UNIVERSITY OF MODENA AND REGGIO EMILIA

DOCTORAL THESIS

---

# Security Analytics and Machine Learning for Cyber Detection: Modern Issues and Novel Solutions

---

*Author:*
Ing. Giovanni APRUZZESE

*Supervisor:*
Prof. Michele COLAJANNI

*PhD Course Coordinator:*
Prof. Sonia BERGAMASCHI

*A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy*

*in the*

International Doctorate in **Information and Communication Technologies**
*Computer Engineering and Science*
Cycle XXXII

UNIVERSITY OF MODENA AND REGGIO EMILIA

# *Abstract*

International Doctorate in **Information and Communication Technologies**
*Computer Engineering and Science*
Cycle XXXII
Department of Engineering "Enzo Ferrari"

Doctoral Thesis

**Security Analytics and Machine Learning for Cyber Detection: Modern Issues and Novel Solutions**

by Giovanni APRUZZESE

Efficient detection of advanced cyber attacks is a complex problem that presents multiple issues and challenges. Skilled attackers are constantly improving their tools and, by adopting original strategies, are able to evade the detection of traditional rule-based approaches. As a consequence, large amounts of data breaches remain undetected for months, causing severe damage to organizations. Humans alone cannot efficiently deal with the increasing velocity, complexity and variety of modern threats. To address these critical menaces, as evidenced by both scientific literature and practical reports, cybersecurity analysts need to be supported with forms of automatic detection mechanisms that exploit the huge volume of data generated by modern systems and networks. This thesis promotes and improves this conviction by leveraging security analytics through machine learning models and mathematical algorithms. We present original solutions for cyber detection of popular threats related to botnets, lateral movements, malicious periodic communications and phishing. We also study the problems affecting these approaches in cybersecurity contexts where solutions are not as straightforward as expected and the balance between true and false detection remains an open issue. In the second part of the thesis, we consider the problem of adversarial attacks against cyber detectors, and we present original solutions to mitigate similar threats. The proposed methods require minimal amounts of information and few assumptions, thus enabling their integration in real defensive frameworks of large enterprises. An important value characterizing the entire thesis is that all the proposed ideas and approaches are implemented and evaluated through experimental campaigns involving real datasets. The presented results improve the state-of-the-art and, in some cases, solve the detection problems. For these reasons, we can conclude that this thesis paves the way to new defensive systems that can support cyber analysts in detecting advanced forms of attacks in several scenarios.

**Keywords: Cybersecurity, Intrusion detection, Security analytics, Machine learning, Adversarial attacks**

UNIVERSITÀ DI MODENA E REGGIO EMILIA

# *Abstract (italiano)*

Scuola di Dottorato in Information and Communication Technologies
Dipartimento di Ingegneria "Enzo Ferrari"

Tesi di Dottorato di Ricerca

**Security Analytics e Machine Learning per la Cyber Detection: Problematiche Moderne e Soluzioni Innovative**

di Giovanni APRUZZESE

La rilevazione efficace dei cyber-attacchi avanzati è un problema complesso che presenta numerose problematiche e sfide. Gli attaccanti più esperti migliorano continuamente i propri strumenti e, attraverso l'attuazione di strategie originali, sono in grado di eludere la rilevazione degli approcci tradizionali basati su regole statiche. Di conseguenza, molte data-breach richiedono mesi prima di essere identificate, provocando ingenti danni alle organizzazioni moderne. Gli operatori umani da soli non sono in grado di gestire il continuo aumento della complessità, varietà e velocità delle minacce recenti. Per risolvere questo problema, come evidenziato sia dalla letteratura scientifica che da appositi report tecnici, gli analisti della sicurezza devono essere supportati da meccanismi di rilevazione aumatici che possano sfruttare le grandi quantità di dati generati dalle reti moderne. Questa tesi promuove e incentiva questa posizione, proponendo tecniche di security analytics che adottano modelli di machine learning e algoritmi matematici. In particolare, vengono presentate soluzioni originali per la cyber detection di minacce diffuse quali botnet, lateral movement, comunicazioni periodiche malevole, e phishing. Viene anche effettuato uno studio dei problemi che affliggono questi approcci nei contesti di cybersecurity, caratterizati da una – non apparente – difficoltà di applicazione di nuove soluzioni, a causa della difficoltà di definire la linea di separazione tra azioni malevole e legittime. Nella seconda parte di questa tesi, si considera il problema degli adversarial attack contro i cyber detector, e si presentano soluzioni originali per ridurre l'impatto di simili minacce. Tutti i metodi proposti richiedono un ridotto quantitativo di informazioni e si basano su assunzioni essenziali, consentendone l'integrazione nei framework di difesa adottati dalle organizzazioni reali. Un valore importante che caratterizza l'intera tesi è che tutte le idee e tecniche proposte sono validate attraverso numerose campagne sperimentali effettuate su dataset realistici di grosse dimensioni. I risultati ottenuti migliorano lo stato dell'arte e, in alcuni casi, risolvono i problemi di detection. Per queste ragioni, si può affermare che la presente tesi costituisca un solido fondamento per la creazione di sistemi difensivi che siano in grado di supportare gli analisti della sicurezza anche in presenza delle forme di cyber-attacchi più all'avanguardia.

**Keywords: Cybersecurity, Intrusion detection, Security analytics, Machine learning, Adversarial attacks**

# *Acknowledgements*

Il senso liberatorio che si prova quando si inizia a scrivere la pagina dei ringraziamenti di una tesi è sempre estremamente piacevole, nonostante questa sia ormai la terza volta (e, spero, l'ultima). Al contrario delle precedenti, che erano basate su lavori molto più limitati (sia per durata che per complessità), questa tesi è frutto di un percorso iniziato oltre tre anni fa. Un percorso che ha sensibilmente cambiato la mia vita, facendomi aprire gli occhi su quello che è il mondo della ricerca scientifica. Un mondo in continuo movimento, in cui le certezze sono poche e ogni giorno può rivelare nuove sorprese – gradite o meno. Scadenze, articoli, progetti, conferenze, convegni, viaggi... difficile descrivere in poche parole tutte le esperienze – completamente nuove – che ho vissuto in questi tre anni. In mezzo a tutte le incertezze che ho avuto durante questo periodo, però, una certezza rimane solida: potessi tornare indietro, rifarei tutto esattamente nella maniera in cui lo ho fatto.
Per questo il più grande "grazie" va al mio tutor (supervisore? advisor? mentore? ancora non capisco quale sia il termine corretto), il Prof. Michele Colajanni, che mi ha invitato a far parte del suo gruppo di ricerca (il *WebLab*) nell'ormai lontano 2016. Grazie, perché le esperienze e le lezioni che ho vissuto in questo periodo non sarebbero mai state possibili senza il tuo continuo supporto.

Un enorme grazie a Fabio, mio *senpai* durante il mio primo anno di dottorato, che è stata la mia figura di riferimento nella fase iniziale di questo percorso e con cui ho prodotto alcuni dei lavori che più mi sono piaciuti. Non dimenticherò mai le sensazioni che provavo quando mi recavo ogni giorno in laboratorio per parlarti dei nuovi risultati ottenuti con i numerosi esperimenti fatti a notte fonda.

Grazie a Mirco, che nonostante i continui impegni a cui era quotidianamente soggetto riusciva comunque a trovare sempre il tempo per "aggiustare " gli articoli in fase di submission, oltre che a fornire molte delle idee che sono poi state alla base dei miei articoli di maggior successo.

Grazie a Luca, per avermi affiancato nel corso del progetto Asgard e per essersi fatto carico di tutta la parte implementativa e di integrazione del mio (quasi inguardabile) jupyter notebook.

An important thank you goes to Prof. VS Subrahmanian, for accepting me as a Visiting Student in his research group (*DSAIL*) at Dartmouth College, allowing me to witness and experience how academic research is conducted in a "top institution". The 6 months that I spent in the USA have been some of the most meaningful in my entire life, both from a professional and personal standpoint.

I also want to express my thanks to the external referees of this thesis, both for the time spent in evaluating my research efforts, and for the valuable suggestions that further improved the quality of this work.

Grazie anche a tutti i rimanenti membri del WebLab, per il supporto, le risate, le cene, la compagnia, il secret santa, i volani in testa, lo sporco sulla scrivania, l'apertura delle finestre quando fa freddo, le luci di natale a ferragosto, i furti del mio antistress: Alessandro, Andrea, Astrid, Dario, Iro, Riccardo, Mauro, Federico... questi 3 anni avrebbero avuto un altro sapore senza di voi.

Infine vanno gli "ovvi" grazie a mamma e papà. Perché ci siete. Sempre. E io lo so. E voi pure.

Giovanni APRUZZESE

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **URL** | **U**niversal **R**esource **L**ocator |
| **API** | **A**pplication **P**rogramming **I**nterface |
| **IDS** | **I**ntrusion **D**etection **S**ystem |
| **NIDS** | **N**etwork **I**ntrusion **D**etection **S**ystem |
| **ML** | **M**achine **L**earning |
| **SL** | **S**hallow **L**earning |
| **DL** | **D**eep **L**earning |
| **NOC** | **N**etwork **O**peration **C**enter |
| **SOC** | **S**ecurity **O**peration **C**enter |
| **SSH** | **S**ecure **SH**ell |
| **IP** | **I**nternet **P**rotocol |
| **ARP** | **A**ddress **R**esolution **P**rotocol |
| **COTS** | **C**ost **O**ff **T**he **S**helf |
| **MITM** | **M**an **I**n **T**he Middle |
| **LAN** | **L**ocal **A**rea **N**etwork |
| **LR** | **L**inear **R**egression |
| **RF** | **R**andom **F**orest |
| **DT** | **D**ecision **T**ree |
| **KNN** | **K**-**N**earest **N**eighbors |
| **DNN** | **D**eep **N**eural **N**etwork |
| **CNN** | **C**onvolutional (Deep) **N**eural **N**etwork |
| **NN** | **N**eural **N**etwork |
| **RNN** | **R**ecurrent **N**eural **N**etwork |
| **MLP** | **M**ulti **L**ayer **P**erceptron |
| **SVM** | **S**upport **V**ector **M**achine |
| **PDF** | **P**ortable **D**ocument **F**ormat |
| **NB** | **N**aive **B**ayes |
| **HMM** | **H**idden **M**arkov **M**odel |
| **DBN** | **D**eep **B**elief **N**etwork |
| **RBM** | **R**estricted **B**oltzmann **M**achine |
| **SAE** | **S**tacked **A**uto**E**ncoder |
| **FNN** | **F**eedforward (Deep) **N**eural **N**etwork |
| **DGA** | **D**omain **G**eneration **A**lgorithm |
| **DNS** | **D**omain **N**ame **S**ystem |
| **GAN** | **G**enerative **A**dversarial **N**etwork |
| **ReLU** | **Re**current **L**inear **U**nit |
| **DOS** | **D**enial **O**f **S**ervice |
| **SIEM** | **S**ystem **I**nformation (and) **E**vent **M**anagement |
| **MAC** | **M**edia **A**ccess **C**ontrol |
| **DBMS** | **D**ata**B**ase **M**anagement **S**ystem |
| **CPU** | **C**entral **P**rocessing **U**nit |
| **GPU** | **G**raphical **P**rocessing **U**nit |
| **RAM** | **R**ead **O**nly **M**emory |
| **NAS** | **N**etwork **A**ttached **S**torage |
| **ACF** | **A**uto**C**orrelation **F**unction |
| **DFT** | **D**igital **F**ourier **T**ransform |
| **ToS** | **T**ype **o**f **S**ervice |
| **CnC** | **C**ommand a**n**d **C**ontrol |
| **PCAP** | **P**acket **CAP**ture |
| **CSV** | **C**omma **S**eparated **V**alues |
| **IANA** | **I**nternet **A**ssigned **N**umbers **A**uthority |
| **DR** | **D**etection **R**ate |
| **SSD** | **S**olid **S**tate **D**isk |

# Summary (Italian)

Le organizzazioni moderne sono spesso bersaglio di molteplici minacce informatiche effettuate da attaccanti con diversi obiettivi, elevate capacità, e forti motivazioni. Nonostante i continui sforzi impiegati nella realizzazione di difese preventive, gli attacchi continuano ad aver successo e nessuna organizzazione può defirsi realmente al sicuro. In questo contesto, la presente tesi sposta l'attenzione dalla *prevenzione* (prevention) delle minacce alla loro *rilevazione* (detection).

L'identificazione degli attacchi avanzati è un procedimento complesso e non esistono soluzioni singole che siano in grado di risolvere questo problema. I difensori possono sfruttare i registri e gli allarmi prodotti dai dispositivi di sicurezza, ma per trasformare l'elevata e ingestibile quantità di questi dati "grezzi" in informazioni utili è necessario l'impiego di tecnologie di *big data analytics*. Non è quindi un caso che il paradigma del *machine learning* stia venendo sempre più adottato anche per compiti inerenti la sicurezza informatica, con lo scopo di supportare o anche sostituire gli operatori umani. Infatti il machine learning è impiegato con successo in molti domini a causa dei suoi vantaggi rispetto ai sistemi tradizionali basati su regole statiche. Nonostante la totale automazione delle procedure di identificazione resti un obiettivo molto ambizioso, bisogna tenere presente che la reale efficacia degli approcci di machine learning in contesti di cybersecurity necessita ancora di approfondite valutazioni.

Tra le molteplici problematiche che affliggono i sistemi di sicurezza basati sul machine learning vi è la loro vulnerabilità ai cosiddetti *adversarial attacks*, che puntano l'algoritmo di detection a produrre output favorevoli all'attaccante mediante l'impiego di specifici samples debitamente modificati. Questo fenomeno è stato ampiamente studiato nel settore della visione artificiale, ma in ambito della cybersecurity le analisi sono limitate e le soluzioni esistenti ancora immature.

Questa tesi si focalizza su questi problemi e propone soluzioni innovative per la cyber detection. I contributi principali di questa tesi si possono riassumere in:

- analisi e miglioramento dello stato dell'arte della security analytics basata sul machine learning;
- analisi approfondita di adversarial attack contro i cyber detector, e sviluppo di difese originali.

In particolare, nella prima parte di questo lavoro verrà fornita una panoramica dello stato dell'arte del machine learning per la cyber detection, e attraverso un'accurata analisi sperimentale verrà valutata la maturità delle soluzioni basate su queste tecniche. Verranno altresì proposte soluzioni innovative per la cyber detection basate sulla security analytics, con il duplice obiettivo di proteggersi contro attacchi avanzati, e ridurre l'impatto degli alti tassi di falsi positivi a cui queste metodologie sono soggette. Quindi, nella seconda parte della tesi, verrà preso in considerazione il problema degli adversarial attacks: attraverso molteplici esperimenti verrà valutato l'impatto di tali minacce sui cyber detector; quindi verranno proposte contromisure originali per contrastare questi pericoli, efficaci sia in contesti di attacchi nelle fasi di training dell'algoritmo, che in fasi di test. Di seguito forniremo una descrizione più dettagliata dei singoli contributi inclusi in questa tesi.

Presentiamo un'analisi, orientata agli specialisti di sicurezza, dell'applicazione delle tecniche di machine learning per la rilevazione delle intrusioni di rete, dei malware, e dello spam. L'obiettivo è duplice: verificare la maturità di questi metodi, e identificare le loro criticità in fase di produzione. Tale analisi è basata su un accurato studio dello stato dell'arte e anche attraverso esperimenti originali in contesti di reti di grandi dimensioni.

Gli strumenti di anomaly detection funzionano offline, o hanno l'obiettivo di segnalare un host come compromesso, denotando un elevato rischio di falsi allarmi e tempi di risposta non ideali per le reti moderne. Pertanto, in questa tesi proponiamo un'architettura innovativa compatibile con analisi online, il cui scopo è il monitoraggio del comportamento di ciascun host interno da più prospettive, al fine di identificare indici di potenziali azioni sospette. Tali indici saranno poi post-correlati al fine di produrre un elenco di host con un elevato tasso di rischio che sia facilmente gestibile da operatori umani.

Molti attacchi avanzati utilizzano la tecnica del "pivoting", attraverso la quale viene creato un tunnel di propagazione di comandi che interessa due o più host, al fine di inoltrare i comandi all'host obiettivo dell'attaccante. In questa tesi si descrive

il primo algoritmo di pivoting detection basato sull'analisi dei network flows; tale algoritmo viene affiancato da un ulteriore algoritmo il cui compito è il ranking delle istanze di pivoting rilevate sulla base della loro pericolosità. L'utilizzo combinato di questi approcci consentirà quindi agli analisti di sicurezza di concentrare il proprio tempo solo sui tunnel più rischiosi.

Infine, viene proposto un metodo innovativo per l'analisi tempestiva ed automatica del traffico generato da reti di grandi dimensioni, che è in grado di rilevare host esterni malevoli anche se questi non generano alcun allarme. Il nostro focus è negli host coinvolti in attività di "beaconing" (comunicazioni periodiche), che sono molto frequenti in caso di host controllati da attaccanti remoti. L'output del metodo è una graylist di poche dozzine di host, facilmente gestibile dagli analisti di sicurezza. La qualità della proposta è misurata attraverso una adeguata valutazione sperimentale in scenari di rete di migliaia di host.

Il machine learning è ampiamente utilizzato in molti contesti, ma l'applicazione di queste tecniche in ambito della cyber security deve affrontare il grave problema degli adversarial attacks. Pertanto, viene fornita un'analisi approfondita di tali minacce verso dispositivi orientati alla detection di intrusioni di rete, di malware, e di spam. Inoltre, attraverso esperimenti originali, verranno anche valutate le contromisure esistenti, evidenziandone le principali limitazioni, e motivando quindi la necessità di sviluppo di strategie di difesa innovative.

Per far fronte a questa esigenza, si propongono metodi originali per contrastare l'efficacia degli adversarial attack contro i cyber detectors, includendo sia attacchi in fase di training del modello, che quelli in fase di testing. L'efficacia dei metodi proposti è valutata attraverso esperimenti estensivi in contesti di (i) botnet e (ii) phishing detection. I risultati di tali valutazioni dimostrano come gli approcci proposti siano in grado di contrastare in maniera efficace tale minaccia, risultando allo stesso tempo non soggetti ai problemi che influenzano i meccanismi difensivi noti.

Pertanto, questa tesi si può considerare come un valido contributo allo stato dell'arte dell'applicazione del machine learning per scopi di cyber detection. Tutte le soluzioni proposte sono facilmente integrabili in sistemi di difesa reali, e molte di esse possono essere accoppiate con altri metodi di cyber detection. Eventuali estensioni del presente lavoro includono la valutazione dell'efficacia di combinazioni delle tecniche innovative proposte, e l'applicazione di tali metodi anche in contesti di sicurezza diversi come la malware detection.

# Chapter 1

# Introduction

Modern organisations are subject to a multitude of cyber threats conceived by a wide range of attackers with different goals, capabilities and motivations. Despite all the efforts spent on preventive defences, the reality is that attacks occur every day and no organisation can consider itself secure. This thesis shifts the focus from the *prevention* to the *detection* phase.

Detecting advanced attacks is increasingly complex and no single solution can work. Defenders can leverage logs and alarms produced by network and security devices, but big data analytics solutions are necessary to transform the huge volumes of raw data into useful information. Recently, the *machine learning* (ML) paradigm is being leveraged also by defensive systems, with the goal of aiding or even substituting the first level of security analysts. Machine learning is adopted in several domains due to its superiority over traditional rule-based methods. Although the complete automation of detection procedures is an enticing goal, the efficacy of machine learning in cyber security must be evaluated with the due diligence.

In this thesis we consider two open issues that affect security analytics solutions based on machine learning: the efficacy of detection in system subject to advanced cyber attacks; the vulnerability of machine learning methods against so called *adversarial attacks* that affect the detection capabilities of machine learning models. We propose original contributions to address both problems. In the former area, we propose novel methods based on prioritizing the most suspicious cyber security alarms, and the first algorithm to detect *pivoting* activities related to advanced threats. In the context of adversarial attacks against cyber detectors, we improve the state of the art by proposing robust methods that mitigate the impact of adversarial attacks at training- and test-time, and avoid the drawbacks of existing countermeasures that reduce the detection performance when applied in non-adversarial scenarios.

The research began by performing an analysis of the state of the art of machine learning techniques for cyber detection, with the purpose of identifying both their benefits and issues. Our goal is devising methods that supporting the human operators, and not replace them. Instead of promising "guaranteed" detection, we leverage security analytics to produce short-lists of hosts that are ranked on the basis of their suspicious behaviours, thus allowing security personnel to focus their attention only on the most significant threats.

We integrate this prioritization approach in an original architecture devoted to the analysis of network traffic in large enterprise networks. Since timely response is paramount in cybersecurity contexts, we make scalability and compatibility with online analyses our top priorities. The proposed architecture monitors the behavior of each internal host from multiple perspectives, detects suspicious activities possibly related to advanced attacks, and correlates these anomaly indicators to produce a list of the most likely compromised hosts that is provided to human analysts. This architecture has a flexible design, allowing its adoption to counter different advanced threats. In this context, this thesis focuses on the identification of malicious *beaconing* and *pivoting* activities.

The term beaconing denotes communications that occur periodically, which is a technique employed by attackers to maintain their control in a target network. Detecting beaconing attacks is a difficult problem: beaconing activities may be legitimate, which increases the risk of false alarms; they can occur at very different time-intervals that are difficult to analyse in large networks. We address these issues by devising a novel method for automatic and timely analysis of traffic generated by large networks, which is able to detect malicious external hosts involved in beaconing activities even when they do not raise any alert by existing defensive systems. The output is a manageable graylist of external hosts characterized by a very high likelihood of being malicious. The proposed method is evaluated in a large network scenario, where its capable of producing short-lists of few external hosts (out of the hundreds of thousands contacted daily) that perform beaconing activities with a high likelihood of maliciousness.

Pivoting attacks involve the creation of command propagation tunnels through two or more hosts in order to reach the intended target, and their identification is a complex challenge: they are a rare event, facilitating elusive attempts; and their timely

discovery is computationally demanding. We solve both of these problems by presenting the first algorithm that detects pivoting activities by means of network flows analysis. We integrate this proposal with an original prioritization algorithm allowing security analysts to investigate just the most suspicious pivoting tunnels. Extensive evaluations in realistic enterprise setting show the quality of our solution: it is able to detect all pivoting instances occurring in the monitored network, and its output includes the hosts involved in pivoting attacks among the Top 5 most suspicious threats in over 97% of the cases.

The increasing diffusion of machine and deep learning gave birth to the topic of adversarial machine learning. This research area focuses on the performance of these methods in settings where an attacker is actively trying to thwart the machine learning algorithm through the production of samples that induce an incorrect output. Literature on computer vision provides multiple use-cases of such adversarial attacks; however, researches that address this issue from a cybersecurity perspective are scarce. After analysing the state of art, we noted a gap in the context of network intrusion detection. We solve this problem by presenting a large evaluation of adversarial attacks against intrusion detectors based on multiple machine learning algorithms and spanning over different publicly available datasets. This preliminary study highlights that modern adversarial attacks are effective on machine learning classifiers for cyber detection. To aggravate this problem, our analysis also evidences that existing countermeasures to adversarial perturbations are affected by several issues: they induce a performance drop in the absence of adversarial attacks; they can be used only for specific algorithms; their integration and maintenance comes at very high costs. In this thesis, we address all of these issues. We propose countermeasures against *poisoning* and *evasion* attacks at test-time. As a practical application, we integrate the proposed solutions in botnet and phishing detectors and evaluate their effectiveness. The results show the threefold value of our proposals: they produce detectors that outperform the state of the art by improving the detection rate from 50% to 250% in adversarial settings; they can be applied to multiple classification algorithms; their application preserves the detection performance in non-adversarial scenarios, which is a critical improvement because in reality we cannot predict when an adversarial attack occurs.

This thesis is divided into six chapters, each presenting multiple contributions

to the state of the art. Chapter 2 analyzes the application of machine learning algorithms to cyber detection and highlights the related issues. Chapter 3 presents three original approaches and solutions for cyber detection based on security analytics. Chapter 4 analyzes the problem of adversarial attacks from a cybersecurity perspective, and presents the results of our focused study on network intrusion detectors in adversarial settings. Chapter 5 describes and evaluates existing countermeasures to adversarial attacks, and proposes novel defensive strategies. Chapter 6 concludes the thesis with final remarks and possible extensions for future work.

# *Part 1 – Security Analytics for Cyber Detection*

# Chapter 2

# State of the Art

The appeal and pervasiveness of machine learning (ML) is growing. Existing methods are being improved, and their ability to understand and answer real issues is highly appreciated. These achievements led to the adoption of machine learning in several domains, such as computer vision, medical analysis, gaming, and social media marketing [1], [2]. In some scenarios, machine learning techniques represent the best choice over traditional rule-based algorithms and even human operators [3]. This trend is also affecting the cybersecurity field where some detection systems are being upgraded with ML components [4]–[10]: indeed, the adoption of semi-automatic defensive techniques to support security operators is an inevitable trend because of the continuous changes and increments of both network traffic and sophistication of the attacks [4], [5], [11]. Although devising a completely automated cyber defence system is yet a distant objective, first level operators in NOC and SOC may benefit from detection and analysis tools based on machine learning. In summary, the domain of security analytics can be greatly improved through the application of machine learning algorithms.

This chapter aims (i) to assess the current maturity of these solutions; (ii) to identify their main limitations; and (iii) to highlight some room for improvement. Our study is based on an extensive review of the literature and on original experiments performed on real, large enterprises and network traffic. In the evaluation we exclude the commercial products based on machine learning (or on the abused Artificial Intelligence label) because vendors do not reveal their algorithms and tend to overlook issues and limitations. We remark that our analyses are tailored for cybersecurity personnel, who are more interested in realistic and practical applications of these novel solutions in real scenarios, rather than on their theoretical benefits. We highlight pros and cons of different methods especially in terms of false positive

or false negative alarms. Moreover, we point out a general underestimation about the complexity of managing ML architectures in cybersecurity, caused by the lack of publicly available and labelled data for training, and by the time required for fine-tuning operations in a domain characterized by continuous changes. All these issues must be known by cybersecurity operators when considering the integration of existing defensive systems with machine learning methods. The evidenced drawbacks pave the way to future improvements that ML components require before being fully adopted in cyber defence platforms.

The remainder of this chapter is structured as follows. Section 2.1 provides a thorough study of the state of the art of machine learning applied to cybersecurity, starting from a general overview of machine learning algorithms and followed by their successful applications to cyber detection. Then, Section 2.2 focuses on the analysis of the main limitations of these methods in the cybersecurity domain, which are based both on a thorough review of existing literature and on original experiments.

## 2.1 Classification of Machine Learning Algorithms for Cybersecurity

This section is based on an extensive review of existing literature on cyber detection through machine learning. We begin by presenting an original taxonomy of machine learning approaches that is oriented to security experts rather than to artificial intelligence specialists. Then, we map the identified classes of algorithms to three detection problems where machine learning found more applications: network intrusion detection, malware detection, spam and phishing email detection.

Machine learning includes a large variety of paradigms in continuous evolution, presenting weak boundaries and cross relationships. Furthermore, different views and applications may lead to different classifications. Hence, we cannot refer to one fully accepted taxonomy from literature, but we prefer to propose an original taxonomy able to capture the differences among the myriad of techniques that are being applied to cyber detection, shown in Figure 2.1. This taxonomy is specifically oriented to security operators and avoids the ambitious goal of presenting the ultimate classification that can satisfy all AI experts and application cases. The first discriminant evidenced in Figure 2.1 is between the traditional ML algorithms, which today can be referred to as **Shallow Learning (SL)**, in opposition to the more recent **Deep**

**Learning (DL)**. Shallow Learning requires a domain expert (that is, a *feature engineer*) who can perform the critical task of identifying the relevant data characteristics before executing the SL algorithm. Deep Learning relies on a multi-layered representation of the input data, and can perform feature selection autonomously through a process defined *representation learning*. We anticipate that all DL algorithms are based on Deep Neural Networks (DNN), which are large neural networks organized in many layers capable of autonomous representation learning.

