 5.1 displays a summary of our features.

We intentionally did not include any feature related to the requests' content or their headers, such as the length of the URLs of a redirection chain, the parameters present in these URLs, the extension and type of any possible files requested through the chain, or any feature

built on the malicious payloads. Through this work, we seek to establish if an exploit kit can be classified using solely the redirection chain, and thus use only features from these chains. While using more features might prove more successful with our dataset, we believe the redirections leading to an exploit kit are more persistent through time and thus should provide better classification in the long term.

The C4.5 decision-tree algorithm [156] was chosen to classify the exploit kits. This algorithm has been used successfully in previous research [148, 155], and provides us with a tree of features which we can interpret in terms of feature importance for each exploit kit. This algorithm was chosen in favor of other classification methods for its interpretability : it provides an observable path of which features link to a certain label, through the tree nodes leading to a leaf. We also opt to build a second classifier with the Random Forest algorithm [180], which consists in an ensemble learning method, i.e., a method which builds multiple decision trees and aggregates their class prediction for each label. While this second method is more costly to compute, it generally provides better accuracy than a single decision tree. Both these algorithms use decision trees for the classification, which supports encoded categorical features, which the majority of our features consist of. A decision tree establishes its predictions by building a tree where a node is a test on a feature, and a leaf is the class prediction. To test a categorical feature, the tree will choose the category which provides the most accurate classification for the node. Our classification will use 10-fold cross-validation as the training method, in order to limit potential overfitting of our data. Overfitting will occur when the classifier models too closely the dataset and not the general phenomenon we aim to identify, and will thus work poorly on new datasets. This method works by randomly partitioning our dataset into, in our case, 10 *folds*. The algorithm uses 9 folds for training and the remaining one for testing, and then repeats the process 9 times varying the fold used for testing. This method has been used with success in previous research [147, 155], and 10 is the most commonly used number of folds in the literature.

As the implementation of the classification, we employ a tool named Weka, an open source machine learning tool developed by Hall *et al.* [154]. It provides multiple classification algorithms that can be applied to a provided dataset, and has been used by previous research [147, 155]. In this tool, the C4.5 decision-tree algorithm implementation in Java is named the *J48 Decision Tree*. We also test our data with the random forests algorithm in Weka, in order to compare results with our initial classification.

Table 5.1 Features of the redirection chains and their category

Category	Feature
Statistical	Chain length
	Number of distinct sites
	Number of IP addresses
	Number of 302 redirections
	Number of script redirections
	Number of iframes
	Number of metahttp tags
	Number of functions redirections
	Number of JavaScript redirections
Contextual	Date
	First hop type
	Last hop type
	Second to last hop type
	Second hop type
Redirections	302 redirections in a row
	Script redirections in a row
	Iframe redirections in a row
	Metahttp redirections in a row
	Function redirections in a row
	JavaScript redirections in a row
	302 to iframe
	iframe to 302
	302 to metahttp
	Metahttp to 302
	302 to JavaScript
	JavaScript to 302
	iframe to metahttp
	Metahttp to iframe
	iframe to JavaScript
	JavaScript to iframe
	Metahttp to JavaScript
	JavaScript to metahttp

5.4 Results

In this section, we present the results of our classification of exploit kit families from our dataset, and explore the impact our features have on the metrics of our overall model, as well as the results per exploit kit family.

5.4.1 Dataset Construction

The redirection chains were extracted from our dataset using our redirection builder tool. We identified exploit kits using the method mentioned in section 5.3.2, and labeled 971 exploit kits from our dataset, which are then used in our classifier. We removed exploit kit families which had under 20 identified instances, due to their low number, and tested our classifier on the following remaining exploit kits found in our dataset: *Angler*, *Infinity*, *Nuclear*, *Sweet Orange*, *Magnitude*, *Styx*, *Blackhole*, *Phoenix*, *Bleeding Life*, *Dotkachef* and *Critx*. The names are those used in the literature and by security software vendors, and as such are the labels we extracted using our regular expressions and Snort. These families are among the most popular exploit kits from the time of our experiments [38, 149]. Our exploit kit family samples vary in number, ranging from 33 instances at the lowest for Bleeding Life to 162 at the highest for Magnitude. This imbalance in the class will impact the classification and skew its prediction towards the classes with more instances. Thus, to prevent this, we apply SMOTE, a technique which oversamples the minority classes by creating synthetic instances [125].

5.4.2 Classification of Exploit Kit Families With Redirection Chains

We first tested our classifier without the date feature, to better assess how the redirection features alone can correctly predict the overall exploit kit family. Our intuition is that possible changes in malicious campaigns and actors are reflected through the date, i.e., an exploit kit is first associated with a certain malicious campaign with a distinct redirection chain pattern, and then used for an entirely new campaign which will have a new redirection chain pattern. Table 5.2 shows our results for both the c4.5 decision tree and random forests algorithms.

As can be observed, both our classifiers show low accuracy, precision and recall, although the random forests classifier performs slightly better on all metrics. These overall results seem to suggest at first glance that an exploit kit can not be correctly identified using solely its redirection chain. It is worth noting that our data is distributed in time over 25 months, which could result in the malicious infrastructure changing through time in our data.

Table 5.2 Results of our decision tree classifier and random forests classifier

	TP Rate / Recall	FP Rate	Precision	F-Measure	ROC Area
Decision Tree	0.585	0.053	0.577	0.574	0.842
Random Forests	0.622	0.049	0.617	0.613	0.897

When analyzing the resulting decision tree from this classifier, we can observe that the tree branches first on the number of 302 redirections in a row, followed by the number of distinct sites and then on the redirection type of the last hop of the redirection chain. A subset of the decision tree is presented in Figure 5.3b. Although our classification does not model the redirections with high accuracy, we can assume that these elements of the redirection chain might provide some help in identifying exploit kit families.

Furthermore, we present the confusion matrix to inspect which exploit kits are better classified, and which ones are mainly assigned to other families. Table 5.3 represents our confusion matrix for the decision tree classifier. The classifier's predictions for each exploit kit family can be observed in the matrix. As can be seen, Infinity, Magnitude, Bleeding life are in majority correctly classified. The classifier's accuracy for these exploit kits are respectively 0.81, 0,86 and 0.92, with comparable precision and recall, although slightly lower. Angler, Nuclear, Blackhole, Phoenix, Dotkachef and Critx are in majority misclassified with other exploit kits, where Nuclear shows the lowest results. When inspecting Nuclear's metrics, we observe an accuracy of 0.302, a precision of 0.382 and recall of 0.302.

Table 5.3 Confusion matrix for the decision tree classifier without the date

a	b	c	d	e	f	g	h	i	j	k	
21	8	2	6	2	7	1	3	1	1	3	Angler(a)
3	105	1	3	4	4	0	3	0	1	6	Infinity(b)
0	2	13	10	7	6	3	0	0	1	1	Nuclear(c)
2	2	5	82	8	18	0	8	1	0	6	Sweet Orange(d)
0	6	1	6	139	6	0	1	0	0	3	Magnitude(e)
5	8	4	16	2	94	1	4	1	1	8	Styx(f)
1	5	3	1	3	4	21	9	0	1	10	Blackhole(g)
2	3	2	14	5	4	3	26	0	2	4	Phoenix(h)
1	1	0	1	1	1	0	0	61	0	0	Bleeding Life(i)
0	3	1	0	0	3	1	14	0	11	4	Dotkachef(j)
3	9	1	19	7	12	5	4	0	1	51	Critx(k)

We further test our data in our classifier, now with the added date feature. This feature is coded as a Unix timestamp, i.e., it consists in the numeric value of seconds that have passed since the 1st of January 1970. Table 5.4 shows the average results for both the decision tree and random forests algorithms. As can be seen, both classifiers obtained better results than our previous classification without the date feature. The accuracy is considerably higher, with a low false positive ratio. The true positive rate, precision and recall are all close to 78%. Contrary to our previous analysis, the classification with the date feature obtains better predictions using the C4.5 decision tree than the random forests algorithm.

Upon analyzing the resulting decision tree, we observed that the most important feature, i.e., the first feature used to branch the tree was the date. A subset of the resulting decision tree with the date as a feature can be observed in Figure 5.3a, and compared to the subset of the decision tree built without the date as a feature in Figure 5.3b. Our decision tree with no date shows that redirection features were mainly used to classify exploit kits, where the top feature was the number of 302 redirections in a row. The number of distinct sites was also a prominent feature, which indicates that some exploit kit families tend to reuse the same domains for different parts of their infrastructure.

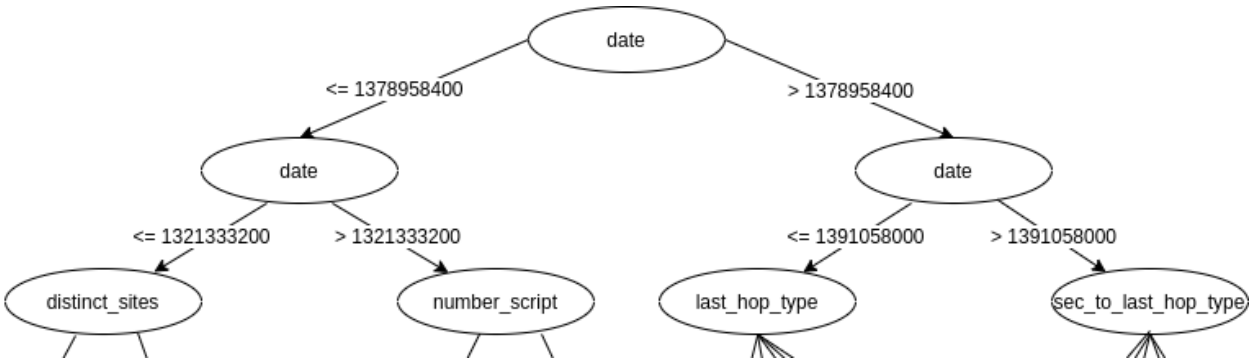
5.4.3 Results per Exploit Kit Family

Our results also vary per exploit kit, where Infinity, Sweet Orange, Magnitude and Phoenix had particularly high results, with close to 90% accuracy or higher, and Angler, Nuclear, Blackhole and Critx had low accuracy, hovering around 50%. Table 5.5 shows the confusion matrix of the decision tree algorithm with the added date feature.

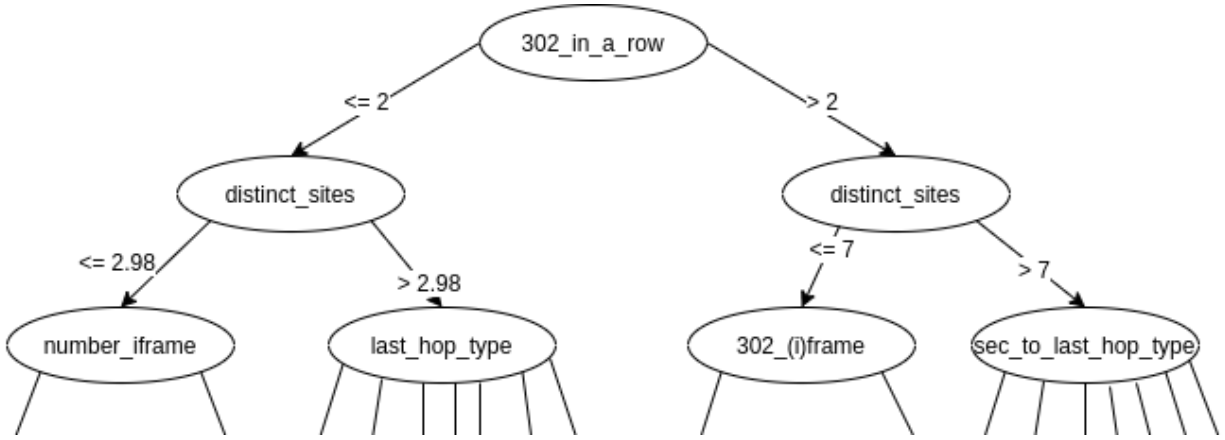
When comparing these results to our previous ones without the date, the exploit kits predictions with the highest improvement are Sweet Orange, Phoenix and Dotkachef, respectively showing close to 32%, 54% and 48% increase in their true positive rate. Overall, almost all exploit kits were better classified, with the highest results consisting in Sweet Orange's classification obtaining a true positive rate of 95% and a precision of 89%, and Magnitude's

Table 5.4 Results of our decision tree classifier and random forests classifier with the date feature

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Decision Tree	0.778	0.025	0.772	0.778	0.774	0.910
Random Forest	0.747	0.032	0.744	0.747	0.742	0.950



(a) With the date as a feature.



(b) Without the date as a feature.

Figure 5.3 A subset of the decision tree model built from the extracted redirection chains and their features

Table 5.5 Confusion matrix for the decision tree classifier

a	b	c	d	e	f	g	h	i	j	k	
29	2	0	2	1	5	4	1	3	4	4	Angler(a)
2	113	3	1	0	5	4	0	1	1	0	Infinity(b)
0	1	19	0	8	6	0	0	2	3	4	Nuclear(c)
1	2	0	125	0	1	0	1	0	0	2	Sweet Orange(d)
1	0	7	0	150	3	1	0	0	0	0	Magnitude(e)
4	5	3	0	2	111	0	4	2	3	10	Styx(f)
2	9	0	0	0	5	34	0	2	1	5	Blackhole(g)
1	0	0	0	0	4	0	56	1	0	3	Phoenix(h)
1	0	0	0	0	2	4	3	50	0	6	Bleeding Life(i)
3	1	0	1	1	2	1	0	0	28	0	Dotkachef(j)
3	0	2	12	2	10	7	8	2	0	66	Critx(k)

classification obtaining a true positive rate of 93% and a precision of 92%. One exception to these improvements is the prediction for the Bleeding Life exploit kit, where the true positive rate decreased from 89% to 76% and the precision decreased from 86% to 79%. This exploit kit does not classify well with the date, and is better identified using its redirection chain features only. Table 5.6 details our results for each exploit kit for both our classification with the date feature and without the date feature.

Two hypotheses can be formed to explain this improved accuracy. A first hypothesis arises from the context of these infections, i.e., the campaigns in which these infections occur. Frequently, an infection with the use of an exploit kit will be part of a malicious campaign spanning a certain time period. During this campaign, one might expect the redirections to an exploit kit to be more similar than redirections from another future campaign. Thus, this would indicate that redirection chains are associated with campaigns of malicious activity using an exploit kit, rather than an exploit kit in itself. However, in the context of EaaS, this would indicate that a single actor employs the services of a given exploit kit at a time, which goes against the idea of widely offering EaaS.

A second hypothesis consists in our dataset having a varying rate of infections through time for one or many of our exploit kits. Indeed, if our dataset consists of clusters of infections by a particular exploit kit at a time, our classification will model the dataset's bias, i.e., that the infections by families are not consistent through time and can be contained entirely in a short period of time.

While the first hypothesis cannot be tested using our dataset, the second one can be tested

Table 5.6 Results of our decision tree classifier per exploit kit, with and without the date feature

	Without the date			With the date		
Exploit Kit	TP Rate	Precision	F-Measure	TP Rate	Precision	F-Measure
Angler	0.418	0.489	0.451	0.527	0.617	0.569
Infinity	0.731	0.674	0.701	0.869	0.850	0.859
Nuclear	0.279	0.343	0.308	0.442	0.559	0.494
Sweet Orange	0.629	0.456	0.529	0.947	0.887	0.916
Magnitude	0.852	0.750	0.798	0.926	0.915	0.920
Styx	0.569	0.577	0.573	0.771	0.721	0.745
Blackhole	0.276	0.533	0.364	0.586	0.618	0.602
Phoenix	0.323	0.356	0.339	0.862	0.767	0.812
Bleeding Life	0.894	0.855	0.874	0.758	0.794	0.775
Dotkachef	0.270	0.400	0.323	0.757	0.700	0.727
Critx	0.429	0.533	0.475	0.589	0.660	0.623

by inspecting the distribution of infections through time in our dataset. Figure 5.4 illustrates the number of infections where each exploit kit family was seen every month of our dataset.

As can be observed, the rate of infection for many of our exploit kits varies greatly from month to month, with Magnitude and Phoenix's infections occurring during only 8 of the 35 months. When calculating the standard deviation of the time of infection for each family, we can order the exploit kits according to how spread they are over time. The exploit kits with the smallest standard deviation are Magnitude, Phoenix and BlackHole, with respectively 46 days, 68 days and 98 days as their standard deviation. As seen in Figure 5.4, these three exploit kits have peaks for their number of infections around certain times, notably the beginning of 2014 for Magnitude, the summer of 2012 for Phoenix and the months close to December 2012 for BlackHole. Following these are Sweet Orange, Dotkachef and Infinity with a slightly higher standard deviation of 129 days, 133 days and 137 days respectively. However, these exploit kits still show a distinct period of high count of infections, where a peak can be observed for Sweet Orange in the fall of 2011, a peak can be observed for Dotkachef around November 2013, and a peak appears at the start of the year 2013 for Infinity. Finally, the exploit kits with the highest spread of infections over time are Bleeding life, Critx, Angler, Nuclear and Styx, with respectively 178 days, 219 days, 229 days, 254 days and 291 days as the biggest standard deviation. We can notice on Figure 5.4 that these exploit kits are spread more evenly over time.

Overall, our exploit kits do not share the same results in their classification by web redirection

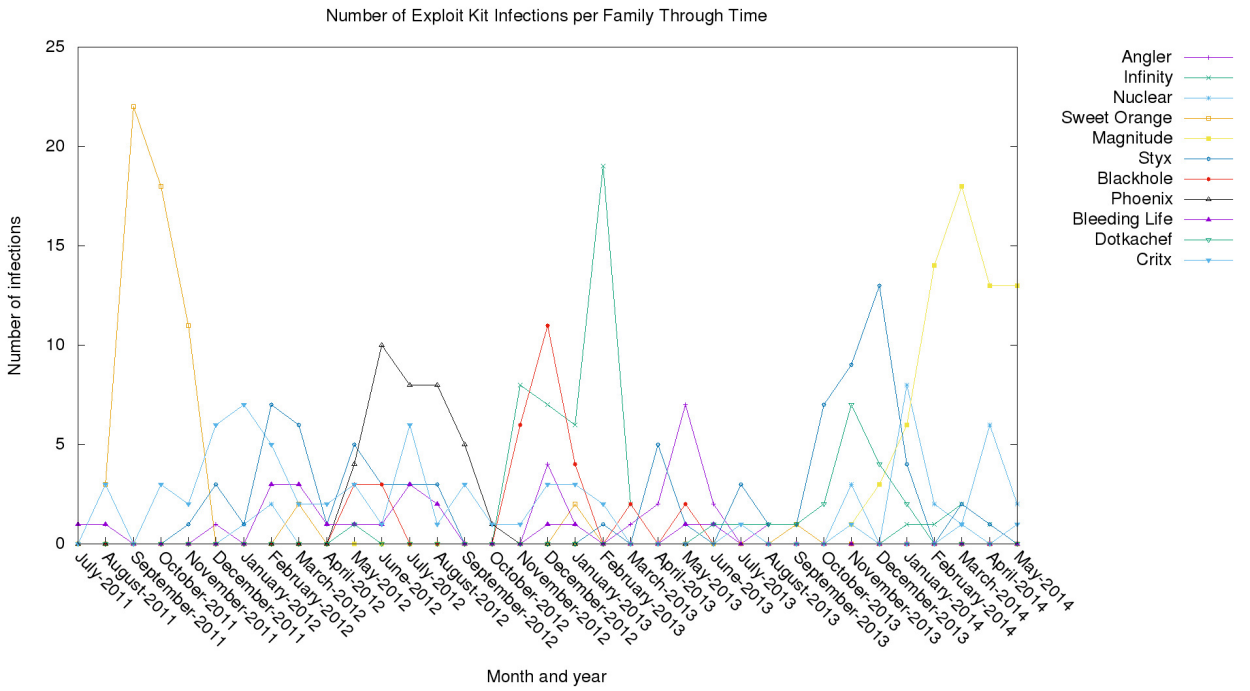


Figure 5.4 The distribution of infections by each exploit kit family for each month of our dataset time period

chains. Firstly, some exploit kit families, while not being very spread out over time, are still relatively well classified both without and with the date feature. These consist in Infinity and Magnitude. Another interesting exploit kit is Bleeding Life, which is well spread over time but is correctly classified in both our analyses. Thus, our approach seems successful in classifying these three exploit kits according only to their redirection chains. Some exploit kits, Sweet Orange and Styx, are in part correctly classified using only their redirection chains, but show an improved classification when combined with their date. Finally, Phoenix and Dotkachef were only correctly classified when the date feature was used. Thus, these two exploit kits are identified by their spread in the dataset, and cannot be considered to be identifiable by their redirection chains according to our analysis. Our initial hypothesis is thus only validated for a subset of our dataset, i.e., for 5 of the 11 exploit kit families analysed: Infinity, Sweet Orange, Magnitude, Styx, Bleeding Life.

5.4.4 Classification of Selected Exploit Kit Families

Following our previous results, we can observe that Angler, Nuclear, Blackhole and Critx are the exploit kits with the worst accuracy in our overall classification, both with the date feature and without. Furthermore, Phoenix and Dotkachef were classified mainly from their

time of infection. Our results seem to indicate that only a subset of exploit kit families can be classified with their web redirection chains. Thus, we explore how well our model would perform if a preselection of exploit kit families was made. In a live implementation of our model, an initial analysis phase would need to be performed on exploit kit families, in order to identify the families that can be identified using their redirection chains.

We explore this scenario, by repeating our classification on a subset of our data, containing the following exploit kit families: Infinity, Sweet Orange, Magnitude, Styx, Bleeding Life. The results are presented in Table 5.7. As can be observed, we obtain a high true positive rate of 89% and low false positive rate of 2%, while the precision is at 89%. This is expected considering this classification is performed on exploit kits that can be correctly identified from their redirection chain, as established by our previous analysis.

Table 5.7 Results of our decision tree classifier and random forests classifier with the filtered dataset

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Decision Tree	0.915	0.022	0.916	0.915	0.915	0.970
Random Forest	0.921	0.021	0.922	0.921	0.921	0.990

Finally, we present an overall comparison of our three analyses in Table 5.8. We chose to compare the decision tree algorithm results for all three, since we obtained the best results on average in our previous results. As expected, our final classification from our filtered dataset performs better.

Table 5.8 Results of our decision tree classifier for every analysis

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Filtered dataset	0.915	0.022	0.916	0.915	0.915	0.970
Date feature	0.778	0.025	0.772	0.778	0.774	0.910
No date feature	0.585	0.053	0.577	0.585	0.574	0.842

5.5 Discussion

Our redirection chain builder tool successfully reconstructs redirection chains from network traces. After applying it to our dataset, we used a classifier to identify the exploit kit with a set of features extracted from the redirection chain.

As presented in section 5.4, we can observe that five exploit kit families were correctly predicted using solely features from the redirection chains. However, a number of them also did not classify well. While this might be due to our small dataset, our further analysis including the date at which each infection occurred proved to be more successful at classifying all exploit kits. While analyzing the decision tree constructed by our C4.5 algorithm, we were able to observe that the classification with the date feature provided the date as a top branching feature for every path of the tree, i.e., for every instance classification. We further inspected the result per exploit kit, and discovered great discrepancies in the classification metrics. Our dataset included 11 exploit kit families, and our results differ from one family to the other.

We established that Phoenix and Dotkachef instances were all dated from a relatively short period of time in our dataset, hinting at the fact that our classifier used this feature to classify them. These exploit kits are thus not classified according to their redirection chains. On the other hand, Infinity, Magnitude and Bleeding Life have been classified with high accuracy using only redirection chain features, as well as Sweet Orange and Styx, to a lesser degree. Our classifier performed badly for the remaining exploit kits, Angler, Nuclear, Blackhole and Critx.

Thus, our classification demonstrated that only some exploit kit families infections could be identified by their redirection chains, while some tend to change through time, possibly because of changing malicious campaigns. However, in the context of EaaS, this result seems counterintuitive. Two possible hypotheses for this phenomenon are: 1) Infinity, Magnitude, Bleeding Life, Sweet Orange and Styx do not generally take part in the EaaS ecosystem and are tied to specific actors or campaigns, or 2) cybercriminals buying the services of these exploit kits use the same redirection infrastructure, i.e., the exploit kit operator possibly controls the redirection infrastructure for its clients.

These results indicate that using redirection chains to identify the family of exploit kits could prove useful for detecting the following exploit kits in a live implementation: Infinity, Magnitude, Bleeding Life, Sweet Orange and Styx. Although our dataset ranges from 2011 to 2014, and some of these exploit kits now show less activity, Magnitude still remains an important exploit kit in 2020 [181]. Additional exploit kit families could also be identified by their redirection chains with our framework, by using network captures containing activity from the relevant exploit kits and recreating the training phase of our model. Additionally, this framework could be combined to other detection tools to provide better protection against malicious software.

5.5.1 Limitations

There are some limitations to our work which will be detailed in this section. First, our dataset was obtained from Threatglass, which is maintained by Barracuda, a cybersecurity firm. Their dataset has been obtained from infections occurring in their virtual machines while visiting compromised websites from the most popular websites according to Alexa, and from infections captured through their clients. The types of infections and families of malware obtained through these two sources might differ from one another, and introduce a bias in our dataset. Thus, it might make it more difficult for our classification to model the data.

Thirdly, while our dataset consisted of 8863 instances, only 971 instances could be labeled with their exploit kit family name, where our smallest used family contained only 33 instances. Thus, the size of our dataset limits our classifier's ability to accurately build a model of the relation between an exploit kit and its web redirection chain.

Finally, some of our redirection chains were incomplete due to the limits of our redirection tree construction. Since our approach employs static analysis of the web pages found in network captures, it is limited by highly obfuscated code hiding a redirection. Thus, our approach might have better success with a dynamic approach.

5.5.2 Future Work

Our research aimed at establishing if the redirection chain leading to an exploit kit can be effectively used to classify it. There are multiple avenues of research rising from our conclusions, which can be established as future work.

First, whereas our work aimed at identifying an exploit kit family, a model could be built using classification to predict a malicious campaign. Similar to our approach, this could be achieved by obtaining or building a dataset of malicious redirection chains, where each instance would instead be identified with its malicious campaign, or even malicious actor. While finding an adequate way to label an infection and its redirection chain with a certain malicious campaign is a complex process in itself, this future work could prove more successful for the exploit kits in our experiments that resulted in a poor classification.

Second, while our work aimed at identifying the exploit kit family associated with a redirection chain, the same approach could be used with the inclusion of benign traffic to predict if a redirection chain is benign or malicious. This would require collecting a large number of legitimate redirection chains, and applying our methodology to these chains as well. One of the challenges in building such an approach is that the dataset of benign redirection chains

must be representative of all types of redirection chains found in the wild, in order to build a model that performs well in a real scenario. For example, most benign traffic do not include obfuscated JavaScript, whereas malicious traffic frequently does. If the benign dataset does not include legitimate obfuscated code, the model might incorrectly assume all redirection chains including obfuscated code are malicious.

Secondly, applying our methodology to a bigger dataset might prove more successful, and permit a number of additional hypotheses to be tested. As mentioned in the previous section, our dataset has multiple limitations that might have affected our results. To avoid testing different types of attacks and possibly introduce a bias, the same analysis could be made with a dataset built from crawling the most popular websites around the world, and identifying any *waterholing attacks* occurring while visiting the website. A waterholing attack consists in a legitimate website that has been compromised by attackers and is now serving its users malware, be it with the addition of a script on the homepage, serving malicious ads or exploiting plugins. These attacks target popular websites, and are generally resolved quickly, although many users have been exposed in the attack's small time frame. These attacks could be searched for by building a framework consisting of many virtual machines, where each VM visits multiple times each of the most popular websites on a daily basis. Any infection resulting from this visit could be kept to construct the dataset. With this setup running over a period of multiple months, it would be possible to build a large dataset of waterholing attacks, and test our methodology on it to obtain more accurate results. With this type of dataset, we could obtain more samples of each exploit kit, possibly over a larger period of time, and test whether exploit kits change from one malicious campaign to another, and how it affects the web redirection chain and its classification.

Thirdly, our redirection tree construction method could be modified to use dynamic analysis of web pages in order to more accurately establish web redirections. Dynamically running the web pages or debugging the JavaScript operating the redirection could provide a means to better identify some redirections, with more certainty, and most importantly bypass obfuscated code. Multiple solutions provide the means to dynamically analyze JavaScript code, with most operating some type of headless web browser, i.e., a browser without the graphical aspect. One such tool consists in JSDetox [182], which provides a framework to analyze malicious JavaScript code, both statically and dynamically.

Finally, another possible approach to obtain redirection chains from web requests is to employ the use of a web browser plugin to capture them. Most browsers provide the possibility to develop plugins for their software. A plugin could be made that simply saves the redirection chains when a user visits a page. This method requires that some type of user visit web

pages while having the plugin installed. This could either be through a security vendor and its customers, where the customers accept the terms of the plugin, or through a series of machines or virtual machines visiting websites through a browser with the plugin installed. With this plugin, there would be no need to reconstruct HTTP requests from network traces, but the data collection and plugin development would require significant effort. This method could be combined to our previous future work explaining the construction of a waterholing attack dataset.

5.6 Conclusion

In summary, we presented an approach which first constructed a web redirection tree from network traces of malicious infections, and then extracted the malicious redirection leading to the exploit kit. Our initial hypothesis consisted in that exploit kit families can be clustered according to the web redirection chains leading to them. Following our extraction of 971 labeled malicious redirection chains from network traces, we applied supervised machine learning to the dataset and tested classification algorithms to predict the labels of our exploit kits. We applied the C4.5 decision tree and random forests algorithms, with statistical, contextual and redirection features extracted from our redirection chains. Our results varied per exploit kit, and showed that some exploit kits can be successfully classified using features solely from their redirection chain, where other exploit kits only perform well in our classification with the date of the infection. Finally, other exploit kits did not classify well at all, which might be the result of the Exploit-as-a-Service business model. Thus, our hypothesis has been verified for a subset of our dataset.

5.7 Acknowledgments

This material is supported by ESET and the Fonds de Recherche du Québec - Nature et technologies (FRQNT).

CHAPTER 6 ARTICLE 1: SHEDDING LIGHT ON THE TARGETED VICTIM PROFILES OF MALICIOUS DOWNLOADERS

Submitted and under review in the *Journal of Information Security and Applications*

Authors: François Labrèche, Enrico Mariconti, José Fernandez, Gianluca Stringhini

6.1 Abstract

Malware affects millions of users worldwide, impacting the daily lives of many people as well as businesses. Malware infections are increasing in complexity and unfold over a number of stages. A malicious downloader often acts as the starting point as it fingerprints the victim's machine and downloads one or more additional malware payloads. Although previous research was conducted on these malicious downloaders and their Pay-Per-Install networks, limited work has investigated how the profile of the victim machine, e.g., its characteristics and software configuration, affect the targeting choice of cybercriminals.

In this paper, we operate a large-scale investigation of the relation between the machine profile and the payload downloaded by droppers, through 151,189 executions of malware downloaders over a period of 12 months. We build a fully automated framework which uses Virtual Machines (VMs) in sandboxes to build custom user and machine profiles to test our malicious samples. We then use changepoint analysis to model the behavior of different downloader families, and perform analyses of variance (ANOVA) on the ratio of infections per profile. With this, we identify which machine profile is targeted by cybercriminals at different points in time.

Our results show that a number of downloaders present different behaviors depending on a number of features of a machine. Notably, a higher number of infections for specific malware families were observed when using different browser profiles, keyboard layouts and operating systems, while one keyboard layout obtained less infections of a specific malware family.

Our findings bring light to the importance of the features of a machine running malicious downloader software, particularly for malware research.

6.2 Introduction

Malicious software, i.e., malware, infects tens of thousands of machines every day around the globe [4].

Once a user has visited a malicious link and has been redirected, an exploit kit or the website's code will infect them with a malicious software. This malicious software can take many forms, one of them being a downloader software. In this scenario, its sole purpose will be to download other malicious software. For example, a malicious actor will infect several machines with their downloader, and then sell the access to one or more other criminals, who will install spam bots, information stealers, ransomware, etc., on the machine. This process is part of the Pay-Per-Install (PPI) distribution model, where the downloader's author will sell access to the infected machine to another malicious actor through the PPI service. Figure 6.1 illustrates the PPI distribution model. These transactions are made privately, and information regarding these transactions, the actors involved, and the platforms on which they occur is difficult to obtain. Particularly, little is known on how cybercriminals select the malicious file(s) to send to the victim through the downloader, i.e. how much, if any, fingerprinting is done on the victim and their machine, to decide which malware to infect them with.

Researchers have studied malware in multiple previous research, while downloader software used for malware is often overlooked in research. Some works have studied downloader software, mainly through a study of PPI models [7, 183, 184], without a large focus on what impact the various features of the victim machine has on the PPI network customer's targeting choice. While the effect of the country and the operating system of the machine on downloaders has been studied in the past, no study has established the potential impact the browser profile, the keyboard layout and the display language has on the choice of victim by a PPI customer. In this paper, we will use these features as part of a machine profile and test whether the machine is targeted by one or more cybercriminals, by executing various families of downloaders daily and identifying downloaded malicious payloads, over a one-year period.

With our approach, we aim at reverse engineering the targeting choices of criminals. However, with respect to classic reverse engineering techniques that analyze the code, we deduce what influenced the decision-making process of cybercriminals by simultaneously running multiple VMs with different configurations (profiles). This approach is more time effective and will not be influenced by the complexity and obfuscation of the downloader's code.

We will achieve this by executing a downloader family's samples over a large set of Virtual Machines (VMs) with different profiles, in order to establish which VM gets infected with which family of malware. Thus, our hypothesis is the following : different malicious software are sent to victims through downloaders according to their profile.

While a downloader and malware can be combined in a single malware, we will, in this work,

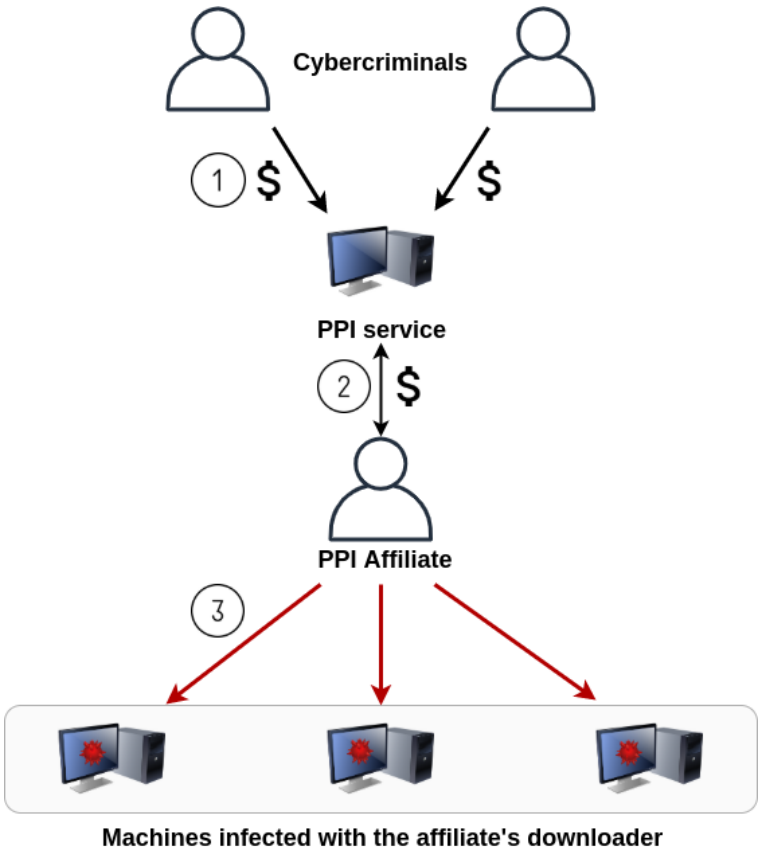


Figure 6.1 The Pay-Per-Install distribution model. 1) Cybercriminals pay a PPI service to install their malware on machines. 2) A PPI affiliate obtains malware to install on victim machines through the PPI service. 3) The PPI affiliate installs malware payloads on victims infected with its malicious downloader

focus on analyzing single downloaders. In order to verify our hypothesis, our research will aim at:

1. Establishing an automated sandboxed testing environment for downloaders.
2. Establishing the correlation between the profile of the victim machine and the downloaded malicious payload(s).

Although previous research has analyzed user profiles more at risk through clinical trials [185], the correlation between the user and the payload requested by a downloader has not been studied in detail. In summary, this paper makes the following contributions :

1. We establish a fully automated sandboxed testing environment, capable of running malicious downloader executables on virtual machines configured to match specific machine and user profiles.
2. We demonstrate what cybercriminals target through PPI networks by observing how a number of downloaders behave differently depending on the profile of the VM running the executable. Our results highlight an existing link between the malicious actors' downloaders and the browser session, the keyboard layout and the display language of machines.

6.3 Related Work

Malicious downloaders, or droppers, are often observed when analyzing malware. Although the downloader is often paired with the malware, researchers focus more on the final malicious payload. We present, in the following section, recent research that analyze downloaders and their PPI networks.

6.3.1 Downloader Families and Samples Analysis

Some researchers have analyzed specific downloaders and large campaigns in detail.

Rossow *et al.* [186] observed a large number of malware downloaders from 23 families between February 2010 and February 2012 and identified their properties and behavior. They identified the means of communication they use to reach their command and control (C&C) server or other infected machines, by reassembling and parsing numerous carrier protocols. The vast majority of families used a centralized C&C architecture, while a small amount used a peer-to-peer (P2P) architecture. Communications with the C&C are often encrypted. They

observed these samples over a period of time and identified how frequently the domains and infrastructure change, and how long the downloader remains active. Their analysis showed that 48% of downloaders actively operated for more than a year. They then inspected how downloaders request their malicious software to install on the machine, and recreated it to farm samples. In the case of downloaders communicating through an encrypted channel, the system was monitored and malicious new processes were gathered. In these samples, they observed the number of executables served, and established that polymorphism was used by 8 of the 9 families of malware gathered.

Kwon *et al.* [187] approached the analysis of downloaders by creating influence graphs of downloaders, and identifying differences between malicious and benign graphs. Their dataset was obtained from the WINE analytics platform, which consists of telemetry reports gathered from users of the security firm Symantec over 19 months, from January 2013 to July 2014. They extracted downloaded files from downloader samples that communicate in plaintext through HTTP, both malicious and benign. From this, they created a downloader graph and an influence graph for each downloader d, representing the downloader subgraph starting with the root d. They extracted a total of 19 million influence graphs from their dataset. An analysis of these graphs revealed that 22.4% (15,115) of downloaders have a valid digital signature, influence graphs with a large diameter are mostly malicious, influence graphs with slow growth rates are mostly malicious, and malware tend to download fewer files per domain. They then used this graph representation of downloaders to extract several features. Using these, they employed supervised learning to create a detection model and to find the most important features which identify malicious downloaders. They used the random forest algorithm [188] for their classifier, with 10-fold cross-validation, and obtained good accuracy.

6.3.2 Pay-per-Install

Downloader software often has a presence in PPI networks. This software is used by PPI providers to install their clients' malware onto compromised machines.

Caballero *et al.* [7] infiltrated four PPI networks and ran their downloaders in a closed environment, from August 2010 to February 2011. They harvested over a million client executables using servers across 15 countries. They observed over time the different malware samples downloaded by the PPI networks and clustered them into their respective families and types, using their network activity. They also observed the repacking rate of these malware. The top 10 families show that they are repacked, i.e., their code is re-obfuscated, every 6.5 day on average. They observed samples running anti-VM techniques; some samples even removed or added them while in a campaign, without any apparent reason. Another

noteworthy observation was that a number of executables extracted from the downloaders are in fact other PPI downloaders, hinting at the fact that there might be arbitrage in the PPI market, i.e., that PPI services buy and resell their services between themselves to make a profit on varying prices. The downloaded samples also differed depending on the location of the machine, and an analysis of PPI forums showed that the price varies according to the location of the purchase of installs.

Kotzias *et al.* [183] analyzed potentially unwanted programs (PUP), which consist in software that is approved by the user, knowingly or not, but exhibit a behavior detrimental to him. This software operates in a gray area, and they are flagged as malicious by certain antivirus and security software. In this work, they used the binary downloads section of the WINE dataset for their analysis. Their first step was to identify top PUP publishers by their signed software name. They then clustered publishers by running a name similarity algorithm, among other techniques. They looked at the prevalence of PUP, and 54% of the WINE dataset hosts had some form of PUP installed. They also listed the top publishers from their dataset. Compared to legitimate software, the top PUP enterprise ranked 15th, which shows how widespread these PUPs are. They then established a relation graph between installers to see which installs which, and then identified PPI services by their high count of outgoing relations and ingoing relations, which suggests they sell installs. They also identified advertisers, and looked at their type of add-ons and their monetization schemes. They also found that the majority of PUP are installed by other PUP. In total, they observed 71 PUP publishers that clustered to malware. However, this number is small in contrast to their total number of publishers.

Thomas *et al.* [184] explored the ecosystem of PPI services to establish what adware they distribute. They explored the 4 largest PPIs from their network : Amonetize, InstallMonetizer, Open-Candy, and Outbrowse. They established an infrastructure to collect software distributed by these PPIs on a regular basis. They installed downloaders from these PPIs, and observed that these downloaders sent to their server the client's OS and service pack version, the Web browsers and their version, and potentially unique identifiers including a MAC address. The server then sent 5 to 50 potential offers to the downloader. In this work, they then implemented *milkers* that recreate the request offers of PPIs, and used them on the C&C servers every hour from January 2015 to January 2016. They collected 446,852 offers through this, and only ran their experiments from US IP addresses. They monitored the price of these offers through forums and various websites advertising these online. They clustered the results into families based on a multitude of characteristics. In the end, advertisement injectors, browser settings hijackers, and system *cleanup* utilities dominated the software installed through these PPIs. Finally, they analyzed the length of campaigns by looking at

offers from the PPI networks. Through the forums advertising these, they found that the prices differ per country. They then assessed how widespread adware was and quantified the overall weekly downloads with the use of Google Safe Browsing and Google's Chrome cleanup tool. They then observed the distribution sites for PPIs and looked at the website of origin to identify their lifetime, which varied from 0.75 hours to the entire monitoring window of 220 days.

Finally, Kwon *et al.* [41] implemented Beewolf, a software built to identify malicious campaigns by using unsupervised learning to identify the *locksteps* of malicious downloaders retrieving their payloads. This work analyzes the global behavior of downloaders in order to improve the detection of malicious campaigns, and not necessarily single malicious files. It identifies overlaps between malware delivery campaigns and PUP delivery campaigns.

6.3.3 Summary

While many projects have focused on improving the detection of malicious software, exploit kits and malicious websites, downloaders have seen less focus from the scientific community. Some of these works [189, 190] have focused on analyzing malware downloads, which includes the analysis of downloaders fetching malicious payloads, although they do not focus on malicious downloaders behavior.

Researchers have analyzed large scale downloader campaigns through adware, i.e., PUP, and, although specific downloaders and PPI networks have been studied in detail, no research has shown the impact the user and its machine configuration have on the downloaded malware by the downloader. More notably, no testing framework has implemented the use of VMs that modify their configuration while testing downloader samples.

After having identified these gaps in research, we decided to systematize the study of how the user and its machine configuration affect the downloader decisions, through an automated environment based on VMs using dynamic analysis, which will be presented in the methodology section.

6.4 Methodology

Our approach consists in building an automated sandboxed environment, containing multiple VMs, where malicious downloader samples of various families are run with multiple machine profiles.

Definition 22 *A* machine profile *consists of a machine M possessing n associated features*

$M_f = \{M_1, M_2, .., M_n\}$.

A sample is automatically tested on multiple machine profiles in order to gather what executable(s) is downloaded depending on which profile was used. An analysis of the time series of infections for every feature and payload family is then performed, in order to assess what profile is targeted by the actors behind malicious downloaders.

Definition 23 *A* feature *F of a machine M can be summarized as a modifiable piece of software or hardware with a subset of l values, such that $F = \{F_1, F_2, .., F_l\}$.*

6.4.1 Testing Environment

Our first research objective is to build an automatic testing environment to run downloaders using various machine profiles.

The design of our platform is inspired from previous research [184], and uses the Cuckoo Sandbox [191] framework to execute samples. This testing environment consists of a cluster of machines automatically launching VMs in a sandbox, according to a 1) Scheduler, while using a predefined user profile for each machine through a 2) Profiler. Each VM is provided with a downloader sample to run, and data is retrieved from the execution and then compiled with the 3) Analyzer, where the downloaded files' malware families are also identified. Our framework is depicted in Figure 6.2. Here we detail the different sections :

Scheduler : This module receives the encrypted downloader sample through a secure stream, asks the profiler to build a VM, and then decrypts the sample and launches it in the VM.

Profiler : This module is responsible for following the experimental design, in terms of features to test. In this module, a profile will be built according to the current features in the queue, and sent back to the scheduler. It will also specify the exit node, i.e., the country, to use through a virtual private network (VPN).

Analyzer : This module is responsible for gathering data on the execution of the downloader samples. It uses the Cuckoo sandbox to capture network traces, process information and identify any downloaded files. This module tests any downloaded file by the downloader on VirusTotal [192], a platform that tests files against multiple antivirus software. The module then establishes the file's family according to the naming of security vendors.

6.4.2 Automation Setup

Our physical testing environment consists of a cluster of 10 servers, each one containing an Intel Xeon E5405 2.00 GHz processor and 8 Gb memory. One server is used as the experiment

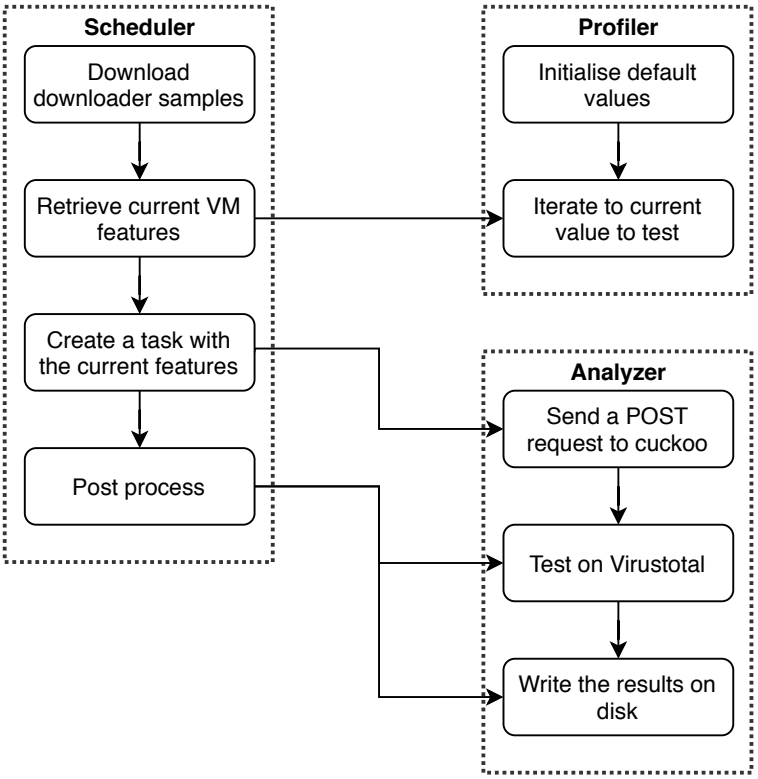


Figure 6.2 Our framework depicting our downloader testing environment

manager, where Cuckoo Distributed and the VPNs are installed. This machine dispatches the tasks of our scheduler to the other servers' Cuckoo frameworks through Cuckoo Distributed. One network interface is used to access the servers, while a separate network interface is used for each of the VPNs, in order to isolate the Internet access of our VMs. Every VM has Internet access only through its assigned VPN by the scheduler. OpenVPN[1] is used to install and manage the VPN connections. Our setup is fully automated: xCAT[2] is used to automate the installation of Ubuntu LTS on our servers, and Ansible[3] scripts are employed to configure our setup, which consists of the following steps:

1. Install and configure libvirt with KVM for the virtualization
2. Copy images of Windows XP, Windows 7 and Windows 10 on each server
3. Create the VMs and their default snapshots
4. Setup 10 VPNs
5. Install and configure the Cuckoo and Cuckoo Distributed frameworks
6. Configure our virtualization software according to Cuckoo's requirements

Each server has three VMs, and our cluster totals 27 VMs which can be used simultaneously. Our experiment and setup was approved by the Information Technology Risks Committee of our university.

6.4.3 VM Hardening

An issue that can arise from testing malicious software in a sandbox, using virtual machines, is the detection of the environment by the malware. Some malware have been shown to use multiple techniques to identify when they are running in a sandbox, and subsequently terminate their malicious activity or delete themselves [193]. They employ sandbox and virtual machine detection to prevent researchers and others from analyzing their software.

While analyzing a subset of our samples through Cuckoo reports, we noted that VM detection techniques were used. Notably, malware from the Banload family included the identification of the VMware MAC address, as well as the execution of opcodes that do not raise an exception in VirtualPC. Since we are not using these platforms, these verification checks did not detect the virtualized environment, and the malware samples operated in our sandbox.

[1] https://openvpn.net/
[2] https://xcat.org/
[3] https://www.ansible.com/

Nonetheless, modifications have been made to the virtual machines to ensure more resilience to virtual environment and sandbox detection from malware. The Cuckoo sandbox environment provides a *disguise* module, which modifies various registry keys identifying the machine as a virtual environment. Since these registry key values are hard-coded in the disguise module, we modified them slightly to evade any malware also trying to identify the Cuckoo sandbox.