SL and DL approaches can be further characterized by distinguishing between *supervised* and *unsupervised* algorithms. The former techniques require a training process with a large and representative set of data that have been previously classified by a human expert or through other means. The latter approaches do not require a pre-labeled training dataset. The ML algorithms applied to cybersecurity considered in this paper appear as the leaves of the classification tree in Figure 2.1.

### 2.1.1 Shallow Learning – supervised algorithms

- *Naïve Bayes (NB)*. These algorithms are probabilistic classifiers which make the a-priori assumption that the features of the input dataset are independent from each other. They are scalable and do not require huge training datasets to produce appreciable results. These algorithms represent a subset of *Bayesian Networks* [12], which are graph algorithms that represent variables as nodes, where the dependencies between variables are the edges of the graph.

- *Logistic Regression (LR)*. They are categorical classifiers that adopt a discriminative model. Like NB algorithms, LR methods make the a-priori independency assumption of the input features. Their performance is highly dependent on the size of the training data.

- *Support Vector Machines (SVM)*. They are non-probabilistic classifiers that map data samples in a feature space with the goal of maximizing the distance between each category of samples. They do not make any assumption on the input features, but they perform poorly in multi-class classifications. Hence, they should be used as binary classifiers. Their limited scalability might lead to long processing times.

- *Random Forest (RF)*. A random forest is a set of *decision trees*, and considers the output of each tree before providing a unified final response. Each decision

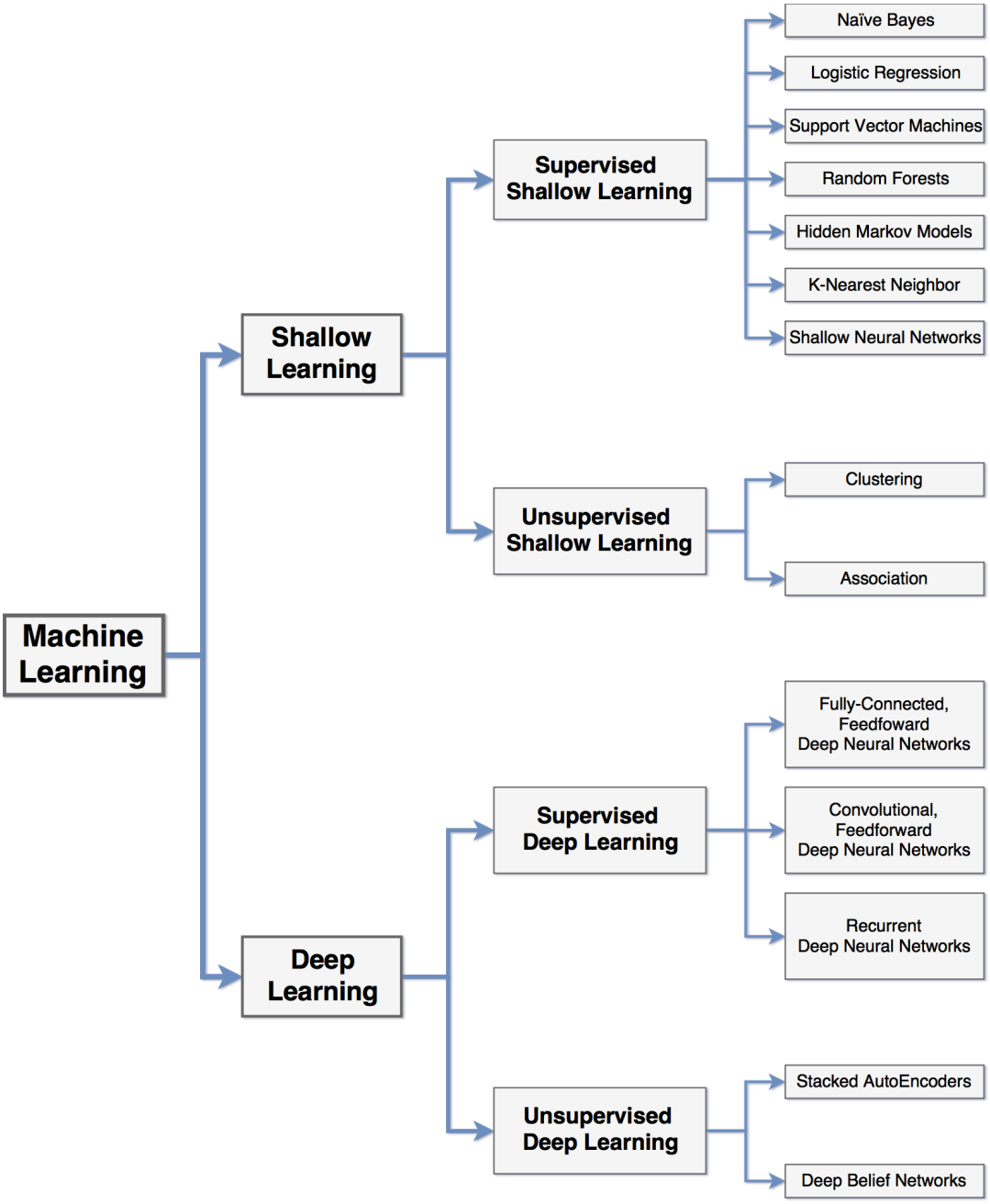



FIGURE 2.1: Classification of ML algorithms for cybersecurity applications.

tree is a conditional classifier: the tree is visited from the top and, at each node, a given condition is checked against one or more features of the analysed data. These methods are efficient for large datasets and excel at multiclass problems, but deeper trees might lead to overfitting.

- *Hidden Markov Models (HMM)*. They model the system as a set of states producing outputs with different probabilities: the goal is to determine the sequence of states that produced the observed outputs. HMM are effective for understanding the temporal behaviour of the observations, and for calculating the likelihood of a given sequence of events. Although HMM can be trained on labelled or unlabeled datasets, in cybersecurity they have mostly been used with labelled datasets.

- *K-Nearest Neighbor (KNN)*. KNN are used for classification and can be used for multi-class problems. However, both their training and test phase are computationally demanding: to classify each test sample, they compare it against all the training samples.

- *Shallow Neural Network (SNN)*. These algorithms are based on neural networks, which consist in a set of processing elements (that is, *neurons*) organized in two or more communicating layers. SNN include all those types of neural networks with a limited number of neurons and layers. Despite the existence of unsupervised SNN, in cybersecurity they have mostly been used for classification tasks.

### 2.1.2 Shallow Learning – unsupervised algorithms

- *Clustering*. They group data points that present similar characteristics. Well known approaches include *k-means* and *hierarchical* clustering. Clustering methods have a limited scalability, but they represent a flexible solution that is typically used as a preliminary phase before adopting a supervised algorithm or for anomaly detection purposes.

- *Association Rules*. They aim to identify unknown patterns between data, making them suitable for prediction purposes. However, they tend to produce an excessive output of not necessary valid rules, hence they must be combined

with accurate inspections by a human expert. This category of algorithms also include *sequential-rule mining* approaches [13].

### 2.1.3 Deep Learning – supervised algorithms

- *Fully-connected Feedforward Deep Neural Networks (FNN)*. They are a variant of DNN where every neuron is connected to all the neurons in the previous layer. FNN do not make any assumption on the input data and provide a flexible and general-purpose solution for classification, at the expense of high computational costs.

- *Convolutional Feedforward Deep Neural Networks (CNN)*. They are a variant of DNN where each neuron receives its input only from a subset of neurons of the previous layer. This characteristic makes CNN effective at analyzing spatial data, but their performance decreases when applied to non-spatial data. CNN have a lower computation cost than FNN.

- *Recurrent Deep Neural Networks (RNN)*. A variant of DNN whose neurons can send their output also to previous layers; this design makes them harder to train than FNN. They excel as sequence generators, especially their recent variant, the *long short-term memory*.

### 2.1.4 Deep Learning – unsupervised algorithms

- *Deep Belief Network (DBN)*. They are modelled through a composition of *Restricted Boltzmann Machines (RBM)*, a class of neural networks with no output layer. DBN can be successfully used for pre-training tasks because they excel in the function of feature extraction. They require a training phase but with unlabelled datasets.

- *Stacked AutoEncoders (SAE)*. They are composed by multiple *AutoEncoders*, a class of neural networks where the number of input and output neurons are the same. SAE excel at pre-training tasks similarly to DBN, and achieve better results on small datasets.

We conclude this section with an important remark. Existing literature (e.g. [14]–[16]) often refers to algorithms based on neural networks with the term *Multi-Layer Perceptron* (MLP). With the advent of deep learning, this term has become outdated,

and most recent studies started to adopt the terminology identifying the "deep" variants of neural networks.

### 2.1.5 Applications of machine learning algorithms to cyber detection

We consider the three areas where most cyber ML algorithms are finding application: *intrusion detection*, *malware detection*, and *spam detection*. An outline of each field is presented below.

**Intrusion detection** aims to discover illicit activities within a computer or a network through Intrusion Detection Systems (IDS). *Network IDS* are widely deployed in modern enterprise networks. These systems were traditionally based on patterns of known attacks, but modern deployments include other approaches for anomaly detection, threat detection and classification based on machine learning. Within the broader intrusion detection area, two specific problems are relevant to our analysis: the detection of *botnets* and of *Domain Generation Algorithms* (DGA). A botnet is a network of infected machines controlled by attackers and misused to conduct multiple illicit activities. Botnet detection aims to identify communications between infected machines within the monitored network and the external command-and-control servers. Despite many research proposals and commercial tools that address this threat, several botnets still exist. DGA automatically generate domain names, and are often used by an infected machine to communicate with external server(s) by periodically generating new hostnames. They represent a real threat for organizations because through DGA, relying on language processing techniques, it is possible to evade defences based on blacklists of static domain names. We consider DGA detection techniques based on ML.

**Malware detection** is an extremely relevant problem because modern malware can automatically generate novel variants with same malicious effects but appearing as a completely different executable file. These polymorphic and metamorphic features defeat traditional rule-based malware identification approaches. ML techniques can be used to analyse malware variants and attributing them to the correct malware family.

**Spam and phishing email detection** includes a large set of techniques aimed at reducing the waste of time and potential hazard caused by unwanted emails. Nowadays, unsolicited emails, namely *phishing*, represent the preferred way through which an attacker establishes a first foothold within an enterprise network. Phishing emails

include malware or links to compromised websites. Spam and phishing detection is increasingly difficult because of the advanced evasion strategies used by attackers to bypass traditional filters. ML approaches can improve the spam detection process.

In Table 2.1 we report the main ML algorithms that have been proposed to address the previously identified cybersecurity problems. In this table, rows report the family of algorithms presented in Sections 2.1.1 through 2.1.4, while columns denote cyber issues. Each cell indicates which ML algorithms are used for each problem; cells with a cross mark denote that, to the best of our knowledge, there is no proposal for that class of problems. From this table, it emerges that SL algorithms are applied to all considered problems. Supervised DL algorithms find wide application to malware analysis, less to intrusion detection; spam detection seems to be better combined with unsupervised DL algorithms. Despite its relatedness to natural language processing [3], no DL algorithm is applied to DGA detection. As expected, the overall number of algorithms based on DL is considerably smaller than those based on SL. Indeed, DL proposals based on huge neural networks are more recent than SL approaches. This gap opens many research opportunities.

TABLE 2.1: Applications of ML to cybersecurity problems.

| | | **Intrusion Detection** | | | **Malware Analysis** | **Spam Detection** |
|---|---|---|---|---|---|---|
| | | Network | Botnet | DGA | | |
| **Deep learning** | Supervised | RNN [8]<br>CNN [37] | FNN [18]<br>CNN [19]<br>RNN [20] | ✗ | | |
| | RNN [8] | ✗ | | | | |
| | Unsupervised | DBN [21]<br>SAE [22] | ✗ | ✗ | DBN [23]<br>SAE [24] | DBN [25]<br>SAE [26] |
| **Shallow learning** | Supervised | RF [4]<br>NB [4]<br>SVM [4]<br>LR [4]<br>HMM [4]<br>KNN [4]<br>SNN [4] | RF [27]<br>NB [27]<br>SVM [27]<br>LR [28]<br>KNN [29]<br>SNN [30] | RF [31]<br>HMM [31] | RF [32]<br>NB [32]<br>SVM [32]<br>LR [32]<br>HMM [33]<br>KNN [32]<br>SNN [34] | RF [35]<br>NB [36]<br>SVM [36]<br>LR [35]<br>KNN [35]<br>SNN [35] |
| | Unsupervised | Clustering [37]<br>Association [38] | Clustering [39] | Clustering [40] | Clustering [32]<br>Association [41] | Clustering [42]<br>Association [43] |

Finally, we highlight a significant difference among supervised and unsupervised approaches: the former algorithms are used for classification purposes and can implement complete detectors; the latter techniques perform ancillary activities [44]. Unsupervised SL algorithms are often used for grouping data with similar characteristics independently of predefined classification criteria, and excel at identifying useful features whenever the data to be analysed present high dimensionality [24].

## 2.2 Issues of Machine and Deep Learning for Cyber Detection

Unlike other fields, such as computer vision, speech recognition or social media marketing, where ML is mature even for autonomous products [3], the integration of cyber detection schemes with ML methods must be evaluated with due diligence: indeed, our analyses evidence that ML is still at an early stage for cybersecurity. We present *seven* issues that must be considered before deciding whether to deploy them in production with no human supervision. We substantiate each conclusion with experimental results from literature or original experiments performed on large enterprises. Our experimental campaign involves the problems of DGA Detection and Network Intrusion Detection, and leverage two ML algorithms: Random Forest (representative of the "Shallow Learning" category) and Feedforward Fully Connected Deep Neural Network (representing Deep Learning methods). We describe the testing environments used for our experiments and the metrics considered for evaluation in Section 2.2.1. Then we present the issues and experimental results in Section 2.2.2.

We anticipate that the our results evidence that machine and deep learning techniques still present several shortcomings that limit their applicability in real contexts. All approaches tend to generate too many false positives, they are vulnerable to adversarial attacks, and require continuous re-training and careful parameter tuning. Moreover, the maturity of deep learning approaches for Security Analytics is still at an early stage and significant improvements may be expected. We can conclude that machine and deep learning techniques can support the security operator activities and automate some tasks, but pros and cons must be known. Overestimating the autonomy of machine and deep learning capabilities can facilitate the possibilities for skilled attackers to infiltrate, steal data and even sabotage an enterprise.

### 2.2.1 Experimental methodology

For **DGA Detection** we compose two labelled training datasets containing both DGA and non-DGA domains. The former dataset contains DGA created through

known techniques, while the latter contains DGA created using more recent approaches. Non-DGA domains are randomly chosen among the Cisco Umbrella top-1 million[1]. We report the meaningful metrics of these training datasets in Table 2.2. Moreover, we build a testing dataset of 10 000 domains extracted evenly from each of the training datasets. We also rely on a real and unlabeled dataset composed of almost 20 000 domains contacted by a large organization. The features extracted for this dataset are: *n-gram* normality score [45], meaningful characters ratio [45], number-to-character ratio, vowel-to-consonant ratio, domain length. These datasets are used to train and test a self-developed Random Forest classifier composed of 100 decision trees leveraging the *CART* (classification and regression tree) algorithm.

TABLE 2.2: Training datasets for DGA Detection experiments.

| Dataset | DGA technique | DGA count | non-DGA count |
|---|---|---|---|
| 1 | Well-known | 21 355 | 20 227 |
| 2 | Well-known and recent | 37 673 | 8 120 |

For **Network Intrusion Detection**, we use three labelled real training datasets composed of benign and malicious network flows collected in a large organization of nearly 10 000 hosts. The labels are created by flagging as malicious those flows [46] that raised alerts by the enterprise network IDS and reviewed by a domain expert. Meaningful metrics of these training datasets are reported in Table 2.3. We generate also a testing dataset of 50 000 flows evenly extracted among the training datasets. The considered features for these datasets include: source/destination IP address, source/destination port, number of incoming/outgoing bytes and packets, TCP flags, protocol used, duration of the flow, list of alerts raised. These datasets are used to test and train two self-developed classifiers: one based on Random Forests and one based on Feedforward Fully-connected Deep Neural Network. Different topologies have been considered for each algorithm. The RF is composed by 100 decision trees leveraging the CART algorithm. For the FNN, the overall number of neurons ranges from 128 to 16 384, distributed between 2 to 16 layers; the hidden layers leverage the *ReLU* activation function, whereas the output layer uses a *sigmoid* activation function.

The quality of each classifier is measured through common performance metrics, namely *Precision* (*Prec*), *Recall* (or Detection Rate, *DR*), *F1-score*, which are computed

[1]Cisco Umbrella top-1 Million: `https://umbrella.cisco.com/blog/2016/12/14/cisco-umbrella-1-million/`

TABLE 2.3: Training datasets for Network Intrusion Detection experiments.

| Dataset | Malicious flows | Benign flows |
|---|---|---|
| 1 | 1 000 | 100 000 |
| 2 | 2 500 | 250 000 |
| 3 | 5 000 | 500 000 |

as follows:

$$Prec = \frac{TP}{TP + FP} \tag{2.1}$$

$$DR = \frac{TP}{TP + FN} \tag{2.2}$$

$$F1\text{-}score = 2 * \frac{Prec * DR}{Prec + DR}, \tag{2.3}$$

where $TP$, $FP$ and $FN$ denote true positives, false positives and false negatives, respectively. For completeness, we remark that we consider a true positive to be a correct detection of a malicious sample. Precision indicates how much a given approach is likely to provide a correct result. Recall is used to measure the detection rate. The F1-score combines Precision and Recall into a single value. We do not rely on Accuracy[2] because, in a real organization, the amount of legitimate events is several orders of magnitude greater than illegitimate events. Hence, all the Accuracy values are close to 1 and these results prevent capturing the true effectiveness of a classifier. Finally, to reduce the possibility of biased results, each evaluation metric is computed after performing 10-fold cross validation.

### 2.2.2 Evaluation results

**Shallow vs Deep Learning** Deep Learning is known to outperform Shallow Learning in some applications, such as computer vision [3]. This is not always the case for cybersecurity where some well configured SL algorithm may prevail, even because the DL proposals are scarce with respect to SL techniques in this domain. Just to give an example, we experimentally compare the performance of the two self-developed classifiers for Network Intrusion Detection, one based on RF (Shallow Learning) and another one based on FNN (Deep Learning). We focus on these techniques due to their academically-proven proficiency in Intrusion Detection scenarios [27], [47]–[51]. Both are trained with the third dataset described in Table 2.3 and tested on

[2] $Accuracy = \frac{FP+FN}{TP+TN+FN+FP}$, where $TN$ denotes true negatives.

the network intrusion detection testing dataset. To obtain more refined results, we repeat the training and test phase of these classifiers multiple times using different topologies. In Table 2.4, we show the classification results achieved by each method; for the FNN we report the results obtained by the best topology consisting in 1 024 neurons spread across 4 hidden layers. The RF classifier performed better than the FNN, with an F1-score of nearly 0.8, against the 0.6 obtained by the FNN. Our takeaway is that security administrators should not be charmed by the alluring neuronal multi-layer approach offered by Deep Learning, as some of these methods might still be immature for cybersecurity.

TABLE 2.4: Comparison between DL and SL classifiers.

| Classifier | *F1-score* | *Precision* | *Recall* |
|---|---|---|---|
| Random Forest (SL) | 0.7985 | 0.8727 | 0.736 |
| Fully-connected Feedforward Deep Neural Network (DL) | 0.6085 | 0.7708 | 0.5027 |

**General vs Specific detectors** Products based on machine learning are often promoted by vendors as catch-all solutions to a broad array of cyber attacks. However, unbiased experimental results show that ML algorithms may provide superior performance when they focus on *specific* threats instead of trying to detect multiple threats at once. We devise multiple intrusion detection systems based on the self-developed RF classifiers for network intrusion detection, each focusing on a specific type of attack, such as buffer overflows, malware infection, DoS. The training dataset for each classifier is based on the third dataset presented in Table 2.3. We train and test each classifier, and then compare their classification results with the classifier described in the first row of Table 2.4 that is our baseline. Table 2.5 shows the Precision, Recall and F1-score of the six classifiers that obtained the best results, alongside the baseline reported in the bottom row. These attack-specific classifiers obtain promising results on real traffic data with F1-scores of over 0.95, while the "general-purpose" classifier performs significantly poorly. We conclude that entrusting a single ML detector to identify malicious flows is an enticing but yet unfeasible goal. On the other hand, by having multiple detectors, each focusing on one attack type, it is possible to produce a defensive scheme with superior detection capabilities.

**Selection of a machine learning algorithm** Unbiased comparison of the effectiveness of two ML algorithms requires that they are both trained on the *same* training

TABLE 2.5: Classification results for attack-specific classifiers and the general classifier.

| Attack name | *F1-score* | *Precision* | *Recall* |
|---|---|---|---|
| DOS attempt | 0.9953 | 0.9938 | 0.9969 |
| Overflow attempt | 0.9939 | 0.9933 | 0.9946 |
| SSH Brute Force | 0.9916 | 0.9941 | 0.9892 |
| Suspicious DNS query | 0.9753 | 0.9953 | 0.9586 |
| Cache Poisoning attempt | 0.9676 | 0.9872 | 0.9506 |
| Possible Malware infection | 0.9587 | 0.9939 | 0.9337 |
| *General approach (baseline)* | 0.7985 | 0.8727 | 0.7360 |

TABLE 2.6: Detection rates of the RF classifier against different DGA before and after hardening. Source: [45].

| DGA method | Baseline *Recall* | Hardened *Recall* |
|---|---|---|
| *corebot* | 0.97 | **0.97** |
| *cryptolocker* | 0.87 | **0.88** |
| *dircrypt* | 0.95 | 0.93 |
| *kraken v2* | 0.72 | **0.76** |
| *lockyv2* | 0.87 | 0.84 |
| *pykspa* | 0.67 | **0.71** |
| *qakbot* | 0.94 | **0.94** |
| *ramdo* | 0.54 | **0.54** |
| *ramnit* | 0.94 | **0.94** |
| *simda* | 0.75 | **0.76** |

dataset and tested on the *same* dataset [4]. Even though many cybersecurity proposals rely on few and old public datasets, their results are not comparable due to several causes: the two algorithms consider different features; one or both algorithms may implement pre-filtering operations that alter the training dataset; they may use a different split between test and training dataset. For these reasons, meaningful comparisons between detection performance in literature are extremely difficult.

For example, papers such as [5] and [39] discuss ML methods for two cybersecurity problems, but they do not consider the different training and testing environments of the analysed works. Hence, although some solutions achieve higher accuracy than others, it is possible that results change significantly under different training settings. Furthermore, there is no guarantee that a method performing best on a test dataset confirms its superiority on different datasets. Security administrators should be aware of this issue, and should thoroughly question the evaluation methodology before accepting the performance results of different machine learning algorithms.

**False Positives and False Negatives** The implicit cost of a misclassification in the cybersecurity domain is a serious problem. False positives in malware classification and intrusion detection annoy security operators and hinder remediation in case of actual infection. In phishing detection, they might cause important, legitimate messages to not be delivered to end users. In contrast, failing to detect a malware, a network intrusion or a phishing email can compromise an entire organization. We explore this problem by considering the performance of ML solutions devoted to malware analysis and phishing detection [35], while we rely on an original experiment for network intrusion detection. For malware analysis, we consider the approach in [32] that proposes an original and effective method for malware classification: this paper contains a detailed analysis and comparison of different ML techniques which were trained and tested on the same datasets, thus satisfying the requirements for valid comparison of different techniques; hence, we deem this paper as a good representation of the state of the art of ML for determining the family to which a malware sample belongs. The evaluation is performed on the DREBIN[3] dataset: for large malware families the proposed approach, which outperforms all other baselines, obtains an F1-score of 0.95, whereas for small malware families it achieves an F1-score of 0.89. For phishing detection, we report the results described in [35] that, to the best of our knowledge, is the only paper on phishing email detection which compares different ML algorithms against the same comprehensive dataset. Therefore, we consider this work as a valid overview of the efficacy of different ML methods. The authors created a custom dataset of nearly 3 000 phishing emails on which several ML classifiers were tested: the best results were obtained by RF (lowest false positives) and LR (lowest false negatives), obtaining an F1-score of 0.90 and 0.89, respectively. The scenario for intrusion detection is different, as modern solutions can achieve higher Accuracy scores [4]. Although near-perfect Accuracy may seem an appreciable result, the massive amounts of events generated daily in a large enterprise account for hundreds to thousands of false positives that need to be manually triaged by security operators. We highlight this problem through an original experiment. We consider two DGA detectors based on the self-developed Random Forest classifiers trained on the first and second datasets of Table 2.2, respectively. We then validate them on the real domain dataset. Results are summarized in Table 2.7 which presents the number of domains that are flagged as DGA by

[3]DREBIN dataset: `https://www.sec.cs.tu-bs.de/~danarp/drebin/`

both classifiers, alongside its percentage on the total amount of domains included in the dataset. We can observe that the two classifiers obtain comparable detection performances on real traffic data, as they both signal about 400 domains. However, manual inspection revealed that they were not DGA, hence all the domains flagged as DGAs are actually false positives. As anticipated, even a false positive rate of 2% can account to hundreds of false alarms in a real organization.

TABLE 2.7: Performance of the DGA detection classifiers when used on real data.

| Classifier | Training Dataset | Domains classified as DGA |
|---|---|---|
| 1 | Well-known | 431(2.16%) |
| 2 | Well-known and recent | 397(1.99%) |

**Re-training issues** A well-known limitation of traditional detection approaches based on static detection rules is the need for frequent and continuous updates (e.g., daily updates of antivirus definitions). A similar issue affects also advanced ML approaches: reliance on outdated training datasets leads to poor detection performance. This is a critical problem for all supervised learning approaches requiring labelled training datasets: the manual creation of similar datasets is an expensive process because they need to be sufficiently large and comprehensive to allow the algorithm to learn the difference between the different classes. Furthermore, these operations are error prone and may lead to incorrect classifications. Finally, the large majority of organizations are unwilling to share their internal network data. This scenario leads to an overall scarcity of publicly available and labelled data for cybersecurity, thus causing periodic retraining extremely difficult or impossible. To show the detrimental effects of obsolete training sets, we perform an experiment comparing the detection rate of two instances of the same self-developed RF classifier for DGA detection. The first and second instances are trained with the first and second datasets reported in Table 2.2, respectively. Both classifiers are tested against the same synthetic domain dataset described at the beginning of this section. We report the results in Table 2.8, which shows the Precision, Recall and F1-score obtained by the two classifiers for DGA detection. As expected, the performance of the second classifier is significantly better because it obtains an F1-score for DGAs of 0.89 against 0.33. These results demonstrate that classifier performances are extremely sensitive to the freshness of the training set.

TABLE 2.8: Performance of the DGA detection classifiers when trained on outdated and recent datasets.

| Classifier | Training Dataset | *F1-score* | *Precision* | *Recall* |
|---|---|---|---|---|
| 1 | Well-known | 0.3306 | 0.1984 | 0.9913 |
| 2 | Well-known and recent | 0.8999 | 0.9126 | 0.8875 |

A potential solution to this issue may involve the application of a recent method by [52], which could be useful in cybersecurity contexts. This paper proposes a retraining approach that puts emphasis on the "when" a given event happens, so that more recent events have a greater influence on the future decisions than "older" events. This is achieved by assigning different weights to each sample, with newer samples having higher impact. However, we stress that a similar approach presents issues in itself, such as high computational costs required to constantly update the weights associated to each data sample.

**Deployment process** Security solutions based on ML achieve valid detection rates only if the training dataset is appropriate and the parameters of the algorithms are finely tuned. In most scenarios, these operations are still executed empirically and represent a resource intensive task that presents several risks. If these steps are not performed rigorously and/or training is not based on the right datasets, the results are underwhelming. We highlight these issues through a set of ML experiments applied to network intrusion detection. The goal is to show the considerably different results achieved by the same ML algorithm in different environments where either the number of features or the training dataset is changed. To this purpose, we rely on the RF classifier for network intrusion detection. We train it using the third dataset reported in Table 2.3 by choosing 5, 7, 10 or 12 features, selected through a *feature agglomeration* process; the testing phase is performed on the test dataset. We report the Precision, Recall and F1-score for the five sets of features in Table 2.9, where we observe that the same classifier yields different results, especially with regards to its Recall, with values ranging from 0.57 to 0.74.