Our virtual machines have been tested using Paranoid Fish [194], a popular virtual machine and sandbox fingerprinting tool, and Al-Khaser [195], a tool detecting virtual machines, emulation, debuggers and sandboxes. For both of these tools, QEMU/KVM provided by default a less identifiable virtualized environment compared to other common virtualization software, notably VMware and VirtualBox. Detection methods appear to be indeed more prevalent for VMware [196], while QEMU/KVM leaves fewer traces in the operating system. For this reason, we opted for KVM as a virtualization software. When running Pafish, 50 out of 54 tests yielded success. Al-Khaser obtained similar results. The following four tests failed on our setup :

- The CPU timestamp counter identification with the RDTSC instruction, which returns different values under a virtualized environment
- Verifiying the hypervisor bit in the CPUID feature bits
- Querying the CPUID vendor ID for known virtual machine vendors
- Verifying if the number of processors is greater than 1

We were not able to modify our VMs to pass these tests, in part because our hardware is limited, which prevents us from assigning more than 1 processor to each VM. Nonetheless, our VM hardening is made in order to gather additional malicious payloads which we would not obtain with fresh VMs. In the end, this setup provided us with a large number of malware detonations.

6.4.4 Choice of Features

Next, we must establish what features identify our machine profile, and what may affect the downloader's behavior, i.e., what features might be targeted by cybercriminals. In this work, we use a *black box* approach, i.e., we do not reverse engineer downloader software to establish what it searches on the machine, but rather change various features of our environment and observe possible changes in behavior. We opted for this approach, considering there are

thousands of different samples of downloaders in our dataset, and they might be modified frequently. Reverse engineering them would not be possible given the time and effort it requires. The malicious hosts serving downloaders might also analyze the machine features server side, in which case reverse engineering the samples would at best only provide partial information. We also opt to change features of our machines instead of monitoring Windows API calls made by the downloader, in order to limit our interactions within the VM as much as possible. This is to prevent the detection of the research environment by the malicious software, and to avoid impeding its execution in any way.

The features of our machines might notably be used in PPI networks, in order to decide which payload to install on a machine. Previous research [184] has analyzed PPI networks and what information adware downloaders send to their control servers. In their analysis of 4 PPIs, they observed the following features being sent by the downloaders:

- The operating system and service pack version
- The Web browsers installed, along with their version
- The IP address
- Potentially unique identifiers, such as the MAC address

Although these PPIs are not necessarily similar to PPIs sending malware, using these features for our experiments will provide a basis to follow. Additionally, we also add features associated with the web history and the profile of the user. In the past, some malware has been shown to look at the keyboard or display language of a system when executed [197,198]. Thus, we opted to also include these features. A summary of our features is described in Table 6.1.

Table 6.1 Tested features of the victim machine

Category	**Feature**
User features	Country Browser History Web session (browser cookies)
Configuration features	IP address Windows OS version Display language Keyboard language

Other user features, such as the age, gender and income, while being relevant, are not always possible to identify on a machine, and thus, will not be considered in our experiments.

6.4.5 Downloader Experiment

Our second research objective is to establish the correlation between user profiles and the malicious payload. Our experiments employ our previously defined environment to run downloader samples.

The dataset, i.e., the downloader samples, is provided on a daily basis by our security vendor partner ESET, an antivirus software company. Each downloader sample is run, when received, on a set of VMs with profiles testing each of the various features. Any duplicate downloader received is discarded. After each set of tests of a downloader, our analyzer establishes what payload was downloaded on the victim machine. Our experimental design is as follows :

Two downloader families are tested, each of them for a period of 15 minutes at a time, so that any execution delay in the sample doesn't impact the test, and our profile set up has time to execute in the VM. Up to 10 samples of each family are tested per day, according to the availability in our data stream. We aim at testing the most current and widespread malicious downloaders, that are known to have dropped multiple types of malware.

Our initial list of families consisted of Waski, Zurgop, Pliskal, Wauchos, Nymaim, Tovkater, Banload and Emotet, which all have the most mentions in security vendor blogs and the research literature, at the time of the beginning of our experiments. However, we could not obtain enough samples of Wauchos, Nymaim and Pliskal to consistently test these three families. While it is not clear why no samples of Pliskal were available, it is likely that Wauchos samples were less prevalent following a large disruptive operation by law enforcement authorities worldwide in late 2017 [199]. As for Nymaim, the lack of available samples might be explained by the fact that the family is older and less active in 2018. We ran some preliminary tests, where samples of each of the remaining families were run through our whole framework. After running these experiments for a week, we noted every family that made at least one successful HTTP connection to an external server, i.e., that had at least one online C&C server. In the end, the only families to have at least one online C&C server were Tovkater and Banload.

The IP address and country are tested through a variation of 10 IP addresses from different countries. A keyboard layout is then chosen from 10 languages, and a display language is also set from the same 10 languages. Furthermore, 9 different browser histories and web

sessions have been established. The operating system is a home version of Windows, and 3 versions of Windows are tested : Windows 10, Windows 7 and Windows XP. By summing these features, we obtain a total of 42 feature variations to test. To obtain a large enough set of tested samples, the variations have been tested each day for a period of 12 months, and the test variations have been executed each day at different times in order to limit a potential bias due to the time of execution. We can establish the number of VMs needed for this experimental design as follows :

Definition 24 *The number of VMs needed can be calculated using* $V = d * s * k * t/p$*, where* d *is the number of downloader families,* s *is the number of samples of a family,* $k = n * l$ *is the number of feature variations,* t *is the time needed to run each sample,* p *is the total minutes in a day, and* V *is the number of VMs needed to run the tests.*

As mentioned previously, for our experiments, 42 feature variations are tested, with a running time of 15 minutes, where a day consists of 1440 minutes, i.e., 24 hours. A full factorial experiment is not possible, given that it would produce 27,000 feature variations, thus requiring 5,625 VMs. We opted to establish default values for each feature, where only a selected feature to test would be modified and tested. With this setup, we do not test combinations of features, notably matching display and keyboard languages. While this may impact the number of detonations we observe, the ones we experience will be more clearly associated to a single feature. Our default profile consists of the most popular feature for each category, namely the United States as the country, English as the keyboard layout and the display language, and *social networks* as the browser session.

Thus, using our formula, a minimum of 9 VMs are required to run our experiments each day.

6.4.6 Labeling

Our final dataset is a list of instances of multiple features of our VMs associated with the data extracted from the execution of a malicious downloader in a sandbox.

To test our hypothesis, we label each dataset entry with the family of the downloaded payloads from the execution of the malicious downloader. Each payload is tested on VirusTotal to establish if it is malicious, and if so, its family. Each sample is scanned once when it is discovered, and another time a month after the end of our experiments, in order to take into account labels that change over time [200]. The second scan result is used if different from the first. In order to filter possible false positives, we only consider malware with at least 5 positive reports in VirusTotal to be malicious, following previous research establishing the ideal threshold at between 2 and 15 [200].

Establishing the family of a malware sample, particularly a fresh new sample, is a difficult task that is still the subject of current research. Avclass [201] is a tool that aims at identifying malicious families through the various entries of a single VirusTotal report. While this tool works well when identifying large generic families, it proved unable to correctly differentiate our various families, labeling our samples as either a singleton (without a family), or all in one unique family. VAMO [202] also uses VirusTotal reports to cluster malware families, but aims at reducing AV label inconsistencies by using supervised learning instead of a majority voting of signatures. MalCommunity [203] uses a different approach when clustering malware, notably by creating a malware relation graph with the VirusTotal reports of samples and employing a graph community detection algorithm to establish clusters, thus identifying the various families. Other works, notably by De La Rosa *et al.* [204], aim at classifying malware using deep learning and various features of the actual malware and its execution. Unfortunately, works such as these are either tested on a specific subset of malicious families, or consist in a proof of concept without any published tool. Thus, we implement our own approach to identify malware families, for our particular setup.

Firstly, we inspect a subset of our payloads' VirusTotal reports, in order to identify the predominant labels for various payloads. These labels often consist in a generic identification, when the family cannot be correctly identified by the antivirus, e.g., a signature such as *Trojan.Generic.XYZ*. These labels introduce noise into our data and are thus discarded. From this, we obtain a set of label families for each payload. The labels differ between antivirus for various reasons : different antivirus software might have different family names for the same malware, some might have more fine-grained family identification than others, and some antivirus software might correctly identify a malicious software but incorrectly identify its family. Previous research has identified that many AV signatures are incomplete or inconsistent [205]. Thus, we initially establish a subset of precise family names used by antivirus software through our manual VirusTotal reports analysis, and we then label the malicious payloads with the most occurring label. If no label from our list is present, we default to the majority label found, similar to Avclass. Our labeling approach simply aims at clustering similar samples together, so as to establish if a cluster behaves differently for a machine profile. Thus, our clustering follows the naming of antivirus companies only as a means of differentiating samples. While previous research demonstrated ways to identify malware by its behavior [205–208], we limit our interactions inside our VMs to the minimum, and thus, do not make an in-depth analysis of our malicious downloaders' behavior to construct similarity profiles.

6.4.7 Time Series Analysis

In order to establish changes in the behavior of our downloader samples, we build time series out of our experiment data. The number of dropped payloads, for one particular family, as established in Section 6.4.6, is extracted from our results for every week of our experiments. This is repeated for every feature tested as to identify all changes of behavior through time for our various downloader families.

To test whether a change has happened in a time series, we employ changepoint analysis. Since our data consists of time series where we aim at identifying a change of behavior, we opted to use changepoint analysis to extract the various changepoints of each value of a feature, and thus, identify if one or more value of a feature changes at a different time than the others. Changepoint analysis is the detection of a change in the distribution of data in a time series, and it establishes the precise time at which a change occurs. Every point in time where a change occurs consists in a *changepoint*. Multiple algorithms have been established in the recent years to improve the detection of changepoints [209–212]. Changepoint analysis has been employed in previous research in computer security, although mainly in network intrusion and anomaly detection [213, 214].

Finally, for every downloader family and feature type, we compute the ratio of infections to total runs for each day of our experiments, and perform a one-way analysis of variance (ANOVA) of the means of the ratio of infections for each feature and each type of payload. With this analysis, we test wether one or more values of a feature's infection rate is statistically different than others and aim at validating our changepoint analysis.

6.5 Evaluation

Our approach was implemented following the framework described in the previous section. The resulting environment consists in a setup permitting the reverse engineering of the targeting choice of criminals employing malicious downloaders.

6.5.1 Machine Setup

We downloaded and installed Chrome as the Internet browser for the machine, and set it as the default Web browser, since it is the most used browser[4] and might be better supported by our downloaders. We did not include every popular browser, since we do not test browser fingerprinting by the downloader, but aim at measuring the impact that the browser history

[4] http://gs.statcounter.com/browser-market-share/desktop/worldwide

and session have on the downloader. A future work could include additional machines and add the browser as a tested feature. Python was installed on the machines to connect back to the sandbox, but no other modification was made in order to limit any malware from identifying our machine as a research project.

6.5.2 Features

As mentioned previously in section 6.4.4, various features have been selected for our experiments.

Firstly, we test multiple OSes, namely Windows XP, Windows 7 and Windows 10. Windows was chosen given that it is the most common desktop OS[5]. Windows 10 and Windows 7 were used since they consist in the most popular desktop OSes, while Windows XP was chosen given that it is not supported anymore by Microsoft but still has users running it worldwide.

We then aim to establish the impact of the location of the machine. We used a VPN located in a specific country to simulate the location. In order to establish the set of countries we wish to test, we identified the countries with the most Internet users[6]. We could not obtain a VPN access in Japan, and thus chose to test Korea instead. We also included Iran in order to have at least one country from the Middle East. We could not obtain a VPN located in China, and instead used Hong Kong as the location. Our intuition is that, at the time of our experiments, malicious downloaders targeting China might also target Hong Kong. In short, our list of countries consists of : Hong Kong, Iran, the United States, Brazil, Nigeria, Korea, Russia, Germany, Mexico, Bangladesh.

Next, we set the keyboard layout and the display language. The languages chosen were the top 10 languages of Internet users worldwide[7]. Thus, the languages chosen are : English, Chinese, Spanish, Arabic, Portuguese, Indonesian, French, Japanese, Russian, German.

While this list does not necessarily match the countries chosen before for our location feature, we still opted for this list, so that we covered the biggest number of Internet users. The keyboard languages and display languages are set by changing the corresponding Windows registry keys. The codes for each language are specified in Microsoft's online documentation for Windows 7 and Windows 10 [8], while some online resources and blogs have kept freely available lists of the Windows XP keyboard layout codes.

Finally, we identified nine Web browser session profiles and browser history profiles to test,

[5]http://gs.statcounter.com/os-market-share/desktop/worldwide/#monthly-201803-201803-bar
[6]https://www.internetworldstats.com/top20.htm
[7]https://www.internetworldstats.com/stats7.htm
[8]https://msdn.microsoft.com/en-us/library/ms776294.aspx

based on Alexa's categories and their list of most popular Web sites[9]. To create browser Web sessions and a browser history, we open in the VM's browser the top ten most popular Web sites of the current category being tested, and wait for them to load before beginning the experiment. Our categories are the following : Business, Games, Health, Kids and teens, Men, News, Social networks, Sports, Women.

6.5.3 Payloads

For our analysis, we recover files downloaded by the downloader, i.e., *payloads*, to establish their correlation with the features used during the execution. For the evaluation of these payloads, we consider every file type, but filter some specific files unrelated to the execution of the malicious file, i.e. files related to Windows updates, software updates, or crash logs. Notably, we noticed that the Internet browser sometimes crashed and sent crash dump files and reports. While the browser issues might have been the result of the malicious downloader's execution, the actual crash dump files are not pertinent for our analysis, and were thus discarded. As for the Windows and software updates, we did not consider *cab* files for our analysis, since they did not show any maliciousness when tested on VirusTotal. It is worth noting that any executable file downloaded as an update was kept for our analysis.

6.5.4 Labeling

As our labeling approach, we identified multiple families from the VirusTotal reports. The following are our identified families and are used as a label when they are the most occurring family : Eldorado, Adware, KillAV, Psyme, Banload, InstallMonster.

KillAV is a term that antivirus software use to label a malicious software which tries to disable or delete antivirus and firewall software installed on the machine.

Psyme is a specific malware which generally download additional malware.

Banload is one of the two downloader families which we test in our experiments. We have seen a number of these download an additional sample of the same family. A possible explanation is that these attacks consist in two stage attacks, or that the initial downloader fetches an updated version from its C&C server. Nonetheless, we consider it as a payload if it is downloaded by a downloader.

Eldorado is a keyword used by multiple antivirus software to label malicious software. It is unclear how specific this label is, but given that it only appears for specific payloads in our dataset, we kept it as a label.

[9] https://www.alexa.com/topsites/category

Adware consists in unwanted software, which either sends advertisements to the user or installs software or an add-on to existing software, without the user's consent. This category is not necessarily malicious software.

InstallMonster is a term used by many antivirus software to label a bundler which installs adware, or is itself an adware. If one of the previous labels is found along with this one, the other label is prioritized given that this label is more generic.

6.5.5 Time Series Analysis

Once our instances are labeled with a payload family, we construct a time series for every combination of a feature and a payload family. These time series show the number of infections of a particular family for all values of a feature used in our experiments. We then employ changepoint analysis on them to establish if one or more value of a feature differs from the others, indicating that for this particular value, the downloader's behavior changed. To execute our changepoint analysis, we've opted to use the Ruptures Python library [215], which implements a multitude of changepoint algorithms. We employ this library on every time series extracted from our results, and write down each value's changepoints time. We then identify, for one particular feature, which time values show one or more different changepoint.

Finally, we compute the ratio of infections for every feature for every day, by dividing the number of infections by the number of runs for this particular feature. Following this, we keep every day with at least one infection for one value of a feature and use this dataset for our analysis. We use the ratio of infections instead of the number of infections to compensate for any imbalance in the number of downloader runs per feature. While our *Profiler* manages the queue of experiments and equally distributes the features to test, some machines in our cluster were the victim of hardware issues and outages during the one year experiment, and were sporadically offline. To identify if the variance in the ratio of infections is significantly different for one or more values of a feature, we run a one-way ANOVA on ranks of our values, since the distribution of the ratio of infection is not a normal distribution. We employ the Kruskal-Wallis test [216], which is a non-parametric method to determine the difference between the means of different groups.

6.6 Results

We employed our framework over a one-year period, from February 2018 to February 2019, and ran malicious downloaders on our predefined machine profiles. We collected data on

these executions and established time series of infections for our various downloader families and payload families.

6.6.1 Dataset

Our final downloader dataset consists of 1,526 unique samples, where 711 were identified as part of the Tovkater family, and 805 were identified as part of the Banload family. On average, each sample was run 100 times, when testing our profile features according to our experimental design. In total, we ran 151,189 tests inside our VMs through the Cuckoo framework, with 72,829 tests run on Tovkater samples and 78,360 on Banload samples. Of these, 18,975 resulted in a *detonation* of the malicious downloader, i.e., these downloaders downloaded other malware payloads from the Internet. While most downloaders only fetched one payload, there were 258 that downloaded two payloads and 163 that downloaded three payloads. Multiple types of files were downloaded by the downloaders, depending on the initial sample and the needs of the C&C. The full list of downloaded file types is shown in Table 6.2, with the number of times they were observed. The file types were identified by inspecting the file header of the downloaded payloads.

We further investigated the downloaded payloads manually, in order to explain the different file types observed.

The high count of octet-stream and text files are in part due to crashes of the Web browser, which generated log files and crash dump files which were picked up by the sandbox.

Some text files also had information about Web cookies and the machine's country, and displayed a link to sendsmtp.com, which has been previously used in malicious campaigns[10]. A number of text files detailed a series of generated email and social network accounts using the machine's username. Since our machines were only active for 15 minutes per malware run, the malware's post-exploitation activity was limited. These text files might have been generated in order to steal the user's online accounts or to generate bot accounts.

The HTML payloads can be explained in part by error pages being downloaded, or pages returning 404 response codes instead of the actual payload. A small number of our downloader runs with HTML payloads did appear to have also downloaded a malicious executable. In these cases, the HTML pages appeared to be phishing websites.

Banload downloaders sometimes appeared not to act maliciously, possibly because the machine did not fit its infection criteria. Notably, they were observed downloading the following files instead of malicious executables:

[10]https://www.threatcrowd.org/domain.php?domain=sendsmtp.com

Table 6.2 File types of downloaded payloads with the number of occurences

File family	**File type**	**Count**
application	octet-stream	12770
	x-dosexec	5218
	vnd.ms-cab-compressed	295
	vnd.openxmlformats-officedocument	476
	vnd.ms-powerpoint(ppt/pptx)	373
	sqlite	38
	x-shockwave-flash	15
	zip	4291
text	plain	19916
	html	5989
	x-msdos-batch	888
	xml	75
	ini file	38
	vbscript	11
	json	509
image	png	1923
	jpeg	563
	gif	34
	svg	18
audio	x-wav	98

- Windows start navigation WAV audio files
- Benign icons, such as social network logos
- The official flash player install executable

While the text, HTML and octet-stream payload files generally indicated a failure to correctly run a downloader, other executions successfully detonated and downloaded up to four additional executables. Upon inspection of these detonated downloader samples, some began the infection with a batch script, a javascript page or a VBScript launching or downloading another malicious executable. A series of Banload runs had an initial batch file, which downloaded an executable, which in turn downloaded a malicious powerpoint file. Another type of multi-stage attack was observed for Banload downloaders, where they retrieved a Microsoft Office document, which then installed another malicious executable. In some cases, we observed an *autoliker* chrome extension being downloaded, hinting at a possible use of the infected machine as a bot on social networks.

A number of ZIP files were downloaded by our downloaders. Upon closer inspection, these did not seem to be valid ZIP files, but did appear to contain a malicious executable. While in these cases, no executable file was captured by our sandbox, the Cuckoo report did point to the Local Security Authority Subsystem Service (LSASS) process being injected with malicious code. Unfortunately, no additional payloads were retrieved in these cases.

Finally, to confirm the maliciousness of the retrieved payloads, we tested each one on VirusTotal, and ended up with 3,608 malicious payloads. In addition to the network traces and the files downloaded, we saved the output of our downloader executions in a basic comma-separated values (CSV) file, where for each execution the following information was stored:

- The date and time of execution
- A unique number identifying the execution of the downloader
- The SHA1 hash of the tested malicious downloader
- The family of the tested malicious downloader
- The display language used
- The keyboard layout used
- The version of Windows used

- The country of our VPN's exit server
- The Internet browser history category used
- The file type(s) of the downloaded payloads
- The file hashes and VirusTotal detections of the downloaded payloads

6.6.2 Labeling

We employed the technique described in section 6.4.6 to label all of our collected instances that resulted in a detonation. These samples had at least 5 positive detections in VirusTotal, and 1298 samples had a majority of positive detections.

In total, our collected payloads consist of 75 instances of KillAV, 134 instances of Psyme, 318 instances of Banload, 346 instances of Eldorado, 1,121 instances of Adware, and finally 1,580 instances of InstallMonster. As can be observed, the majority of our downloaded payloads were labeled as Adware or InstallMonster. Previous research identified that PUP and malware share infrastructure [41, 190], which explains the presence of Adware samples in the retrieved payloads of our downloaders.

6.6.3 Infections Through Time

Our experiments rely on a stream of malicious downloaders, where the goal is to detonate them in our infrastructure, i.e., successfully make them execute their malicious activity. One aspect we aimed at exploring was how the activity of a malicious family evolved over time in our setup. Figure 6.3 shows the number of dropped payloads from our infections over the course of our experimentations. The figure shows a very high quantity of detonations during the first months of our experiments, while there are periods of inactivity during the summer. We observe a more regular activity in the last months of 2018, although it is not as constant as in the spring.

The pauses in the droppers activities were thoroughly investigated : although we were using VPNs as the exit nodes in our experiments, we first considered the possibility that our VPN servers' IPs might be blacklisted from the C&Cs with which our samples communicate. But as Figure 6.3 shows, our experiments began detonating once more in the following two months, although with less frequency than in our initial month. Multiple explanations can be proposed for this decrease in activity : our families might have gotten less active over the months, our infrastructure or VPN servers could have been in part blocked, or our partner samples' *freshness* could have diminished over time. Previous work [149] has shown that

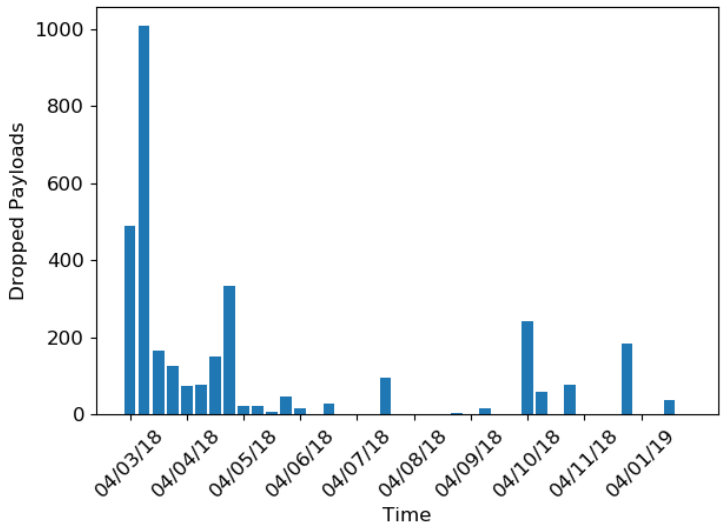


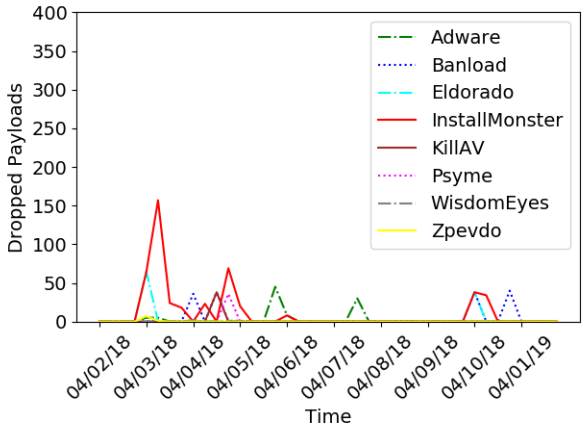
Figure 6.3 The number of dropped payloads over time

malicious websites hosting exploit kits remain active only for a median of 2.5 hours. Thus, if our provided samples became older, e.g. because our partner was detecting them less rapidly, they might not have been active anymore, similarly to these malicious websites hosting exploit kits.

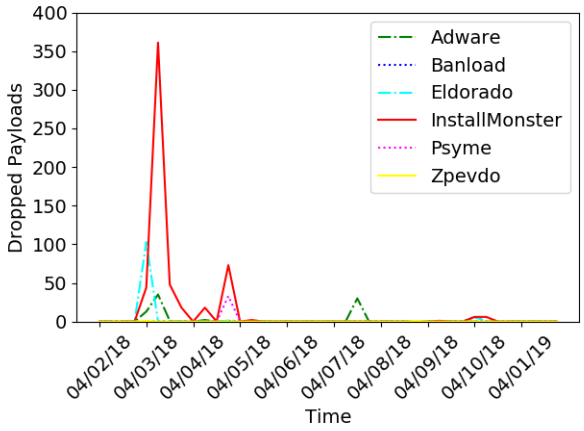
Since we do not control our stream of downloader samples, we cannot guarantee that the samples are new and that their C&C are still active. Thus, the consistent variations of activity could be due to our source of data. Another possible explanation would be that the actors controlling these downloaders only used them in these specific periods.