TABLE 2.9: Performance of the intrusion detection classifier when trained with different features.

| Features | *F1-score* | *Precision* | *Recall* |
|---|---|---|---|
| 12 | 0.7985 | 0.8727 | 0.7361 |
| 10 | 0.7801 | 0.8684 | 0.7093 |
| 7 | 0.7476 | 0.8893 | 0.6448 |
| 5 | 0.6920 | 0.8724 | 0.5734 |

Then, we keep the number of features fixed at 12 and we repeat the training process two more times by using the first and then the second dataset reported in Table 2.3, and then test them on the same testing dataset. Table 2.10 reports the Precision, Recall and F1-score for the three training datasets. These results confirm that the Recall between the best and the worst case may differ by 10% and over.

TABLE 2.10: Performance of the intrusion detection classifier when trained on different datasets.

| **Dataset** | ***F1-score*** | ***Precision*** | ***Recall*** |
|---|---|---|---|
| 1 | 0.7306 | 0.8753 | 0.6270 |
| 2 | 0.7757 | 0.8703 | 0.6996 |
| 3 | 0.7985 | 0.8727 | 0.7361 |

**Vulnerability to adversarial attacks** Competent adversaries use advanced strategies to evade detectors based on machine learning algorithms [39]. These malicious actions, namely *adversarial attacks*, can affect any machine learning model, and are one of the main reasons why developing a fully autonomous cyber defence platform is a daunting task. We explore the topic of adversarial attacks with greater depth in the second part of this thesis, starting from Chapter 4.

# Chapter 3

# Novel Solutions for Cyber Detection

Detecting advanced cyber attacks is increasingly difficult as attackers have several ways to penetrate a network and to hide their activities. The huge volume of logs generated by the multitude of servers, firewalls and devices are useful only when they are integrated with security analytics systems for automatic detection and triage. Considering the attacker ability and the difficulty of signaling an infected host without causing false alarms, there is high demand of novel solutions that can be used to protect modern enterprises.

In this chapter, we propose original approaches that leverage Security Analytics to detect even sophisticated forms of attacks while, at the same time, reducing the human effort to manually inspect thousands of logs. We begin by presenting an original architecture that aims to detect cyber threats through a *multi-layer* approach, whose output is produced by means of a *prioritization* mechanism: indeed, instead of signaling an impossible "guaranteed" detection, our system ranks the most suspicious hosts and leaves to the security analyst the task of inspecting only a manageable number of hosts. Additional features include online processing for early prioritization and scalability over thousands of hosts as most analyses can be carried out independently for each host. Then, we propose a novel method to identify malicious *pivoting* activities occurring in large networks; our method involves devising an original algorithm that detects all pivoting activities within the monitored environment, in combination with a prioritization scheme to facilitate the inspection of the detected pivoting instances on the basis of their suspicious characteristics. Finally, we present an innovative method to identify malicious external hosts involved in *beaconing* activities by combining the analysis of time series with an unsupervised

machine learning algorithm; the output of this method is a manageable gray-list of few dozens of hosts. We remark that all these solutions share the common goal of (i) detecting recent threats; (ii) being easily integrated in modern enterprises; and (iii) producing an actionable output that can be easily managed by human operators.

The remainder of this chapter has the following structure. Section 3.1 presents the multi-layer architecture for countering advanced cyber threats. Section 3.2 describes our work on detection and prioritization of malicious pivoting activities. Finally, Section 3.3 shows our innovative approach to detect malicious external hosts involved in periodic activities.

## 3.1 Scalable Architecture for Online Prioritization of Cyber Threats

The information systems of modern organizations are subject to a multitude of cyber attacks conceived by a wide range of attackers with different goals, capabilities and motivations. Despite all efforts spent in preventive defenses, the reality is that attacks occur every day and no organization can consider itself secure. Existing proposals in literature perform detection of specific attacks through heuristics and statistical analysis (e.g., [53]–[56]). Most approaches ([54], [57]) rely on offline post-event analyses. Other online anomaly detectors assume that statistically detectable changes involve huge numbers of hosts (e.g., worm propagation in [58], [59]) or that compromised hosts share similar behavior (e.g., botnet detection in [60]–[63]). However, these assumptions are not true anymore in modern human-driven advanced cyber attacks [64], hence existing proposals can be affected by many false positive and false negative alarms.

As no security operator accepts to be annoyed by hundreds of alarms notified at the same priority level, we take a different direction and focus on *ranking* suspicious hosts. To this purpose, our online analysis begins by monitoring and analyzing the behavior of individual hosts over time and by identifying suspicious events involving even single or few hosts; we then post-correlate outputs to compute various indicators corresponding to different attacker activities. These indicators are finally aggregated to produce a ranking of the most suspicious hosts, which are then provided to the security operator in a timely fashion, thus allowing to focus only on the most suspicious hosts and activities.

Due to the amount of data to be managed online, we propose a scalable design and implementation of our approach. All initial phases before the final aggregation scale linearly with the number of hosts and can be parallelized. The proposed approach is general enough to be adopted with different types of data (such as internal traffic, external traffic, alarms coming from IDS and SIEM [65]–[67]), yet our goal is not to present a complete framework, but rather to propose the idea that the combination of autonomous triage with manual inspection increases the probability of detecting even advanced attacks. The effectiveness of our solution is shown through experiments applied to networks of more than one thousand hosts. We consider five main attack scenarios, representative of the activities that an attacker will likely perform from a compromised internal host: reconnaissance, data transfer to a dropzone, man in the middle, watering hole through DNS spoofing, and lateral movement activities. The appreciable results of our experiments demonstrate the feasibility and scalability of the proposed approach for online autonomous triage of different attack scenarios.

### 3.1.1 Related work

We identify three main areas of related works: offline forensics analysis, advanced malware detection, online traffic monitoring.

The large majority of related proposals in literature concern offline analysis for forensics purposes that differ from our online approach. Just to give some representative examples, we can cite [54] on heterogeneous logs analyses, [57] for its original graph-based approach for forensics, *BeeHive* [55] that correlates logs through histogram analysis to identify suspicious activities and corporate policy violations, [68] on forensics for cloud environments, and [69] on mobile forensics. Literature on advanced malware detection focuses on specific attack sequence patterns based on past APT campaigns (e.g., [64], [70]–[73]) instead of detecting suspicious activities in each possible phase of an attack. Other more general solutions (e.g., [74], [75]) share our idea of prioritizing suspicious hosts, but they are designed for offline or batch analysis.

The proposals based on online analyses focus on detection of DDoS [76]–[78], worm propagation and botnets, where the last two are the most related to our work. In worm propagation detection [59], [79], the internal network is usually modeled as a graph, where huge changes in the overall structure are identified as possible

infection propagations. These works differ from our proposal because they focus on a specific threat, and their analyses look for huge changes in traffic volumes and patterns, whereas we prioritize signals of malicious activities related to behavioral changes of individual hosts. Moreover, our solution is scalable with respect to the number of monitored hosts, while worm propagation analyses depend on the size of the network graph. Botnet detection proposals [60]–[63] are based on online scalable solutions for finding hosts that are possibly compromised. However, their underlying assumption is that a large number of hosts are compromised and share a similar network behavior, which is not true in the case of advanced cyber attacks where only few hosts may be compromised and malicious actions are often human-driven. In summary, we can outline the major contributions that differentiate our work with respect to the state of the art:

- *ranking* of suspicious activities instead of specific detection(s);
- *online* analysis instead of offline post-mortem analysis;
- analysis based on *individual hosts* behavior that guarantees parallel analyses and scalability;
- possibility of capturing suspicious actions involving even *few hosts*.

### 3.1.2 Proposed method

We aim to detect anomalous network activities concerning each host of a corporation, and to use this information to rank the most suspicious hosts. In this section we outline the proposed architecture and design choices for achieving scalability, and then describe our proposal in more detail. Figure 3.1 emphasizes that the input is represented by raw network data gathered by internal probes. Without loss of generality, we consider just network flows of traffic among internal hosts, which are feasible to collect and analyze for online contexts [80]. These logs are processed through three main steps: *analytics core*, *attack prioritization* and *autonomous triage*. The final output is a list of internal hosts ranked by a risk score representing the likelihood that each host is involved in one or more attacks.

Starting from raw network data, the analytics core builds different *layers* that are graph models in which nodes represent internal hosts and edges represent a metric of interest. Each layer portrays a different perspective of the events occurring

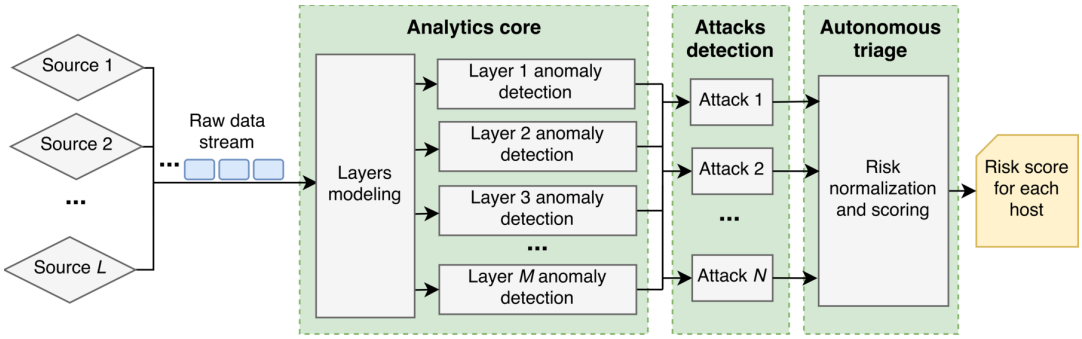



FIGURE 3.1: Framework overview.

in the monitored network. For example, if we consider three layers, then edges may represent the number of packets, the number of bytes, and the average duration of the transmissions between two hosts, respectively. Then, the analytics core applies anomaly detection algorithms on the activities of each internal host within each layer. This fine-grained analysis is motivated by the observation that an attack related to a single host within a large internal network cause very small alterations that are not visible in an aggregated model comprising all layers and all hosts. Similar "global" approaches work well only to identify massive attacks or network-wide anomalies [56], [81]. As a further advantage, since anomaly detection on different layers and hosts can be performed in parallel, the analytics core scales linearly with respect to the number of monitored hosts and layers. In such a way, we can extend and improve an instance of the framework by adding more layers and/or nodes without having to change the information flow and the overall architecture. The attack prioritization module takes as its input the anomalies identified by the analytics core, and correlates them with the goal of detecting different attack scenarios, each one corresponding to activities that an attacker may perform from a compromised internal host. It is also possible to include novel attack detection algorithms with limited computational effort because they can leverage the common fine-grained analyses already performed by the analytics core. The details of the attack prioritization algorithms are discussed in Section 3.1.3. The output of the attack prioritization module is a risk score assigned to each internal host for each considered attack. Attack specific risk scores for all hosts represent the input of the *autonomous triage* module that aids security operators by visualizing the few hosts with higher ranks and the attacks in which they are likely involved.

We now describe the algorithms used by the analytics core for *layers modeling* and for *anomaly detection* within each layer. The objective is to identify statistical

anomalies for each host on all the layers, which will be correlated and ranked by the attack prioritization module. The analytics core is designed for *online* processing and *scalability*.

**Layers Modeling** Raw data are collected from the probes as soon as they are produced, and temporarily stored for a time defined by the *current time window* of size $\Delta$. If $t$ denotes the current time, then the layers modeling module maintains all raw data generated between $t - \Delta$ and $t$. Since literature shows that most network activities are characterized by a daily periodicity [75], [81], it is convenient to set $\Delta$ equal to one day. At every *sampling interval* $\tau$, all raw data in the current time window are used to compute the *current representation* of all layers. Since anomalies can be detected only after their appearance in the current representation of a layer, "early" prioritization is influenced by the choice of the parameter $\tau$ that is conveniently chosen in the order of few minutes. Lower values cause useless oversampling of data (as an example, Netflow records [82] related to long-lived connections are refreshed every 2 minutes), while higher values introduce detection delays. We use the notation $L_i(t)$ to identify the current representation of the layer $i$ that is built using raw data in the current time window.

As shown in Figure 3.2, each $L_i(t)$ is modeled as a graph, in which the nodes represent hosts of the internal network, and the edges denote some specific features of network activities occurring between the two hosts. As an example, a layer representing the number of bytes exchanged between internal hosts can be defined as a directed and weighted graph, in which edge direction denotes the direction of data transfer (from source to destination) and the weight represents the amount of transferred bytes.

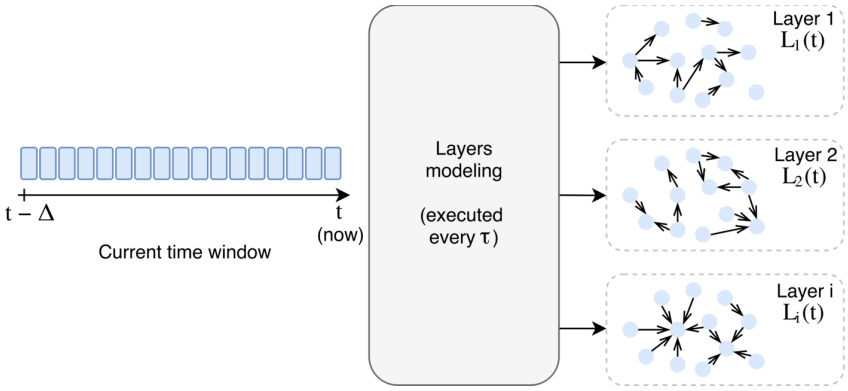



FIGURE 3.2: Activities of the layer modeling module.

Table 3.1 reports the list of considered layers and their descriptions. These characteristics are commonly adopted to identify anomalies in traffic [83]. For example, time series of flows, packets, bytes and ports are used to identify reconnaissance activities [58] and data exfiltration [75]; graphs of internal communications are adopted for identification of worm propagation [59]; `arp` messages can be useful for detecting eavesdropping activities [84].

TABLE 3.1: Considered Layers.

| | Layer | Description |
|---|---|---|
| $L_1$ | *Packets* | Directed weighted graph. Nodes are internal hosts and edges connect two nodes that exchange packets using any protocol. Direction is from source to target, and the weight of the edge is the total number of packets transmitted. |
| $L_2$ | *Bytes* | Directed weighted graph. Nodes are internal hosts and edges connect two nodes that exchange packets using any protocol. Direction is from source to target, and the weight of the edge is the total number of bytes transmitted. |
| $L_3$ | *Flows* | Directed weighted graph. Nodes are internal hosts and edges connect two nodes that exchange packets using any protocol. Direction is from source to target, and the weight of the edge is the total number of network flows. |
| $L_4$ | *Ports* | Directed weighted graph. Nodes are internal hosts and edges connect two nodes communicating through TCP or UDP protocols. Direction is from source node to target node, and the weight of the edge is the number of different destination port numbers. |
| $L_5$ | *Durations* | Directed weighted graph. Nodes are internal hosts and edges connect two nodes that exchange packets using any protocol. Direction is from source to target, and the weight of the edge is the average duration of network flows. |
| $L_6$ | *Conns* | Directed unweighted graph. Nodes are internal hosts and edges connect two nodes that exchange IP datagrams. Direction is from source to target. |
| $L_7$ | *Paths* | Bipartite directed graph. Both sets of nodes represent internal hosts. Edges connect each host from the first set, to the hosts of the second set that are reachable by it through a "path" composed of at least 3 hosts. |
| $L_8$ | *DNS* | Bipartite directed graph. One set of nodes represents hostnames of internal hosts, the other set of nodes represents IP addresses. Edges connect a hostname to the associated IP address in DNS resolutions. |
| $L_9$ | *ARP* | Bipartite directed graph. One set of nodes represents IP addresses of internal hosts, the other set of nodes represents MAC addresses. Edges connect the IP address and the MAC address that are bound as part of an `arp` transaction. |

The representations of all layers are passed in input to the processing modules that perform anomaly detection. We anticipate that an implementation of a possible way to correlate all these layers is provided in Section 3.1.3.

**Layer Anomaly Detection** The goal is to identify hosts that exhibit anomalous behaviors in any of the layers. This step does not depend on the identifiable attacks nor on the feature represented by each layer, hence all layers are subject to the same anomaly detection algorithms, which can be executed in parallel and independently. For each layer, we adopt two complementary detection approaches, as shown in Figure 3.3: the former identifies quantitative anomalies and state changes; the latter detects novel or uncommon events.

The former approach processes all current layer representations $L_i(t)$. The goal is to extract scalar values from graphs and to build time series. For each host, the

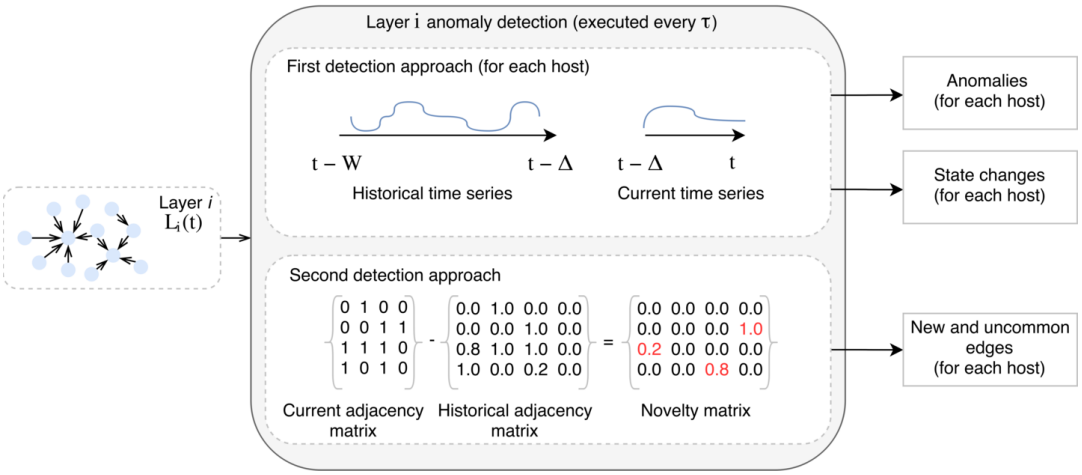



FIGURE 3.3: Structure of a layer anomaly detection module.

framework computes two scalar values: the weighted in-degree and the weighted out-degree [85] representing the number of incoming and outgoing connections of each host in the current layer, respectively. Since a new $L_i(t)$ is received by the layer anomaly detection module (one at every sampling interval $\tau$), scalar values for consecutive $L_i(t)$ are used to build two current time series representing recent values of in-degree and out-degree for each host. If $t$ denotes the current instant of time, the current time series includes values between $t - \Delta$ and $t$. Moreover, to perform anomaly detection, it is necessary to build two historical time series including older scalar values between $t - W$ and $t - \Delta$ (excluded), where $W$ represents the size of the historical window. $W$ should be large enough (in the order of few weeks) to have a reliable baseline for the past behavior of each host [80]. Traffic among internal hosts exhibits more stability with respect to traffic among internal and external hosts, characterized by higher variability and consequent difficulties to achieve stable baseline models [81]. Anomaly detection is performed on each current time series through the online and adaptive detection algorithm proposed in [80] trained over the period $W$. This algorithm identifies both point anomalies and state changes [53] that reflect different kinds of relevant deviations between the current and past behaviors of an internal host.

The latter approach (see Figure 3.3) identifies new edges that never appeared in the historical window. For example, these edges may represent novel persistent connections of an attacker trying to perform lateral movement. For each $L_i(t)$, the detection algorithm computes its current adjacency matrix [85], which is a mathematical representation of the edges in $L_i(t)$, whose rows and columns represent the

internal hosts: the matrix element $(j, k)$ is set to 1 if $L_i(t)$ has an edge from host $j$ to $k$, to 0 otherwise.

Older versions of the current adjacency matrix, built on previous $L_i(t)$ belonging to the historical time window $W$, are used to compute the historical adjacency matrix. Its values are rational numbers between 0 and 1. In particular, $(j, k)$ denotes the frequency of occurrence of the edge from $j$ to $k$ in the older instances of $L_i(t)$. For example, the value $(j, k)$ is set to 1 if all older instances of $L_i(t)$ layer contain an edge from $j$ to $k$; if one fifth of older $L_i(t)$ layers include an edge from $j$ to $k$, then the value $(j, k)$ is set to 0.2. The historical time window is updated every $\Delta$.

At every sampling interval $\tau$, the detection algorithm subtracts the historical adjacency matrix to the current adjacency matrix. The result, defined as novelty matrix, allows an immediate identification of new or uncommon edges that are present in $L_i(t)$, but never or seldom appeared in the historical time window. Uncommon edges having a low value in the historical adjacency matrix will result in values that are close to 1 in the novelty matrix; common edges with high values in the historical adjacency matrix will result in values close to 0 in the novelty matrix. The layer anomaly detection algorithm can sum the values included in each row of the novelty matrix to evaluate the "novelty" of all the edges starting from the corresponding host. Similarly, the "novelty" of all edges that end in any internal host is computed by summing the values on the corresponding column of the novelty matrix.

Anomalies, state-changes and novel edges detected by the analytics core are then used by the algorithms that evidence malicious activities and prioritize them.

### 3.1.3 Evaluation results

The main goal is to prioritize signals of malicious activities that may be part of an advanced attack. To this purpose, we correlate the anomalies, state-changes and novel edges detected by the analytics core. We first present the experimental testbed, and then show the results of our evaluation for each different attack scenarios.

There are several possible indicators associated with malicious activities. Here, we consider: reconnaissance (R), data transfer to a dropzone (DTD), Man in the Middle (MITM), watering hole through DNS spoofing (WH), lateral movement through (LM). Table 3.2 indicates which layer models are included in the analysis of each attack scenario. The presence of multiple layers increases confidence that a suspicious activity is actually occurring. The attack prioritization module evaluates a risk score

for each internal host by combining the anomalies, state-changes and novel edges detected by the analytics core. We refer to the following notations:

- $A^{in}_{L_i}$ (resp. $A^{out}_{L_i}$) denotes the intensity of the biggest point anomaly in the incoming (resp. outgoing) time series related to layer $L_i$ of an internal host during the observed window [55]. For example, a burst in the outgoing bytes.
- $C^{in}_{L_i}$ (resp. $C^{out}_{L_i}$) denotes the intensity of possible state-changes in the incoming (resp. outgoing) time series related to layer $L_i$ of an internal host. For example, a state-change is detected if the average number of packets in the current window doubles for a long period (hence, it is not only a point anomaly [56]).
- $N^{in}_{L_i}$ (resp. $N^{out}_{L_i}$) denotes the number of new incoming (resp. outgoing) edges of an internal host in the graph of layer $L_i$. For example, it can be used to detect the number of newly contacted hosts in the current time window.

All formulas and scores in this section are computed for each host.

TABLE 3.2: Layers used to prioritize different types of attacker activities.

| | $L_1$ | $L_2$ | $L_3$ | $L_4$ | $L_5$ | $L_6$ | $L_7$ | $L_8$ | $L_9$ |
|---|---|---|---|---|---|---|---|---|---|
| **Attack** | **Packets** | **Bytes** | **Flows** | **Ports** | **Durations** | **Conns** | **Paths** | **DNS** | **ARP** |
| *Reconnaissance* | | | ✓ | ✓ | ✓ | ✓ | | | |
| *Data transfer* | ✓ | ✓ | | | ✓ | ✓ | | | |
| *MITM* | ✓ | ✓ | ✓ | | | ✓ | | | ✓ |
| *Watering hole* | | | | | ✓ | ✓ | | ✓ | |
| *Lateral movement* | | | | | ✓ | | ✓ | | |

In the experiments, we consider an internal network consisting of more than 1 000 hosts composed of about 800 clients and 200 servers. The client machines have heterogeneous operating systems including several versions of Mac OS, Linux, and Windows. The server machines host mainly websites and DBMS, but also high performance computations, code versioning and NAS storage. We place monitoring probes in the main 1Gbit switches of the network. Our algorithms are executed on a cluster of eight blades, each having an Intel Xeon 2.6GHz CPU and 16GB of RAM. Network flows are sampled every five minutes.

To evaluate scalability, we consider three incremental scenarios consisting of 96, 287 and 1 012 hosts, respectively. One cluster node is sufficient for computations related to 96 and 287 hosts, while four nodes are necessary for the scenario with 1 012 hosts. This scalability is achieved because all computations of the analytics core are performed independently for each host and for each layer. Operations of

the attack prioritization module do not scale linearly, but their computational cost is negligible with respect to the anomaly detection algorithms of the analytics core. We present the details about prioritization of the five attack scenarios, and how risk scores are shown to the security operators. For each scenario, we inject multiple attacks in some hosts of the network, we apply our analytics and evaluate a *risk score* for each host. Finally, we report that we made a snippet of the network data used for these experiments publicly available[1].

**Reconnaissance in internal network** An attacker having control of an internal host likely scans neighbor hosts looking for (known or zero-day) vulnerabilities [58], [78]. We define the risk score $R$ for reconnaissance as follows:

$$R = \frac{A^{out}_{Flows} + A^{out}_{Ports} + N^{out}_{Conns}}{1 + A^{out}_{Durations}} \tag{3.1}$$

where a higher value of $R$ denotes a higher likelihood that an internal host is performing a scan. Intuitively, when an internal host performs a reconnaissance activity, the average duration of its connections decreases (due to many volatile communications) while the numbers of flows, ports and contacted hosts increase. To evaluate the risk score for this attack, we carry out reconnaissance activities from 10 hosts by varying the scan intensity in terms of number of scanned hosts and ports, as described in Table 3.3.

TABLE 3.3: Reconnaissance attacks injected in the internal network from 10 hosts.

| Attack | #ports scanned | #hosts scanned |
|---|---|---|
| *horizontal scan* | 1 single port | from 50 to 1 000 distinct hosts |
| *vertical scan* | from 50 to 1 000 distinct ports | 1 single host |
| *block scan* | from 50 to 1 000 distinct ports | from 50 to 1 000 distinct hosts |

Since our approach produces a ranking, we evaluate how many times an internal host performing the attack is prioritized within the Top-K hosts. Table 3.4 reports the results of multiple experiments executed over several weeks, where each row represents the percentage of times an internal host performing the attack has been ranked within the Top-K. Each column corresponds to horizontal, vertical, or block scan experiments as described in Table 3.3.

Table 3.4 shows that in more than 99% of the cases a host performing a reconnaissance activity is ranked within the Top 10. Horizontal scans are easier to detect

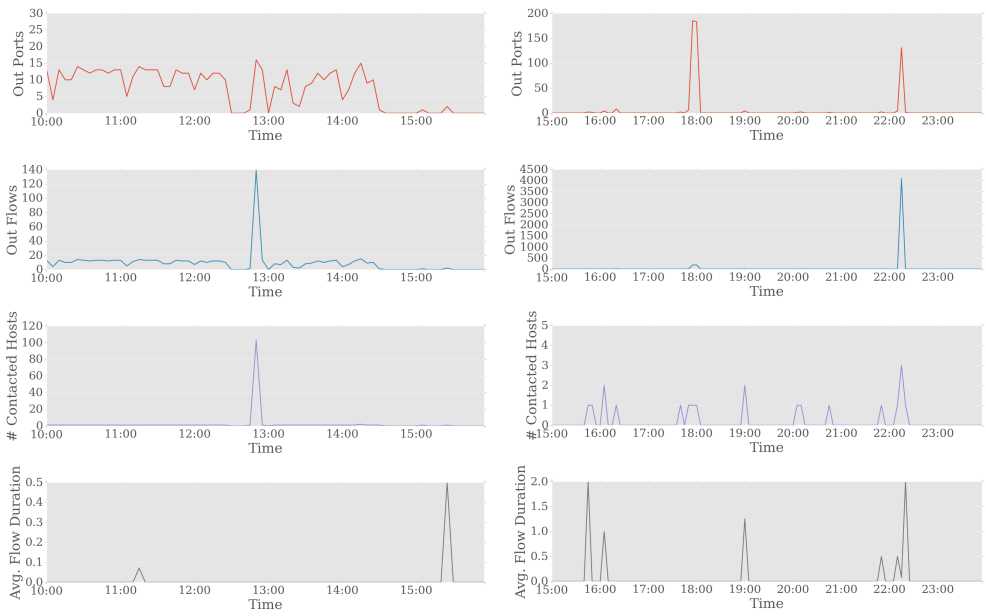
[1]Data Snippet: `https://weblab.ing.unimore.it/people/apruzzese/datasets/pivoting.html`

TABLE 3.4: Percentage of times a host performing a reconnaissance is ranked within the Top-K.

| In Top-K | Horizontal scan | Vertical scan | Block scan |
|---|---|---|---|
| *in Top 5* | 94.3% | 92.4% | 99.2% |
| *in Top 10* | 99.5% | 99.1% | 99.7% |
| *in Top 25* | 100% | 99.7% | 100% |
| *in Top 50* | 100% | 100% | 100% |

because they span over multiple hosts, whereas vertical scans are harder because it is more common for clients to contact servers on multiple ports if they offer more than one service. As expected, block scans have higher rankings, because they span both over multiple ports and multiple hosts.

Figures 3.4 report an example of the traffic time series used to compute the risk score $R$, extracted from the layers *Ports*, *Flows*, *Conns*, *Durations*. The x-axis represents time, and the y-axis reports the value of different metrics. Small arrows highlight significant anomalies: a horizontal scan (in Figure 3.4a) around 12:48 and two vertical scans (in Figure 3.4b) around 17:55 and 22:20.




(A) Horizontal Scan (B) Vertical Scan.

FIGURE 3.4: Time series of an internal host performing horizontal and vertical scans.

**Data transfer to dropzone** Attackers often move data to be exfiltrated towards an internal dropzone [64], [75], used as intermediate point from which the exfiltration is easier to perform. These activities can be detected through the risk score *DTD* defined as follows:

$$DTD = \frac{A^{out}_{Durations} + A^{out}_{Bytes} + A^{out}_{Packets}}{1 + N^{out}_{Conns}} \tag{3.2}$$

where a higher value of $DTD$ suggests that an internal host is likely transferring data to an internal dropzone. In the numerator, we consider point anomalies instead of state changes because the higher bandwidth of internal networks – typically in the order of Gbps – allows for short transfer times. In the denominator, we consider $N^{out}_{Conns}$ to rule out legitimate intensified network activity, such as p2p protocols. We perform several experiments in which we simulate DTD attacks of increasing transfer sizes from 10MB-50MB to 100MB-1GB. We use five controlled hosts as possible attackers and we transfer data to a Web server of the organization as an emulated dropzone. Table 3.5 reports the percentage of times a host performing a DTD is ranked within the Top-K: for small amounts of data (10-50MB), in about 92% of the cases the hosts are ranked within the Top 10. This is the most challenging scenario for detection because clients may use multiple hosts/devices for backup, hence it is tough to identify anomalous transfers unless we integrate anomaly detection with white/black lists of internal hosts (where storage is or is not allowed), but similar integrations are out of the scope of this work. The most compelling result is that in 99% of the cases, the 50-100MB internal transfers are ranked within the Top 10.

TABLE 3.5: Percentage of times a host performing a DTD is ranked within the Top-K.

| In Top-K | 10-50MB | 50-100MB | 100MB-1GB |
|---|---|---|---|
| *in Top 5* | 87.5% | 95.4% | 99.7% |
| *in Top 10* | 91.7% | 99.1% | 100% |
| *in Top 25* | 95.6% | 99.8% | 100% |
| *in Top 50* | 99.5% | 100% | 100% |

As an example, Figure 3.5 reports the time series that show the evolution of several layers referring to an internal host used for injecting data transfers to dropzone. The x-axis represents time, and the y-axis reports the different metrics. We highlight peaks of about 100MB (occurring at 17:00) and 1GB (occurring at 20:00), corresponding to the injected data exfiltration. We also observe an increment of the average flow duration in correspondence of the two data transfers, whereas other statistics (e.g., number of contacted hosts) remain stable.

**MITM: Man in the Middle attack** Man in the Middle (MITM) attacks are important to perform advanced reconnaissance or to steal credentials, because an attacker

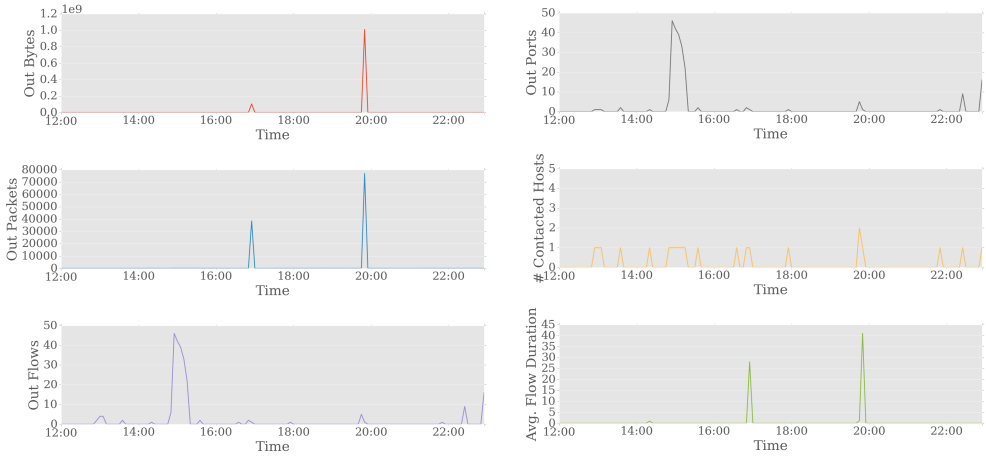



FIGURE 3.5: Time series of an internal host in which two DTDs of 100MB and 1GB are injected.

can eavesdrop communications of hosts within the same subnet. Here, we consider one of the most subtle forms of MITM performed through `arp` spoofing [86]. In this scenario, an attacker sends fake correspondences between IP and MAC addresses with the goal of acting as "hidden" proxy between a victim and the gateway of its subnet. Netflows record no explicit communication between the eavesdropper and the victim, but our experiments evidence that once a host becomes victim of MITM, then all packets sent and received by the victim pass twice through the switch. This attack can be captured by the state-change detection algorithm of the analytics core. In order to prioritize possible victims of MITM we define the following risk score:

$$MITM = \frac{C^{in}_{Bytes} + C^{out}_{Bytes} + C^{in}_{Pkts} + C^{out}_{Pkts}}{1 + C^{in}_{Flows} + C^{out}_{Flows} + N^{in}_{Conns} + N^{out}_{Conns}} * N^{out}_{ARP} \tag{3.3}$$

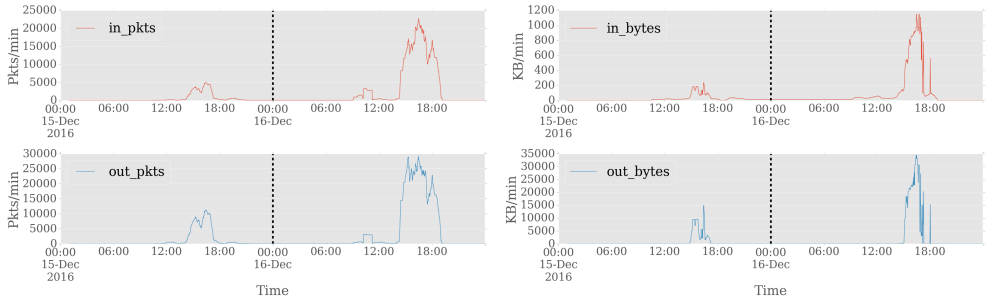
In the numerator we consider state-changes instead of point anomalies because MITM is usually an activity that lasts for some time to get useful information. The parameter $N^{out}_{ARP}$ is a multiplicative factor because if $N^{out}_{ARP} = 0$ there is no new correspondence in the ARP layer (see Section 3.1.2). In the denominator, we include state-changes and novel edges in *Flows* and *Conns* layers, because they must remain approximately stable with respect to a past window even if MITM is occurring. Experimental results are achieved through controlled Man in the Middle attacks of varying durations where we use one host as the eavesdropper and other 10 hosts as victims. From Table 3.6, we can observe that in more than 95% of the cases, even MITM lasting for just 15-30 minutes are prioritized in the Top 10; if an attack lasts for at least 1 hour, the victim hosts are ranked within the Top 5 in more than 98% of

the cases.

TABLE 3.6: Percentage of times a host victim of a MITM is ranked within the Top-K.

| In Top-K | 15-30min | 1-2hr | 12-24hr | 24-72hr |
|---|---|---|---|---|
| *in Top 5* | 89.8% | 98.2% | 99.4% | 99.8% |
| *in Top 10* | 95.4% | 99.1% | 99.8% | 100% |
| *in Top 25* | 99.0% | 99.8% | 100% | 100% |
| *in Top 50* | 99.7% | 100% | 100% | 100% |

As a motivation, in Figures 3.6 we report the time series of a host related to *Packets* (Figure 3.6a) and *Bytes* (Figure 3.6b) layers, where the y-axis denotes the different metrics, and the x-axis reports time. The plots report two days separated by a vertical dashed line. When the MITM is occurring on the second day, it is possible to observe that the number of packets and bytes increases significantly.




(A) Time series of the *Packets*. (B) Time series of the *Bytes*.

FIGURE 3.6: Comparison showing changes in *Packets* and *Bytes* layers when MITM occurs.

**Watering hole through DNS spoofing** Watering hole is a technique used by attackers to increase their coverage and persistence by infecting multiple hosts of an organization simultaneously. We consider a particular type of *watering hole* attack performed through DNS spoofing [87], where the attacker spoofs DNS responses to redirect victims to a compromised sever. To prioritize internal hosts that may correspond to watering holes, we define the risk score $WH$ as follows:

$$WH = (N^{in}_{Conns} + C^{in}_{Conns} + C^{in}_{Durations}) * N^{out}_{DNS} \tag{3.4}$$

where a high value of $WH$ represents a higher likelihood that a host is performing a watering hole through DNS spoofing. Intuitively, this can be prioritized when a host has many new incoming connections, its IP corresponds to a new DNS resolution, and has a state-change in the number and duration of incoming connections. We

observe that $N_{DNS}^{out}$ is a multiplicative factor, because $N_{DNS}^{out} = 0$ implies that no DNS spoofing occurred.

To evaluate the risk score $WH$, we use five internal clients as "spoofers" of three internal Web servers offering different services: *Server 1 (small)*, *Server 2 (medium)* and *Server 3 (large)* that are used by an average number of clients per hour of about 10, 50 and 250, respectively. We perform a DNS spoofing at times 10am, 2pm, 4pm, 6pm. Table 3.7 reports the percentage of times a "watering hole" host (that was redirecting traffic to itself through DNS spoofing) has been ranked within the Top-K hosts in the different scenarios. This table shows that for *Server 1* (having small activity) the spoofer is prioritized in the Top 10 in more than 96% of the cases, while this percentage is even higher for servers with high number of clients where the intensity of the redirect is more evident.

TABLE 3.7: Percentage of times a host performing "watering hole" is ranked within the Top-K.

| In Top-K | Server 1 (small) | Server 2 (medium) | Server 3 (large) |
|---|---|---|---|
| *in Top 5* | 94.7% | 98.9% | 99.9% |
| *in Top 10* | 96.2% | 99.8% | 100% |
| *in Top 25* | 99.5% | 100% | 100% |
| *in Top 50* | 99.8% | 100% | 100% |

As an example, Figure 3.7 reports a bipartite graph representation of the Conns layer over different days: on day 6, the red circle highlights a host that started performing a watering hole attack through DNS spoofing. We can observe an increase in the number of the incoming communications.

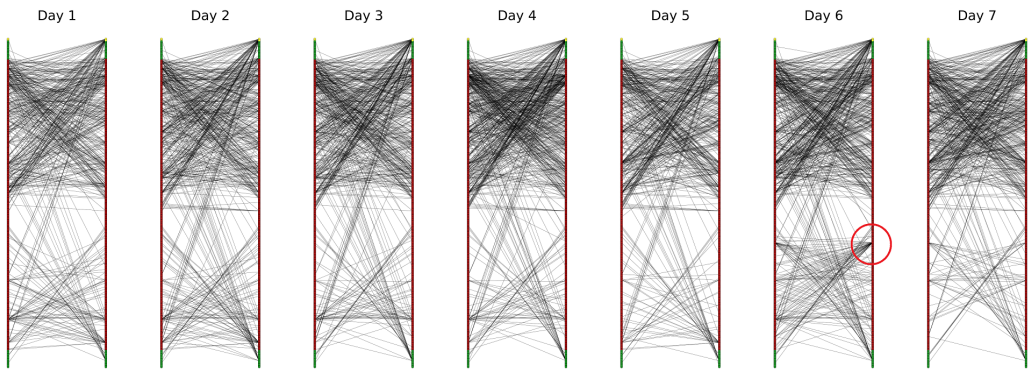



FIGURE 3.7: Bipartite communications graph derived from *Conns* layer over 7 different days.

**Lateral movement** To get closer to his target, an attacker tends to compromise several internal hosts with higher privileges or to access to the most internal parts of the corporate network. This activity is called *lateral movement* [64]. Figure 3.8 reports an example where an attacker expands his control over multiple hosts to access a LAN that cannot be reached directly from the external.

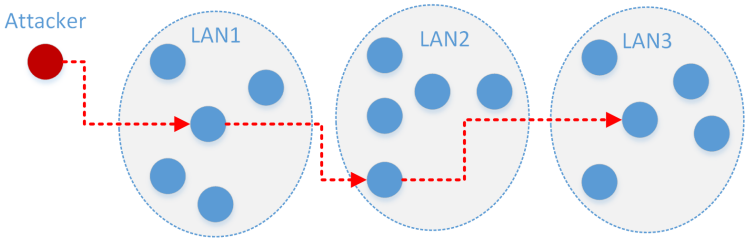



FIGURE 3.8: Example of lateral movement.

To prioritize lateral movements, we define the risk score $LM$ as follows:

$$LM = N^{in}_{Paths} + N^{out}_{Paths} + C^{in}_{Durations} + C^{out}_{Durations} \tag{3.5}$$

where $N^{in}_{Paths}$ and $N^{out}_{Paths}$ take into account new paths in the communications graph (see Section 3.1.2), while $C^{in}_{Duration}$ and $C^{out}_{Duration}$ check whether an increment in the average duration of the flows has occurred. Indeed, we recall that the commands issued by the attacker generate communications that have to last for some time [88]). We perform several experiments involving up to 10 controlled clients as intermediate hosts in the lateral movement (see Figure 3.8). Table 3.8 reports the percentages of times one of the controlled hosts is ranked within the Top-K risky nodes. The different columns correspond to different *lengths* of the controlled host chains. For example, Figure 3.8 presents a chain of length 2 with two intermediate hosts.

TABLE 3.8: Percentage of times a host performing LM is ranked within the Top-K.

| In Top-K | 1 host | 3-5 hosts | 8-10 hosts |
|---|---|---|---|
| *in Top 5* | 96.2% | 99.7% | 99.9% |
| *in Top 10* | 97.9% | 99.9% | 100% |
| *in Top 25* | 99.1% | 100% | 100% |
| *in Top 50* | 99.8% | 100% | 100% |

**Autonomous triage to support security analysts** We present how results can be combined to produce an overall ranking useful for security analysts to focus on few suspicious hosts.

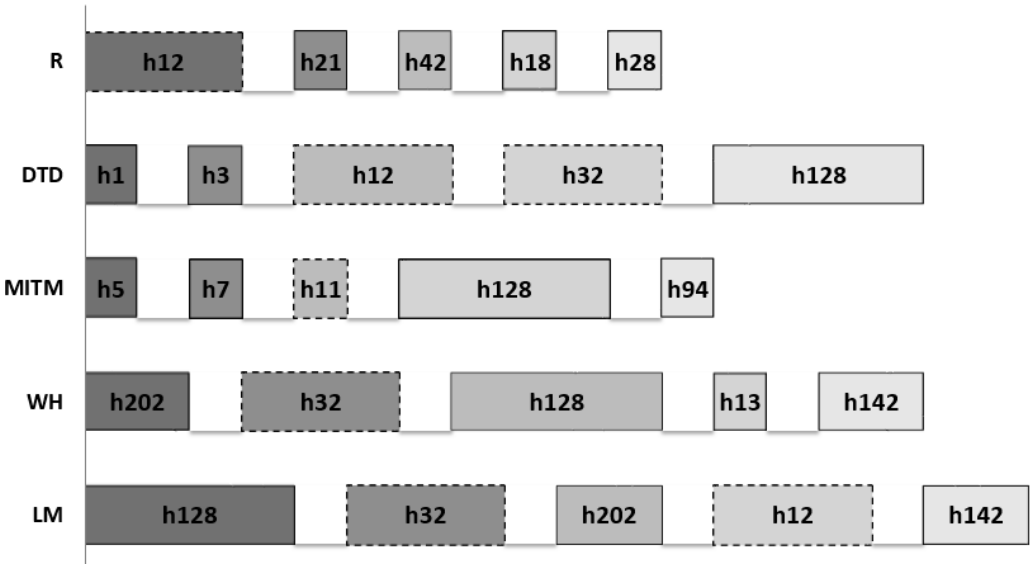



FIGURE 3.9: Online autonomous triage of internal hosts for different attack scenarios.

Figure 3.9 reports the overall rankings, with each line corresponding to a different attack: R for reconnaissance, DTD for data transfer to dropzone, MITM for Man in the Middle, WH for watering hole through DNS spoofing, LM for lateral movement. On the leftmost (resp. rightmost) side, there is the host with higher (resp. lower) risk score on that line. All hosts are represented as rectangles, where the size is proportional to the number of Top-K rankings in which a host appears. For example, host *h202* appears in two Top 5 rankings (first for WH, and third for LM), hence its rectangle has a double size. As ranking operations are evaluated online, rectangles with a dashed outline denote hosts that recently entered a Top 5 ranking. A similar visualization supports security analysts in monitoring several cyber threats occurring in the core of large networks. The number of Top-K hosts to show can be chosen adaptively depending on the size of the organization and the amount of human resources. As a final remark, it is important to observe that our proposal makes it hard for an attacker to elude ranking, because we monitor changes in activity of each *individual host* with respect to its history, and we produce an overall ranking considering all hosts of the internal network. Hence, an attacker would require a *complete* view of all hosts behaviors/history to evade prioritization successfully.

## 3.2 Detection and Threat Prioritization of Pivoting Attacks

Defending large enterprise systems is an increasingly challenging task. Modern attacks may combine social engineering strategies with malware to exploit software vulnerabilities, allowing attackers to find their ways into the network. Attackers typically begin by compromising any vulnerable internal host and then try to reach the most valuable targets by moving host-to-host laterally and deeper into the enterprise network. To this purpose, attackers are increasingly adopting the so called *pivoting* technique [89] in which a command propagation tunnel is created through one or more compromised internal host called *pivoters*. Several examples of pivoting can be found in the *Operation Aurora* [90], in the *Operation Night Dragon* [91], in the *BlackEnergy* [92] malware that in December 2015 compromised the Ukrainian power grid, in the *MEDJACK* [93] attack in 2016 where attackers stole health data by using medical devices as pivoters. Among the most recent cases at the time of writing, we can cite the *Archimedes* tool [94] that leverages pivoting to reach the LAN of target hosts, passively gathers their Web traffic, and injects forged Web pages with the goal of compromising hosts or stealing credentials. All these examples demonstrate that pivoting is an emerging and challenging research problem.

To address these issues, we propose the first algorithm for detection and threat prioritization of pivoting that analyzes internal network flows and does not rely on any a-priori knowledge about the adopted protocols and compromised hosts, which are instead needed by related solutions making them impractical for real contexts. For example, the algorithms in [95], [96] consider only a specific protocol (e.g., SSH brute-forcing), while another paper [97] considers only hosts that generate some IDS security alerts. Approaches in [59], [98] may work for detecting pivoting only if aggressive scan activities are performed over hundreds of hosts. We initially propose an original formalization of the pivoting detection problem into the temporal graph analytics domain, and then we present a pivoting detection algorithm that identifies pivoting tunnels through efficient network flow analyses that do not require any a-priori assumption about involved protocols and hosts. As some paths may correspond to benign activities (e.g. SSH tunnels created by network administrators), we add an original threat prioritization mechanism that ranks the detected pivoting activities on the basis of a maliciousness score, therefore reducing the rate of false alarms related to benign pivoting activities. We validate the effectiveness of

our proposal through an extensive experimental campaign, whose aim is to measure its accuracy (which is also compared to those of related algorithms [59], [95]) but also its computational cost. This evaluation is conducted on huge amounts of network traffic data pertaining to a large organization, which consist of over 650M communications collected over more than five months, in which we inject instances of synthetic pivoting attacks. The appreciable results obtained by our implementation show that the proposed approach is able to effectively detect and prioritize malicious pivoting activities even against attackers that adopt evasion techniques; moreover, its short execution times (few minutes to analyze one day of traffic of a large organization) make the proposal applicable to real contexts. Finally, we remark that a modern defensive system adopts multi-layer analyses, hence it is important to highlight that the proposed algorithm can be integrated with any other detection schemes based on black- and white-lists of hosts, on analyses of DNS traffic and of internal-to-external traffic. Similar extensions can further decrease the potential damage of malicious pivoting activities to modern enterprises.

### 3.2.1 Related work

To the best of our knowledge, we present the first algorithms for detection and threat prioritization of malicious pivoting activities: our proposal relies on the analysis of network flows, does not make assumptions about involved protocols and hosts, and is based on an original formulation of the pivoting detection problem in the temporal graph analytics domain.

A research area broadly related to pivoting studies malware propagation, whose two representative examples are worm detection [98] and botnet [60] discovery. These works define statistical models of normality for communications among internal hosts, where anomalies represent possible bot/worm propagations. Other papers in this area analyze network flows (e.g., [61], [63], [99]) as we do, but their solutions work only under two major assumptions that are not valid for pivoting: they require from tens to hundreds of hosts involved in malicious activities; they assume that an aggressive propagation model is adopted by the bot/worm. Unlike these scenarios, pivoting activities typically involve few internal hosts out of the thousands and more of a medium-large organization. Moreover, several research papers based on network flows analyses [100] aim to detect quite different attacks, such as data exfiltration [75], DDoS [101], port-scanning [83]. Other works focus on

attack detection in Wireless Sensor Networks that does not include the identification of command propagation tunnels which is the core of our proposal. For example, the authors in [102] propose a protocol for detecting selective forwarding attacks, and the researchers in [103] present a framework for detecting ongoing attacks (such as DoS and ARP replay) through usage control and chance discovery.

Research on pivoting is still at the beginning. Many articles (e.g., [90]–[93]) have a useful descriptive approach of popular pivoting-based attacks, but they do not propose any novel detection algorithm. Other works (e.g., [104], [105]) aim to train security experts on pivoting-related plans and attacks but they do not consider countermeasures. Some papers focus on prevention of pivoting-related activities, without proposing any detection approach. For example, Chapman et al. [106] simulate pivoting-based attacks through a game-theoretic framework and propose some best practices based on their observations. Johnson et al. [107] present an original graph metric that quantifies whether granting a certain privilege to an employee may increase chances of pivoting in Windows domains.

Other research efforts related to pivoting detection make strong assumptions that are not applicable to real contexts. Some papers [95], [96] are oriented to detect pivoting-related activities involving the SSH protocol and brute-force password guessing, while we do not make any assumption on host and protocol types. The heuristics proposed in [97] are valid to detect one-step pivoting activities, but just those related to security alerts issued by a signature-based network intrusion detection system. We can detect pivoting paths of any length and do not leverage alerts of other defensive systems, although integrations are feasible. Some works focus on *lateral movement* [64] activities where attackers move within the network to get closer to their final target. For example, the paper in [108] proposes a game-theoretic framework that models attackers behavior and simulates network responses aiming to prevent attackers from reaching valuable hosts. The problem is that expert attackers can easily evade similar defensive schemes by acting differently from the expected model. The detection algorithm proposed by Fawaz et al. [109] requires analyses of huge amounts of host-based logs, which are hard to collect in large organization and may be easily altered by attackers because we remark that they control the hosts of the pivoting tunnel. Unlike these proposals, we consider network flows because they are easier to collect and store; moreover, our analysis requires less processing costs than investigations carried out on raw traffic data [83].

### 3.2.2 Problem description

In pivoting, attackers use a command propagation tunnel created among three or more internal hosts with the purpose of controlling a specific target. Past literature sometimes refers to pivoting with the term *island-hopping* [97], where attackers propagate commands through hosts in multiple LANs ("islands"). Many recent cyberattacks [90]–[93] used pivoting in their propagation phase. It is of paramount importance to find novel approaches for early detection of pivoting because it would prevent attackers from reaching their target and damaging an organization.

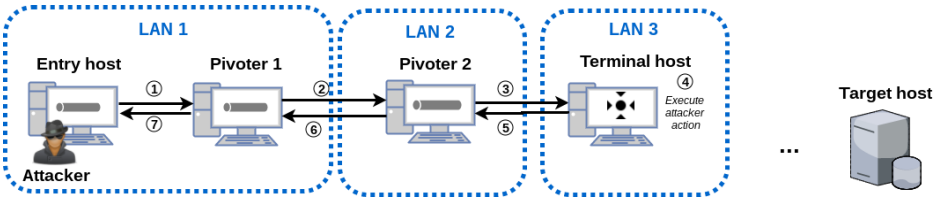



FIGURE 3.10: Example of pivoting activity.

Figure 3.10 reports an example of pivoting activity, where attackers have created a pivoting tunnel between four internal hosts belonging to three different LANs. From the *entry host*, malicious commands are forwarded through intermediate hosts (called *pivoters*) to the last host of the pivoting chain, called *terminal host*, which executes these commands and returns the results. This last host can be the final target or it can be used to further extend the pivoting tunnel.

After having established a foothold within the internal network of the target organization, attackers typically perform the following steps:

1. **Reconnaissance**: attackers gather information about neighbor hosts through some active (e.g., port-scan) or passive means (e.g., by looking at the `arp` table).

2. **Compromise**: attackers compromise another internal host using a known or zero-day vulnerability; in such a way, they can increase the length of the pivoting chain, and the newly compromised host will become the terminal host as in Figure 3.10.

3. **Pivoting**: the commands of the attackers are propagated from the entry host to the terminal host through the pivoting chain. Any command issued to the terminal host is executed and the results are forwarded towards the entry host through the pivoting chain.

The first two actions are part of *lateral movement* [64] where attackers assume control of other hosts to get closer to their final targets, and the *pivoting* is the actual command propagation activity through the tunnel of compromised hosts. To perform pivoting, attackers may adopt different protocols and tools, which can be either standard (e.g., SSH or netcat), or ad-hoc software designed to avoid easy detection (e.g., meterpreter [110]). Depending on the chosen protocols, payloads may or may not be subject to encapsulation. For example, if attackers adopt SSH, then data is encrypted and encapsulated within protocol-specific headers; if they use netcat, then the original payload is forwarded without modifications.

It is important to highlight that detection techniques based on signatures [111] cannot detect pivoting, because they only perform pattern matching against packet headers and payloads, which is not sufficient to detect active command propagation tunnels. Some parts of the lateral movement phase (i.e., reconnaissance and compromise) may be detected by signature-based systems, but only if attackers perform active reconnaissance and uses publicly disclosed exploits [97].

We focus on the detection of pivoting involving internal hosts. Looking for suspicious communications among the internal and external traffic is a different and widely investigated problem by the literature on intrusion detection through signature-based [81], [112], [113], anomaly-based [75], [114], protocol-specific [95], [115] and attack-specific proposals [59], [98]. The approaches can be combined for a more accurate detection, but their integration is out of the scope of this work.

There are several issues that make pivoting detection a tough problem. Command propagation is a rare event immersed in a huge amount of activities characterizing the network traffic of a medium-large organization. Since some pivoting activities are benign (e.g., for network administration), detection algorithms are affected by false alarms because malicious pivoting is an even rarer event. Finally, attackers can use a variety of tools and protocols in the attempt to evade detection. For these reasons, we propose an innovative approach that through the analysis of network flows identifies pivoting flow sequences and then integrates pivoting detection with the threat prioritization algorithm presented in Section 3.2.5. In such a way, we allow security analysts to inspect only few pivoting activities exhibiting the most suspicious behavior. Figure 3.11 outlines the main steps of the proposal.