When observing the rate of infections for each of our labeled families, according to our various features, we can establish where a peak of infections occur for a specific feature and a specific malware family. We applied changepoint analysis to our various time series, and noted the feature values for which these vary. Our results also differ when testing our two downloader families. As the tests were running every day and the changepoint analysis needed less granularity, we created four bins for each month (on the 4th, 12th, 20th, and 28th) grouping the activities of the 7-8 days around them. Additionally, we ran a one-way ANOVA of the ratio of infection per day to identify if the features identified as statistically different match with our features identified in the changepoint analysis. Next, we will show the results returned by the application of the changepoint analysis on the different features, along with the results of our analysis of variance.

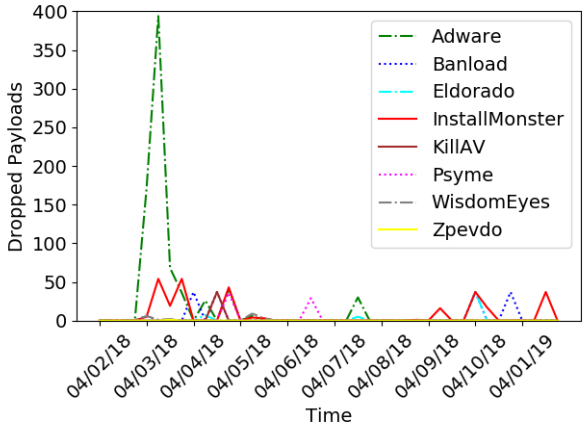
One of the most impactful features in our tests was, unsurprisingly, the operating system



(a) Infection rate when using Windows XP



(b) Infection rate when using Windows 7



(c) Infection rate when using Windows 10

Figure 6.4 The VM infection rate per dropped family

used in the VM running the downloader executable. Figure 6.4a shows that, when running Windows XP, we obtained less than half the number of infections of other OSes. Over time, we see more infections in the first months. The number of infections for each family on this OS is more distributed than on the other OSes. Particularly, Figure 6.4b highlights the infection rate for each dropper family on Windows 7. As can be seen, the bulk of the infections are in the first months, primarily in March 2018. The majority of the infections belong to the InstallMonster family, whereas Windows 10 is mainly a victim of Adware, as shown in Figure 6.4c. Whereas Windows 7 machines have seen a peak of infections from InstallMonster in a small period of time, Windows 10 has seen more distributed infections of this family over the course of our experiments. It is also worth noting that Banload infections occur at different periods for each OS.

The Windows version used in the VM is not the only feature highlighted by our analysis. Firstly, Figure 6.5 highlights infections that downloaded Adware as a payload while testing our browser profile feature and running the Tovkater downloader. As can be observed, there is an increase to 8 infections with solely a *news* browser profile, where the closest other browser profile is *sports* with 2 infections. Our changepoint analysis identified four common changepoints among the different browser profiles:

1. The 4th of March 2018 : The start of our experiments and an increase in the number of infections
2. The 12th of April 2018 : The moment of a second sudden peak of infections
3. The 28th of June 2018 : The end of the first infection pause mentioned earlier
4. The 4th of August 2018 : The separation between the activity in the summer and the decrease following it

The algorithm identified another relevant point in its analysis: the 20th of May 2018. The changepoint analysis highlighted that the *news* browser profile was showing different activities from the other ones. In fact, this profile has seen more detonations, resulting in a higher number of payloads downloaded in this configuration.

We ran a one-way ANOVA to compare the means of the ratio of infections by Adware for different browser profiles when running Tovkater, for the time period with the most activity: from March 6 2018 to June 6 2018. One browser profile was identified as being different: the *news* browser profile results were identified as statiscally higher than the *business* and *health* browser profiles with a p-value$<$0.05. Indeed, the news browser profile shows an average ratio of infection more than two times higher than infections with the health or business profile, which confirms that our machines with the news browser profile received more Adware payloads. The means of the ratio of infections for each browser profile are identified in Figure 6.6.

Secondly, we tested our various display languages, and observed that Spanish and German display languages obtained more Adware infections in May 2018, when running the Tovkater downloader. Figure 6.7 displays our results. The changepoint analysis identified the same moments in the timeline as for the profiles presented in the previous case. However, it also highlighted another relevant moment in June. This moment was relevant for all the VM settings, except when German is the display language. The June changepoint is not present when evaluating only the German display language. This is due to the fact that there is not a drop of infections for this language, contrary to the other languages. One additional

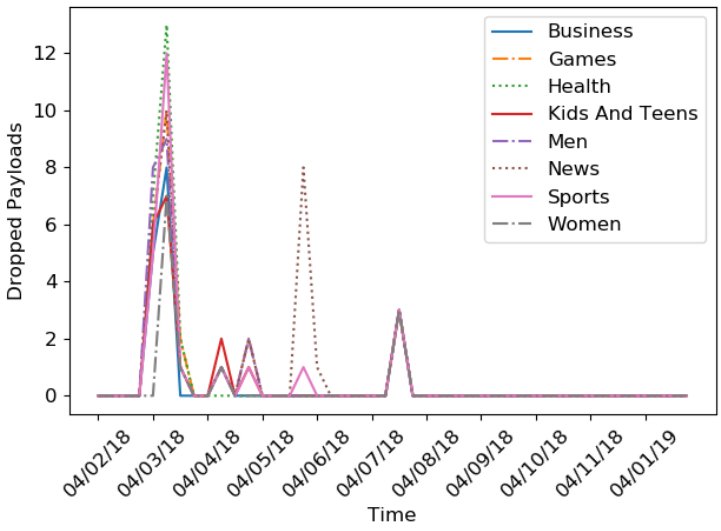


Figure 6.5 The number of dropped Adware payloads over time according to the browser profile

changepoint was identified solely for the Spanish display language, on the 20th of May 2018. In fact, Figure 6.7 shows an increased number of infections in this particular setting.

We ran an ANOVA on the means of the ratio of Adware infections for each display language when running Tovkater, and they were not found to be statistically different. While a changepoint was identified, our data might not be sufficient to conclude a difference between the number of infections per display language.

Another interesting activity was observed when testing the different keyboard layouts with the Banload downloader, as shown in Figure 6.8. Specifically for dropped payloads part of the Banload family, more activity was seen over time when the keyboard layout consisted in Portuguese. This result can be further confirmed by analyzing online activity of the Banload downloader[11][12], which is most active in Brazil, a country with Portuguese as its official language. Moreover, another keyboard layout with which the downloader is behaving differently is Chinese, where there are fewer detonations than with our other keyboard layouts. Our changepoint analysis identified four changepoints for all our keyboard layout values except Chinese:

1. The 4th of March 2018 : The start of our experiments

[11] `https://www.virusradar.com/en/Win32_TrojanDownloader.Banload/map`

[12] `https://securityboulevard.com/2019/05/cybercrime-groups-behind-banload-banking-malware-implement-new-techniques/`

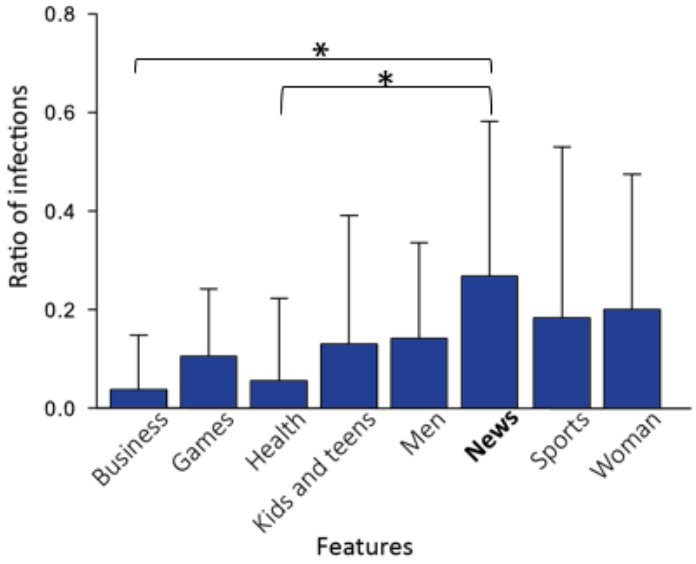


Figure 6.6 The average ratio of infections of Tovkater with Adware payloads over time according to the browser profile. Significantly different averages are marked in bold

2. The 20th of May 2018 : A drop in the number of infections
3. The 12th of September 2018 : Infections suddenly increased
4. The 28th of December 2018 : The infections' final drop

Some of these changepoints are different from our previous results, which is due to the infections occurring at different times. It is worth noting that there is a smaller number of infections than for our previous results. The Chinese keyboard language does not have any changepoints identified, which highlights the fact that VMs with a Chinese keyboard language are less infected. The changepoint analysis on the data related to the Portuguese keyboard language, however, has highlighted two additional moments in the timeline, the 28th of June 2018 and the 20th of October 2018. These moments correspond to an increased number of infections which is not happening when using the other layouts.

We also ran an ANOVA on the ratio of Banload infections for keyboard layouts when running the Banload downloader, and found one of them to be statistically different from March 20 2018 to June 6 2018. The Portuguese keyboard layout was found to be statistically different than the Chinese and Russian keyboard layouts, with a p-value$<$0.05. The means of the ratio of infection of each keyboard layout is shown in Figure 6.9, where we can observe that the Portuguese keyboard layout obtains more than twice the ratio of infections of the Russian and Chinese keyboard layouts.

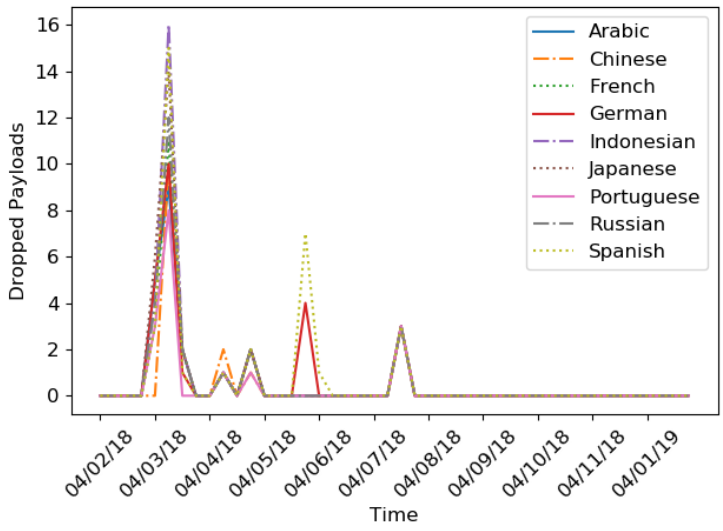


Figure 6.7 The number of dropped Adware payloads over time according to the display language

The Chinese keyboard layout appears to be the victim of less Adware infections when running Tovkater downloader samples as well. In fact, no Adware infection was registered at all for this keyboard layout. We ran an ANOVA on this feature and identified the Chinese keyboard layout to be different than almost all other layouts: it is statistically different than the Arabic, German, Russian, Japanese, French and Spanish keyboard layouts with a p-value$<$0.05. Machine profiles with a Chinese keyboard layout are indeed the victim of less infections by Adware payloads. The means of the ratio of infections per keyboard layout can be observed in Figure 6.10.

Finally, the last of our tested features is the location of our machines, identified by the country of their VPN server's exit IP address. While running Tovkater downloaders, we observed a peak of Adware infections in May 2018 when running our samples from Mexico. Our results are shown in Figure 6.11. The changepoint analysis highlighted the four changepoints identified in our first two analyses for all the locations.

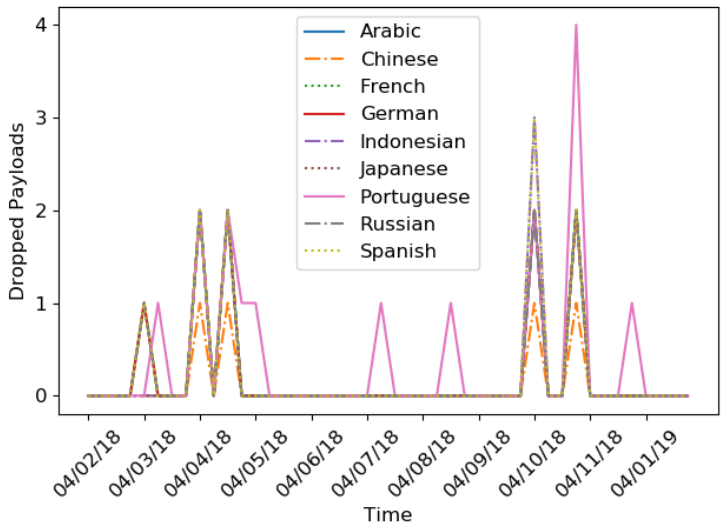


Figure 6.8 The number of dropped Banload payloads over time according to the keyboard layout

The *Mexico* location obtained an additional point of interest at the 20th of May 2018, when infections occurred more frequently specifically for this country.

Another noticeable phenomenon can be observed when Tovkater samples have dropped payloads of the InstallMonster family. When using the Hong Kong location, the infections in March and April 2018 were more numerous than for the other locations, as shown in Figure 6.12. On the other side, the VMs using Nigerian locations appeared to be less infected than the other machines. Our changepoint analysis identified only two relevant moments for all the locations:

1. The 4th of March 2018 : The beginning of our experiments
2. The 12th of April 2018 : A decrease in the number of infections

One additional changepoint was identified solely for the Hong Kong and Iran locations : the 20th of May 2018, corresponding to a slight increase in the number of infections. While analyzing our time series, only 9 infections were identified for the Nigeria location, compared to 24 at the highest for Hong Kong. Since a peak of infections still occurs for the Nigeria location, no changepoint was different for this location.

While changepoints were identified for some peaks of infections for machines in different locations, we ran an ANOVA of the ratio of infections and did not find any statistically different locations.

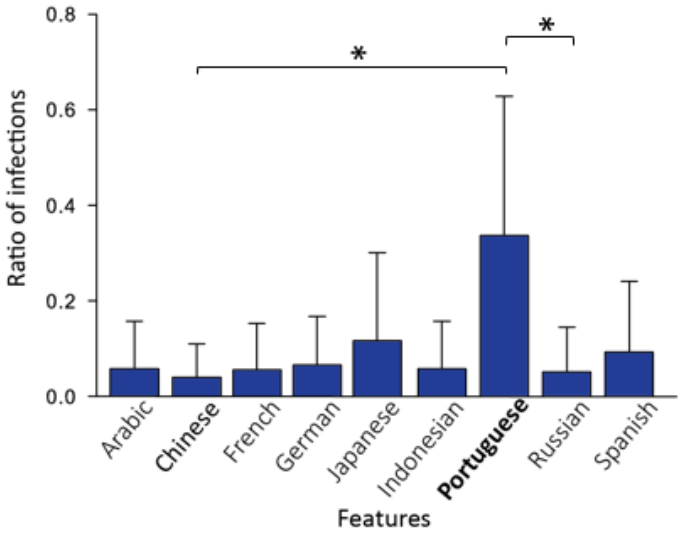


Figure 6.9 The average ratio of infections of Banload dropping Banload payloads over time according to the keyboard layout. Significantly different averages are in bold

6.7 Discussion

Through our experiments, we aimed at reverse engineering the targeting choice of criminals by establishing the link between a malicious downloader and the profile of the victim machine. Our approach did not look at the code of malicious samples, but empirically determined this link by setting up different VM profiles and analyzing whether samples were reacting to different profiles in different ways. In Section 6.6, we described the findings of our experiments, i.e., which profiles attracted a certain family of malware, and to what degree, with the number of infections. The changepoint analysis highlighted the important moments during the 12 months of experiments, and an analysis of variance identified if the average ratio of infection for a given feature value was significantly different than other values.

The highest number of malicious payloads was experienced on machines running Windows 10, while Windows XP resulted in the least infections. The popularity of Windows 10 among these malware families is particularly interesting as the OS has become more prevalent than Windows 7 only at the beginning of 2019[13]. Windows XP is not supported anymore by Microsoft, and is thus less secure than other modern OSes, however, these results show how it is becoming less relevant to the cybercriminals' purposes. One explanation might be that malware is targeting the most used OSes, and as such avoid Windows XP. Additionally,

[13]https://www.theverge.com/2019/1/2/18164916/microsoft-windows-10-market-share-passes-windows-7-statistics

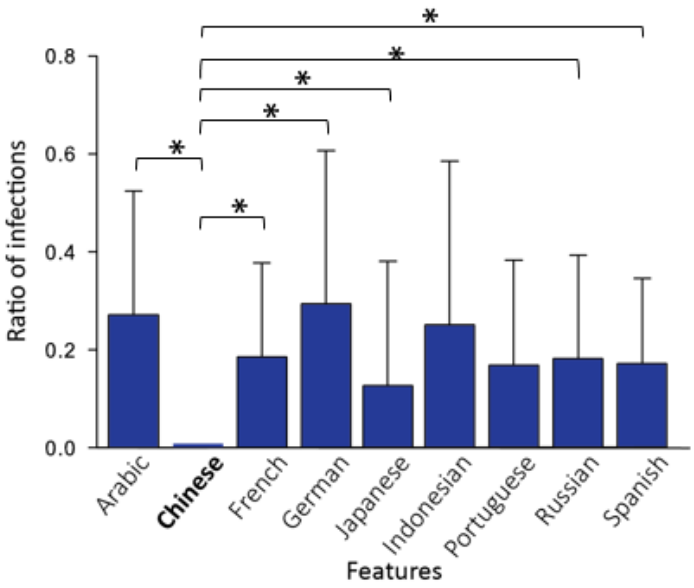


Figure 6.10 The average ratio of infections of Tovkater dropping Adware payloads over time according to the keyboard layout. Significantly different averages are in bold

droppers downloaded different families of malware depending on the OS. Windows XP and Windows 7 were mainly infected with InstallMonster payloads, while Adware payloads were used to target Windows 10 VMs.

Similar to previous work, our analysis has highlighted how the OS is a feature taken into consideration by criminals. However, our results also presented a new phenomenon, i.e., that the browser profile, keyboard layout and display language can have an impact on what payload a downloader downloads.

In the previous section we mentioned that at one point, Tovkater samples focused on the download of Adware payloads when observing a *news* browser profile. Indeed, users visiting news websites might be more susceptible to adware, or correspond more closely to the target demographic of the malicious actors. We also noted an increase in Banload downloaders downloading additional Banload payloads when running a machine with a Portuguese keyboard layout, particularly compared to the Chinese and Portuguese keyboard layouts.This is corroborated by news articles showing that a Banload campaign has run since May 2018[14]. Malicious actors appear to target Portuguese speaking countries through the Banload downloader. The Chinese keyboard layout also did not receive any Adware infections when running Tovkater samples, further confirming that this keyboard layout is less targeted by Adware

[14]`https://securityboulevard.com/2019/05/cybercrime-groups-behind-Banload-banking-malware-implement-new-techniques/`

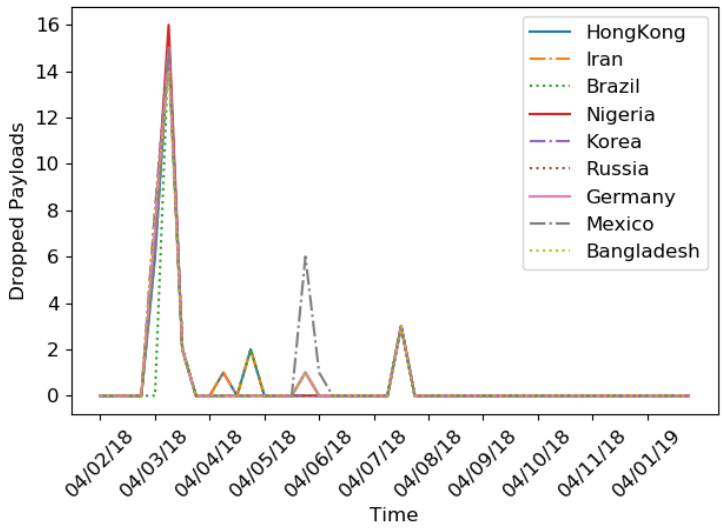


Figure 6.11 The number of dropped Adware payloads, over time, according to the country

samples. One explanation for this phenomenon can be that malicious actors avoid targeting their country of residence. In can also be that laws around adware infections are stricter in China, or that the downloader operators fear harsher penalties. Finally, it can simply be that Chinese speaking countries are less attractive to these malicious actors, due to there being less profits to be made.

The OSes and the Chinese keyboard layout results are showing some particular design choices made by cybercriminals; a selection of targets depending on their interests in what could be the best way to monetize their activities. However, the other findings are showing how the situations and the events influenced the choices of the attackers, aiming to earn as much as possible from particular situations.

These findings highlight how crucial it is to consider the context in which a malicious downloader is executed when trying to detonate it and observe its behavior. One of the key issues security researchers face when analyzing malware is effectively executing them in a research environment in order to identify their malicious behavior and build methods to detect and mitigate them. Our results show how identifying important profile features for a malicious downloader can not only have an impact on the number of downloaded payloads, but also on the type of downloaded payloads as well. It can help security researchers improve how they analyze malicious software in a research environment, thus leading to a better understanding of malicious downloaders and, ultimately, better countermeasures. While malware has been shown to target specific countries, our findings show how many more features of the victim machine can have an impact on what malware is received. An effective malware analysis

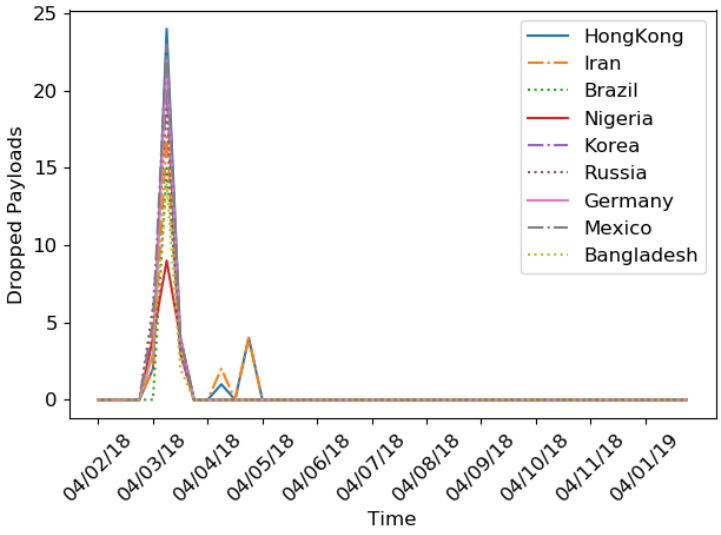


Figure 6.12 The number of dropped InstallMonster payloads, over time, according to the country

setup should consider the features of the VMs in order to obtain the best results.

6.7.1 Limitations

While our results have shown how Tovkater and Banload downloaders behave differently given various machine profiles, the downloader samples used in our experiments were provided by our antivirus partner, and as such, are only samples that could be detected and retrieved by them. Thus, these samples might not necessarily be representative of the general ecosystem of malicious downloaders. We also have limited information regarding the source and context from which originated the downloader samples.

Running live samples in our testing environment had some limitations. Testing too many samples that connect to the same C&C server might have gotten our VPN IP addresses blacklisted. Some downloaders might also have had virtualization detection techniques that we did not evade. Others had their C&C server already offline, in which case they did not download any malicious software. Overall, this results in fewer payloads accumulated through our experiments.

Another limitation of our dataset consists in the number of detonations of malicious downloaders we were able to produce. While we ran 151,189 tests, only 18,975 resulted in a detonation. This number gets further reduced when we split the payloads by family and analyze them by feature, as can be observed in section 6.6. Nonetheless, we feel confident

that the behavior changes observed are accurate.

Finally, it is worth noting that we tested each feature independently, in order to clearly identify if one feature impacted the execution of a downloader. We did not possess enough resources to test combinations of features, such as a matching keyboard and display language. However, while this setup might have negatively impacted the number of noteworthy detonations of malicious downloaders, we still obtained a number of results solely with our single feature tests.

6.7.2 Future Work

In this paper, we bring light to the behavior of malicious downloaders and PPI networks. From our findings, a number of research avenues can be established.

Firstly, our results show how malicious downloaders behave differently depending on what features are tested. While our setup was limited in terms of resources, many additional features could be tested to further establish which machine profile is targeted by PPI networks. For example, the use of finance software inside the VM or the setup of the machine as to appear in a health care network could provide valuable insights on the behavior of malicious actors. Furthermore, with enough experiments, using our approach with highly customized machines could in turn provide risk factors, i.e., which profile is more at risk of an infection from a particular family of malware.

Secondly, our automated analysis framework can be improved by incorporating a more thorough creation of customized virtual machines, by utilizing tools such as Malboxes [217], which permits the automatic building of a virtual machine image according to a specified configuration file. With this tool, we could build VMs that include a number of customized settings as well as a varying list of pre-installed software. This could improve our framework so that it includes more complex VM profiles, which provide more features to test. Eventually, this could be used to test more general profiles, i.e., profiles which include dozens of custom features, such as the network it's in, its installed software, and fake personalized files and system options. The correlation between a malware payload family and a general profile, instead of single features, could then be established with new experimentations. This approach could benefit our first future work as well.