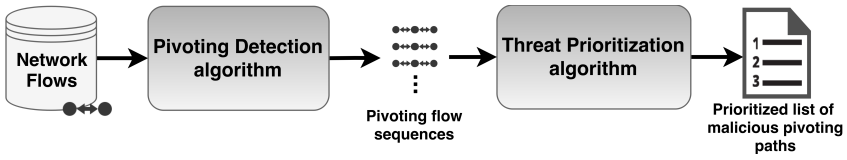



FIGURE 3.11: Overview of proposed method.

### 3.2.3 Pivoting detection algorithm

The proposed pivoting detection algorithm has the purpose of identifying all active pivoting tunnels within an internal network. This algorithm is integrated with a novel prioritization approach presented in Section 3.2.5.

We detect pivoting activities by searching for active command propagation tunnels. Network flows are the input data of our algorithm. A network flow represents a common way of summarizing traffic data [100] because it contains all information about connections, such as start-time, duration, transmitted bytes, involved IP addresses and port numbers. Using network flows instead of raw traffic data has the twofold benefit of simplifying data gathering and decreasing processing costs. A network flow $f$ can be defined as an ordered sequence:

$$f = (src, dst, p_{src}, p_{dst}, b_{in}, b_{out}, d, t) \tag{3.6}$$

where $src$ and $dst$ are the IP addresses of the source and destination hosts of the flow $f$; $p_{src}$ and $p_{dst}$ are the source and destination ports of their communications; $b_{in}$ and $b_{out}$ are the incoming and outgoing bytes; $d$ is the duration (by default 120$s$ [82]), and $t$ is the timestamp corresponding to the beginning of the communication. For each network flow, $src$ and $dst$ represent the flow direction with respect to the node that started the communication, even if they keep exchanging packets in both directions. Flow direction can be reliably computed by traditional network security appliances through the analysis of packet headers and timings [82].

From network flows it is possible to model communications among internal hosts as a temporal graph [116] corresponding to a time window $W$, where nodes correspond to internal hosts and directed temporal edges represent flows. Command propagations in pivoting tunnels are represented by network flows connecting pivoters sequentially and separated by a low latency. For this reason, we introduce the concept of *maximum propagation delay* $\varepsilon_{max}$, defined as the maximum amount of time that can pass between two consequent communications to consider them as a

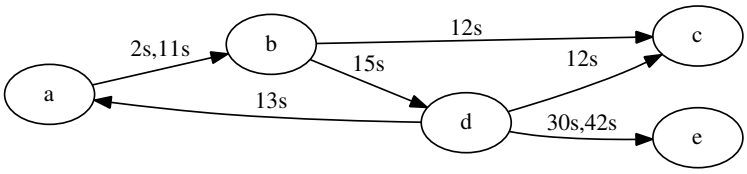



FIGURE 3.12: Temporal graph representation of network flows between five hosts (a,b,c,d,e).

part of the same pivoting activity.

We define a *pivoting flow sequence* $\mathcal{F}$ as an ordered set of flows $(f_1, f_2, ..., f_L)$, $L \in \mathbb{N}$, $L > 1$, where:

- all consecutive flows must be adjacent (*dst* of a flow is *src* for the following flow);
- all connected nodes appear only once in the sequence;
- consecutive flows are chronologically ordered;
- consecutive flows are separated by at most $\varepsilon_{max}$ time units.

We define a *pivoting path* $\mathcal{P}$ as an ordered set of hosts $(h_1, h_2, ..., h_N)$ for which at least one pivoting flow sequence exists. A pivoting path corresponds to one or more pivoting flow sequences, whereas each pivoting flow sequence corresponds to a specific pivoting path.

We illustrate some examples of flow sequences and pivoting paths by referring to the temporal graph shown in Figure 3.12. For the sake of clarity, multi-edges are represented by timestamp labels separated by a comma as in [117]. If we consider any value of $\varepsilon_{max} \geq 27s$, the temporal graph contains all the paths identified in Table 3.9. For the sake of simplicity, the represented flows only report *src*, *dst* and $t$.

We observe that some paths are subsets of other paths, and paths involving the same vertices can occur at different timestamps. If we consider a maximum propagation delay $\varepsilon_{max} = 5s$, the pivoting paths of interest are reported in Table 3.10. This propagation delay defines the admitted tolerance for considering two flows as part of the same pivoting path.

**Algorithm for pivoting detection** In Algorithm 1 we present a novel algorithm for pivoting detection that finds all the pivoting flow sequences within a temporal graph representing network communications among internal hosts. We observe that

TABLE 3.9: Example of pivoting paths and corresponding flow sequences from Figure 3.12 for $\varepsilon_{max} \geq 27s$.

| Path | Length | Flow sequences |
|---|---|---|
| a,b,d | 2 | (a,b,2s),(b,d,15s)<br>(a,b,11s),(b,d,15s) |
| a,b,c | 2 | (a,b,2s),(b,c,12s)<br>(a,b,11s),(b,c,12s) |
| b,d,e | 2 | (b,d,15s),(d,e,30s)<br>(b,d,15s),(d,e,42s) |
| a,b,d,e | 3 | (a,b,11s),(b,d,15s),(d,e,30s)<br>(a,b,11s),(b,d,15s),(d,e,42s)<br>(a,b,2s),(b,d,15s),(d,e,30s)<br>(a,b,2s),(b,d,15s),(d,e,42s) |

TABLE 3.10: Example of pivoting paths and corresponding flow sequences from Figure 3.12 for $\varepsilon_{max} = 5s$.

| Path | Length | Flow sequences |
|---|---|---|
| a,b,d | 2 | (a,b,11s),(b,d,15s) |
| a,b,c | 2 | (a,b,11s),(b,c,12s) |

**Algorithm 1:** Algorithm for pivoting detection.

**Input:** List of $m$ temporal edges corresponding to time window $W$ (*Flows*), maximum propagation delay $\varepsilon_{max}$, minimum incoming and outgoing bytes $B^{in}_{min}$ and $B^{out}_{min}$, maximum flow duration $\delta_{min}$, maximum pivoting path length $L_{max}$

**Output:** List of *pivoting flow sequences* of length $\geq 2$ (corresponding to *pivoting paths*)

```
// Initialization
PivotingSequences ← emptyList();
CandidateFlows ← emptyList();
for flow f in Flows do
    if (f.d ≥ δ_min) and (f.b_in ≥ B^in_min and f.b_out ≥ B^out_min) then
        Insert flow f in PivotingSequences;
        Insert flow f in CandidateFlows;
// Look for possible pivoting flow sequences of length ≥ 2
for flow sequence F in PivotingSequences do
    if length(F) ≥ L_max then
        break;
    FoundSequences ← ExtendPivotingSequence(F, CandidateFlows, ε_max)
    Include FoundSequences in PivotingSequences;
return List of elements in PivotingSequences with length ≥ 2;
// Function to find flow sequences of length (L+1) given a sequence F of
length L
Function ExtendPivotingSequence(F,CandidateFlows,ε_max)
    FoundSequences ← emptyList();
    h_F ← last host in pivoting flow sequence F
    t_F ← lastest timestamp of F
    FlowsWithinDelay ← BinarySearch(CandidateFlows[t_F : t_F + ε_max])
    for flow f in FlowsWithinDelay do
        if ((f.src equal to h_F) and (f.dst not in sequence F)) then
            NewSequence ← (sequence F with flow f);
            Insert NewSequence in FoundSequences;
    return FoundSequences;
```

once the pivoting flow sequences are found, it is immediate to enumerate the corresponding pivoting paths (as in Table 3.9). The pivoting detection algorithm takes the following input parameters:

- the temporal graph of internal network communications built over the time window $W$, represented as the list of its $m$ edges. Without loss of generality, we consider that the list of edges (representing network flows) is ordered according to their timestamp;
- the maximum propagation delay $\varepsilon_{max}$, which is the maximum amount of time tolerated between two consecutive flows to be part of the same pivoting flow sequence;
- the minimum flow duration $\delta_{min}$, which is the minimum duration of a network flow to consider it part of a pivoting path;
- the minimum incoming and outgoing bytes $B^{in}_{min}$ and $B^{out}_{min}$, which is the minimum number of bytes transmitted in a network flow to consider it as a possible portion of pivoting;
- the maximum pivoting sequence length $L_{max}$, which is the maximum length of the pivoting paths that the algorithm will search for. This parameter may be seen as a terminating condition.

In Table 3.11, we report the most relevant symbols used in this section.

TABLE 3.11: Symbol table.

| Symbol | Description |
|---|---|
| $\mathcal{F}$ | Pivoting flow sequence. |
| $\mathcal{P}$ | Pivoting path. |
| $\mathcal{F}_{\mathcal{P}}$ | Set of pivoting flow sequences associated with path $\mathcal{P}$. |
| $W$ | Time window (e.g. 1 hour) analyzed by the pivoting detection algorithm. |
| $m$ | Number of network flows within $W$. |
| $B^{in/out}_{min}$ | Minimum number of incoming/outgoing bytes of a network flow for building flow sequences. |
| $\delta_{min}$ | Minimum network flow duration for building flow sequences. |
| $L$ | Length of a pivoting flow sequence (number of edges). |
| $L_{max}$ | Maximum pivoting flow sequence length tolerated by the pivoting detection algorithm. |
| $\varepsilon$ | Command propagation delay in a pivoting tunnel. |
| $\varepsilon_{max}$ | Maximum command propagation delay tolerated by the pivoting detection algorithm. |

Algorithm 1 can be divided into two main phases: *initialization*, and *extending pivoting sequences*.

1. **Initialization**: This phase takes the $m$ edges of the temporal graph as its input and stores them into two separate lists: *PivotingSequences* contains the list of all the sequences of length $L = 1$ and is used to store new sequences as the algorithm proceeds; *CandidateFlows* contains the flows that are evaluated for extending the existing sequences. In line 5, an initial pruning condition is reported as a function of the inputs: for the analysis of pivoting detection, only flows with at least $\delta_{min}$ duration and $B^{in}_{min}$ and $B^{out}_{min}$ bytes are considered for this analysis. If $B^{in}_{min} = B^{out}_{min} = \delta_{min} = 0$, then the $m$ input flows are considered without pruning. Conditions in line 5 are considered because any pivoting activity has to last for some time (since attackers are interacting through some command interpreter), and has to transfer some bytes in both directions (the propagated command, and the corresponding response).

2. **Extending pivoting sequences**: This is the core phase of the algorithm that repeats a common function multiple times. In line 9 there is a cycle that iterates through all the *PivotingSequences* found so far. Initially, it considers only sequences of length $L = 1$ corresponding to the flows themselves. For each flow sequence $\mathcal{F}$ of length $L$ in the *PivotingSequences*, the algorithm executes a function that searches for all possible flow sequences of length $(L + 1)$ that extend $\mathcal{F}$. This function is called *ExtendPivotingSequence* (line 16), and takes as its input a pivoting flow sequence $\mathcal{F}$, a maximum propagation delay $\varepsilon_{max}$ and the *CandidateFlows*.

The algorithm extracts the last host in the pivoting flow sequence $h_{\mathcal{F}}$ and its most recent timestamp $t_{\mathcal{F}}$. Then, it performs a binary search in line 20 on the *CandidateFlows* in the time interval $[t_{\mathcal{F}}, t_{\mathcal{F}} + \varepsilon_{max}]$. This binary search is admissible because the *CandidateFlows* are listed in sorted order of timestamp. The results of the binary search are stored in *FlowsWithinDelay*, which are then iterated to check whether a flow $f \in$ *FlowsWithinDelay* can extend the currently analyzed pivoting flow sequence $\mathcal{F}$. There are two conditions in line 22 that must be verified for $\mathcal{F}$ to be extended with $f$:

- the source node of $f$ must be equal to $h_{\mathcal{F}}$ (which is the last host of $\mathcal{F}$);

- the destination node of $f$ must not be in sequence $\mathcal{F}$ (it must be a new host in the sequence).

Whenever both conditions are satisfied, a new pivoting sequence is added to the *FoundSequences* list, and then returned. By repeating the cycle on line 9 until a pivoting path of length $L_{max}$ is reached (line 11) or until no flow sequences can be further extended, all the pivoting sequences of length at most $L_{max}$ are enumerated. We observe that in line 14 the algorithm returns only pivoting sequences of length $L \geq 2$, because those with $L = 1$ are the flows themselves.

Let us describe the algorithm by considering the example graph in Figure 3.12 and the function *ExtendPivotingSequence* with parameters $\varepsilon_{max} = 5s$, flow sequence $\mathcal{F} = ((a, b, 11s))$ of length $L = 1$. For the sake of simplicity, we report only $(src, dst, t)$ of each flow. After the binary search in line 20, the flows within delay $[11s, 11s + 5s]$ are: $(a, b, 11s), (b, c, 12s), (d, a, 13s), (d, c, 12s), (b, d, 15s)$. Only the flows starting with $b$ and not containing $a$ can extend $\mathcal{F} = ((a, b, 11s))$. Only the flows $(b, c, 12s)$ and $(b, d, 15s)$ satisfy both conditions, and hence the new extended sequences found are $((a, b, 11s), (b, c, 12s))$ and $((a, b, 11s), (b, d, 15s))$. It is immediate to get the corresponding pivoting paths $(a, b, c)$ and $(a, b, d)$.

### 3.2.4 Computational complexity

We evaluate the computational complexity of the proposed pivoting detection algorithm and compare it against two alternatives: subgraph isomorphism and brute force enumeration algorithms.

**Subgraph isomorphism** If we consider a pivoting path (or sequence) as a subgraph, then the pivoting detection problem could be seen as that of finding occurrences of specified subgraphs within a graph. This approach, which is known as the subgraph isomorphism problem, is *NP-complete* [118] even for static graphs without temporal edges. Hence, it is not a viable solution.

**Brute force enumeration** A possible approach to pivoting detection is to enumerate all possible sequences that can be derived from existing flows, and then evaluate whether these flow sequences are consistent with a maximum propagation delay

$\varepsilon_{max}$. The complexity of this brute force enumeration algorithm is:

$$\sum_{L=1}^{m} \left[ \binom{m}{L} \cdot L! \right] \sim \Omega(2^m) \tag{3.7}$$

where $\binom{m}{L}$ represents the number of possible combinations (subsets) of length $L$ given $m$ flows; $L!$ denotes all possible permutations of such elements, and counts the number of possible re-orderings of a length $L$ path. The computational complexity is more than exponential in the number of edges $m$, and is always higher than $\Omega(2^m)$. If we simplify the problem by considering only contiguous subsequences as in [117], then the complexity diminishes (that is, $L^2$ instead of $L!$ as a multiplicative factor in Eq. 3.7), but still remains more than exponential in the number of edges $m$.

**Pivoting detection** The algorithm for pivoting detection proposed in this paper has an overall worst-case time complexity of:

$$\mathcal{O}(m^{L_{max}} \cdot \log_2(m) \cdot \tau) \tag{3.8}$$

where $m$ is the number of network flows within the window $W$, $L_{max}$ is the maximum pivoting tunnel length we are looking for, and $\tau$ is the maximum number of flows between any $[t, t + \varepsilon_{max}]$ interval. For small values of $\varepsilon_{max}$ representing the common case, the parameter $\tau \ll m$, hence the complexity may be simplified as follows:

$$\mathcal{O}(m^{L_{max}} \cdot \log_2(m)) \tag{3.9}$$

For the demonstration, we assume that the $m$ flows arrive in order of timestamp. The initialization phase requires $\mathcal{O}(m)$ operations to initialize the list of flow sequences with length $L = 1$. The computational complexity of the initialization phase is then:

$$\mathcal{O}(m) \tag{3.10}$$

The number of iterations of the core part starting on line 9 of the Algorithm 1 depends on the total number of possible flow sequences with length between 1 and $L_{max}$. We recall that the flow sequences are included in the *PivotingSequences* list in Algorithm 1 while they are being found.

Let $k_i$ be the number of $i$-length pivoting flow sequences that can be seen as the number of permutations without repetition of $m$ flows in ordered groups of $i$ elements [119]:

$$k_i = \frac{m!}{(m-i)!} \tag{3.11}$$

$$= m \cdot (m-1) \cdot ... \cdot (m-i+1) \tag{3.12}$$

$$= \mathcal{O}(m^i) \tag{3.13}$$

As we have to consider the total number of flow sequences of length $i = \{1, 2, ..., L_{max}\}$, we analyze $\sum_{i=1}^{L_{max}}(m^i)$ sequences, which can be approximated to the following known *geometric series* [120]:

$$\sum_{i=0}^{L_{max}} m^i = \frac{1 - m^{L_{max}+1}}{1-m} = \mathcal{O}(m^{L_{max}}) \tag{3.14}$$

Then, we have to consider that for each iteration on line 9, the function *ExtendPivotingPath* is executed.

In the *ExtendPivotingPath* function, we have a binary search in the sorted list of $m$ flows, that takes $\mathcal{O}(\log_2(m))$ time. Then, if $\tau$ is the maximum number of flows between any $t$ and $t + \varepsilon_{max}$ timestamps, we have an overall time complexity of the function *ExtendPivotingPath* equal to:

$$\mathcal{O}(\log_2(m) \cdot \tau) \tag{3.15}$$

From Eq. 3.7 and Eq. 3.13 we obtain a worst-case complexity of:

$$\mathcal{O}(m + m^{L_{max}} \cdot \log_2(m) \cdot \tau) \tag{3.16}$$

where $L_{max} \ll m$ and $\tau \ll m$ (for small values of $\varepsilon_{max}$).

We conclude this section with some observations. Although the computational complexity of the Eq. 3.8 remains high, in Section 3.2.6 we verify through a large set of experiments that the proposed algorithm is efficient and applicable to real contexts. In particular, we can limit the propagation time $\varepsilon_{max}$ because in many real attacks [90]–[93], [110] commands are propagated as fast as possible, thus leading to values of $\varepsilon$ of few milliseconds. Although advanced attackers could introduce some delays to evade detection, values of $\varepsilon$ higher than few seconds make pivoting

unpractical. Moreover, post-exploitation activities, such as command execution or data exfiltration, require flows lasting for at least few seconds, and comprising at least some tens of bytes. These conditions allow the pruning phase of the pivoting detection algorithm to reduce up to 90% the number of edges with respect to the worst case. As shown in Section 3.2.6, in practice the computational cost decreases significantly, and the execution time in a COTS machine renders the proposed algorithm suitable even for large network environments.

### 3.2.5 Threat prioritization

Some pivoting activities may be benign, hence it is important to avoid or limit the false alarms and let the security analysts focus on the most likely threats. For this reason, we integrate Algorithm 1 with a threat prioritization approach that ranks the detected paths on the basis of their suspicious factors. We define the following threat indicators for each detected pivoting path:

- path novelty,
- reconnaissance activities,
- involved LANs,
- use of uncommon ports,
- anomalous data transfers.

We now discuss these scores individually, and then show our approach to combine them in an overall threat score.

**Path novelty** The appearance of paths involving hosts that have never previously communicated may be related to some unauthorized activities, as also highlighted in [64]. Hence, the score of the communication path novelty should be high if all the communications occurring among the involved hosts and their ports have never been seen in the recent past. Taking into consideration the pivoting path $\mathcal{P}$, we consider the set $\mathcal{F}_\mathcal{P}$ of the flow sequences $\mathcal{F}$ associated with $\mathcal{P}$ (see Table 3.10), we define the score $N(\mathcal{P})$ as the highest percentage of novel communications between hosts involved in the path $\mathcal{P}$:

$$N(\mathcal{P}) = \max_{\mathcal{F} \in \mathcal{F}_\mathcal{P}} \left\{ \left( \frac{len(\mathcal{F}) - len(S_\mathcal{F})}{len(\mathcal{F}) + 1} \right) \right\} \tag{3.17}$$

where $N(\mathcal{P}) \in [0,1]$, $len(\mathcal{F})$ is the number of edges (flows) in $\mathcal{F}$, and $S_{\mathcal{F}}$ is the longest common subsequence of $\mathcal{F}$ found among the flow sequences related to the path detected in the recent past runs of Algorithm 1. We define a common subsequence as any subsequence of $\mathcal{F}$ where all flows have the same *src* and *dst*, and either the same $p_{src}$ or $p_{dst}$ ports but not necessarily both. This latter condition is useful because in client-server communications it is common to have different clients ports over time but only one server port. If $N(\mathcal{P}) = 1$, all communications within the path are novel; if $N(\mathcal{P}) = 0$, all communications have occurred in the past.

**Reconnaissance activities** Attackers perform reconnaissance to look for neighbor hosts that they can compromise. The paths including hosts involved in such activities should have a high risk score $R(\mathcal{P})$ as defined below:

$$R(\mathcal{P}) = \frac{\text{n.hosts of } \mathcal{P} \text{ involved in recon.}}{len(\mathcal{P})} \tag{3.18}$$

where $R(\mathcal{P}) \in [0,1]$, and $len(\mathcal{P})$ is the number of hosts in the pivoting path $\mathcal{P}$. This score reports the percentage of hosts within a path that have been involved in reconnaissance activities. As the reconnaissance detection problem is out of the scope of this work, we rely on existing techniques, such as those proposed in [83].

**Involved LANs** Reaching internal hosts not easily accessible from the external network is one of the main reasons leading attackers to adopt pivoting techniques [121]. Thus, paths including hosts of different LAN segments may be a risk signal that should be prioritized. We define the $Z(\mathcal{P})$ score as:

$$Z(\mathcal{P}) = \frac{\text{n.hosts of } \mathcal{P} \text{ in different LANs}}{len(\mathcal{P})} \tag{3.19}$$

where $Z(\mathcal{P}) \in [0,1]$, and $len(\mathcal{P})$ is the number of hosts in a path $\mathcal{P}$. This score denotes the percentage of different LANs included in a path, where $Z(\mathcal{P}) = 0$ if all hosts belong to the same LAN, and $Z(\mathcal{P}) = 1$ if each host belongs to a different LAN.

**Use of uncommon ports** Some ports, such as number 22 (SSH), 23 (Telnet), 443 (SSLH Multiplexing), 3389 (Windows remote desktop protocol) or 5938 (Team Viewer),

are commonly used for benign tunneling activities. Others are uncommon and possibly risky. We consider an edge (flow) to be uncommon if both $p_{src}$ and $p_{dst}$ are uncommon ports. The related score is computed for each path as the maximum of the following ratio:

$$S(\mathcal{P}) = \max_{\mathcal{F} \in \mathcal{F}_{\mathcal{P}}} \left\{ \frac{\text{n.flows in } \mathcal{F} \text{ with uncomm. ports}}{len(\mathcal{F})} \right\} \tag{3.20}$$

where $S(\mathcal{P}) \in [0,1]$, $\mathcal{F}_{\mathcal{P}}$ is the set of flow sequences associated with the pivoting path $\mathcal{P}$, and $len(\mathcal{F})$ is the number of flows contained in $\mathcal{F}$. This score represents the maximum percentage of uncommon ports used in any communication of a pivoting path. For example, $S(\mathcal{P}) = 0.5$ when there is at least one flow sequence of $\mathcal{P}$ whose 50% of the flows use uncommon ports.

**Anomalous data transfers** Attackers often leverage pivoting tunnels to perform data exfiltrations. A method to prioritize such activities is to verify whether the amount of data exchanged among the hosts of a given path, considering all its corresponding flow sequences, has increased with the respect to the recent past. To this purpose, we define the $E(\mathcal{P})$ score as follows:

$$E(\mathcal{P}) = \begin{cases} 1, & \text{if a data transfer anomaly is detected} \\ 0, & \text{otherwise} \end{cases} \tag{3.21}$$

To detect whether an anomaly occurred, we apply the boxplot rule [120] to the exchanged traffic among hosts.

**Overall threat score** We compute an overall threat score $\mathcal{T}(\mathcal{P})$ of the path $\mathcal{P}$ as the sum of the indicators (we omit $\mathcal{P}$ for readability) where higher values denote paths that are more likely malicious:

$$\mathcal{T} = (N + R + Z + S + E), \mathcal{T} \in [0,5] \tag{3.22}$$

All pivoting paths identified with the proposed detection algorithm are ranked according to their threat score. Section 3.2.6 validates the effectiveness of our threat prioritization score in different scenarios and with respect to other baseline algorithms.

### 3.2.6 Evaluation results

We demonstrate the effectiveness and feasibility of the proposed pivoting detection and threat prioritization algorithms when deployed on a real network of a large organization. Hence, we perform our experiments on a test dataset consists of real network traffic captured by probes situated in a large enterprise, collected over a time span of over five months (160 days). The dataset is composed by more than half a billion (657 213 849) network flows describing the internal network communications among 8 198 hosts. This scenario can be modeled as a graph where the number of temporal edges (flows) is significantly larger than the number of nodes (hosts).

We organize our experiments through the following outline:

- we demonstrate that our algorithm is able to detect all existing pivoting activities.
- We inject different pivoting attacks into real network traffic, and show that our algorithms are able to detect and prioritize all the injected threats.
- We analyze the additional challenge posed by possible evasion strategies that may be adopted by expert attackers; we show the robustness of our proposals even against these attempts.
- We compare our solutions against two related algorithms.
- Finally, we measure the execution times of the proposed pivoting detection algorithm with the goal of demonstrating its applicability to real contexts.

We remark that we publicly released a snippet of the data used for these experiments[2].

**Pivoting detection and prioritization** We evaluate how the proposed algorithms for pivoting detection and threat prioritization perform under varying attack conditions. We initially execute the pivoting detection algorithm on the test dataset once per hour ($W = 60$ minutes) until each of the 160 days of the dataset has been analyzed. In typical pivoting activities the propagation delay is in the order of milliseconds [90]–[93], hence we initially consider the performance with $\varepsilon_{max} = 1s$ and no limit on the maximum length of the pivoting path ($L_{max} = \infty$).

[2]Data Snippet: `https://weblab.ing.unimore.it/people/apruzzese/datasets/pivoting.html`

Manual inspection of each path detected by the algorithm reveals that they are all true and benign pivoting activities corresponding to SSH tunneling and proxying. As the algorithm has been able to identify all tunneling activities occurring within the network for propagation delay $\varepsilon \leq 1s$, we can conclude that, in absence of attacks with propagation delay $\varepsilon > 1s$, it achieves 100% Accuracy.

As no malicious pivoting activity occurs in the testbed considered for evaluation, we emulate five classes of pivoting attacks, which are summarized in Table 3.12, and inject them into the real traffic traces. AC1 refers to an attacker creating an SSH tunnel and performing port-scans to detect the next victim, which is compromised after a brute-force SSH password guessing. In AC2, the attacker uses the SSH protocol for exchanging 30MB of data, but no brute-forcing nor active reconnaissance is performed. AC3, AC4 and AC5 are performed through the Metasploit toolset. The Attack Classes in Table 3.12 consider also increasing length of the pivoting chain to show that our algorithm can still detect longer chains and how this affects prioritization.

TABLE 3.12: Pivoting Attack Classes.

| Attack Class | Vector | Len | Recon | LANs | Data |
|---|---|---|---|---|---|
| **AC1** | SSH | 2 | ✓ | 2 | 10 MB |
| **AC2** | SSH | 2 | ✗ | 2 | 30 MB |
| **AC3** | Metasploit | 4 | ✓ | 5 | 100 MB |
| **AC4** | Metasploit | 3 | ✗ | 4 | $<$ 1 MB |
| **AC5** | Metasploit | 4 | ✗ | 1 | 5 MB |

The network traffic generated by each Attack Class is collected as netflow data and then injected into each day of the real traffic dataset. This injection process is realized through an ad-hoc script that merges each attack flow with the dataset by replacing the IP addresses of the virtual pivoter hosts with those of real hosts. To avoid bias when choosing real hosts as pivoters, we identify two main sets of hosts: $\omega$ and $\beta$. The $\omega$ set contains the hosts with a total number of communications above the 95-th percentile, while the $\beta$ set those below the 5-th percentile. In other words, $\omega$ and $\beta$ sets contain high-activity and low-activity hosts, respectively. Since it is easier to prioritize pivoting activities in $\beta$ because it contains hosts that rarely interact, we can observe that $\omega$ and $\beta$ represent a worst- and best-case scenarios for prioritization, respectively. We execute the pivoting detection algorithm on the entire injected dataset for $\varepsilon_{max} = 1s$, no bounds on the maximum length of the pivoting paths ($L_{max} = \infty$), and $W = 60$ minutes. Our algorithm is able to correctly detect

all the injected pivoting paths thus achieving a 100% Precision. Unfortunately, in addition to malicious pivoting paths, the algorithm detects some benign pivoting activities present in the dataset. This result motivates our choice of relying on the threat prioritization algorithm presented in Section 3.2.5. The goal now is to assess whether this algorithm is actually able to prioritize malicious pivoting activities by placing them at the top positions in the ranking. Table 3.13 reports the results of the threat prioritization algorithm for each Attack Class and each set of injected hosts. Each row displays the average rank and its standard deviation obtained by each Attack Class after considering all the pivoting paths found within each day in the dataset. Lower values of rank correspond to higher prioritization. For example, if a pivoting path is ranked in position 1, it implies that it has the highest likelihood of being malicious among all pivoting paths detected in the same day. We observe that all attacks of each class have always an average rank lower than 2 even for the worst-case high-activity hosts belonging to the $\omega$ set. Moreover, a standard deviation that is always below 1.4 indicates that the vast majority of malicious pivoting paths are ranked in the top positions. These important results show that the threat prioritization algorithm is capable of assigning a high, stable rank to all the considered Attack Classes, thus ensuring the prioritization of the malicious pivoting activities.

TABLE 3.13: Performance of the threat prioritization algorithm.

| Attack Class | average rank | standard deviation |
|---|---|---|
| **AC1** ($\omega$) | 1.38 | 1.32 |
| **AC1** ($\beta$) | 1.17 | 0.72 |
| **AC2** ($\omega$) | 2.01 | 1.18 |
| **AC2** ($\beta$) | 1.55 | 1.04 |
| **AC3** ($\omega$) | 1.00 | 0.00 |
| **AC3** ($\beta$) | 1.00 | 0.00 |
| **AC4** ($\omega$) | 1.13 | 0.51 |
| **AC4** ($\beta$) | 1.14 | 0.68 |
| **AC5** ($\omega$) | 1.15 | 0.83 |
| **AC5** ($\beta$) | 1.14 | 0.78 |

We can also observe that AC3 is always ranked first with standard deviation equal to zero, because the attacks in this class span over five LANs and involve about 100MB of exfiltration traffic (see Table 3.12). On the other hand, the attacks in AC1 and AC2 exhibit slightly lower ranks. This is motivated by the fact that these attacks are performed through SSH, which is a standard protocol often used by system administrators for legitimate tunneling operations; as expected, the threat scores of AC1 and AC2 are slightly lower than those performed through Metasploit

(see Section 3.2.5). Nevertheless, the average ranks of these two classes are always lower than 3.

**Evasion techniques** Our pivoting detection and prioritization algorithms are able to detect and properly prioritize malicious pivoting tunnels occurring in internal networks. We now further stress the algorithms by considering skilled attackers that employ evasion techniques based on the introduction of some propagation delay in their pivoting communications. Detecting these stealthy attacks requires that the algorithm works with higher values of the parameter $\varepsilon_{max}$. Increasing this parameter has two consequences on the results of the detection algorithm: its execution times increase; it may label as pivoting activities some flow sequences that pertain to normal traffic, that is, it may be affected by false positives. For these reasons, it is of paramount importance to evaluate the robustness of the detection algorithm against evasion techniques of expert attackers. To this purpose, we repeat the attacks of Table 3.12 by adding the propagation delays reported in Table 3.14, and then inject the delayed attacks in each day of the entire dataset. For the sake of completeness, we also consider delays equal or longer than 10 seconds, although they are unpractical in reality. The motivation is simple: if for example the attackers want to perform an action in a pivoting chain of 5 hosts with a delay of $\varepsilon = 25$s, they should wait for $5\,hosts \times 25s \times 2\,directions$, which is more than four minutes delay for each issued command.

TABLE 3.14: Emulated propagation delays $\varepsilon$ for the Attack Classes.

| | **Attack Class** | | | | |
|---|---|---|---|---|---|
| | **AC1** | **AC2** | **AC3** | **AC4** | **AC5** |
| **Delay** | 2$s$ | 4$s$ | 8$s$ | 10$s$ | 15$s$ |

We then execute the detection algorithm on the injected dataset for values of $\varepsilon_{max}$ up to 30s and we do not set any bound on the maximum length of the pivoting path ($L_{max} = \infty$), while the size of the time-window is still set to $W = 60$ minutes. The detection results are reported in Table 3.15. From this table we can conclude that our detection algorithm is able to detect all the Attack Classes when it is executed with an adequate value of $\varepsilon_{max}$.

We then execute the threat prioritization algorithm and report in Table 3.16 the daily average and standard deviation of the rank obtained by attacks belonging to

TABLE 3.15: Pivoting attack detection for increasing $\varepsilon_{max}$.

| Attack Class | 1s | 5s | 10s | 15s | 20s | 25s | 30s |
|---|---|---|---|---|---|---|---|
| **AC1** | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **AC2** | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **AC3** | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **AC4** | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **AC5** | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ |

each Attack Class over the entire injected dataset. Rows refer to the attacks belonging to different Attack Classes and columns to different values of the parameter $\varepsilon_{max}$. Each cell reports the average rank and its standard deviation between parentheses. We observe that the average rank produced by our algorithm is always lower than 3 and its standard deviation is always lower than 2.

TABLE 3.16: Threat prioritization: average ranking for increasing $\varepsilon_{max}$.

| Attack Class | 1s | 5s | 10s | 15s | 20s | 25s | 30s |
|---|---|---|---|---|---|---|---|
| **AC1** ($\omega$) | ✗ | ✓ 1.48 (1.67) | ✓ 1.55 (1.84) | ✓ 1.48 (1.58) | ✓ 1.62 (1.91) | ✓ 1.65 (1.93) | ✓ 1.69 (1.98) |
| **AC1** ($\beta$) | ✗ | ✓ 1.21 (1.09) | ✓ 1.21 (1.12) | ✓ 1.21 (1.10) | ✓ 1.21 (0.92) | ✓ 1.21 (0.93) | ✓ 1.21 (0.99) |
| **AC2** ($\omega$) | ✗ | ✓ 2.11 (1.23) | ✓ 2.24 (1.26) | ✓ 2.27 (1.46) | ✓ 2.52 (1.57) | ✓ 2.65 (1.66) | ✓ 2.80 (1.94) |
| **AC2** ($\beta$) | ✗ | ✓ 1.61 (1.11) | ✓ 1.72 (1.19) | ✓ 1.81 (1.34) | ✓ 2.04 (1.29) | ✓ 2.09 (1.54) | ✓ 2.21 (1.65) |
| **AC3** ($\omega$) | ✗ | ✗ | ✓ 1.00 (0.00) | ✓ 1.00 (0.00) | ✓ 1.00 (0.00) | ✓ 1.00 (0.00) | ✓ 1.00 (0.00) |
| **AC3** ($\beta$) | ✗ | ✗ | ✓ 1.00 (0.00) | ✓ 1.00 (0.00) | ✓ 1.00 (0.00) | ✓ 1.00 (0.00) | ✓ 1.00 (0.00) |
| **AC4** ($\omega$) | ✗ | ✗ | ✓ 1.26 (0.86) | ✓ 1.26 (1.14) | ✓ 1.21 (1.31) | ✓ 1.21 (1.00) | ✓ 1.21 (1.63) |
| **AC4** ($\beta$) | ✗ | ✗ | ✓ 1.21 (0.75) | ✓ 1.21 (1.06) | ✓ 1.17 (1.23) | ✓ 1.17 (1.32) | ✓ 1.17 (1.37) |
| **AC5** ($\omega$) | ✗ | ✗ | ✗ | ✓ 1.26 (1.16) | ✓ 1.21 (1.44) | ✓ 1.21 (1.56) | ✓ 1.21 (1.86) |
| **AC5** ($\beta$) | ✗ | ✗ | ✗ | ✓ 1.21 (1.15) | ✓ 1.17 (1.28) | ✓ 1.17 (1.29) | ✓ 1.17 (1.54) |

TABLE 3.17: Detection rate in top 5 for increasing $\varepsilon_{max}$.

| Attack Class | 1s | 5s | 10s | 15s | 20s | 25s | 30s |
|---|---|---|---|---|---|---|---|
| **AC1** ($\omega$) | ✗ | 98.1% | 97.5% | 97.5% | 97.0% | 97.0% | 97.0% |
| **AC1** ($\beta$) | ✗ | 98.1% | 98.1% | 98.1% | 98.1% | 97.5% | 97.5% |
| **AC2** ($\omega$) | ✗ | 94.4% | 94.4% | 94.4% | 93.8% | 93.8% | 93.8% |
| **AC2** ($\beta$) | ✗ | 97.5% | 97.0% | 97.0% | 95.0% | 95.0% | 95.0% |
| **AC3** ($\omega$) | ✗ | ✗ | 100.0% | 100.0% | 100.0% | 100.0% | 100.0% |
| **AC3** ($\beta$) | ✗ | ✗ | 100.0% | 100.0% | 100.0% | 100.0% | 100.0% |
| **AC4** ($\omega$) | ✗ | ✗ | 98.1% | 98.1% | 98.8% | 98.8% | 98.8% |
| **AC4** ($\beta$) | ✗ | ✗ | 99.4% | 99.4% | 98.8% | 98.8% | 98.8% |
| **AC5** ($\omega$) | ✗ | ✗ | ✗ | 98.1% | 98.8% | 98.8% | 98.8% |
| **AC5** ($\beta$) | ✗ | ✗ | ✗ | 99.4% | 99.4% | 98.8% | 98.8% |

The accuracy of the prioritization algorithm is evaluated by denoting the *detection rate* as the percentage of days in which an injected Attack Class has been prioritized within the top 5 threats. Table 3.17 reports the detection rates for increasing values of $\varepsilon_{max}$. The best results are achieved on attacks of type AC3, where the algorithm always prioritizes the injected attacks within the top 5. As expected, the lowest detection rate (between 93% and 97%) is obtained for AC2 attacks because

they represent a stealthy activity with standard SSH protocol and no reconnaissance (see Table 3.12). We also recall that $\omega$ represents the set of high-activity internal hosts, which are much more challenging to prioritize. Overall, we can be satisfied by the results in Table 3.17 because our algorithm prioritizes the large majority of Attack Classes with detection rates between 97% and 99%.

We can motivate these results by considering that the proposed approach integrates a *pivoting detection* with a *threat prioritization* phase. If the value of $\varepsilon_{max}$ is high enough to tolerate the propagation delay inserted by the attackers, then the pivoting detection algorithm achieves 100% Recall, that is, all pivoting tunnels are detected. In particular, if $\varepsilon_{max} = 1s$ then all the detected flow sequences belong to actual pivoting paths, therefore achieving 100% Precision and 100% Recall. If we set $\varepsilon_{max} = 1s$ and the attackers perform some evasion techniques, the algorithm may be affected by some false negatives. In these cases, detection of evasive pivoting tunnels is achieved by setting $\varepsilon_{max} > 1s$ because any flow sequence with a propagation delay greater than $1s$ is either a false positive or an evasive pivoting attack, and is never benign because benign operations have short latencies. To overcome the limitations of the pivoting detection algorithm in case of evasive attackers, we designed the threat prioritization algorithm to rank the pivoting tunnels most likely related to threats in top positions, despite the presence of benign tunnels and false positives. We can conclude that the combination of the proposed pivoting detection and threat prioritization algorithms allows effective triage of even the most evasive attackers, as demonstrated by the results in Table 3.16.

**Comparison with other detection algorithms** To the best of our knowledge, this is the first approach targeted to pivoting detection and threat prioritization that relies solely on network flows without any assumption on protocols and hosts. Nonetheless, we find meaningful to compare our solution with two algorithms relying on network flows, *SSHC* [95] and *WHL* [59], which we consider the most related to our approach. SSHC detects SSH-based attacks involving port-scans and brute-force password guessing; when these actions are performed through a remotely controlled internal host, they are true pivoting attacks. WHL evaluates variations in the graph of internal communications to detect malware propagations; in this case, rapid malware propagation through different hosts can be modeled as a pivoting attack.

Comparative results are presented in Table 3.18, where each row refers to a specific Attack Class. For this set of experiments we assume an advanced attacker that is trying to evade detection by applying the delays described in Table 3.14. Each column of Table 3.18 summarizes the results achieved by a different detection algorithm. The first column refers to the detection and prioritization algorithms, in which $\varepsilon_{max} = 15$ seconds, $W = 60$ minutes and $L_{max} = \infty$. The last two columns refer to the detection results obtained by SSHC and WHL. Detected and not detected attacks are marked by ✓ and ✗ symbol, respectively. The proposed algorithm detects all pivoting paths, but it is also able to prioritize those representing real threats. For this reason, in Table 3.18 we also include the daily average rank and standard deviation associated to the attacks belonging to the same class. Since SSHC and WHL perform just detection with no prioritization, we report whether the injected pivoting attack has been detected or not.

TABLE 3.18: Comparison of detection algorithms.

| Attack Class | Alg ($\varepsilon_{max}$=15s) | SSHC | WHL |
|---|---|---|---|
| **AC1** ($\omega$) | ✓ 1.48 (1.58) | ✓ | ✓ |
| **AC1** ($\beta$) | ✓ 1.21 (1.10) | ✓ | ✓ |
| **AC2** ($\omega$) | ✓ 2.27 (1.46) | ✗ | ✗ |
| **AC2** ($\beta$) | ✓ 1.81 (1.34) | ✗ | ✗ |
| **AC3** ($\omega$) | ✓ 1.00 (0.00) | ✗ | ✓ |
| **AC3** ($\beta$) | ✓ 1.00 (0.00) | ✗ | ✓ |
| **AC4** ($\omega$) | ✓ 1.26 (1.14) | ✗ | ✗ |
| **AC4** ($\beta$) | ✓ 1.21 (1.06) | ✗ | ✗ |
| **AC5** ($\omega$) | ✓ 1.26 (1.16) | ✗ | ✗ |
| **AC5** ($\beta$) | ✓ 1.21 (1.15) | ✗ | ✗ |

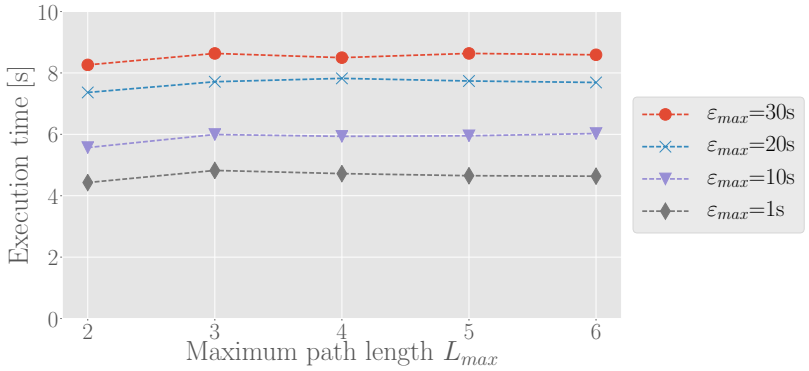
From these results, we can observe that attacks in AC1 can be detected by all three approaches. Our algorithm detects and prioritizes it within the top 5 hosts; SSHC recognizes the attack associated with a brute-force attempt, and WHL detects a sudden change in the communication structure due to the reconnaissance performed after pivoting. AC3 is always detected by our approach and by WHL, however it is not detected by SSHC because it does not involve brute-forcing of an individual host, but a reconnaissance on multiple hosts. Finally, we can observe that attacks in AC2, AC4 and AC5 cannot be detected by SSHC and WHL because they involve more subtle pivoting activities and they include a propagation delay between 2$s$ and 15$s$ introduced by the attackers to improve his chances of evasion.

We can conclude that our algorithm is by far the most effective in detection and prioritization of pivoting activities even if the attackers adopt evasive attack strategies.

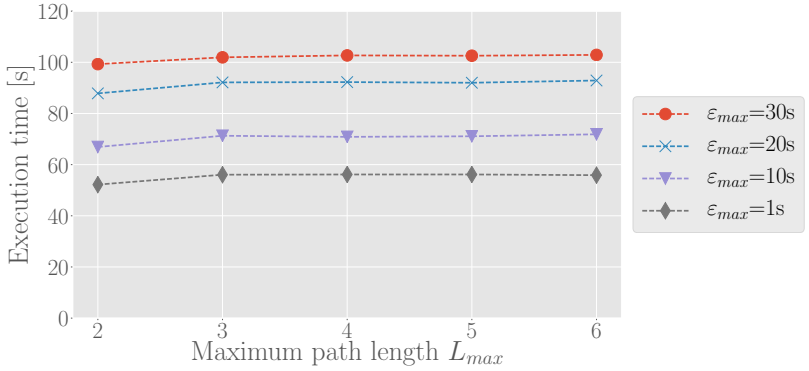
**Execution times** The computational complexity of pivoting detection algorithms is a fundamental challenge for evaluating if they can be applied to real contexts. As discussed in Section 3.2.3, the worst-case computational complexity of our algorithm is exponential with respect to the parameter $L_{max}$ (see Eq. 3.8), that is, the maximum pivoting path length. Nevertheless, the parameter $L_{max}$ is just a theoretical worst-case upper boundary because in real networks the large majority of pivoting paths have length $L = 2$ or $L = 3$. Hence, we show that the time required to analyze even large datasets has little dependence on $L_{max}$.

We execute the algorithm multiple times on the injected dataset, each time providing different input values for $\varepsilon_{max}$ (1s, 10s, 20s, 30s) and $L_{max}$ (2, 3, 4, 5, 6, $\infty$). The size of the time-window $W$ is set to 1 hour and 12 hours. Analyses are performed on a COTS server equipped with one Intel Xeon E5-2609 v2 CPU with 4 physical cores and 128GB RAM. Figures 3.13 report the average execution times required for each run of the algorithm. The x-axis in both figures corresponds to different input values of $L_{max}$; the y-axis represents the average time (in seconds) required for the computation, and each line refers to different input values of $\varepsilon_{max}$. For the sake of completeness, we report that the maximum standard deviation $\sigma_{max}$ among all the experiments with time window $W = 1$ hour and $W = 12$ hours are of 4.8 seconds and 17.8 seconds, respectively. The results in these figures show that the execution times are almost constant with respect to the parameter $L_{max}$ for $L_{max} > 3$ thus confirming that the upper bound $L_{max}$ has no practical influence on the paths present in the real dataset.

As expected, the execution times increase for increasing $\varepsilon_{max}$ values because this parameter affects the research space of the detection algorithm (see Section 3.2.3). Nevertheless, these results show that the amount of time required for analyzing even large datasets of flows is short (the majority of execution times are below 2 minutes even when for analyses of 12 hours of traffic), therefore demonstrating the applicability of the detection algorithm to large real networks.




(A) Analysis of 1 hour of data ($\sigma_{max} = 4.8$s).




(B) Analysis of 12 hours of data ($\sigma_{max} = 17.8$s).

FIGURE 3.13: Execution times of the pivoting detection algorithm.

## 3.3 Detection of Malicious Beaconing Activities

The following work is focused to support automatic security analyses by identifying malicious *external* hosts, even if their activities do not raise any NIDS alert. We propose a method that analyzes network flows by combining time series analysis ( [122]) with clustering algorithms [123], and is able to automatically generate a graylist of few external hosts characterized by a likelihood of being malicious that is several orders of magnitude greater with respect to all the external hosts contacted by the monitored network. In particular, we focus on periodic communications (also referred to as *beaconing*) at different time intervals [124]. The detection of malicious beaconing activities is still an open research problem [61], [125], [126], which is further complicated in large networks due to the difficulty of performing accurate and timely analyses of huge volumes of network traffic [127]. Moreover, we have experimentally verified that external hosts exhibiting periodic connections present a higher rate of malicious behaviors when compared to hosts with irregular communication patterns. As an additional benefit, our proposal is capable of identifying beaconing activities even in the absence of communications with a strict periodic pattern, thus

allowing the detection of possible evasion attempts.

Our solution is evaluated with a thorough set of experiments performed on a large, real network without the creation of any synthetic traffic. The results of these experiments are validated through external sources, and demonstrate the efficacy of our proposal: it produces concise graylists of few dozens of hosts, containing malicious hosts that evaded traditional NIDS detection; and the execution time is compatible with online analyses for timely detection. As a final remark, our approach can be deployed even on very large networks and can be integrated in any detection system, and its performance can be further improved by easily combining with other detection algorithms.

### 3.3.1 Related work

We present a novel method for automatically generating a graylist of external hosts with a higher probability of being involved in malicious beaconing activities with respect to the entire set of external hosts contacted by the monitored organization. Our proposal leverages clustering techniques applied to network flows. There are two main areas of related work: NIDS alarm optimization and detection of malicious beaconing activities.

Each NIDS generates huge amounts of alerts whose manual inspection is often unfeasible for human operators, hence several solutions aim to improve the information presented to security analysts by presenting shorter, comprehensive records. The authors in [128] discuss an algorithm to reduce the volume of alarms produced by multiple NIDSs through clustering alerts raised by similar malicious actions. Other papers, such as [129], propose to cluster alarms to detect their root-causes. Valeur et al. [97] transform groups of correlated alarms into intrusion reports; an evolution of similar approaches can be found in [130]. More recent works propose prioritization techniques for *internal* hosts. Authors of [66], [67], [131] focus on multistep attacks. The proposal in [132] leverages the alarms raised by the most critical assets, whereas the one in [133] leverages detailed information of the monitored network to provide more accurate rankings. All these papers share the broad goal of supporting security analysts by allowing them to focus on the most relevant alarms detected by a NIDS. In contrast, our proposal combines intrusion alerts together with network flow analyses and clustering algorithms to identify the most suspicious *external* hosts even if their actions do not raise any NIDS alert.

The detection of malicious beaconing activities is a well known problem in the field of botnet detection. Gu et al. [63] devise a framework for detecting internal hosts belonging to botnets through clustering of network traffic, based on the assumption that bots belonging to the same botnet have similar network behaviors. Other proposals to expose bots involve monitoring specific communication protocols (such as IRC- or TOR-related information [134]–[136]). Authors of [137] plan to discover botnet infected hosts through supervised machine learning algorithms applied to network flows by identifying the key features of Command and Control communications. A similar solution is proposed in [61], although its main focus is on detecting Command and Control servers instead of bots.

Our method is not limited to detecting botnet-related malware, but extends to any possible external threat that is performing beaconing activities. Unlike botnet-related proposals, we do not make any assumption about the characteristics displayed by the analyzed traffic. Related work, such as [125], inspects DNS logs to discover malicious beaconing activities performed by internal hosts, whereas the proposal in [126] relies on the analysis of both DNS and web-proxy logs. On the other hand, we aim to detect malicious external hosts, which is a tougher problem because a large organization may contact hundreds of thousands of external hosts daily.

Moreover, our proposal is based on the analysys of network flows, which can be easily gathered and stored [83], leverages an unsupervised machine learning algorithm (unlike [61], [137]), and its execution time on a large network is compatible to online traffic analyses.

### 3.3.2 Proposed method

The main objective is to provide a graylist of external hosts involved in periodic communications with a high likelihood of being malicious. The basic assumption is that although novel variants of attacks are likely to evade NIDS detection [32], some features of malware network behavior persist and can be used to identify likely malicious activities.

The proposed method works on two inputs that can be easily obtained in modern infrastructures: network flows related to communications between internal and external hosts, and security alerts generated by a signature-based NIDS. These inputs are processed by the three modules shown in Figure 3.14. The *Periodicity Detector* is

responsible for identifying network communications between internal and external hosts occurring at regular intervals. The *Behavioral Aggregator* clusters periodic connections according to their network behavior. The *Graylist Builder* creates the final graylist of suspicious external hosts.

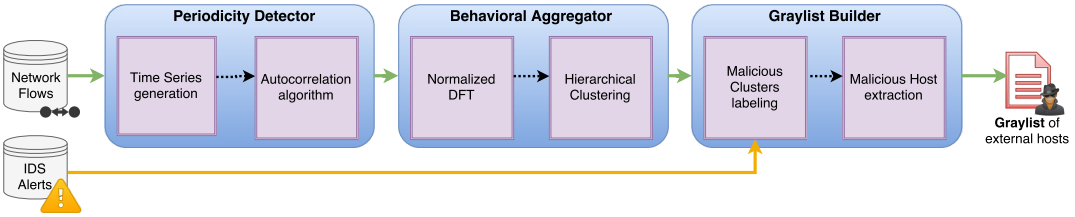



FIGURE 3.14: Workflow of the proposed method.

As the number of connected devices in enterprise networks continues to increase, the detection of periodic activities is becoming a challenging task, since they can occur at different degrees of granularity spanning from few seconds to hours (to the interested reader, we point out that similar problems are also tackled in [124]). Instead of looking for periodicities in raw traffic, we consider network flows which offer aggregated metadata summarizing relevant network traffic features. Each flow record is defined as an unidirectional sequence of packets that share specific network properties, such as source/destination IP address, transport layer protocol type, and source/destination port. Using network flows as input source is a popular choice in the cybersecurity domain [83], as they lower the amount of storage space required, make analyses faster, and reduce privacy concerns due to the absence of packet-specific payloads.

NIDSs are a valuable asset for detecting malicious activities, but they are unable to detect novel malware variants that do not contain any known signature. However, some characteristics of malware behavior, such as beaconing, are stable across a wide array of automatically generated malware variants, thus resulting in similar communication patterns. Since our approach clusters network communications that share similar periodic behaviors, different variants of the same piece of malware are likely to be clustered together. Our approach only requires that a single malware variant generates a NIDS alert to pinpoint as suspicious the entire cluster of periodic communications containing that variant.

We now describe each module that composes our solution.

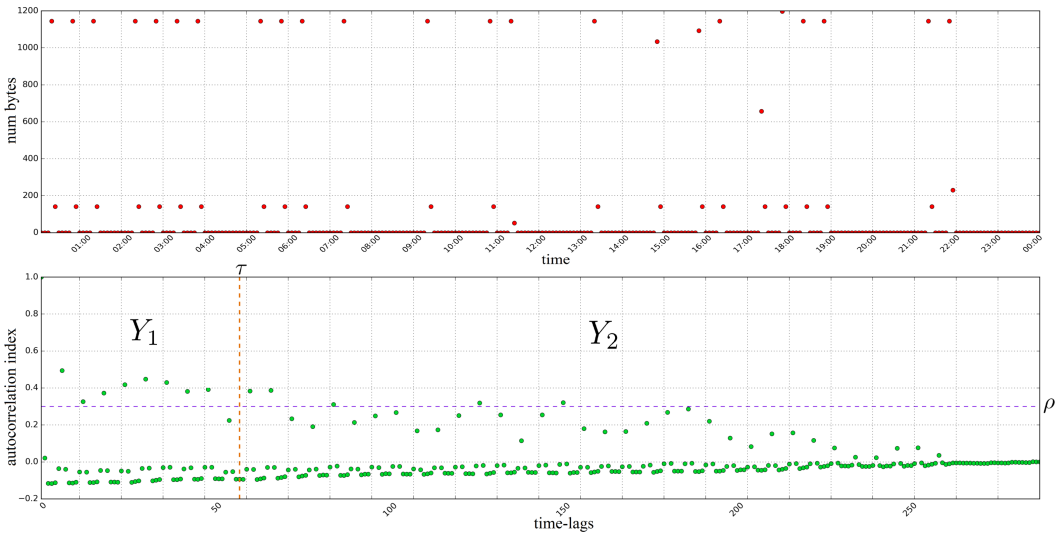



FIGURE 3.15: Example of time series and related ACF generated by two host with a noisy periodic behavior.

**Periodicity Detector** The Periodicity Detector module detects periodic communications from network flows in two phases: first, it generates time series from the network flows; then, it analyzes these time series through an autocorrelation algorithm to determine whether they are periodic or not. The adopted techniques are robust and tolerate possible pertubations caused by noise or introduced by an attacker to escape detection.