Finally, our results could be compared to data retrieved from online black markets and dark web forums, in order to establish how well our approach mapped the behavior of PPI networks. From online black markets, we could obtain actual listings of criminals selling access to infected machines. The offers of malware installs for different machine profiles

could be used to confirm our findings, and possibly improve them. However, this would require a reliable way to obtain data on the selling of malware on black markets.

6.8 Conclusion

In this work, we aimed at reverse engineering the targeting choice of cybercriminals acting through PPI networks, by establishing if a link between the features of a machine and the payload(s) downloaded by a malicious downloader exists. We successfully built an automated sandboxed environment framework, which is capable of changing the configuration of a VM for a specific run of a malicious downloader executable. Using this setup, we ran 151,189 tests on Tovkater and Banload downloader families. Of these, 18,975 resulted in the download of at least one additional payload. With the use of changepoint analysis applied on time series of our infections, we identified different malicious payloads downloaded depending on the operating system.

More notably, we showed that malicious downloader families download different payloads depending on the browser history, the keyboard layout of the machine, highlighting the targeting choices of cybercriminals when infecting victims through downloaders.

Our findings show that an effective setup to analyze malicious downloaders should consider the features of the virtual machines in order to obtain better rates of detonation and to gather a larger array of malware families. Our work helps in assisting malware researchers in better analyzing malware and PPI networks, ultimately improving malware detection methods and countermeasures.

CHAPTER 7 GENERAL DISCUSSION

In this work, our research objective consisted in doing a deep analysis of the complete chain of infection by malicious software.

First, to better understand and mitigate cybercriminals in the attraction step of the infection, we built POISED. Through our approach, we aimed at improving our understanding of user connections in social networks, an important infection attraction vector, by establishing that structural communities of users share common topics of interest, distinct from other communities. With this new insight, we built a probabilistic model which predicts spam messages according to the path they take through communities of interest. In the attraction step, spam messages are a common vector malicious actors use to infect new unsuspecting users in a large number of attacks, and spam delivery services are accessible to cybercriminals through CaaS. With this framework, we were able to predict spam messages with high accuracy, precision and recall, and thus, improve on previous studies with this innovative approach at employing communities of users to identify spam. We additionally identified spam user accounts by identifying that the majority of users either post only legitimate messages or only spam messages. With a defined threshold, we used our approach's predictions to further identify spam accounts based on the percentage of spam messages an account posts. This approach also aims at being more resilient to evasion techniques, given that it does not model specific features of spam accounts or messages, but the way information travels through communities. We have shown that this approach is succesfull at identifying spam messages, and is scalable enough to be used in large networks.

While the attraction of victims is the starting point of the infection, understanding the further steps cybercriminals take to infect and take advantage of users is essential in properly establishing protections against infections. Following a spam message, a user will typically be redirected through a series of websites, until a final malicious page hosting an exploit kit fingerprints their system and infects one of its vulnerable components. As previously explained in Chapter 2, the redirection step can be provided through CaaS by a number of redirection providers, be it botnets, TDS, advertisements, legitimate proxies or a combination of them, rending its analysis complex.

To better understand this step, we successfully built a framework which reconstructs web redirections from a network packet capture file in Chapter 5. With this framework, we then improved over current methods of exploit kit family identification, by building a classification model capable of identifying the exploit kit family solely using features derived from the web

redirection chain leading to it. Our approach was partially successful, where a subset of tested exploit kit families were correctly identified. Some exploit kit families appeared linked to a number of different malicious campaigns, which is consistent with CaaS where exploit kits are sold as a service to other cybercriminals. Some exploit kit families were linked to a highly heterogenenous infrastructure, or a single campaign at a time, since our classification model could only predict their family when including the date as a feature to the model. While our previous project studied spam messages specifically, this project used data from malicious or compromised websites, and thus contains redirection chains that might originate from a number of infection vectors. With this framework, one can identify a subset of exploit kit families in network packet captures, or, with the use of the date as a feature, identify exploit kit families and their campaigns in historical network data. Furthermore, this approach can be useful in identifying which exploit kit families use CaaS, so that security analysts can adjust their countermeasures accordingly. This web redirection chain reconstruction framework has also been used successfully in another published work to establish the value chain of click fraud, through an analysis of the Boaxxe malware [218].

The infection step concludes when the exploit kit infects the machine with malware. What follows is one or more monetization schemes by the malware operator. However, an exploit kit will often first infect the user with a malicious downloader software. This software will download additional malware on the machine. Capturing this activity for research purposes is a difficult task, primarily due to the fact that downloader executables need to be relatively new and their C&C still active for them to successfully contact their C&C servers and retrieve one or more payloads. These must be run in a sandboxed environment, with the correct protections in place to prevent an infection of the host machine. The downloader might identify the testing environment and abort its operation. Our latest project presented in Chapter 6 aimed at testing malicious downloaders' behavior with different machine profiles, in order to shed light on how PPI services target victims, once the attraction and redirection steps of the infection are complete. As mentioned in Chapter 2, PPI services sell access to compromised machines to other cybercriminals as part of CaaS.

To better understand this step of the infection chain, we built an automated testing environment, which was run on a cluster of servers running a modified Cuckoo sandbox on each node. Our framework automated the testing of malicious executables on different Virtual Machine (VM) profiles. Using this setup, we successfully ran samples from two downloader families over a year and accumulated the payloads downloaded on profiles with 42 different feature variations. The samples tested were samples directly extracted from infected machines by our research partner, after the victims accessed malicious links, followed redirections to an exploit kit and got infected. With the use of changepoint analysis applied to

time series of our downloaded payloads, along with analyses of variance of the ratio of infections per day for different features, we finally extracted malware families linked to specific VM profiles at various points in time, and provided valuable insights on how cybercriminals target their victims. With this novel information, security analysts can better setup their testing environment by considering the profile of the VM running their malicious executables, and adjusting it according to the tested malware families.

In summary, our different projects brought light to how malware infections occur in the attraction step, the redirection step and the infection step. We built a framework to detect spam more effectively, and then classified a number of exploit kit families according to their redirection chains, and finally operated a large-scale analysis of malicious downloaders, showing how certain malicious payloads are linked to specific features of a machine. These projects helped in better understanding the complex chain of operations in malware infections, now operating through CaaS, and provided tools and insights to help security specialists protect users from malicious software.

CHAPTER 8 CONCLUSION

8.1 Summary of Works

In summary, this thesis aimed at analyzing the entire infection chain of malicious software in the hopes of better understanding the operations of malicious actors. We described the different steps of an infection, ranging from the *attraction* of victims, to the *redirection* through intermediaries, followed by the *infection* through exploit kits.

In the context of attraction, we established communities of interest in social networks, followed by the creation of a detection model of the diffusion of malicious messages through these communities. Previous research has focused on detecting malicious messages in social networks by analyzing the accounts or messages characteristics. Unfortunately, spam messages constantly evolve to evade detection techniques. In this work, we proposed a novel detection method based on the paths messages take through communities of interest. Our communities of interest were built by establishing structural communities from user connections graphs, and then combining them to topics extracted from messages shared in the community using LDA. A probabilistic model of groups of similar messages traveling through these communities was then built, followed by the use of an SVM classifier to establish if a message was spam or not. Our implementation was tested on data collected from Twitter, namely 15,751,198 English tweets posted by a total of 82,275 users. Our tweets were regrouped into documents, and LDA assigned a topic to each of these documents. Through the use of clustering metrics on our communities of interest, we were able to establish the validity of our first hypothesis that members of a networked community have similar topics of interest. We then implemented a probabilistic model of which communities a message passes through, and used SVM to build a predictor of spam messages using the probabilistic tables as input. Our approach performed well in detecting spam messages, obtaining an accuracy of 0.90, a precision of 0.91 and a recall of 0.93. From these predictions, we inferred spam accounts based on a threshold of the proportion of spam message publication.

In the context of the redirection step, we reconstructed the HTTP redirections leading to an exploit kit, and then classified the families of exploit kits according to their redirections. Previous research has focused on analyzing exploit kits' behavior and content features, while we propose a novel classification method of exploit kit families based on the entire malicious infrastructure. In this work, we established a methodology to accurately extract and identify the various HTTP redirections from a network capture, through the construction of a redirection tree. We also developed a heuristic to identify the malicious redirection chain

from this tree, and applied it to a dataset of network captures of infections of machines. We extracted 971 redirection chains, and used classification to predict the exploit kits associated with them. Our results showed that while the Infinity, Magnitude, Bleeding Life, Sweet Orange and Styx exploit kit families were predicted with high accuracy, Phoenix and Dotkachef could only be identified using the time and date as feature, and Angler, Nuclear, Blackhole and Critx could not be correctly identified. These results show that some exploit kits appear assigned to a single malicious campaign, while others subscribe to the CaaS model and are likely shared between different attacks. Finally, when filtering our dataset, we obtained high accuracy, precision and recall, respectively 0.921, 0.922 and 0.921. Thus, we validated our hypothesis for a subset of exploit kit families, where these 5 families can be classified solely according to their web redirection chains.

Finally, in the context of the infection step, we built an automated framework permitting the testing of malicious downloaders, through the use of customized virtual machines. Our framework consists in the first downloader testing environment that enables the reversing of cybercriminals' targeting choice, through an analysis of the downloaded malicious payloads matched against a virtual machine's features. We built our framework using a modified Cuckoo sandbox, with an automated setup capable of launching executables in a number of different VM profiles. With the use of this framework, we ran a one-year experiment where we tested 151,189 executables from two downloader families, Banload and Tovkater. Of these executions, 18,975 resulted in at least one additional payload downloaded. We investigated the various file types of downloaded payloads, along with the number of downloaded payloads. We used changepoint analysis applied to time series of downloaded payloads' families for each tested feature, and performed an analysis of variance of the ratio of infection per day for every feature. Our detected changepoints showed how, at specific points in time, the browser history, the keyboard layout and the display language of the machine were linked to specific malware families. Our analysis of variance further confirmed that the ratio of infection for the *news* browser profile was statiscally higher than the business and health browser profiles, when receiving an Adware payload from the Tovkater downloader. Additionally, the Portuguese keyboard layout was identified as statistically higher than the Chinese and Russian keyboard layouts, when receiving a Banload payload from a Banload Downloader. Finally, the Chinese keyboard layout was statistically lower than the majority of other keyboard layouts, in the context of a Tovkater downloader obtaining Adware payloads.

8.2 Limitations

In this work, we explored the malware infection chain, beginning with the attraction of victims through social network spam messages. We chose this attraction vector given that it has been on the rise over the past decade. However, this choice limits the scope of our work, and the framework presented in Chapter 4 was only tested on this type of attraction specifically. Future research can expand on this, as will be discussed in the following Section 8.3.

Additionally, although we analyze the entire malicious infection chain, our data is not from a single source and is thus not homogeneous. The ideal dataset for our work would have consisted in a list of malicious links in social networks with their captured web redirections towards an exploit kit, along with the downloader and malware installed through it. Unfortunately, to the best of our knowledge, no such dataset is publicly available, and our antivirus partner could not provide one. While this does not invalidate the results of our individual projects, it would have provided us with a single malicious path and would have permitted us to better map the results of our projects between themselves.

Finally, as explained in Chapter 2, many types of infections exist, and they do not necessarily employ all the same elements of EaaS. Our three projects presented in this work have distinct sources of data, which are not guaranteed to include the same types of attack. Thus, they might not directly match into a single path of attack, which might introduce some bias in our general analysis of the ecosystem based on what cyberattacks were present in our respective datasets.

8.3 Future Research

In the implementation of POISED, we used Twitter as our source of data. An interesting future work would be to explore applying the same methodology to other types of malicious attraction of victims, and compare the results to the effectiveness of our detection of spam messages on social networks. Particularly, it would enable us to observe how messages and information in general disseminate through communities of interest in different types of attraction vectors, and adjust our approach accordingly.

As mentioned in Chapter 2, web redirections following the visit of a malicious link can be the result of TDS, botnets, advertisement markets, or legitimate websites. These can be connected to an attraction campaign, where both the attraction and redirection steps are managed by the same actor, or they can be part of a single malicious campaign entirely managed by one actor. The different steps can also be managed completely separately as differet services, having no link to other parts of the infection. Considering this, exploring the

link between the attraction and redirection steps would provide valuable insights into how malicious campaigns operate, and how prevalent EaaS is. This could be done by combining our projects presented in Chapter 4 and Chapter 5. It would explore how spam travels through communities of interest, and how it continues through web redirections. Particularly, specific families or types of exploit kits might follow different spam dissemination patterns. Identifying these could provide an excellent avenue for detecting malicious attacks in their early stages.

Another future work would be to extend our downloader testing framework presented in Chapter 6, in order to catch a larger part of the malicious infection chain. This framework could be used to test the malicious attraction of victims, in order to capture the entire malicious chain, from attraction, to redirection, to downloader software. Such a framework would permit the construction of a complete dataset, which could lead to a number of new avenues of research. Finding links between specific types of activities in different steps of the infection could provide useful information in identifying the actors or groups behind malicious activities. Additionally, such a complete dataset could be used to build a more generalizable prediction model for different types of malicious attacks. Indeed, features could be extracted from all steps of the infection, so that a model can identify a malicious activity with more certainty even if part of the infection chain was modified. This model could be used in detecting malware, exploit kits, botnets, or malicious websites in general. Although producing this dataset would prove challenging, it would provide the highest value in terms of attack identification and prediction. Finally, this dataset would also provide future researchers with a way to explore malicious activity and test their own hypotheses.

Finally, in this work, we looked at the malicious infection chain from a technical point of view, analyzing its operation through real malicious data. A comparative study could be made between the various findings detailed in this work to data gathered from dark markets, where CaaS thrive. These criminal forums offer numerous services to help malicious actors carry out attacks. With an adequate source of forum data, one could compare malicious attraction campaign offerings to what was observed in Chapter 4, or compare the offering of exploit kits and redirections to our redirection patterns found in Chapter 5, or finally compare the offering of malicious installs with what was observed in Chapter 6.

REFERENCES

[1] World Wide Web Consortium, "Internet live stats," http://www.internetlivestats.com/internet-users/, accessed: 2020-07-09.

[2] B. Krebs, "The scrap value of a hacked pc, revisited," https://krebsonsecurity.com/2012/10/the-scrap-value-of-a-hacked-pc-revisited/, accessed: 2017-03-23.

[3] S. J. Choi, M. E. Johnson, and C. U. Lehmann, "Data breach remediation efforts and their implications for hospital quality," *Health services research*, vol. 54, no. 5, pp. 971–980, 2019.

[4] Kaspersky, "Statistics of local infections in the last 24 hour," https://statistics.securelist.com/en, accessed: 2020-07-09.

[5] "2019 CIRA Cybersecurity Survey," library Catalog: www.cira.ca. [Online]. Available: https://www.cira.ca/resources/cybersecurity/report/2019-cira-cybersecurity-survey

[6] "2019 Cost of Cybercrime Study | 9th Annual | Accenture," library Catalog: www.accenture.com. [Online]. Available: https://www.accenture.com/us-en/insights/security/cost-cybercrime-study

[7] J. Caballero *et al.*, "Measuring pay-per-install: The commoditization of malware distribution." in *Usenix security symposium*, 2011, p. 15.

[8] Statista, "Number of social media users worldwide from 2017 to 2025 (in billions)," https://www.statista.com/statistics/278414/number-of-worldwide-social-network-users/, accessed: 2021-03-23.

[9] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[10] L. Huang *et al.*, "Adversarial machine learning," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, 2011, pp. 43–58.

[11] P. Laskov and R. Lippmann, "Machine learning in adversarial environments," *Machine learning*, vol. 81, no. 2, pp. 115–119, 2010.

[12] J. Echeverría *et al.*, "LOBO–evaluation of generalization deficiencies in Twitter bot classifiers," *arXiv preprint arXiv:1809.09684*, 2018.

[13] C. Liu *et al.*, "Exploiting temporal dynamics in sybil defenses," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2015.

[14] A. Calleja, J. Tapiador, and J. Caballero, "A look into 30 years of malware development from a software metrics perspective," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2016, pp. 325–345.

[15] R. Broadhurst *et al.*, "An analysis of the nature of groups engaged in cyber crime," *An Analysis of the Nature of Groups engaged in Cyber Crime, International Journal of Cyber Criminology*, vol. 8, no. 1, pp. 1–20, 2014.

[16] J. Lusthaus, "How organised is organised cybercrime?" *Global Crime*, vol. 14, no. 1, pp. 52–60, 2013.

[17] R. Broadhurst *et al.*, "Malware trends on 'darknet'crypto-markets: Research review," *Available at SSRN 3226758*, 2018.

[18] P. H. Meland, Y. F. F. Bayoumy, and G. Sindre, "The ransomware-as-a-service economy within the darknet," *Computers & Security*, p. 101762, 2020.

[19] R. Van Wegberg *et al.*, "Plug and prey? measuring the commoditization of cybercrime via online anonymous markets," in *27th {USENIX} security symposium ({USENIX} security 18)*, 2018, pp. 1009–1026.

[20] K. Thomas *et al.*, "Framing dependencies introduced by underground commoditization," 2015.

[21] K. Huang, M. Siegel, and S. Madnick, "Systematically understanding the cyber attack business: A survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–36, 2018.

[22] S. Hinde, "Spam: the evolution of a nuisance," *Computers & Security*, vol. 22, no. 6, pp. 474–478, 2003.

[23] C. Kanich *et al.*, "Spamalytics: An empirical analysis of spam marketing conversion," in *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008, pp. 3–14.

[24] D. McCoy *et al.*, "Pharmaleaks: Understanding the business of online pharmaceutical affiliate programs," in *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12)*, 2012, pp. 1–16.

[25] J. I. Helfman and C. L. Isbell, "Ishmail: Immediate identification of important information," in *AT&T Labs*. Citeseer, 1995.

[26] W. W. Cohen *et al.*, "Learning rules that classify e-mail," in *AAAI spring symposium on machine learning in information access*, vol. 18. Stanford, CA, 1996, p. 25.

[27] "Apache SpamAssassin." [Online]. Available: https://spamassassin.apache.org/

[28] C. Pu and S. Webb, "Observed trends in spam construction techniques: A case study of spam evolution." in *CEAS*, 2006, pp. 104–112.

[29] Y. Xie *et al.*, "Spamming botnets: signatures and characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 171–182, 2008.

[30] E. Blanzieri and A. Bryl, "A survey of learning-based techniques of email spam filtering," *Artificial Intelligence Review*, vol. 29, no. 1, pp. 63–92, 2008.

[31] T. S. Guzella and W. M. Caminhas, "A review of machine learning approaches to spam filtering," *Expert Systems with Applications*, vol. 36, no. 7, pp. 10 206–10 222, 2009.

[32] N. Spirin and J. Han, "Survey on web spam detection: principles and algorithms," *Acm Sigkdd Explorations Newsletter*, vol. 13, no. 2, pp. 50–64, 2012.

[33] K. Levchenko *et al.*, "Click trajectories: End-to-end analysis of the spam value chain," in *2011 ieee symposium on security and privacy*. IEEE, 2011, pp. 431–446.

[34] G. Stringhini *et al.*, "The harvester, the botmaster, and the spammer: on the relations between the different actors in the spam landscape," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*, 2014, pp. 353–364.

[35] R. A. Rodríguez-Gómez, G. Maciá-Fernández, and P. García-Teodoro, "Survey and taxonomy of botnet research through life-cycle," *ACM Computing Surveys (CSUR)*, vol. 45, no. 4, pp. 1–33, 2013.

[36] A. K. Sood and R. J. Enbody, "Crimeware-as-a-service—a survey of commoditized crimeware in the underground market," *International Journal of Critical Infrastructure Protection*, vol. 6, no. 1, pp. 28–38, 2013.

[37] D. Manky, "Cybercrime as a service: a very modern business," *Computer Fraud & Security*, vol. 2013, no. 6, pp. 9–13, 2013.

[38] M. Hopkins and A. Dehghantanha, "Exploit kits: The production line of the cybercrime economy?" in *2015 second international conference on Information Security and Cyber Forensics (InfoSec)*. IEEE, 2015, pp. 23–27.

[39] A. Zarras *et al.*, "The dark alleys of madison avenue: Understanding malicious advertisements," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, 2014, pp. 373–380.

[40] A. K. Sood and R. J. Enbody, "Malvertising–exploiting web advertising," *Computer Fraud & Security*, vol. 2011, no. 4, pp. 11–16, 2011.

[41] B. J. Kwon *et al.*, "Catching worms, trojan horses and pups: Unsupervised detection of silent delivery campaigns," in *Proceedings of the 20th Network and Distributed Systems Security Symposium (NDSS)*, 2017.

[42] "ESET Threat Report Q2 2020," Jul. 2020, section: Threat Reports. [Online]. Available: https://www.welivesecurity.com/2020/07/29/eset-threat-report-q22020/

[43] "Early Findings: Review of State and Local Government Ransomware Attacks," May 2019, section: Research. [Online]. Available: https://www.recordedfuture.com/state-local-government-ransomware-attacks/

[44] M. Porolli, "Cybercrime black markets: Dark web services and their prices," Jan. 2019, section: Cybercrime. [Online]. Available: https://www.welivesecurity.com/2019/01/31/cybercrime-black-markets-dark-web-services-and-prices/

[45] R. Van Wegberg, A. Klievink, and M. Van Eeten, "Discerning novel value chains in financial malware," *European Journal on Criminal Policy and Research*, vol. 23, no. 4, pp. 575–594, 2017.

[46] D. Y. Huang *et al.*, "Botcoin: Monetizing stolen cycles." in *NDSS*. Citeseer, 2014.

[47] S. Eskandari *et al.*, "A first look at browser-based cryptojacking," in *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2018, pp. 58–66.

[48] M. Saad, A. Khormali, and A. Mohaisen, "End-to-end analysis of in-browser cryptojacking," *arXiv preprint arXiv:1809.02152*, 2018.

[49] H. Nguyen, "2013 state of social media spam," *Publication of NexGate, USA, from websites http://nexgate. com/wpcontent/uploads/2013/09/Nexgate-2013-State-of-Social-Media-Spam-Research-Report. pdf*, 2013.

[50] "Community Standards Enforcement report," library Catalog: transparency.facebook.com. [Online]. Available: https://transparency.facebook.com/community-standards-enforcement#fake-accounts

[51] M. Chakraborty *et al.*, "Recent developments in social spam detection and combating techniques: A survey," *Information Processing & Management*, vol. 52, no. 6, pp. 1053–1073, 2016.

[52] C. Grier *et al.*, "spam: the underground on 140 characters or less," in *Proceedings of the 17th ACM conference on Computer and communications security.* ACM, 2010, pp. 27–37.

[53] M. Egele *et al.*, "Compa: Detecting compromised accounts on social networks." in *NDSS*, 2013.

[54] T. N. Jagatic *et al.*, "Social phishing," *Communications of the ACM*, vol. 50, no. 10, pp. 94–100, 2007.

[55] G. Wang *et al.*, "You are how you click: Clickstream analysis for sybil detection." in *Usenix Security*, vol. 14, 2013.

[56] G. Danezis and P. Mittal, "Sybilinfer: Detecting sybil nodes using social networks." in *NDSS*. San Diego, CA, 2009.

[57] E. Ferrara *et al.*, "The rise of social bots," *arXiv preprint arXiv:1407.5225*, 2014.

[58] G. Stringhini, C. Kruegel, and G. Vigna, "Detecting spammers on social networks," in *Proceedings of the 26th Annual Computer Security Applications Conference.* ACM, 2010, pp. 1–9.

[59] A. H. Wang, "Detecting spam bots in online social networking sites: a machine learning approach," in *Data and Applications Security and Privacy XXIV.* Springer, 2010, pp. 335–342.

[60] B. Viswanath *et al.*, "Towards detecting anomalous user behavior in online social networks." in *Usenix Security*, vol. 14, 2014.

[61] Q. Cao *et al.*, "Uncovering large groups of active malicious accounts in online social networks," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 2014, pp. 477–488.

[62] C. Yang, R. C. Harkreader, and G. Gu, "Die free or live hard? empirical evaluation and new design for fighting evolving Twitter spammers," in *Recent Advances in Intrusion Detection (RAID)*, 2011.

[63] M. McPherson, L. Smith-Lovin, and J. M. Cook, "Birds of a feather: Homophily in social networks," *Annual review of sociology*, pp. 415–444, 2001.

[64] L. Weng, F. Menczer, and Y.-Y. Ahn, "Virality prediction and community structure in social networks," *Scientific reports*, vol. 3, 2013.

[65] A. Nematzadeh *et al.*, "Optimal network modularity for information diffusion," *Physical review letters*, vol. 113, no. 8, p. 088701, 2014.

[66] K. Lerman and R. Ghosh, "Information contagion: An empirical study of the spread of news on digg and Twitter social networks." *ICWSM*, vol. 10, pp. 90–97, 2010.

[67] Twitter, "Reporting spam on Twitter," https://support.twitter.com/articles/64986, 2009.

[68] K. Thomas *et al.*, "Suspended accounts in retrospect: an analysis of Twitter spam," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM, 2011, pp. 243–258.

[69] H. Gao *et al.*, "Detecting and characterizing social spam campaigns," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 35–47.

[70] ——, "Towards online spam filtering in social networks." in *NDSS*, 2012.

[71] C. Fleizach, G. M. Voelker, and S. Savage, "Slicing spam with occam's razor," in *CEAS*, 2007.

[72] S. Lee and J. Kim, "WarningBird: Detecting suspicious URLs in Twitter stream." in *NDSS*, 2012.

[73] K. Thomas *et al.*, "Design and evaluation of a real-time url spam filtering service," in *Security and Privacy (SP), 2011 IEEE Symposium on*. IEEE, 2011, pp. 447–462.

[74] M. Egele *et al.*, "Towards detecting compromised accounts on social networks," *Transactions on Dependable and Secure Computing (TDSC)*, 2015.

[75] M. J. Culnan, P. J. McHugh, and J. I. Zubillaga, "How large US companies can use Twitter and other social media to gain business value," *MIS Quarterly Executive*, vol. 9, no. 4, pp. 243–259, 2010.

[76] A. Java *et al.*, "Why we Twitter: understanding microblogging usage and communities," in *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*. ACM, 2007, pp. 56–65.

[77] K. H. Lim and A. Datta, "Finding Twitter communities with common interests using following links of celebrities," in *Proceedings of the 3rd international workshop on Modeling social media*. ACM, 2012, pp. 25–32.

[78] L. Tang and H. Liu, "Community detection and mining in social media," *Synthesis Lectures on Data Mining and Knowledge Discovery*, vol. 2, no. 1, 2010.

[79] S. Yardi *et al.*, "Detecting spam in a Twitter network," *First Monday*, vol. 15, no. 1, 2009.

[80] W. Xu, F. Zhang, and S. Zhu, "Toward worm detection in online social networks," in *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM, 2010, pp. 11–20.

[81] F. Benevenuto *et al.*, "Detecting spammers on Twitter," in *Collaboration, electronic messaging, anti-abuse and spam conference (CEAS)*, vol. 6, 2010, p. 12.

[82] Q. Cao *et al.*, "Aiding the detection of fake accounts in large scale social online services," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 15–15.

[83] Z. Cai and C. Jermaine, "The latent community model for detecting sybil attacks in social networks," in *Proc. NDSS*, 2012.

[84] L. Liu *et al.*, "Detecting" smart" spammers on social network: A topic model approach," *arXiv preprint arXiv:1604.08504*, 2016.

[85] S. Ghosh *et al.*, "Understanding and combating link farming in the Twitter social network," in *Proceedings of the 21st international conference on World Wide Web*. ACM, 2012, pp. 61–70.

[86] Y. Boshmaf *et al.*, "Integro: Leveraging victim prediction for robust fake account detection in osns." in *NDSS*, vol. 15, 2015, pp. 8–11.

[87] G. Stringhini *et al.*, "Evilcohort: detecting communities of malicious accounts on online services," in *USENIX Security Symposium*, 2015.

[88] S. Ye and S. F. Wu, *Measuring message propagation and social influence on Twitter.* Springer, 2010.

[89] C. Tan, L. Lee, and B. Pang, "The effect of wording on message propagation: Topic-and author-controlled natural experiments on Twitter," *arXiv preprint arXiv:1405.1438*, 2014.

[90] G. Mezzour and K. M. Carley, "Spam diffusion in a social network initiated by hacked e–mail accounts," *International Journal of Security and Networks*, vol. 9, no. 3, pp. 144–153, 2014.

[91] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann, "Link communities reveal multiscale complexity in networks," *Nature*, vol. 466, no. 7307, pp. 761–764, 2010.

[92] G. Palla *et al.*, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, no. 7043, pp. 814–818, Jun. 2005.

[93] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[94] A. Lancichinetti *et al.*, "High-reproducibility and high-accuracy method for automated topic classification," *Physical Review X*, vol. 5, p. 011007, JAN 2015.

[95] J. Li, C. Cardie, and S. Li, "Topicspam: a topic-model based approach for spam detection," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2013, pp. 217–221.

[96] I. Bíró, J. Szabó, and A. A. Benczúr, "Latent dirichlet allocation in web spam filtering," in *Proceedings of the 4th international workshop on Adversarial information retrieval on the web*, 2008, pp. 29–32.

[97] V. Ramanathan and H. Wechsler, "Phishing detection and impersonated entity discovery using conditional random field and latent dirichlet allocation," *Computers & Security*, vol. 34, pp. 123–139, 2013.

[98] L. Hong and B. D. Davison, "Empirical study of topic modeling in Twitter," in *Proceedings of the first workshop on social media analytics.* ACM, 2010, pp. 80–88.

[99] O. Tsur, A. Littman, and A. Rappoport, "Efficient clustering of short messages into general domains." in *ICWSM*. Citeseer, 2013.

[100] K. D. Rosa *et al.*, "Topical clustering of tweets," *Proceedings of the ACM SIGIR: SWSM*, 2011.

[101] S. Dann, "Twitter content classification," *First Monday*, vol. 15, no. 12, 2010.

[102] G. Stringhini *et al.*, "Poultry Markets: On the Underground Economy of Twitter Followers," in *Proceedings of the Workshop on Online Social Network (WOSN)*. Helsinki, Finland: ACM, August 2012.

[103] Twitter, "Twitter usage," https://about.twitter.com/company, 2015.

[104] S. B. Seidman, "Network structure and minimum degree," *Social networks*, vol. 5, no. 3, pp. 269–287, 1983.

[105] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proceedings of the National Academy of Sciences*, vol. 105, no. 4, pp. 1118–1123, 2008.

[106] E. Eaton and R. Mansbach, "A spin-glass model for semi-supervised community detection." in *AAAI*. Citeseer, 2012.

[107] P. Pons and M. Latapy, "Computing communities in large networks using random walks," in *Computer and Information Sciences-ISCIS 2005*. Springer, 2005, pp. 284–293.

[108] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[109] A. Clauset, M. E. Newman, and C. Moore, "Finding community structure in very large networks," *Physical review E*, vol. 70, no. 6, p. 066111, 2004.

[110] V. D. Blondel *et al.*, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008.

[111] A. Lancichinetti and S. Fortunato, "Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities," *Physical Review E*, vol. 80, no. 1, p. 016118, 2009.

[112] ——, "Community detection algorithms: a comparative analysis," *Physical review E*, vol. 80, no. 5, p. 056117, 2009.

[113] A. K. McCallum, "{MALLET: A Machine Learning for Language Toolkit}," http://mallet.cs.umass.edu/, 2002.

[114] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure." in *EMNLP-CoNLL*, vol. 7, 2007, pp. 410–420.

[115] K. Y. Yeung and W. L. Ruzzo, "Details of the adjusted rand index and clustering algorithms, supplement to the paper "an empirical study on principal component analysis for clustering gene expression data"," *Bioinformatics*, vol. 17, no. 9, pp. 763–774, 2001.

[116] J. M. Santos and M. Embrechts, "On the use of the adjusted rand index as a metric for evaluating supervised classification," in *International Conference on Artificial Neural Networks*. Springer, 2009, pp. 175–184.

[117] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: is a correction for chance necessary?" in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 1073–1080.

[118] B. Csákány, "Homogeneity and completeness," in *Fundamentals of Computation Theory*. Springer, 1981, pp. 81–89.

[119] P. O. Perry and P. J. Wolfe, "Null models for network data," *arXiv preprint arXiv:1201.5871*, 2012.

[120] M. E. Newman and J. Park, "Why social networks are different from other types of networks," *Physical Review E*, vol. 68, no. 3, p. 036122, 2003.

[121] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: User movement in location-based social networks," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2011.

[122] J. Su and S. Wu, "Null models for social networks," http://snap.stanford.edu/class/cs224w-2013/projects2013/cs224w-003-final.pdf, 2013.

[123] P. J. Mucha *et al.*, "Community structure in time-dependent, multiscale, and multiplex networks," *science*, vol. 328, no. 5980, pp. 876–878, 2010.

[124] D. Chatzakou *et al.*, "Mean birds: Detecting aggression and bullying on Twitter," in *International ACM Web Science Conference (WebSci)*, 2017.

[125] N. V. Chawla *et al.*, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[126] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'95. Morgan Kaufmann Publishers Inc., 1995.

[127] P. Refaeilzadeh, L. Tang, and H. Liu, "Cross-validation," in *Encyclopedia of database systems*. Springer, 2009, pp. 532–538.

[128] U. M. Braga-Neto and E. R. Dougherty, "Is cross-validation valid for small-sample microarray classification?" *Bioinformatics*, vol. 20, no. 3, pp. 374–380, 2004.

[129] R. Aldecoa and I. Marín, "Exploring the limits of community detection strategies in complex networks," *Scientific reports*, vol. 3, 2013.

[130] Y. Wang *et al.*, "PLDA: Parallel latent dirichlet allocation for large-scale applications," in *International Conference on Algorithmic Applications in Management*. Springer, 2009, pp. 301–314.

[131] R. Nallapati, W. Cohen, and J. Lafferty, "Parallelized variational EM for latent Dirichlet allocation: An experimental evaluation of speed and scalability," in *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*. IEEE, 2007, pp. 349–354.

[132] P. Smyth, M. Welling, and A. U. Asuncion, "Asynchronous distributed learning of topic models," in *Advances in Neural Information Processing Systems*, 2009, pp. 81–88.

[133] R. Gomes, M. Welling, and P. Perona, "Memory bounded inference in topic models," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 344–351.

[134] I. Porteous *et al.*, "Fast collapsed Gibbs sampling for latent Dirichlet allocation," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 569–577.

[135] C. Kim and K. Shim, "Text: Automatic template extraction from heterogeneous web pages," *IEEE Transactions on knowledge and data Engineering*, vol. 23, no. 4, pp. 612–626, 2011.

[136] A. S. Das *et al.*, "Google news personalization: scalable online collaborative filtering," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 271–280.

[137] O. Chum *et al.*, "Near duplicate image detection: min-hash and tf-idf weighting." in *BMVC*, vol. 810, 2008, pp. 812–815.

[138] S. Nilizadeh *et al.*, "POISED: Spotting Twitter spam off the beaten paths," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1159–1174.

[139] B. Alghamdi, Y. Xu, and J. Watson, "A hybrid approach for detecting spammers in online social networks," in *International Conference on Web Information Systems Engineering*. Springer, 2018, pp. 189–198.

[140] P. Andriotis and A. Takasu, "Emotional bots: Content-based spammer detection on social media," in *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE, 2018, pp. 1–8.

[141] D. Mulamba, I. Ray, and I. Ray, "On sybil classification in online social networks using only structural features," in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 2018, pp. 1–10.

[142] B. Wang, L. Zhang, and N. Z. Gong, "SybilBlind: Detecting fake users in online social networks without manual labels," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018, pp. 228–249.

[143] C. Xiao, D. M. Freeman, and T. Hwa, "Detecting clusters of fake accounts in online social networks," in *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*. ACM, 2015, pp. 91–101.

[144] D. Yuan *et al.*, "Detecting fake accounts in online social networks at the time of registrations," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1423–1438.

[145] C. A. Tuttle, S. Patel, and H. Yue, "Malicious message detection on Twitter via dissemination paths," in *2020 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2020, pp. 400–404.

[146] J. Zhang *et al.*, "Error-sensor: Mining information from HTTP error traffic for malware intelligence," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018, pp. 467–489.

[147] G. Stringhini, C. Kruegel, and G. Vigna, "Shady paths: leveraging surfing crowds to detect malicious web pages," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 133–144.

[148] H. Mekky *et al.*, "Detecting malicious HTTP redirections using trees of user browsing activity," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 1159–1167.

[149] C. Grier *et al.*, "Manufacturing compromise: the emergence of exploit-as-a-service," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 821–832.

[150] G. De Maio *et al.*, "PExy: The other side of exploit kits," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2014, pp. 132–151.

[151] B. Eshete and V. Venkatakrishnan, "WebWinnow: Leveraging exploit kit workflows to detect malicious urls," in *Proceedings of the 4th ACM conference on Data and application security and privacy*. ACM, 2014, pp. 305–312.

[152] Z. Ma, "The decline of exploit kits as an exploitation strategy," 2018.

[153] M. Cova, C. Kruegel, and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious JavaScript code," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 281–290.

[154] M. Hall *et al.*, "The WEKA data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[155] L. Lu, R. Perdisci, and W. Lee, "SURF: detecting and measuring search poisoning," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 467–476.

[156] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[157] S. Lee and J. Kim, "Warningbird: A near real-time detection system for suspicious urls in Twitter stream," *IEEE transactions on dependable and secure computing*, vol. 10, no. 3, pp. 183–195, 2013.

[158] Z. Li *et al.*, "Hunting the red fox online: Understanding and detection of mass redirect-script injections," in *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 2014, pp. 3–18.

[159] T. Nelms *et al.*, "WebWitness: investigating, categorizing, and mitigating malware download paths," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 1025–1040.

[160] M. Akiyama *et al.*, "Analyzing the ecosystem of malicious url redirection through longitudinal observation from honeypots," *Computers & Security*, 2017.

[161] H. Kwon, M. B. Baig, and L. Akoglu, "A domain-agnostic approach to spam-URL detection via redirects," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2017, pp. 220–232.

[162] K. Hans, L. Ahuja, and S. Muttoo, "Detecting redirection spam using multilayer perceptron neural network," *Soft Computing*, pp. 1–12, 2017.

[163] J. Burgess *et al.*, "REdiREKT: Extracting malicious redirections from exploit kit traffic," in *2020 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2020, pp. 1–9.

[164] Z. Li *et al.*, "Finding the linchpins of the dark web: a study on topologically dedicated hosts on malicious web infrastructures," in *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 2013, pp. 112–126.

[165] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1, pp. 107–117, 1998.

[166] J. Zhang *et al.*, "ARROW: Generating signatures to detect drive-by downloads," in *Proceedings of the 20th international conference on World wide web*. ACM, 2011, pp. 187–196.

[167] T.-K. Huang, N. C. Valler, and M. Faloutsos, "Characterizing the scam hosting infrastructure," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*. IEEE, 2010, pp. 1–5.

[168] B. Chen and Y. Shi, "Malicious hidden redirect attack web page detection based on css features," in *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*. IEEE, 2018, pp. 1155–1159.

[169] T. Taylor *et al.*, "Detecting malicious exploit kits using tree-based similarity searches," in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*. ACM, 2016, pp. 255–266.

[170] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks." in *Lisa*, vol. 99, no. 1, 1999, pp. 229–238.

[171] B. Stock, B. Livshits, and B. Zorn, "Kizzle: A signature compiler for exploit kits," in *International Conference on Dependable Systems and Networks (DSN)*, 2015.

[172] V. Kotov and F. Massacci, "Anatomy of exploit kits," in *International Symposium on Engineering Secure Software and Systems*. Springer, 2013, pp. 181–196.

[173] N. Šrndić and P. Laskov, "Hidost: a static machine-learning-based detector of malicious files," *EURASIP Journal on Information Security*, vol. 2016, no. 1, p. 22, 2016.

[174] N. Šrndic and P. Laskov, "Detection of malicious PDF files based on hierarchical document structure," in *Proceedings of the 20th Annual Network & Distributed System Security Symposium*. Citeseer, 2013, pp. 1–16.

[175] S. Kim and B. B. Kang, "FriSM: Malicious exploit kit detection via feature-based string-similarity matching," in *SecureComm*, 2018.

[176] A. Dell'Aera, "Thug - Python low-interaction honeyclient." [Online]. Available: https://buffer.github.io/thug/

[177] S. Kim, S. Kim, and D. Kim, "LoGos: Internet-explorer-based malicious webpage detection," *ETRI Journal*, vol. 39, no. 3, pp. 406–416, 2017.

[178] Barracuda Labs, "Threatglass," http://threatglass.com/, accessed: 2017-03-03.

[179] S. Ostermann, "Tcptrace," 2005.

[180] A. Liaw, M. Wiener *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[181] B. Larin, "Magnitude exploit kit – evolution," https://securelist.com/magnitude-exploit-kit-evolution/97436/, 2020. [Online]. Available: https://securelist.com/magnitude-exploit-kit-evolution/97436/

[182] sven_t, "Jsdetox," http://www.relentless-coding.org/projects/jsdetox/, accessed: 2019-02-01.

[183] P. Kotzias, L. Bilge, and J. Caballero, "Measuring pup prevalence and pup distribution through pay-per-install services," in *Proceedings of the USENIX Security Symposium*, 2016.

[184] K. Thomas *et al.*, "Investigating commercial pay-per-install and the distribution of unwanted software," in *USENIX Security Symposium*, 2016.

[185] F. Lalonde Levesque *et al.*, "A clinical study of risk factors related to malware infections," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 97–108.

[186] C. Rossow, C. Dietrich, and H. Bos, "Large-scale analysis of malware downloaders," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2012, pp. 42–61.

[187] B. J. Kwon *et al.*, "The dropper effect: Insights into malware distribution with downloader graph analytics," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1118–1129.

[188] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[189] B. Rahbarinia, M. Balduzzi, and R. Perdisci, "Real-time detection of malware downloads via large-scale URL-> file-> machine graph mining," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM, 2016, pp. 783–794.

[190] C. C. Ife *et al.*, "Waves of malice: A longitudinal measurement of the malicious file delivery ecosystem on the web," in *ACM ASIA Conference on Computer and Communications Security*. Association for Computing Machinery, 2019.

[191] Cuckoo Sandbox, "Automated malware analysis," https://cuckoosandbox.org, 2013.

[192] Virus Total Scanner, "VirusTotal-free online virus, malware and URL scanner," *Online: https://www.virustotal.com/en*, 2012.

[193] X. Chen *et al.*, "Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware," in *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*. IEEE, 2008, pp. 177–186.

[194] A. Ortega, "Pafish is a demonstration tool that employs several techniques to detect sandboxes and analysis environments in the same way as malware families do," Jun. 2019, original-date: 2012-07-01T11:06:40Z. [Online]. Available: https://github.com/a0rtega/pafish

[195] Noteworthy, "Public malware techniques used in the wild: Virtual Machine, Emulation, Debuggers, Sandbox detection," Jun. 2019, original-date: 2015-11-12T18:35:16Z. [Online]. Available: https://github.com/LordNoteworthy/al-khaser

[196] P. Ferrie, "Attacks on more virtual machine emulators," *Symantec Technology Exchange*, vol. 55, 2007.

[197] J. Jain, "Banload trojan targets brazilians with malware downloads," https://securingtomorrow.mcafee.com/mcafee-labs/banload-trojan-targets-brazilians-with-malware-downloads/, 2016, accessed: 2018-06-28.

[198] P.-M. Bureau, "Malware trying to avoid some countries," https://www.welivesecurity.com/2009/01/15/malware-trying-to-avoid-some-countries/, 2009, accessed: 2018-06-28.

[199] T. Foltýn, "ESET research: Wauchos now headed for extinction?" https://www.welivesecurity.com/2018/01/04/wauchos-now-headed-extinction/, 2018, accessed: 2018-07-23.

[200] S. Zhu *et al.*, "Measuring and modeling the label dynamics of online anti-malware engines," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020, pp. 2361–2378.

[201] M. Sebastián *et al.*, "AVclass: A tool for massive malware labeling," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2016, pp. 230–253.

[202] R. Perdisci *et al.*, "VAMO: towards a fully automated malware clustering validity analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 2012, pp. 329–338.

[203] Y. Chen *et al.*, "MalCommunity: A graph-based evaluation model for malware family clustering," in *International Conference of Pioneering Computer Scientists, Engineers and Educators*. Springer, 2018, pp. 279–297.

[204] L. De La Rosa *et al.*, "Efficient characterization and classification of malware using deep learning," in *2018 Resilience Week (RWS)*. IEEE, 2018, pp. 77–83.

[205] M. Bailey *et al.*, "Automated classification and analysis of internet malware," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2007, pp. 178–197.

[206] K. Rieck *et al.*, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

[207] I. K. Cho *et al.*, "Malware similarity analysis using API sequence alignments." *J. Internet Serv. Inf. Secur.*, vol. 4, no. 4, pp. 103–114, 2014.

[208] G. Wagener, A. Dulaunoy *et al.*, "Malware behaviour analysis," *Journal in computer virology*, vol. 4, no. 4, pp. 279–287, 2008.

[209] V. Guralnik and J. Srivastava, "Event detection from time series data," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1999, pp. 33–42.

[210] Y. Kawahara and M. Sugiyama, "Change-point detection in time-series data by direct density-ratio estimation," in *Proceedings of the 2009 SIAM International Conference on Data Mining*. SIAM, 2009, pp. 389–400.

[211] S. Liu *et al.*, "Change-point detection in time-series data by relative density-ratio estimation," *Neural Networks*, vol. 43, pp. 72–83, 2013.

[212] V. Jandhyala *et al.*, "Inference for single and multiple change-points in time series," *Journal of Time Series Analysis*, vol. 34, no. 4, pp. 423–446, 2013.

[213] A. G. Tartakovsky *et al.*, "A novel approach to detection of intrusions in computer networks via adaptive sequential and batch-sequential change-point detection methods," *IEEE Transactions on Signal Processing*, vol. 54, no. 9, pp. 3372–3382, 2006.

[214] A. G. Tartakovsky, A. S. Polunchenko, and G. Sokolov, "Efficient computer network anomaly detection by changepoint detection methods," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 1, pp. 4–11, 2012.

[215] C. Truong, L. Oudre, and N. Vayatis, "ruptures: change point detection in python," *arXiv preprint arXiv:1801.00826*, 2018.