The sequence of network flows among two hosts represents an unevenly spaced time series, which cannot be immediately used to detect periodic communications. Hence, we initially compute one evenly spaced time series for each pair of internal and external hosts exchanging packets within a time window $W$. Given a sampling period $P$, this time series contains a total of $W/P$ elements. Each element is built by aggregating all the network flows between the involved hosts occurring within the same sampling period. As beaconing activities require repeated exchanges of some data, to capture these data transfers we compute each element of the time series by adding together the amount of bytes exchanged between the involved hosts within the related sampling period. This design choice allows us to better differentiate beaconing activities that exchange different volumes of data. After this phase, each pair of internal and external hosts is associated to one time series.

Then, we adopt autocorrelation to detect periodicities in each time series because this technique can signal time series exhibiting more than one period [138]. By computing the autocorrelation on a time series we obtain an autocorrelation function (ACF) containing $W/P$ elements, each one representing the similarity of the time

series with a delayed copy of itself. The analysis of the local maxima of the ACF determines whether a time series exhibits or not periodicities. In particular, looking for periodicities in the ACF involves determining the coordinates of local maxima, since strictly periodic time series tend to have local maxima with high amplitude at the beginning of the related ACF. Existing works relying on this technique for detecting periodicities in time series (e.g., [139]) are only able to identify strictly periodic time series. The problem is that skilled attackers may insert some perturbations to avoid detection, either by delaying or anticipating the communications, or by changing the volume of data exchanged during each interaction by random amounts. Moreover, network traffic may be subject to noise induced by inactivity periods, temporal disconnections, or by the presence of packets retransmissions and other network-related artifacts. To address these issues, we propose an innovative algorithm that is capable of labeling as periodic even time series that do not display a strictly periodic pattern. The main intuition is that restricting the analysis of the ACF only on the first very high local maxima does not allow to identify noisy periodic time series: time series with noisy periodicities are characterized by a limited amplitude of local maxima, hence they cannot be detected by conventional approaches. However, with respect to aperiodic time series, they present several local maxima with a similar amplitude, as well as high amplitudes between a local maximum and its next local minimum. To achieve a more flexible algorithm to detect noisy periodicities, we introduce two thresholds in the ACF:

- the *local maximum-location threshold* $\tau$ identifies the initial set of local maxima, splitting the ACF into two subseries: $Y_1$, containing all the first $\tau$ elements of the ACF and determining the initial set of local maxima; and $Y_2$, containing all the other elements, determining the remaining set of local maxima;
- the *local maximum-amplitude threshold* $\rho$ is used to determine the amplitude required for an ACF element to be considered a local maximum: we consider those elements whose value is greater than $\rho$ as local maxima, and those elements whose value is lower than $\frac{\rho}{2}$ as local minima.

To illustrate the idea, we report in Figure 3.15 the time series and its related ACF obtained from the communications between two hosts. We observe the existence of two noisy periodic behaviors in the time series, evidenced by the exchange of about 1.1KB and 180B of data every 30 minutes. For the ACF plot, we represent $\tau$ and $\rho$

with a vertical and horizontal dashed line, respectively. We note that the amplitude of the local maxima in the ACF decreases irregularly, caused by the presence of noise in the original time series; furthermore, the initial local maxima set has a similar amplitude as the remaining set.

Our algorithm labels as periodic those time series whose ACF satisify at least one of the following criteria:

- $Y_2$ has at least $d$ elements $\geq \rho$ **and** $Y_1$ has at least $2d$ elements $\leq \frac{\rho}{2}$;
- $Y_1$ has at least $r$ elements $\geq \rho$ **and** $Y_1$ has at least $r$ elements $\leq \frac{\rho}{2}$.

Where $d$ is the *period duration sensitivity* and $r$ is the *periodic rate sensitivity* that must be chosen manually. Higher values of $d$ imply that those time series that are labeled as periodic are characterized by periods of shorter length; higher values of $r$ result in periodic time series whose periodicities occur for longer time-frames. We remark that the first and second criteria are designed to detect time series with periods of greater and shorter length, respectively. Upon the completion of this phase, all those time series that have been labeled as periodic are forwarded to the Behavioral Aggregator module.

**Behavioral Aggregator** The Behavioral Aggregator clusters periodic communications exhibiting similar patterns. Although clustering techniques have already been employed in the information security field, to the best of our knowledge we are the first to propose the leveraging of clustering algorithms to detect communications with a similar periodic behavior. This task is performed in two phases: we compute the Discrete Fourier Transform (DFT) for each periodic time series to obtain its spectrogram; then, these spectrograms are used as input for a hierarchical clustering algorithm.

By applying the DFT to a periodic time series it is possible to generate a spectrogram. This representation is useful to describe the behavior of network communications, since periodic time series that are out of phase may look very different, while their spectrograms exhibit the same profile. The problem is that the shape of each spectrogram also depends on the amounts of bytes exchanged between the involved hosts. For example, two hosts that regularly exchange 1MB of data will have a spectrogram with a smaller amplitude than a different pair of hosts that regularly

exchange 10MB, although their frequency components are the same. To address this issue we normalize the amplitudes of each spectrogram between 0 and 1.

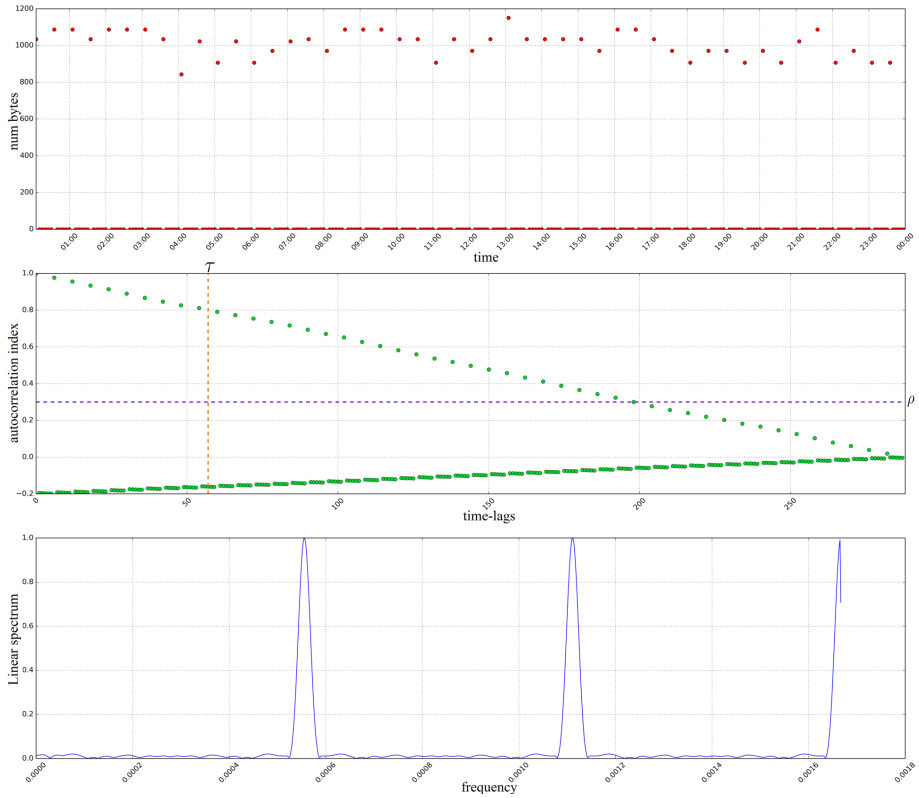
Then, each spectrogram is used as input for a hierarchical clustering algorithm, an unsupervised machine learning algorithm that takes as its input a matrix of distances. We create this distance matrix by means of the Pearson correlation coefficient [140], which is computed among all the normalized spectrograms. The output of the hierarchical clustering algorithm is a dendrogram. By cutting the dendrogram at a given height $h$, it is possible to create clusters of objects that are similar to each other. We tune the parameter $h$ as to minimize intra-cluster variance and maximize the inter-cluster variance. At the end of this phase we obtain a variable number of clusters of periodic communications with similar behaviors, which are used as input for the Graylist Builder module.

**Graylist Builder** The final graylist of malicious external hosts is produced by the Graylist Builder module. It initially identifies malicious clusters of periodic communications by mapping NIDS alerts into clusters of similar periodic communications. More specifically, those clusters containing at least one communication that has raised a NIDS alert are labeled as malicious; this process allows us to detect malicious hosts that are not signaled by the NIDS. Then, this module extracts all the external hosts belonging to malicious clusters and uses them to populate the final graylist.

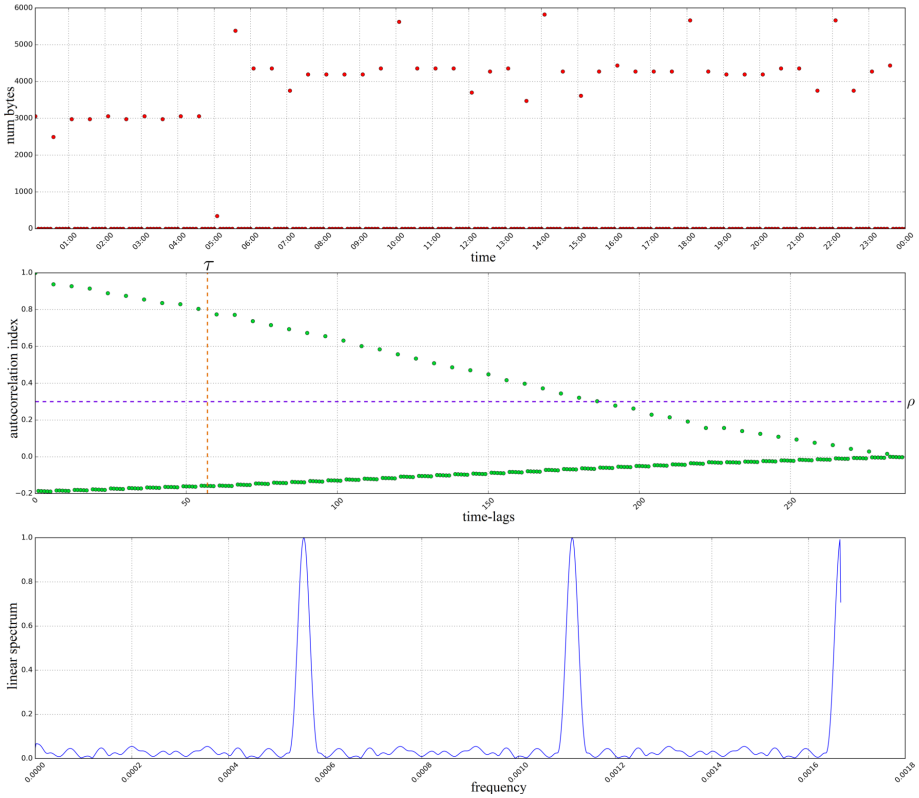
### 3.3.3 Experimental methodology

The proposed method is validated on real traffic generated by a large network of nearly ten thousand hosts during an entire week, consisting of about half a billion of network flows. The outgoing traffic has been monitored by a NIDS equipped with Suricata [141], used and configured by security operators with the most recent rulesets [142]. Table 3.19 reports the most meaningful metrics of the testbed for the different days of the considered week. The second and third days, marked with an asterisk, represent weekend days and are characterized by a lower activity.

All the experiments discussed in this section refer to a time window set to one day ($W = 1\text{d}$), while network flows are sampled every five minutes ($P = 300\text{s}$). The parameters of the autocorrelation algorithm are determined through a comprehensive sensitivity analysis performed through multiple executions of the algorithm,




(A) Communications involving a malicious external host with one periodic behavior.




(B) Communications involving a malicious external host with three periodic behaviors.

FIGURE 3.16: Time series, ACF and normalized spectrogram of two communications involving distinct malicious external hosts.

TABLE 3.19: Traffic information of each day of the dataset.

| Day | Distinct external hosts | Distinct time series | Network flows |
|---|---|---|---|
| 1 | 296 945 | 1 915 186 | 109 302 224 |
| 2* | 105 884 | 541 844 | 53 500 389 |
| 3* | 89 283 | 393 077 | 47 789 977 |
| 4 | 298 241 | 1 835 351 | 101 314 287 |
| 5 | 314 313 | 1 935 982 | 110 875 503 |
| 6 | 249 768 | 1 667 168 | 99 359 716 |
| 7 | 258 439 | 1 789 238 | 106 304 916 |

and the resulting values are summarized in Table 3.20. The values of the parameters $\rho$ and $\tau$ are chosen equal to those suggested by the literature on periodicity evaluation in time series [138]. The height at which the dendrogram is cut to generate the clusters is set to $h = 0.95$, because sensitivity analyses show that this value minimizes intra-cluster variance and maximizes inter-cluster variance for the monitored environment.

TABLE 3.20: Parameter values used as input.

| Symbol | Description | Value |
|---|---|---|
| $\rho$ | Local maximum-height threshold | 0.30 |
| $\tau$ | Local maximum-location threshold | $\frac{W}{5P}$ |
| $d$ | Period duration sensitivity | 6 |
| $r$ | Periodic rate sensitivity | 2 |

### 3.3.4 Evaluation results

The detection framework is executed every day. The goal is to demonstrate its capability of producing a manageable graylist of external hosts with a considerably higher likelihood of being malicious when compared to the original set of contacted external hosts. In addition we show that the rate of malicious external hosts performing periodic communications is considerably higher with respect to those involved in aperiodic communications. Finally, we demonstrate that the graylist includes even external hosts that did not raise a NIDS alert, and that the execution time of our method is compatible with online traffic analyses.

We initially assess the amount of malicious external hosts in the entire set of external hosts that have been contacted by the monitored network. Then, we let the Periodicity Detector module generate time series and determine which of them are periodic. We remark that our analyses are executed on the unmodified network traffic produced by a large organization: we did not inject synthetic attacks or malicious traffic, but we leverage the APIs provided by VirusTotal [143] to validate malicious

external hosts. More specifically, we consider an external host to be malicious if it has been signaled by more than half of the sources queried by VirusTotal.

To demonstrate that the rate of malicious external hosts involved in periodic communications is considerably higher than the rate of malicious hosts involved in aperiodic communications, we present the results of the validation process performed on these two sets of hosts in Table 3.21. For each column, the rows with gray and white background report the number of external hosts involved in periodic and aperiodic communications, respectively. We observe that the average ratio of malicious external hosts exhibiting periodic communications is 2.7%, whereas the one of hosts involved in irregular communications is 0.51%. These results show that periodic communications display a greater rate of maliciousness with respect to aperiodic communications, thus supporting our decision to focus on this set of hosts. Furthermore, these results indicate that malicious external communications can be considered as rare events in the overall traffic, and motivates our effort of building a manageable graylist in which the likelihood of finding a malicious host is higher.

TABLE 3.21: Validation of external hosts involved in periodic (gray) and aperiodic (white) communications.

| Day | External hosts | Malicious external hosts |
|---|---|---|
| 1 | 3 139 | 97 (3.09%) |
| | 293 806 | 1 224 (0.42%) |
| 2* | 2 284 | 59 (2.58%) |
| | 103 600 | 785 (0.76%) |
| 3* | 2 123 | 53 (2.49%) |
| | 87 160 | 603 (0.69%) |
| 4 | 3 194 | 74 (2.31%) |
| | 295 047 | 1 198 (0.41%) |
| 5 | 3 288 | 91 (2.77%) |
| | 311 025 | 1 153 (0.37%) |
| 6 | 3 044 | 80 (2.63%) |
| | 246 724 | 1 202 (0.48%) |
| 7 | 3 034 | 90 (2.97%) |
| | 255 405 | 1 283 (0.50%) |

To illustrate that our method is capable of detecting periodicities, we execute the Behavioral Aggregator module and we report in Figures 3.16 the time series, ACF and normalized DFT pertaining to communications belonging to the same cluster and involving two distinct malicious external hosts. The first and second plots in each figure display the time series and related ACF, while the third plot displays the normalized spectrogram of the DFT. We observe that both time series exhibit a periodic behavior, although some noise is present. More specifically, the hosts associated to the first time series exchange about 1KB of data every 30 minutes; whereas those

associated to the second time series present three periodical behaviors, evidenced by the exchange of about 3KB and 4KB of data every 30 minutes, and of about 5.5KB of data every 4 hours. These results demonstrate that our algorithm is able to identify even periodic communications affected by some perturbations. Moreover, we observe that the spectrograms of Figures 3.16 are very similar despite featuring different data exchanges, leading to their inclusion in the same cluster. This result indicates that our approach based on normalized DFT provides a good representation of the periodic behavior of a time series and is robust against alterations in exchanged data volume.

TABLE 3.22: Comparison of the amount of external hosts.

| Day | All external hosts | External hosts with periodic behavior | External hosts in graylist |
|---|---|---|---|
| 1 | 296 943 | 3 139 | 127 |
| 2* | 105 884 | 2 284 | 90 |
| 3* | 89 283 | 2 123 | 70 |
| 4 | 298 241 | 3 194 | 31 |
| 5 | 314 313 | 3 288 | 120 |
| 6 | 249 768 | 3 044 | 119 |
| 7 | 258 439 | 3 034 | 115 |

Finally, we generate the graylists by executing the Graylist Builder module for each day of the dataset. We present in Table 3.22 the amount of hosts included in our graylists alongside both the entire set of hosts that have been contacted and the number of hosts displaying a periodic behavior. We appreciate that our graylists comprise about one hundred of entries down from the initial set of hundreds of thousands hosts, thus allowing further security inspections to focus on a restricted amount of external threats.

TABLE 3.23: Validation of the graylist and comparison with NIDS.

| Day | Malicious hosts in graylist | Malicious hosts detected by NIDS |
|---|---|---|
| 1 | 19 (14.96%) | 3 (2.36%) |
| 2* | 17 (18.89%) | 3 (3.33%) |
| 3* | 6 (8.57%) | 3 (4.29%) |
| 4 | 3 (9.68%) | 3 (9.68%) |
| 5 | 17 (14.17%) | 4 (3.33%) |
| 6 | 7 (5.58%) | 3 (2.52%) |
| 7 | 15 (13.04%) | 4 (3.48%) |

The evaluation of the graylist produced by our method is performed by determining the rate of malicious graylisted hosts, and by showing that the graylist contains even malicious hosts that do not raise NIDS alarms. The results are reported in Table 3.23, where the first column indicates different days while the second and third columns show the number of malicious hosts included in the graylist and the number of malicious hosts that raised a NIDS alarm, respectively. We manually inspect these results and find that all the hosts detected by the NIDS were included in the graylist for the same day.

By correlating the values of Table 3.23 with those presented in Table 3.21, we understand that the ratio of malicious hosts in our graylist is an order of magnitude greater than the ratio of malicious host in the entire set of contacted hosts. Moreover, by comparing the values of the second and third column of Table 3.23 we understand that our method is capable of graylisting up to six times as many malicious hosts with respect to those detected by the NIDS.

These results indicate that our method is able to produce a manageable graylist consisting of about one hundred of entries, down from the original set of hundreds of thousands entries, which is characterized by a ratio of malicious hosts that is an order of magnitude greater and containing malicious hosts that do not raise any NIDS alarm. Validating several dozens of IP addresses through external public APIs only requires few minutes, whereas the validation of hundreds of thousands of addresses requires almost one week. Finally, we remark that all these analyses are performed on real network traffic, as we did not inject any artificial attack.

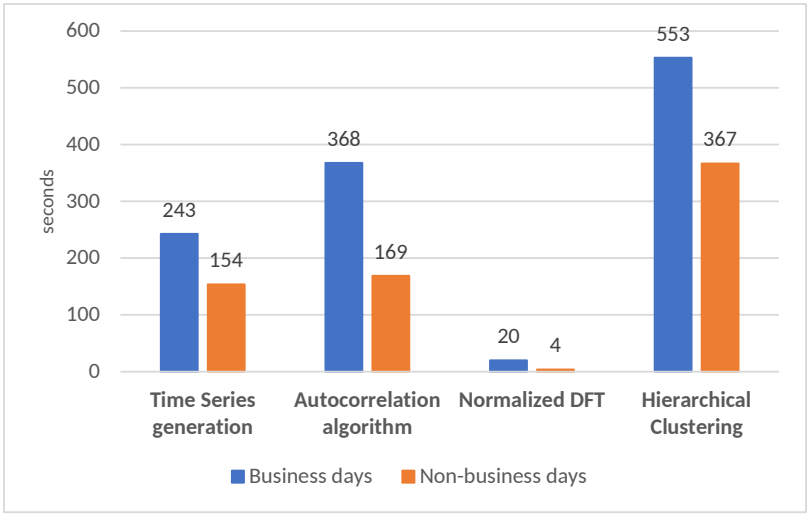



FIGURE 3.17: Average execution time of the main phases of the proposed method for 24 hours of traffic.

Early detection of ongoing attacks is of paramount importance for modern organizations. Hence, we evaluate the execution time of the proposed method for each day of the dataset by distinguishing business and weekend days. The results are illustrated in Figure 3.17, where each pair of histograms represents the average execution time (in seconds) for a different phase of the proposed method, calculated for business days (left rectangle) and non-business days (right rectangle). For the sake of completeness, we report that these analyses have been performed on a COTS server equipped with an Intel Xeon E5-2609 v2 CPU with 4 physical cores and 4 threads, and 128GB RAM. It is important to observe that the execution time for the analysis of 24 hours of traffic is below 20 minutes even for contexts generating hundreds of millions of network flows daily. In particular, we note that the longest phase (Hierarchical Clustering) requires less than 10 minutes. Hence, by pipe-lining our algorithms, it is possible to obtain detailed reports once every $\sim$10 minutes. These results prove that the proposed method is applicable to online security analyses.

# *Part 2 – Adversarial Attacks against Cyber Detectors*

# Chapter 4

# Adversarial Attacks against Cyber Detectors

In this chapter, we consider the problem presented by the inherent vulnerability of machine learning methods to adversarial attacks, through which opponents can thwart the system by inducing the generation of incorrect or undesirable results [144]. This issue is aggravated by the multiple variations of malicious actions that can be performed during the training- or test-time of the machine learning algorithms [145], [146].

Adversarial attacks against machine learning have been explored in image processing [147], but lack adequate analyses in the cybersecurity domain. The papers that evaluate the performance of cyber detectors in adversarial settings (e.g., [145], [148]) consider a limited number of cybersecurity problems, few machine learning classifiers, and a restricted subset of adversarial attacks. The main focus is on spam and malware detection [149], [150], while we consider this issue from a network intrusion detection perspective, where experimental evaluations and novel solutions are lacking [39]. In summary, the contributions of this chapter are the following: (i) a thorough analysis of adversarial attacks against cyber detectors; (ii) original evaluations of realistic adversarial attacks against botnet detectors; and (iii) an original method to generate realistic adversarial samples from labelled network data.

This chapter presents the following structure. Section 4.1 presents a thorough study of the state of the art of adversarial attacks against cyber detectors, outlining the characteristics of these threats. Then, Section 4.2 focuses on measuring the effectiveness of adversarial perturbations on cyber detectors through original experiments. Due to their importance for the remaining parts of this thesis, we remark that Section 4.2.3 describes the datasets adopted for the evaluation, whereas Section 4.2.5

presents the procedure to generate the adversarial samples.

## 4.1 Categories of Adversarial Attacks in Cybersecurity

To defend against cyber threats, security operators rely on techniques borrowed from the machine learning domain [4], [5] because of their *anomaly detection* capabilities, which may identify novel attacks and which are not recognizable through *signature-based* approaches [6], [151]. Machine learning algorithms can be divided into *supervised* and *unsupervised* techniques, depending on the requirement of the training phase, with a set of labelled data. Both groups can be applied to address cybersecurity problems [152], but supervised methods are appreciated due to their ability to provide actionable results, such as detecting an attack [153]. On the other hand, unsupervised techniques are employed for ancillary tasks such as data clustering [37]. All these methods present several open issues that must be considered when integrating them into security systems, as we have shown in Chapter 2. Here, we focus on the topic of adversarial attacks.

Adversarial attacks against machine learning solutions represent a major limitation to the adoption of a fully autonomous cyber defence platform. These threats are based on the generation of specific samples that induce the model to produce an output that is favourable to the attacker, and leverage the intrinsic sensitivity of machine learning models to their internal configuration settings [4], [154], [155]. Although adversarial perturbations affect all applications of machine learning, the cybersecurity field presents several characteristics that further aggravate this menace: there is a constantly evolving arms race between attackers and defenders; the system and network behaviour of an organization can be subject to continuous modifications. These unavoidable and unpredictable changes are denoted as the *concept drift* [156] problem, which decreases the performance of any model based on anomaly detection. Mitigations involve periodic retraining and adjustment processes that can identify behavioural modifications and recent related threats. While performing such operations is a challenging task in itself (see Section 2.2.2), it also facilitates the execution of adversarial attacks [157].

Many research results (e.g., [144], [146], [158]) show that machine learning algorithms are unsuitable to face adversarial settings. The first examples of adversarial attacks date back to 2004 [159], but the advent of deep learning drew the attention of

the research community to this issue [160]. Possible countermeasures have appeared in the computer vision literature [147], with several papers proposing solutions for improving the robustness of deep neural networks for image classification in adversarial environments [161]. However, the performance of machine learning algorithms depends on their application contexts, hence it is of paramount importance to understand the effects of adversarial threats against cyber detectors. We consider different classes of attacks by proposing a taxonomy inspired by the work of Huang et al. [144], where threats are classified on the basis of two properties: the *influence* determines whether an attack is performed at training-time or test-time; the *violation* denotes the type of security violation that may affect availability or integrity of the system.

- **Influence**
  - *Training-time*: these attacks include the manipulation of the training set used by the machine learning model through the insertion or removal of specific samples that alter the decision boundaries of the algorithm. They are also known as *poisoning attacks*.
  - *Test-time*: These attacks assume that the detector has been deployed and aim to subvert its behaviour through the submission of specific samples during its operational phase.
- **Violation**
  - *Integrity*: often referred to as *evasion attacks*, these attacks aim to increase the false negative rate of the model by introducing malicious samples that are classified as benign. Hence, when successful, these stealthy threats do not cause any defensive action to be taken by the targeted organization.
  - *Availability*: these attacks make the targeted model useless, for example by causing overwhelming spikes of false alarms. For this reason, attacks of this type usually induce some sort of response action by the defending side, such as temporary shut-down and recalibration of the model.

A comprehensive classification of adversarial attacks requires a definition of the attacker model. According to Biggio et al. [158], we should consider the following main features.

- The **goal** is related to the security violation purpose of the adversarial attack.
- The **knowledge** denotes the information possessed by the attacker on the machine learning system that may include the adopted algorithm, its parameters, and its training data set. Depending on the type of information, we can distinguish between *black box* attacks (zero knowledge), *grey box* attacks (partial knowledge), and *white box* attacks (complete knowledge).
- The **capability** determines the type of actions that an attacker can perform against the targeted environment that includes, but is not limited to, the machine learning system. As a strict requirement, it is important to specify which kind of access the attacker has to the cyber detector: he can have full access (that is, reading its output and modifying its internals), limited access (can only read its output) or no access at all.
- The **strategy** denotes the workflow pursued by the attacker to achieve his goal by leveraging previous knowledge and capabilities.

The attacker model distinguishes the adversarial attacks against cybersecurity systems from offences against other domains of application of machine learning. For example, most papers on image recognition [147], [162] assume that the attacker has complete knowledge and capability. These assumptions are unrealistic in cybersecurity applications for two reasons: cyber detectors are protected by multiple defence layers; if an attacker overcomes these barriers and can modify the detector, he can achieve his goals without relying on adversarial attack strategies. Thus, in the remainder of this work, we consider attacks in which the attacker has limited or no access to the machine learning system.

In Table 4.1, we classify the most important examples of adversarial attacks on three cybersecurity detection problems (network intrusion, malware, spam & phishing) representing scenarios where machine learning methods are achieving appreciable results (e.g., [4], [5], [32]). In this table, columns indicate the cybersecurity areas while rows denote the adversarial attack class. Each cell reports the machine learning algorithms that have been tested against the related class of attacks. We remark that algorithms written in bold are evaluated for the first time in this thesis. Existing literature focuses mainly on integrity attacks, with several algorithms evaluated for malware and spam analysis. Few solutions exist and are tested in

the network intrusion detection context, and this observation motivates this chapter. There are few documented attacks targeting the system availability, and there are no specific studies at test-time.