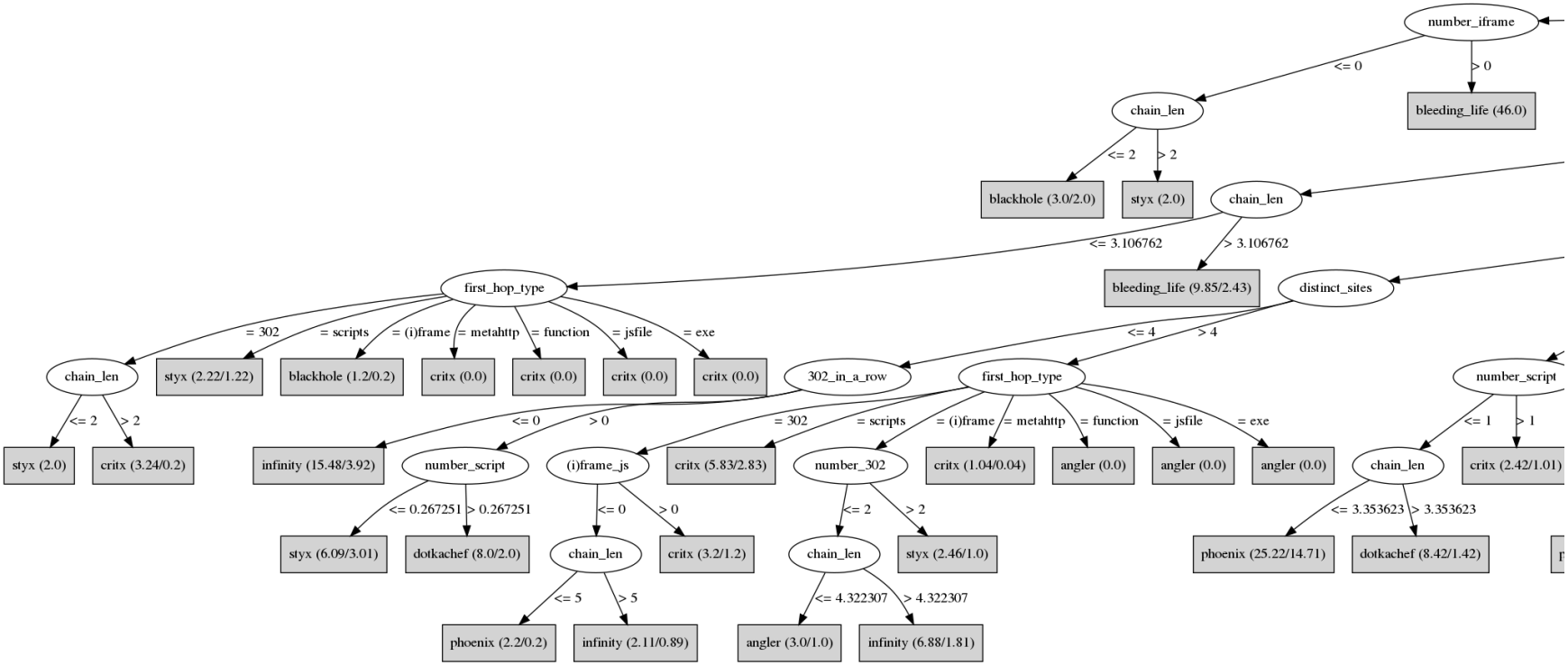
[216] W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

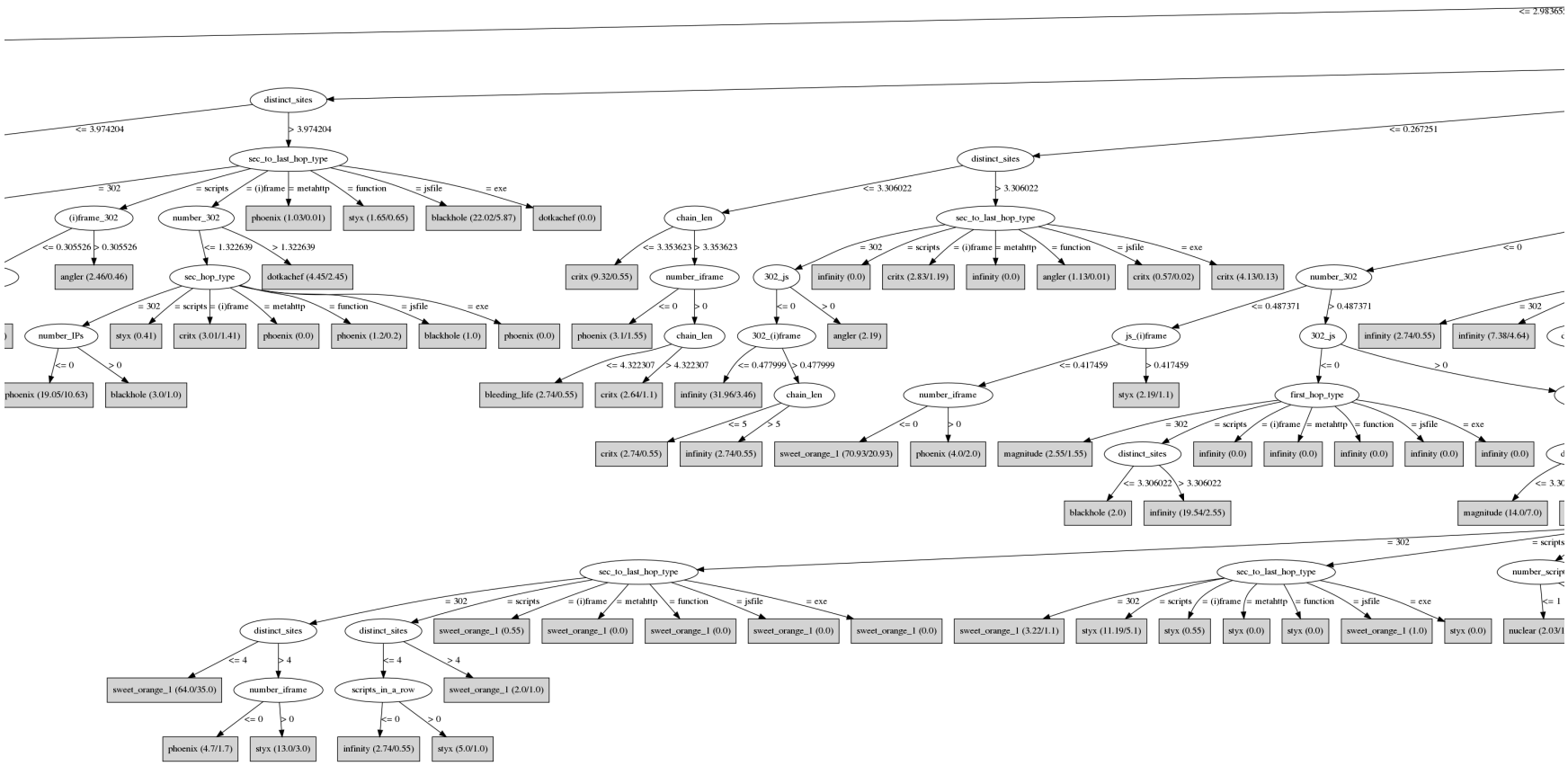
[217] "GoSecure/malboxes," Nov. 2019, original-date: 2016-04-01T17:50:36Z. [Online]. Available: https://github.com/GoSecure/malboxes

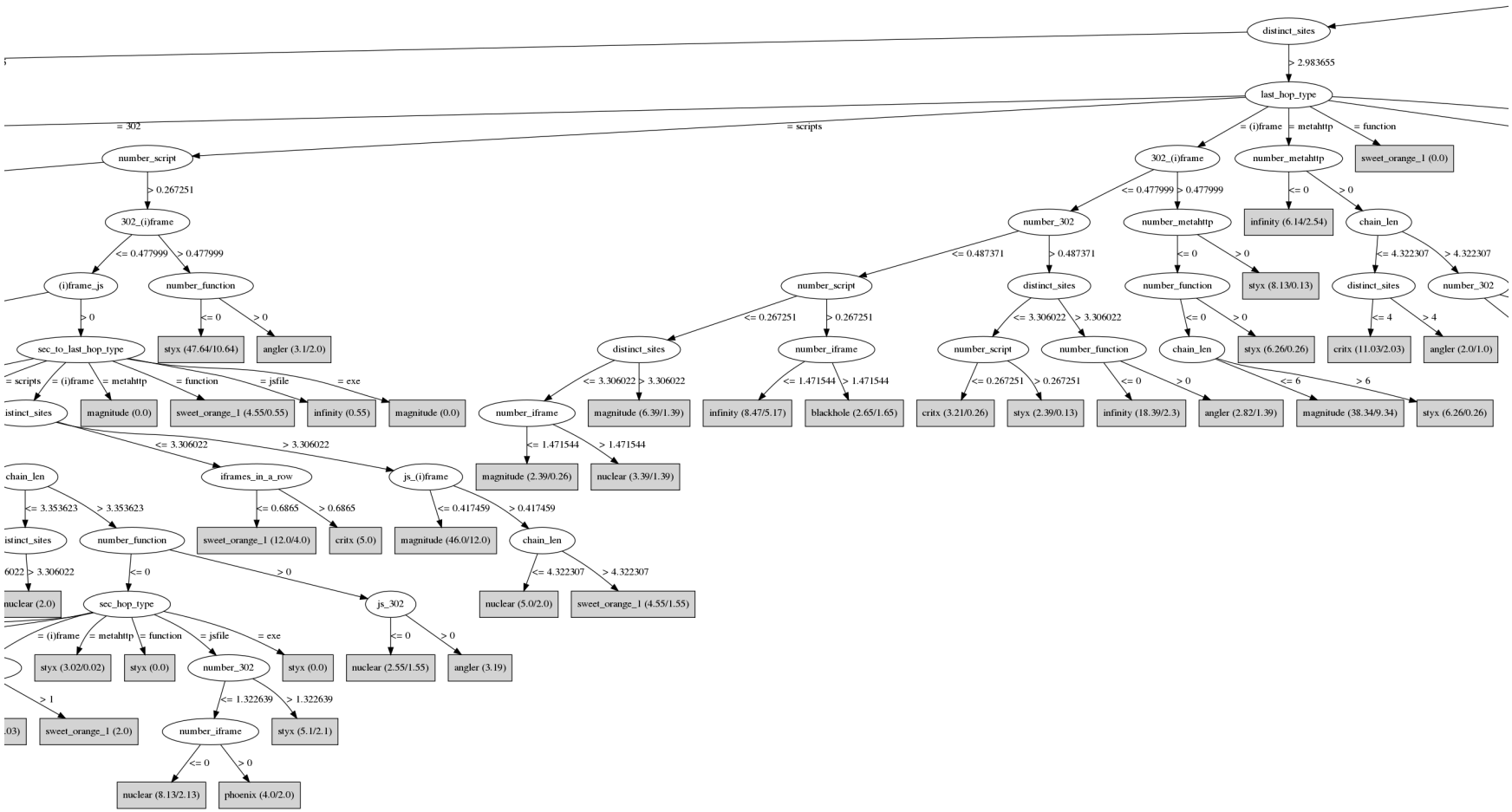
[218] M. Faou *et al.*, "Follow the traffic: Stopping click fraud by disrupting the value chain," in *2016 14th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 2016, pp. 464–476.

APPENDIX A EXPLOIT KIT PREDICTION MODEL DECISION TREE

Chapter 5 presented our approach at classifying exploit kit families by the patterns in the web redirection chain leading to the exploit kit. In Section 5.4, we showed the accuracy and a number of other metrics our decision tree classifier obtained at classifying exploit kit families, both with the date feature and without it. Here we show, in Figure A.1, the complete decision tree built by the C4.5 algorithm, and the respective branching conditions, when building the model without the date. Figure A.2 presents the model when built with the addition of the date feature. It is worth noting that the date feature has a large impact on the resulting decision tree, since it is chosen as a top branching feature for classifying exploit kits.


number_iframe
<= 0
> 0
chain_len
bleeding_life (46.0)
<= 2
> 2
blackhole (3.0/2.0)
styx (2.0)
chain_len
<= 3.106762
> 3.106762
first_hop_type
bleeding_life (9.85/2.43)
distinct_sites
= 302
= scripts
= (i)frame
= metahttp
= function
= jsfile
= exe
<= 4
> 4
chain_len
styx (2.22/1.22)
blackhole (1.2/0.2)
critx (0.0)
critx (0.0)
critx (0.0)
critx (0.0)
302_in_a_row
first_hop_type
number_script
<= 2
> 2
<= 0
> 0
= 302
= scripts
= (i)frame
= metahttp
= function
= jsfile
= exe
<= 1
> 1
styx (2.0)
critx (3.24/0.2)
infinity (15.48/3.92)
number_script
(i)frame_js
critx (5.83/2.83)
number_302
critx (1.04/0.04)
angler (0.0)
angler (0.0)
angler (0.0)
chain_len
critx (2.42/1.01)
<= 0.267251
> 0.267251
<= 0
> 0
<= 2
> 2
<= 3.353623
> 3.353623
styx (6.09/3.01)
dotkachef (8.0/2.0)
chain_len
critx (3.2/1.2)
chain_len
styx (2.46/1.0)
phoenix (25.22/14.71)
dotkachef (8.42/1.42)
<= 5
> 5
<= 4.322307
> 4.322307
phoenix (2.2/0.2)
infinity (2.11/0.89)
angler (3.0/1.0)
infinity (6.88/1.81)


distinct_sites
<= 3.974204
> 3.974204
sec_to_last_hop_type
= 302
= scripts
= (i)frame
= metahttp
= function
= jsfile
= exe
(i)frame_302
number_302
phoenix (1.03/0.01)
styx (1.65/0.65)
blackhole (22.02/5.87)
dotkachef (0.0)
<= 0.305526
> 0.305526
angler (2.46/0.46)
sec_hop_type
<= 1.322639
> 1.322639
dotkachef (4.45/2.45)
number_IPs
styx (0.41)
critx (3.01/1.41)
phoenix (0.0)
phoenix (1.2/0.2)
blackhole (1.0)
phoenix (0.0)
<= 0
> 0
phoenix (19.05/10.63)
blackhole (3.0/1.0)
<= 0.267251
distinct_sites
<= 3.306022
> 3.306022
chain_len
sec_to_last_hop_type
<= 3.353623
> 3.353623
critx (9.32/0.55)
number_iframe
infinity (0.0)
critx (2.83/1.19)
infinity (0.0)
angler (1.13/0.01)
critx (0.57/0.02)
critx (4.13/0.13)
<= 0
> 0
phoenix (3.1/1.55)
chain_len
302_js
302_(i)frame
angler (2.19)
<= 4.322307
> 4.322307
bleeding_life (2.74/0.55)
critx (2.64/1.1)
infinity (31.96/3.46)
chain_len
<= 0.477999
> 0.477999
critx (2.74/0.55)
infinity (2.74/0.55)
sweet_orange_1 (70.93/20.93)
number_302
<= 0.487371
> 0.487371
js_(i)frame
302_js
infinity (2.74/0.55)
infinity (7.38/4.64)
<= 0.417459
> 0.417459
number_iframe
styx (2.19/1.1)
<= 0
> 0
phoenix (4.0/2.0)
first_hop_type
<= 0
> 0
magnitude (2.55/1.55)
distinct_sites
infinity (0.0)
infinity (0.0)
infinity (0.0)
infinity (0.0)
infinity (0.0)
<= 3.306022
> 3.306022
blackhole (2.0)
infinity (19.54/2.55)
magnitude (14.0/7.0)
= 302
= scripts
sec_to_last_hop_type
distinct_sites
distinct_sites
sweet_orange_1 (0.55)
sweet_orange_1 (0.0)
sweet_orange_1 (0.0)
sweet_orange_1 (0.0)
sweet_orange_1 (0.0)
<= 4
> 4
sweet_orange_1 (64.0/35.0)
number_iframe
scripts_in_a_row
sweet_orange_1 (2.0/1.0)
<= 0
> 0
phoenix (4.7/1.7)
styx (13.0/3.0)
infinity (2.74/0.55)
styx (5.0/1.0)
sweet_orange_1 (3.22/1.1)
styx (11.19/5.1)
styx (0.55)
styx (0.0)
styx (0.0)
sweet_orange_1 (1.0)
styx (0.0)
nuclear (2.03/1


distinct_sites
> 2.983655
last_hop_type
= 302
= scripts
= (i)frame
= metahttp
= function
number_script
> 0.267251
302_(i)frame
<= 0.477999
> 0.477999
(i)frame_js
number_function
> 0
<= 0
> 0
sec_to_last_hop_type
styx (47.64/10.64)
angler (3.1/2.0)
= scripts
= (i)frame
= metahttp
= function
= jsfile
= exe
istinct_sites
magnitude (0.0)
sweet_orange_1 (4.55/0.55)
infinity (0.55)
magnitude (0.0)
<= 3.306022
> 3.306022
chain_len
iframes_in_a_row
js_(i)frame
<= 3.353623
> 3.353623
<= 0.6865
> 0.6865
<= 0.417459
> 0.417459
istinct_sites
number_function
sweet_orange_1 (12.0/4.0)
critx (5.0)
magnitude (46.0/12.0)
chain_len
6022
> 3.306022
<= 0
> 0
<= 4.322307
> 4.322307
nuclear (2.0)
sec_hop_type
js_302
nuclear (5.0/2.0)
sweet_orange_1 (4.55/1.55)
= (i)frame
= metahttp
= function
= jsfile
= exe
<= 0
> 0
styx (3.02/0.02)
styx (0.0)
number_302
styx (0.0)
nuclear (2.55/1.55)
angler (3.19)
> 1
<= 1.322639
> 1.322639
.03)
sweet_orange_1 (2.0)
number_iframe
styx (5.1/2.1)
<= 0
> 0
nuclear (8.13/2.13)
phoenix (4.0/2.0)
number_iframe
<= 1.471544
> 1.471544
magnitude (2.39/0.26)
nuclear (3.39/1.39)
distinct_sites
<= 0.267251
> 0.267251
<= 3.306022
> 3.306022
number_iframe
magnitude (6.39/1.39)
<= 1.471544
> 1.471544
infinity (8.47/5.17)
blackhole (2.65/1.65)
302_(i)frame
number_metahttp
sweet_orange_1 (0.0)
<= 0.477999
> 0.477999
<= 0
> 0
number_302
number_metahttp
infinity (6.14/2.54)
chain_len
<= 0.487371
> 0.487371
<= 0
> 0
<= 4.322307
> 4.322307
number_script
distinct_sites
number_function
styx (8.13/0.13)
distinct_sites
number_302
<= 0.267251
> 0.267251
<= 3.306022
> 3.306022
<= 0
> 0
<= 4
> 4
number_script
number_function
chain_len
styx (6.26/0.26)
critx (11.03/2.03)
angler (2.0/1.0)
<= 0.267251
> 0.267251
<= 0
> 0
<= 6
> 6
critx (3.21/0.26)
styx (2.39/0.13)
infinity (18.39/2.3)
angler (2.82/1.39)
magnitude (38.34/9.34)
styx (6.26/0.26)

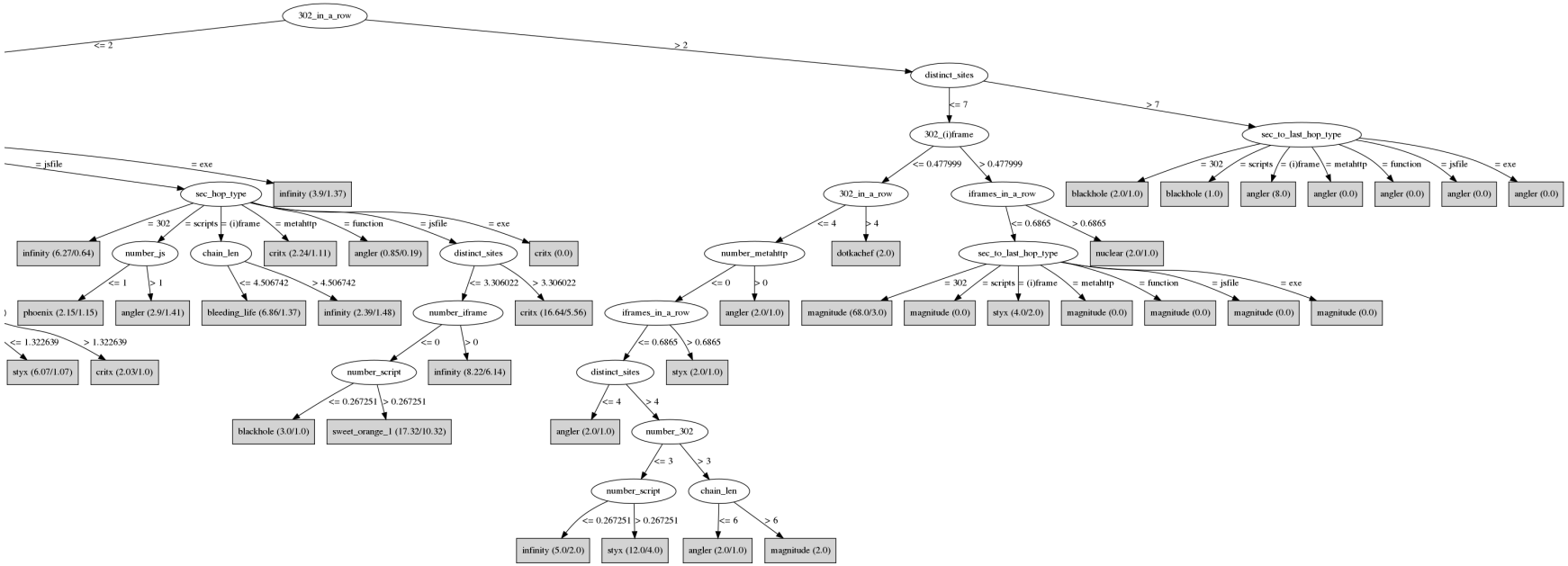
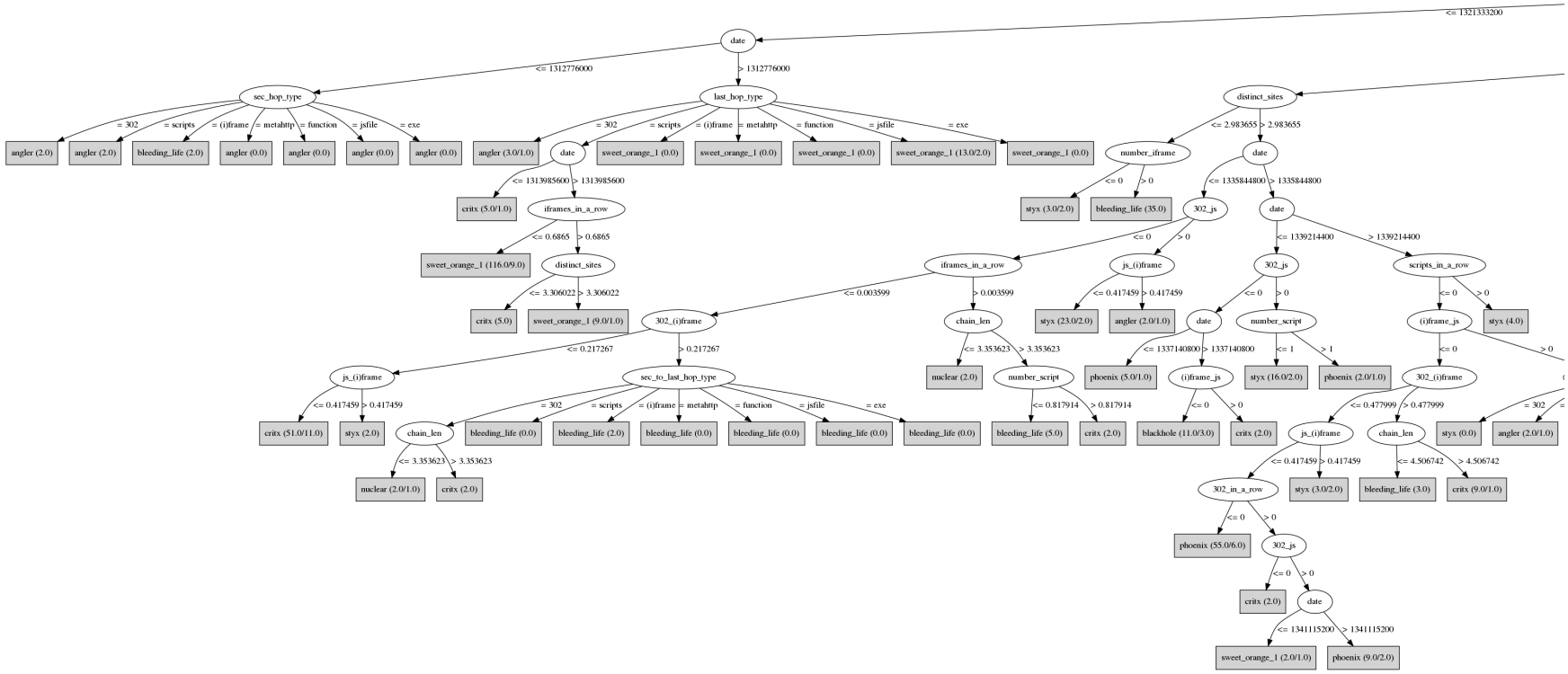
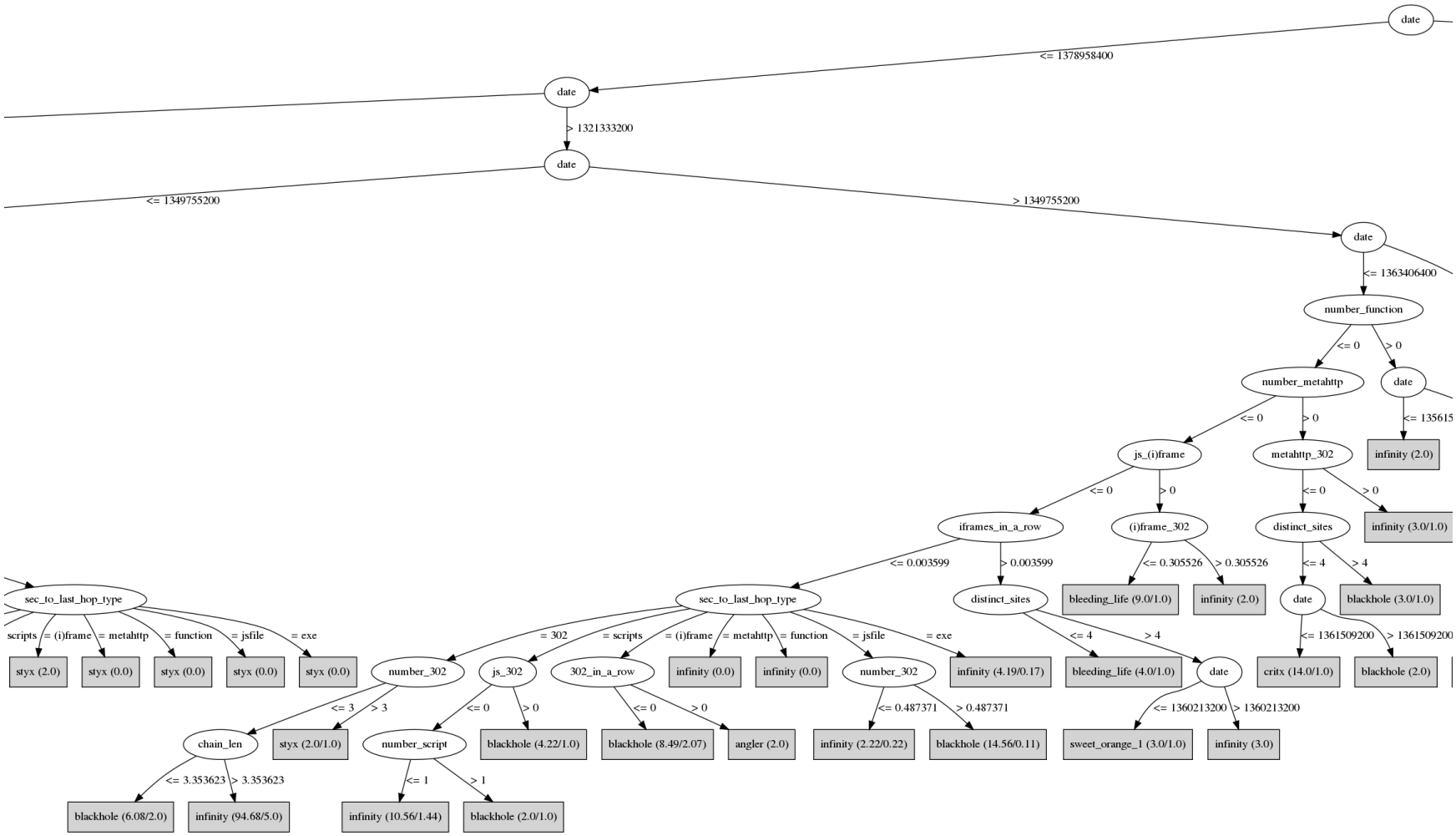
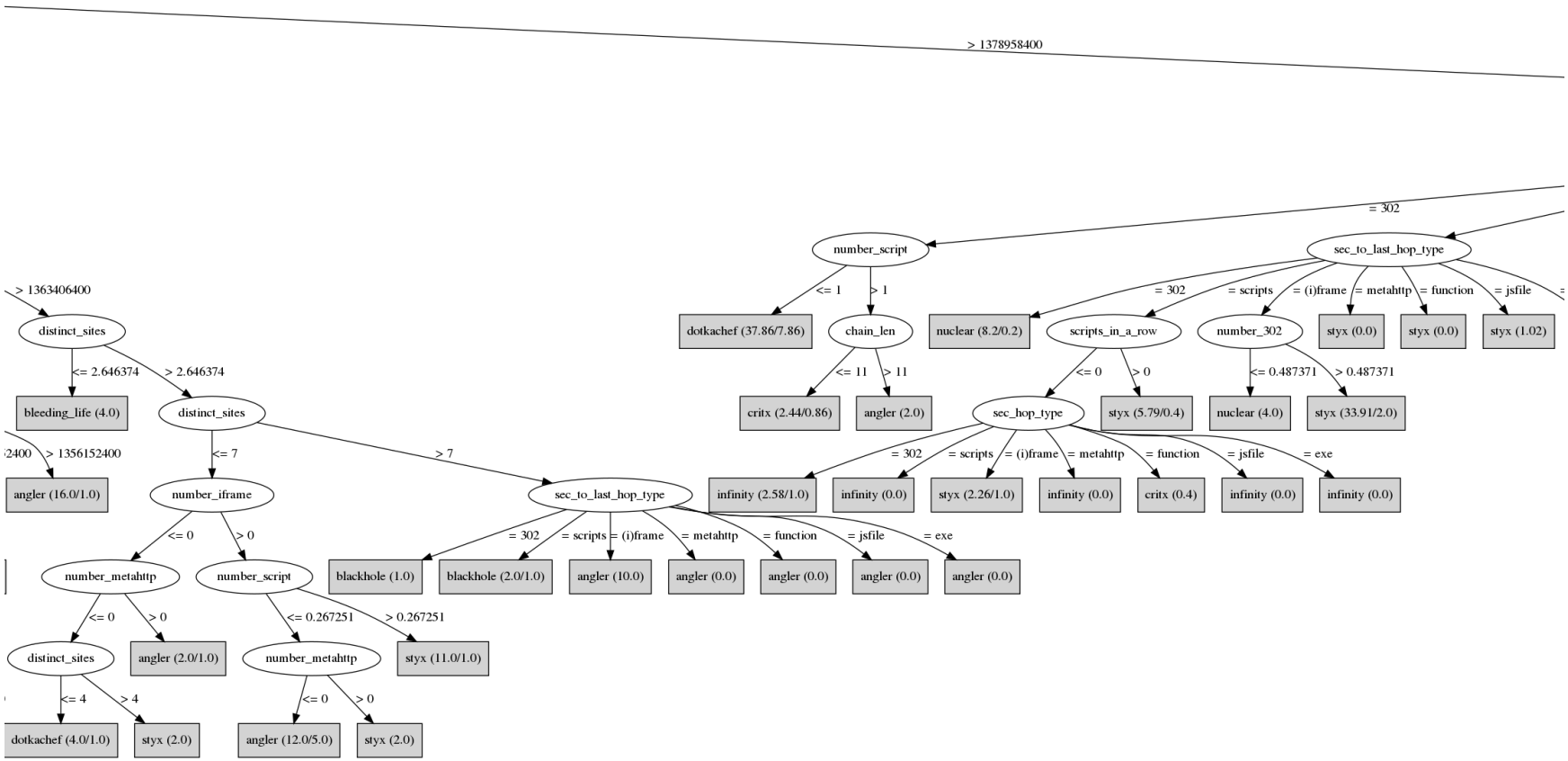


Figure A.1 The decision tree of our exploit kit family prediction model, without the date feature


date
<= 1321333200
<= 1312776000
> 1312776000
sec_hop_type
= 302
= scripts
= (i)frame
= metahttp
= function
= jsfile
= exe
angler (2.0)
angler (2.0)
bleeding_life (2.0)
angler (0.0)
angler (0.0)
angler (0.0)
angler (0.0)
last_hop_type
= 302
= scripts
= (i)frame
= metahttp
= function
= jsfile
= exe
angler (3.0/1.0)
date
sweet_orange_1 (0.0)
sweet_orange_1 (0.0)
sweet_orange_1 (0.0)
sweet_orange_1 (13.0/2.0)
sweet_orange_1 (0.0)
<= 1313985600
> 1313985600
critx (5.0/1.0)
iframes_in_a_row
<= 0.6865
> 0.6865
sweet_orange_1 (116.0/9.0)
distinct_sites
<= 3.306022
> 3.306022
critx (5.0)
sweet_orange_1 (9.0/1.0)
distinct_sites
<= 2.983655
> 2.983655
number_iframe
<= 0
> 0
styx (3.0/2.0)
bleeding_life (35.0)
date
<= 1335844800
> 1335844800
302_js
<= 0
> 0
iframes_in_a_row
<= 0.003599
> 0.003599
302_(i)frame
<= 0.217267
> 0.217267
js_(i)frame
<= 0.417459
> 0.417459
critx (51.0/11.0)
styx (2.0)
sec_to_last_hop_type
= 302
= scripts
= (i)frame
= metahttp
= function
= jsfile
= exe
chain_len
bleeding_life (0.0)
bleeding_life (2.0)
bleeding_life (0.0)
bleeding_life (0.0)
bleeding_life (0.0)
bleeding_life (0.0)
<= 3.353623
> 3.353623
nuclear (2.0/1.0)
critx (2.0)
chain_len
<= 3.353623
> 3.353623
nuclear (2.0)
number_script
<= 0.817914
> 0.817914
bleeding_life (5.0)
critx (2.0)
js_(i)frame
<= 0.417459
> 0.417459
styx (23.0/2.0)
angler (2.0/1.0)
date
<= 1337140800
> 1337140800
phoenix (5.0/1.0)
(i)frame_js
<= 0
> 0
blackhole (11.0/3.0)
critx (2.0)
302_js
<= 0
> 0
number_script
<= 1
> 1
styx (16.0/2.0)
phoenix (2.0/1.0)
date
<= 1339214400
> 1339214400
scripts_in_a_row
<= 0
> 0
(i)frame_js
<= 0
> 0
styx (4.0)
302_(i)frame
<= 0.477999
> 0.477999
js_(i)frame
<= 0.417459
> 0.417459
302_in_a_row
<= 0
> 0
phoenix (55.0/6.0)
302_js
<= 0
> 0
critx (2.0)
date
<= 1341115200
> 1341115200
sweet_orange_1 (2.0/1.0)
phoenix (9.0/2.0)
styx (3.0/2.0)
chain_len
<= 4.506742
> 4.506742
bleeding_life (3.0)
critx (9.0/1.0)
= 302
styx (0.0)
angler (2.0/1.0)


date
<= 1378958400
date
> 1321333200
date
<= 1349755200
> 1349755200
date
<= 1363406400
number_function
<= 0
> 0
number_metahttp
date
<= 0
> 0
<= 135615
js_(i)frame
metahttp_302
infinity (2.0)
<= 0
> 0
<= 0
> 0
iframes_in_a_row
(i)frame_302
distinct_sites
infinity (3.0/1.0)
<= 0.003599
> 0.003599
<= 0.305526
> 0.305526
<= 4
> 4
sec_to_last_hop_type
distinct_sites
bleeding_life (9.0/1.0)
infinity (2.0)
date
blackhole (3.0/1.0)
scripts
= (i)frame
= metahttp
= function
= jsfile
= exe
= 302
= scripts
= (i)frame
= metahttp
= function
= jsfile
= exe
<= 4
> 4
<= 1361509200
> 1361509200
styx (2.0)
styx (0.0)
styx (0.0)
styx (0.0)
styx (0.0)
number_302
js_302
302_in_a_row
infinity (0.0)
infinity (0.0)
number_302
infinity (4.19/0.17)
bleeding_life (4.0/1.0)
date
critx (14.0/1.0)
blackhole (2.0)
<= 3
> 3
<= 0
> 0
<= 0
> 0
<= 0.487371
> 0.487371
<= 1360213200
> 1360213200
chain_len
styx (2.0/1.0)
number_script
blackhole (4.22/1.0)
blackhole (8.49/2.07)
angler (2.0)
infinity (2.22/0.22)
blackhole (14.56/0.11)
sweet_orange_1 (3.0/1.0)
infinity (3.0)
<= 3.353623
> 3.353623
<= 1
> 1
blackhole (6.08/2.0)
infinity (94.68/5.0)
infinity (10.56/1.44)
blackhole (2.0/1.0)


> 1378958400
= 302
number_script
sec_to_last_hop_type
> 1363406400
distinct_sites
<= 1
> 1
= 302
= scripts
= (i)frame
= metahttp
= function
= jsfile
dotkachef (37.86/7.86)
chain_len
nuclear (8.2/0.2)
scripts_in_a_row
number_302
styx (0.0)
styx (0.0)
styx (1.02)
<= 2.646374
> 2.646374
<= 11
> 11
<= 0
> 0
<= 0.487371
> 0.487371
bleeding_life (4.0)
distinct_sites
critx (2.44/0.86)
angler (2.0)
sec_hop_type
styx (5.79/0.4)
nuclear (4.0)
styx (33.91/2.0)
2400
> 1356152400
<= 7
> 7
= 302
= scripts
= (i)frame
= metahttp
= function
= jsfile
= exe
angler (16.0/1.0)
number_iframe
sec_to_last_hop_type
infinity (2.58/1.0)
infinity (0.0)
styx (2.26/1.0)
infinity (0.0)
critx (0.4)
infinity (0.0)
infinity (0.0)
<= 0
> 0
= 302
= scripts
= (i)frame
= metahttp
= function
= jsfile
= exe
number_metahttp
number_script
blackhole (1.0)
blackhole (2.0/1.0)
angler (10.0)
angler (0.0)
angler (0.0)
angler (0.0)
angler (0.0)
<= 0
> 0
<= 0.267251
> 0.267251
distinct_sites
angler (2.0/1.0)
number_metahttp
styx (11.0/1.0)
<= 4
> 4
<= 0
> 0
dotkachef (4.0/1.0)
styx (2.0)
angler (12.0/5.0)
styx (2.0)

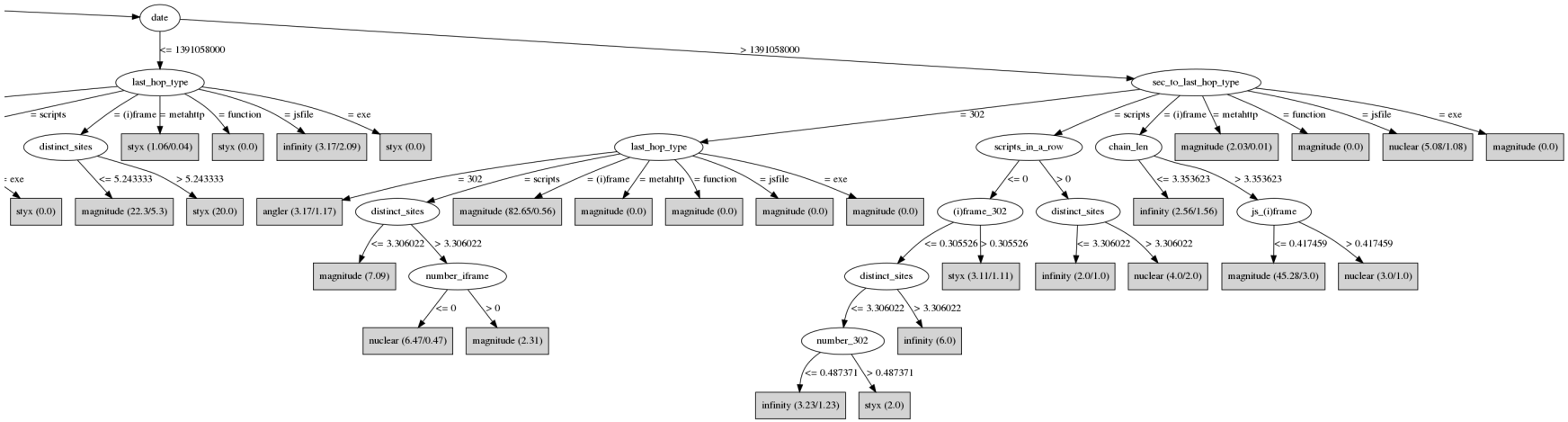
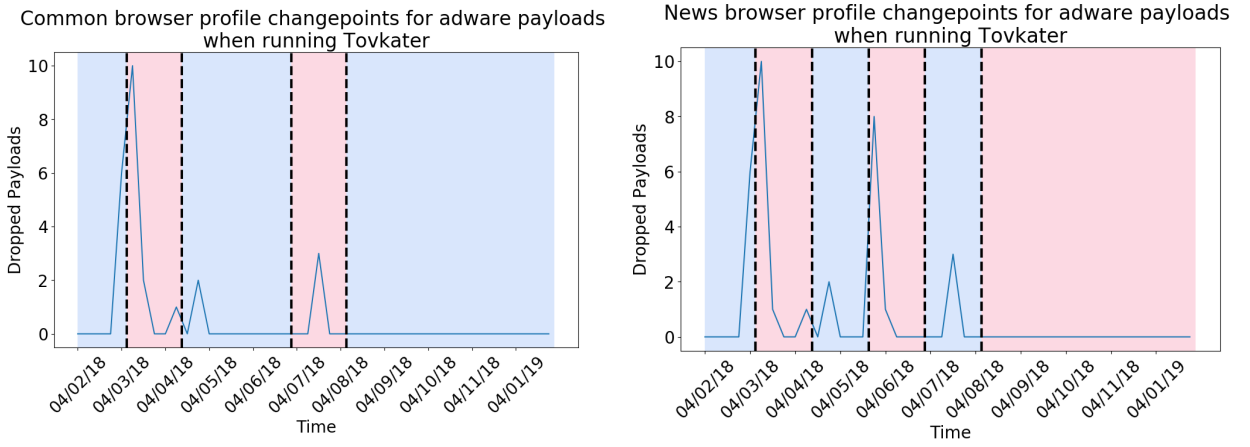


Figure A.2 The decision tree of our exploit kit family prediction model, with the added date feature

APPENDIX B CANGEPOINT ANALYSIS VISUAL OUTPUT

In establishing the different behaviors of malicious downloaders in Chapter 6, we employed changepoint analysis in order to identify changes in malicious infections through time for different features of the machine running the downloader. Our analysis in Section 6.6.3 cited multiple changepoints for our various features. We used the Ruptures Python library [215] to identify our changepoints and to produce the visual representations. Here we show these changepoints over the plots of our results. The changepoints are identified by a dotted vertical line over the plot of a result, and the background color around a changepoint alternates between red and blue to highlight the change.

Figure B.1a identifies the changepoints over our overall browser profiles for adware payloads when running the Tovkater downloader, while Figure B.1b shows an additional changepoint in May of 2018 detected for the *News* browser profile.

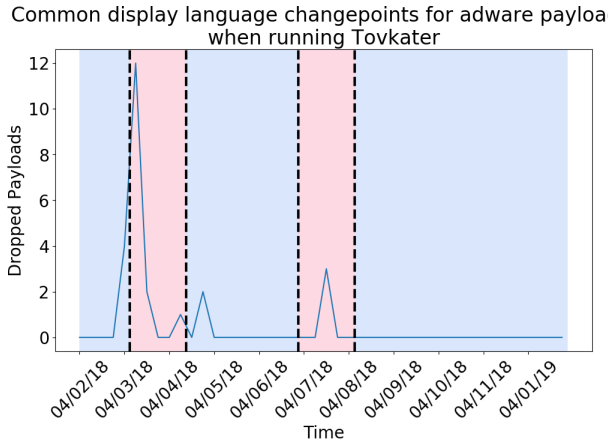


(a) Changepoints identified for all browser profiles

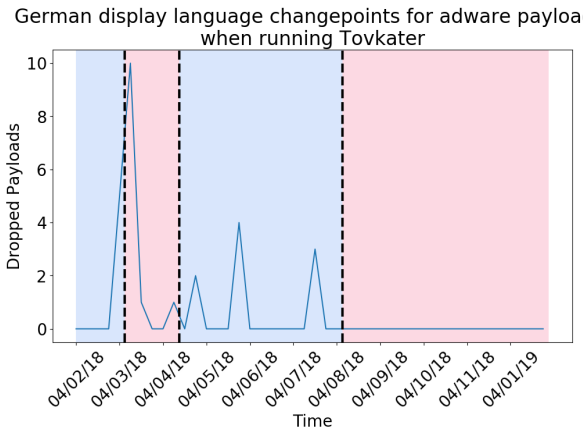
(b) Changepoints identified for the news browser profile

Figure B.1 Changepoints identified when using the News browser profile

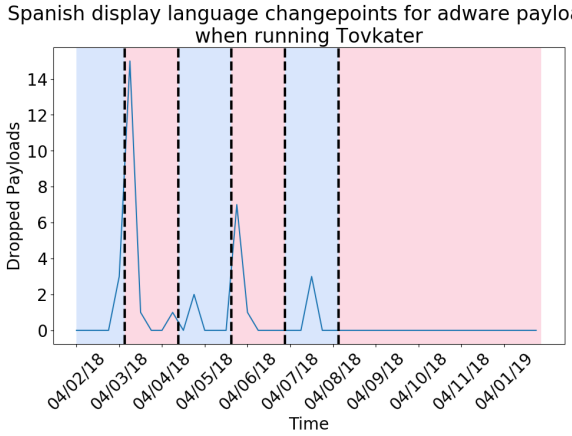
When testing the display language feature with Tovkater samples, we obtained four changepoints on most of our features with an adware payload, as can be observed in Figure B.2a. We obtained one less changepoint when testing with the German language, as shown in Figure B.2b, while we obtained an additional changepoint in May when executing malicious downloaders with a Spanish display language, as can be seen in Figure B.2c.



(a) Changepoints identified for all display languages except German



(b) Changepoints identified for the German display language

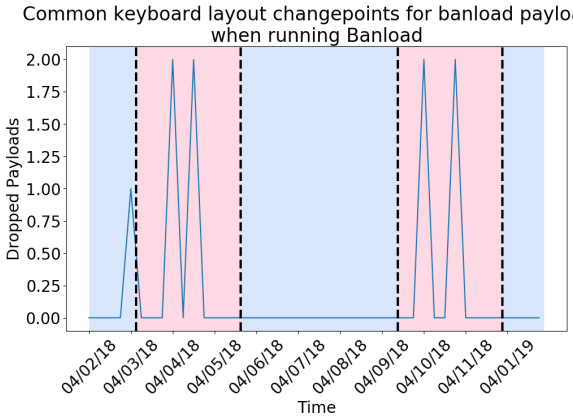


(c) Changepoints identified for the Spanish display language

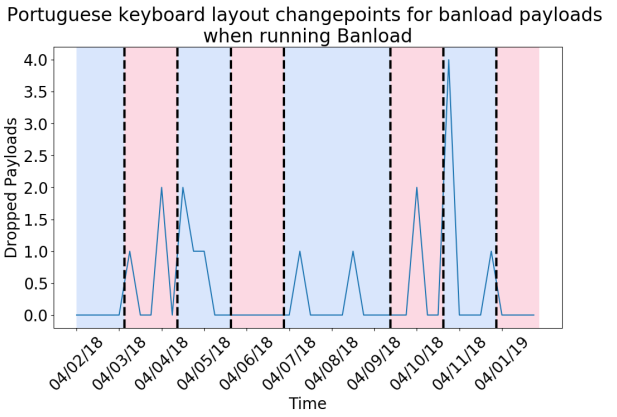
Figure B.2 Changepoints identified when using different display languages

As highlighted by our resuts in Section 6.6.3, we also obtained different changepoints with different keyboard layouts, specifically when running a Banload downloader and obtaining a Banload payload. Figure B.3a shows the various changepoints identified with most keyboard layouts. The portuguese keyboard layout has two additional changepoints identified, as can be observed in Figure B.3b.

The last tested feature which had a changing behavior from downloaders is the location of the VM, as set by our VPN. We identified four changepoints for all our locations when obtaining an adware payload from Tovkater downloaders, as can be seen in Figure B.4a. Our VPN in Mexico is the only location which obtained an additional changepoint, which can be observed in Figure B.4b.

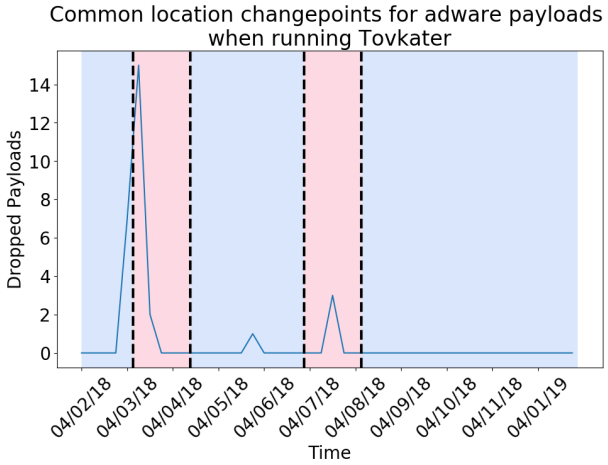


(a) Changepoints identified for most keyboard layouts

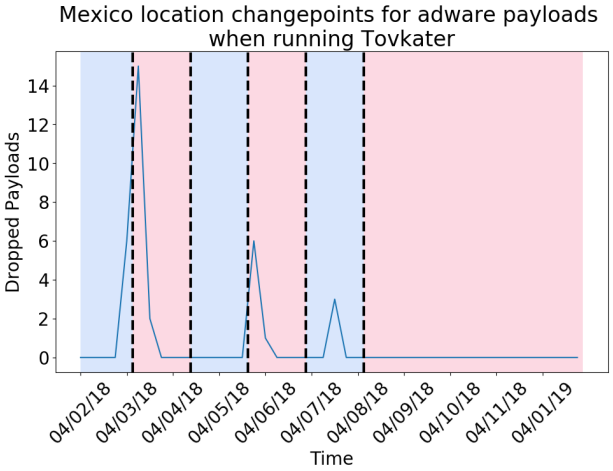


(b) Changepoints identified for the Portuguese keyboard layout

Figure B.3 Changepoints identified when using different keyboard layouts



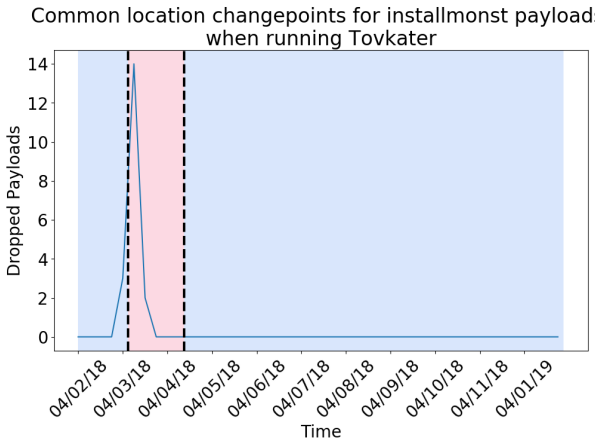
(a) Changepoints identified for all locations



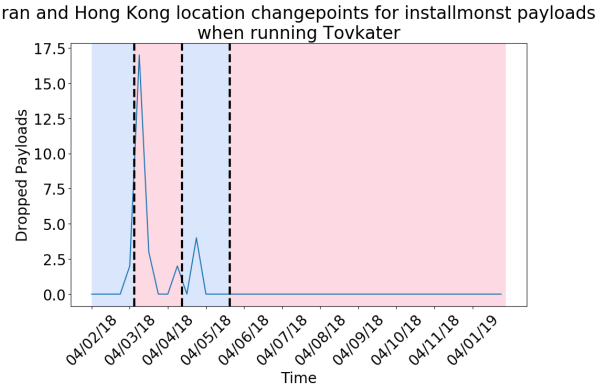
(b) Changepoints identified with the Mexico VPN

Figure B.4 Changepoints identified for Adware payloads when using different VPNs

Different changepoints when also identified when obtaining InstallMonster payloads. Figure B.5a highlights the changepoints for all our locations, while Figure B.5b shows an additional changepoint due to increased infections in Iran and Hong Kong.



(a) Changepoints identified for all locations



(b) Changepoints identified with the Iran and Hong Kong VPN

Figure B.5 Changepoints identified for InstallMonster payloads when using different VPNs