TABLE 4.1: Mapping of the categories of adversarial attacks to cybersecurity problems.

| | | **Network intrusion detection** | **Malware detection** | **Spam & phishing detection** |
|---|---|---|---|---|
| **Test-time** | Availability violation | ✗ | ✗ | ✗ |
| | Integrity violation | **RF**<br>**FNN**<br>**KNN**<br>NB [163] | RF [164]<br>SVM [145]<br>LR [165]<br>MLP [145] | SVM [166]<br>LR [167]<br>NB [168] |
| **Training-time** | Availability violation | ✗ | NB [169] | NB [149]<br>Clustering [170] |
| | Integrity violation | **RF**<br>**FNN**<br>**KNN** | LR [171]<br>DNN [172] | NB [159]<br>DNN [172] |

## 4.2 Effectiveness of Adversarial Attacks

In this section, we study adversarial perturbations in the context of network intrusion detection systems. Our objective is to provide a comprehensive evaluation of the impact of these malicious actions against state of the art detectors of botnets, due to their relevance in the current cybersecurity landscape [11]. Indeed, despite extensive research efforts [173], botnets remain one of the most serious threats to modern enterprises. Several research papers address this issue by devising botnet detectors relying on machine learning [39], involving both supervised (e.g.: [174]) and unsupervised (e.g.: [37]) algorithms; however, the problem still persists [11], [175].

In this study, we perform an extensive experimental campaign of realistic adversarial attacks, involving different machine learning techniques and multiple large datasets of network flows that are publicly accessible. Thus, our broad analysis captures a multitude of scenarios that represent the heterogeneous nature of modern networked systems. To the best of our knowledge, this is the first study that presents such a vast range of novel experiments on this subject. The results highlight the vulnerability of all the considered detectors to adversarial samples. The problem is further aggravated by the fact that similar alterations can be introduced at a very low cost for the adversary: they do not require to change the communication scheme nor the control logic of the botnet; can be easily introduced without the need of deeper

compromise and privilege escalation on the infected bots; and the attacker does not need advanced knowledge of the defensive scheme. Indeed, attackers can evade detection with high probability just by introducing slight changes in the network flows of the controlled bots, such as inserting small delays or adding a few bytes to the network packets. As an example, for some malware families increasing the duration of network communications by just 1 second causes the detection rate to drop from over 99% to less than 20%. Moreover in many cases it is possible to easily produce adversarial samples that cause a detection rate below 1%. These results highlight the high effectiveness of adversarial samples against all of the considered classifiers, evidencing a critical vulnerability of machine learning applied to cybersecurity.

### 4.2.1 Related work

Our study highlights and evaluates the fragility of flow-based botnet detectors built on machine learning classifiers against adversarial attacks at test-time. Therefore, we identify two main areas of related work: *network intrusion detection* focused on botnets and based on machine learning; and *adversarial attacks* against machine learning.

The scientific literature on network intrusion detection has been applying techniques and algorithms borrowed from the machine learning domain for several years [6], and detection schemes involving machine learning have been integrated to some extent even in many recent cyber defense platforms and commercial products [9], [10]. Many machine learning algorithms have been proposed to specifically address the problem of botnet detection [39], ranging from supervised to unsupervised techniques and even to deep learning. A meaningful and direct comparison among different detection methods is difficult to achieve due to known reasons [4], [39]. However, many research papers [27], [45], [176], [177] adopt classifiers based on machine learning due to their appreciable results: among these, we highlight that those built on random forest algorithms have been empirically shown to outperform several other methods for network intrusion detection tasks [51].

The authors of [27] devised a network intrusion detection system for identifying network traffic related to communications between infected hosts and their Command and Control infrastructure. This approach is based on the analysis of network flows, rather than full network packets, and applies a classifier based on random forests as suggested by related literature. We remark that inspecting network flows

for network intrusion detection is a common practice (e.g.: [61]) because of the following advantages with respect to full packet traces including communication payloads: reduced privacy concerns; smaller storage space and lower computational requirements; the pervasive adoption of end-to-end encryption that prevents collection and analysis of the content of network packets. These reasons motivated us to focus our experiments on a flow-based botnet detector leveraging proficient machine learning classifiers as a relevant, realistic and representative example of the best practices currently adopted by network security researchers and industry.

The increasing pervasiveness of machine learning led to question its performance in adversarial settings. Recent research papers have mostly addressed this problem from an image processing perspective [146], [178], [179], providing clear use-cases of adversarial attacks. However, literature on evasion attempts against cybersecurity detectors based on machine learning is more focused on the theoretical vulnerabilities of these techniques rather than showing and evaluating the actual effectiveness and consequences of adversarial attacks [39], [144]. Hence, the evaluation of adversarial samples in the cybersecurity domain is still an open research problem [180]. This is the main motivation behind our work, since a clear understanding of the impact of adversarial attacks against cyber defence systems based on machine learning is of crucial importance for the development of modern cybersecurity technologies.

As shown in Section 4.1, related literature has mostly focused on analysing the effects of these threats against malware and spam detectors [144], [145], [181]–[184], but few papers address this problem from a network intrusion detection perspective [185]. The authors of [186] present an analysis of attacks against NIDS, but they do not consider machine learning techniques. Other works raise the awareness on this subject by proposing mechanisms to evade machine learning-based NIDS, but they do not provide any experimental evidence to sustain their claims [39]. Previous work also include research efforts [187], [188] based on the deprecated [189] KDD-99 dataset[1], and cannot be considered as a good representation of modern large network environments. Other papers only consider attack scenarios where the adversary has extensive or perfect knowledge of the detector, which is an unrealistic assumption in a true cybersecurity context [190]–[192].

To the best of our knowledge, this is the first study that presents an extensive evaluation of *realistic* adversarial attacks performed against botnet detectors based

[1]KDD99 Dataset: https://www.unb.ca/cic/datasets/nsl.html

on *multiple* machine learning algorithms, which are deployed in *different*, *recent* and *heterogeneous* network scenarios. Thus, the present analysis portrays a much needed and representative overview of the current state of the art of machine learning botnet detectors in adversarial settings.

### 4.2.2 Threat model

We now describe the realistic scenario of the threat model considered in our work. Indeed, to conduct a meaningful evaluation of the impact of realistic adversarial attacks, it is necessary to describe the characteristics of the considered network environment and of the considered attacker. We begin by modeling the target network, and then present the characteristics of the attacker.

**Defensive model** The defensive model is represented by a large enterprise network of over a thousand of internal hosts. At its edge, the network presents a border router connected to a network flow collector. Despite the existence of several software products aimed at collecting network flows, each presenting diverse characteristics and allowing to capture different pieces of data, we assume that the generated flows only contain the following essential information:

- source and destination *IP address*;
- flow Start- and End-time (flow *duration*);
- source and destination *ports*;
- *protocol*;
- source and destination *Type of Service (ToS)*;
- source and destination exchanged *bytes*;
- Total *packets* transmitted.

The produced network flows are then inspected by a botnet detector that relies on a machine learning classifier to distinguish between legitimate and botnet samples. The classifier is trained to identify the malicious flows produced by specific botnet variants. We assume that some machines within the network are infected by a botnet-related malware (for example through a zero-day attack, successfull spear

phishing attempts, or insider threats) that communicates with an external Command and Control (CnC) server.

A representation of the scenario described above is provided in Figure 4.1, which shows a large enterprise network with many internal hosts and a border router connected to a network flow exporter. The generated flows are inspected by a network intrusion detection system based on machine learning that aims to identify malicious activities (e.g., botnet) by leveraging the random forest algorithm.

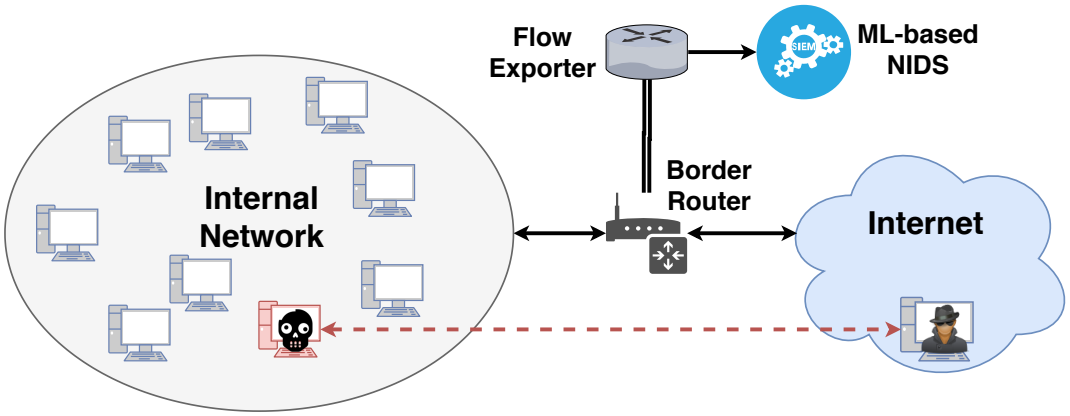



FIGURE 4.1: Example of network considered in our use-case.

**Attacker model** We describe the model of the considered attacker by following the guidelines proposed in related literature [180], outlining their *goal*, *knowledge*, *capabilities* and *strategy*.

- **Attacker Goal**: The attacker aims to evade the botnet detector in order to maintain access to the network and perform additional malicious activities (such as *pivoting* – see Section 5.2).

- **Attacker Knowledge**: The attacker has partial knowledge on the defenses adopted by the target network. They know that network communications might be monitored by a network intrusion detection system based on supervised machine learning. However, they do not possess knowledge on the algorithm itself (e.g.: they do not know the parameters, the weights or the feature set); nor they have complete knowledge of the dataset used to train the detector. Nonetheless, we assume that the attacker knows that this detector is trained over a dataset containing malicious flows generated by the very same malware variant deployed on the infected machines. Thus, the attacker is aware that they need to quickly devise some countermeasure to evade

the botnet detector, since any delay may increase the chance that the infected machine is manually inspected and cleaned due to the alerts triggered by the detector. We remark that these are realistic assumptions for any commercial cybersecurity appliance.

- **Attacker Capabilities**: We assume that the attacker does not have full control and privileges on the infected machines. The attacker is limited to issuing commands to the bots through the Command and Control infrastructure, possibly modifying their behavior. However the attacker cannot interact with the detector in any way: this includes both read and write operations. For example, he cannot use the detector as an "oracle" [146] by submitting certain inputs and reading the respective outputs.

- **Attacker Strategy**: To avoid detection, the attacker leverages their limited knowledge and capabilities to produce a *targeted exploratory integrity attack* [145]. More specifically, they insert some slight modifications in the communication sequences between bots and their Command and Control. These alterations need to keep the internal logic of the employed piece of malware intact, as their sole purpose is to make the (malicious) samples different from the ones that have possibly been used to train the detector; furthermore, they need to be stealthy enough to avoid triggering detection through other defensive mechanisms [74]. The purpose is inducing the botnet detector to misclassify the network flows generated by bot communications due to their different characteristics with respect to the malicious flows contained in the training dataset. Possible alterations include slight increases of flow *duration*, exchanged *bytes* and exchanged *packets*. We remark that similar modifications can be applied without interfering with the application logic of the bots (e.g. [193]), which can continue to operate as designed by the attacker.

We highlight that our assumptions portray a realistic scenario: models that involve attackers with perfect (or near-perfect) knowledge are unrealistic, as the NIDS is usually protected by multiple layers of defenses. Adversaries that have possess such confidential information – or that have direct access to the detector – are also capable of launching attacks of higher magnitude that are out of the scope of this analysis. Also adversaries that are able to generate a brand new malware characterized by different communication patterns between bots and CnC will still be able to

evade detection. Hence we focus on the wide majority of attackers that leverage and customize common malware toolkits.

### 4.2.3 Testbed

In our evaluation we rely on 4 recent, public and labelled datasets of network traffic that include communications generated by botnet-related pieces of malware: CTU-13[2] [194], IDS2017[3] [189], CIC-IDS2018[4] [189], UNB-CA Botnet[5] [195]. As these datasets involve different types of attacks, we only consider those portions that include botnet-related traffic. Some of these datasets are readily available in netflow format; for those that only include raw packet data, we generate the corresponding flows through Argus[6]. We summarize the meaningful metrics of each dataset in Table 4.2, which reports the total amount of packets, internal hosts, flows and number of botnet families included. We exclude those families that present less than 100 samples, as their scarcity may lead to the creation of training sets that would generate poor detection results [195], [196]. We can observe that the considered datasets are a valid representation of medium-to-large network scenarios.

TABLE 4.2: Datasets metrics.

| Dataset | Packets | Devices | Botnet Flows | Legitimate Flows | Botnet Families |
|---|---|---|---|---|---|
| CTU-13 | 855 866 143 | 150 | 443 906 | 19 199 170 | 6 |
| IDS2017 | 5 776 888 | 111 | 1 966 | 189 067 | 1 |
| CIC-IDS2018 | 13 486 990 | 450 | 283 429 | 760 824 | 1 |
| UNB-CA Botnet | 14 502 782 | 369 | 238 415 | 345 113 | 10 |

We now provide a more thorough description of the CTU-13 dataset, due to its relevance in Network Intrusion Detection scenarios that involve machine learning algorithms [197], [198]. The CTU-13 is a public dataset composed of multiple collections of network data captured at the Czech Technical University in Prague, and contains labelled network traffic generated by various botnet variants and mixed with normal and background traffic. These flows are captured in a network environment with hundreds of hosts, while the malicious traffic is generated by infecting machines with malware related to several botnet families [194]. Overall, the CTU-13 contains 13 distinct collections of different botnet activity, where each collection

[2] CTU-13 dataset: https://www.stratosphereips.org/datasets-ctu13
[3] IDS2017 dataset: https://www.unb.ca/cic/datasets/ids-2017.html
[4] CIC-IDS2018 dataset: https://www.unb.ca/cic/datasets/ids-2018.html
[5] UNB-CA Botnet dataset: https://www.unb.ca/cic/datasets/botnet.html
[6] Argus software: https://qosient.com/argus/argusnetflow.shtml

refers to one botnet variant of the 6 considered families: `Neris`, `Rbot`, `Virut`, `Menti`, `Murlo`, `NSIS.ay`. We report the meaningful metrics of each collection in Table 4.3, which also includes the botnet-specific piece of malware and the number of infected machines. This Table highlights the massive amount of included data, which can easily represent the network behavior of a medium-to-large real organization. Nevertheless, we remark that in our evaluation, we prefer not to consider the `Sogou` botnet because of the limited amount of its malicious samples.

TABLE 4.3: Meaningful metrics of the `CTU-13` dataset. Source: [194].

| Dataset | Duration (hrs) | Size (GB) | Packets | Netflows | Malicious | Benign | Malware | # Bots |
|---|---|---|---|---|---|---|---|---|
| 1 | 6.15 | 52 | 71 971 482 | 2 824 637 | 40 959 | 2 783 677 | Neris | 1 |
| 2 | 4.21 | 60 | 71 851 300 | 1 808 122 | 20 941 | 1 787 181 | Neris | 1 |
| 3 | 66.85 | 121 | 167 730 395 | 4 710 638 | 26 822 | 4 683 816 | Rbot | 1 |
| 4 | 4.21 | 53 | 62 089 135 | 1 121 076 | 1 808 | 1 119 268 | Rbot | 1 |
| 5 | 11.63 | 38 | 4 481 167 | 129 832 | 901 | 128 931 | Virut | 1 |
| 6 | 2.18 | 30 | 38 764 357 | 558 919 | 4 630 | 554 289 | Menti | 1 |
| 7 | 0.38 | 6 | 7 467 139 | 114 077 | 63 | 114 014 | Sogou | 1 |
| 8 | 19.5 | 123 | 155 207 799 | 2 954 230 | 6 126 | 2 948 104 | Murlo | 1 |
| 9 | 5.18 | 94 | 115 415 321 | 2 753 884 | 184 979 | 2 568 905 | Neris | 10 |
| 10 | 4.75 | 73 | 90 389 782 | 1 309 791 | 106 352 | 1 203 439 | Rbot | 10 |
| 11 | 0.26 | 5 | 6 337 202 | 107 251 | 8 164 | 99 087 | Rbot | 3 |
| 12 | 1.21 | 8 | 13 212 268 | 325 471 | 2 168 | 323 303 | NSIS.ay | 3 |
| 13 | 16.36 | 34 | 50 888 256 | 1 925 149 | 39 993 | 1 885 156 | Virut | 1 |

To generate each collection, the authors first capture the network data in specific packet-capture (PCAP) files, and then convert them into *network flows*. This conversion process is performed by means of Argus, a network audit system. Argus presents a client-server architecture: the server component processes packets (either PCAP files or live packet data) and generates detailed status reports of all the netflows in the packet stream, which are then provided to the dedicated clients. By inspecting the `CTU-13`, we can assume that the client used by the authors to extract the netflows from each individual PCAP file is `ra`. The output of this conversion process is a CSV file. The final step is the labeling of each individual network flow: indeed, the authors provide an additional "Label" field, which separates legitimate from illegitimate flows. More specifically, benign flows correspond to the *normal* and *background* labels; whereas the *botnet* and *CnC-channel* labels denote malicious samples.

### 4.2.4 Cyber detectors

Our experiments involve multiple botnet detectors, each based on a different machine learning classifier among the following: Random Forest (RF), Decision Trees

(DT), AdaBoost (AB), Feedforward Deep Neural Network (FNN), K-Nearest Neighbor (KNN), Gradient Boosting (GB), Linear Regression (LR), Support Vector Machines (SVM), Naive Bayes (NB), ExtraTrees (ET), Bagging (Bag), Stochastic Gradient Descent Linear Classifier (SGD). We focus on these algorithms since previous work demonstrate their applicability to the task of identifying botnet traffic [174], [196], [199]. As each dataset may include multiple botnet families, we build our detectors by using a multi-instance approach: every instance is developed by training every classifier on each individual botnet family per dataset. This is motivated by the fact that machine learning methods tend to perform better when they address a specific problem (that is, a specific botnet family) rather than being used as a catch-all solution [196], [199], [200]. More formally, let $A$ be the number of considered machine learning algorithms (i.e., $A = 12$), let $S$ be the number of involved datasets ($S = 4$), and let $S_f^i$ the number of botnet families included in dataset $S^i$ ($0 < i \leq S$); then, we devise a total of $G = A \cdot \sum_{i=1}^{S} S_f^i$ (instances of) detectors – in our case, $G = 216$. Such a large amount of models is necessary in order to gauge how different machine learning classifiers, trained in different network environments, respond against adversarial attacks.

Each detector is trained, validated and evaluated individually. For each botnet variant, we generate a dedicated training set containing both benign and malicious samples belonging to that family; all instances share the same legitimate-to-illegitimate flow ratio in the training sets. Formally, let $D$ be the set of all the traces of network flows considered in the testbed, and let $D^l \subset D$ and $D^m \subset D$ be the sets of all legitimate and malicious samples in $D$, respectively (so that $D^l \cup D^m = D$, and $D^l \cap D^m = \emptyset$). Now, let $D^b$ be the set of malicious flows corresponding to the $b$ botnet family, so that $\bigcup_{b=1}^{6} D^b = D^m$. We train each detector's instance corresponding to the $b$ botnet family with samples randomly extracted from $D^l$ and $D^b$, in a $20:1$ ratio. (The randomized extraction of samples is done to reduce the impact of selection bias.) The $20:1$ ratio is similar to that in [27], and it is motivated by the fact that in realistic settings the legitimate flows largely outnumber the botnet-generated flows. Other studies use even greater ratios [201]. The instances of each detector are trained with 80% of the botnet flows generated by each malware variant, and validated on the remaining 20%. These splits are close to those adopted in [27]. These models adopt feature sets that are similar to those adopted in related literature [174], [202] because they achieve appreciable detection rates. We integrate these features

with information about the IANA *port type* and with the *IP address type* (it can be either *internal* or *external*) for both the source and destination hosts, thus obtaining the list summarized in Table 4.4. For completeness, we remark that the code for the experiments is implemented in *Python3* and uses the *scikit-learn* [203], [204] toolkit.

TABLE 4.4: Features of the machine learning models.

| # | Feature name | Feature type |
|---|---|---|
| 1,2 | source/destination IP address type | Boolean |
| 3,4 | source/destination port | Numerical |
| 5 | flow direction | Boolean |
| 6 | connection state | Categorical |
| 7 | duration (seconds) | Numerical |
| 8 | protocol | Categorical |
| 9,10 | source/destination ToS | Numerical |
| 11,12 | outgoing/incoming bytes | Numerical |
| 13 | total transmitted packets | Numerical |
| 14 | total transmitted bytes | Numerical |
| 15,16 | source/destination port type | Categorical |
| 17 | bytes per second | Numerical |
| 18 | bytes per packet | Numerical |
| 19 | packets per second | Numerical |
| 20 | ratio of outgoing/incoming bytes | Numerical |

After performing multiple grid-search operations to determine its optimal configuration settings, each classifier is validated through 3-fold cross validation. We measure the performance of each detector through the *Precision* (see Eq. 2.1), *Detection Rate* (DR, or Recall – see Eq. 2.2) and *F1-score* (see Eq. 2.3). We anticipate that those detectors that obtain a score lower than 0.9 for any of these metrics are not considered in the evaluation, as such values are inadequate for NIDS deployed in real contexts.

### 4.2.5 Generation of adversarial datasets

We produce multiple adversarial datasets by manipulating the botnet netflows $D^b$ through feature modifications. Since the produced adversarial samples are used to evaluate the proposed approach, we consider the portion of botnet netflows from $D^b$ contained in the datasets used for the testing-phase, thus avoiding the submission of samples contained in the training set.

An attacker can evade detection by increasing the flow duration through a small latency; and the number of bytes (or packets) by adding random junk data. All these modifications can be introduced in the network behavior of the bots without altering their underlying logic. To reproduce a similar adversarial attack pattern, we generate adversarial samples by manipulating combinations of up to 4 features, such as the duration of the flows, the total number of transmitted packets, the number of outgoing(Src) or incoming(Dst) bytes. Table 4.5 reports the 15 groups of altered features denoted by $G$. As an example, adversarial samples belonging to group `1a` alter only the flow *duration*, while those of group `3c` include modifications to the *duration*, *dst_bytes* and *tot_packets* features. The feature manipulation is performed by augmenting each of these groups through 9 increment steps denoted by $S$; these steps are fixed for all the possible combinations. Hence, for each botnet family, we produce 135 adversarial collections, thus resulting in a total of 2430 adversarial datasets (given by 15[groups of altered features] $\times$ 9[increment steps] $\times$ 18[botnet families]).

TABLE 4.5: Groups of altered features.

| **Group** (g) | **Altered features** |
|---|---|
| **1a** | Duration (in seconds) |
| **1b** | Src_bytes |
| **1c** | Dst_bytes |
| **1d** | Tot_pkts |
| **2a** | Duration, Src_bytes |
| **2b** | Duration, Dst_bytes |
| **2c** | Duration, Tot_pkts |
| **2d** | Src_bytes, Tot_pkts |
| **2e** | Src_bytes, Dst_bytes |
| **2f** | Dst_bytes, Tot_pkts |
| **3a** | Duration, Src_bytes, Dst_bytes |
| **3b** | Duration, Src_bytes, Tot_pkts |
| **3c** | Duration, Dst_bytes, Tot_pkts |
| **3d** | Src_bytes, Dst_bytes, Tot_pkts |
| **4a** | Duration, Src_bytes, Dst_bytes, Tot_pkts |

Table 4.6 reports the relationship between each step and the corresponding feature increments where *Duration* is measured in seconds. As an example, the adversarial datasets obtained through the `VI` step of the group `1b` have the values of their flow outgoing bytes increased by 128. The adversarial datasets obtained through the `II` step of the group `3c` have the values of their flow duration, incoming bytes and total packets increased by 2. There is a greater focus on small increments since they are easier to achieve and they are still able to generate samples that evade detection. The rationale behind the choice of the values shown in Table 4.6 is the following: our

objective is to generate adversarial malicious samples that are only marginally different from their original counterparts, as shown in [179]. Although the exact numbers have been selected arbitrarily by adopting the powers of 2 for convenience, our goal is to represent the effects of small, but sensible variations of these features. Furthermore, introducing these small perturbations is a realistic and sensible task for the type of attacker considered in our threat model (e.g. [193]). On the other hand, excessive increases higher than those shown in Table 4.6 may generate anomalous network flows that can be detected by different defensive mechanisms (e.g., [81]). Moreover, increasing the duration of each flow above 120 seconds may exceed the duration limits of the flow collector. The generated adversarial flows represent a realistic attack pattern that is compatible with the considered threat model, and which can be easily achieved by botmasters [185], [193].

TABLE 4.6: Increment steps of each feature for generating realistic adversarial samples.

| **Step** (s) | **Duration** | **Src_bytes** | **Dst_bytes** | **Tot_pkts** |
|---|---|---|---|---|
| **I** | +1 | +1 | +1 | +1 |
| **II** | +2 | +2 | +2 | +2 |
| **III** | +5 | +8 | +8 | +5 |
| **IV** | +10 | +16 | +16 | +10 |
| **V** | +15 | +64 | +64 | +15 |
| **VI** | +30 | +128 | +128 | +20 |
| **VII** | +45 | +256 | +256 | +30 |
| **VIII** | +60 | +512 | +512 | +50 |
| **IX** | +120 | +1024 | +1024 | +100 |

The generation of the adversarial datasets is described in Algorithm 2, where $\mathcal{A}(\cdot)$ denotes the operator indicating an adversarially manipulated input. We remark the importance of the operation on line 19, because it shows that some features are mutually dependent. For example, for consistency reasons, increasing the flow duration requires to update also the bytes per second and the packets per second.

The obtained adversarial datasets (which include only the manipulated malicious samples) are then used to test each classifier. The effectiveness of these attacks is measured through the following *Attack Severity* ($AS$) score:

$$AS = 1 - \frac{DR(\text{after the attack})}{DR(\text{before the attack})} \tag{4.1}$$

Higher (lower) values of $AS$ imply attacks in which greater (lower) amounts of adversarial samples have evaded detection.

**Algorithm 2:** Algorithm for generating datasets of adversarial samples.

**Input:** List of datasets of malicious flows $X^m$ divided in botnet-specific sets $X^b$; list of altered features groups $G$; list of feature increment steps $S$.
**Output:** List of adversarial datasets $\mathcal{A}(X^m)$.

$\mathcal{A}(X^m) \leftarrow$ emptyList();
**foreach** *group* $g \in G$ **do**
  **foreach** *step* $s \in S$ **do**
    **foreach** *dataset* $X^b \in X^m$ **do**
      $\mathcal{A}_s^g(X^b) \leftarrow CreateOneDataset(s, g, X^b)$;
      Insert $\mathcal{A}_s^g(X^b)$ in $\mathcal{A}(X^m)$;
**return** $\mathcal{A}(X^m)$
// Function for creating a single adversarial dataset $\mathcal{A}_s^g(X^b)$ corresponding to a botnet-specific dataset $X^b$, a specific altered feature group $g$, and a specific increment step $s$.
**Function** *CreateOneDataset*$(s, g, X^b)$
  $\mathcal{A}_s^g(X^b) \leftarrow$ emptyList();
  **foreach** *sample* $x^b \in X^b$ **do**
    $\mathcal{A}_s^g(x^b) \leftarrow AlterSample(s, g, x^b)$;
    Insert $\mathcal{A}_s^g(x^b)$ in $\mathcal{A}_s^g(X^b)$;
  **return** $\mathcal{A}_s^g(X^b)$
// Function for creating a single adversarial sample $\mathcal{A}_s^g(x^b)$ corresponding to a botnet-specific sample $x^b$, a specific altered feature group $g$, and a specific increment step $s$.
**Function** *AlterSample*$(s, g, x^b)$
  $\mathcal{A}_s^g(x^b) \leftarrow x^b$;
  Increment features $g$ of $\mathcal{A}_s^g(x^b)$ by $s$;
  Update features of $\mathcal{A}_s^g(x^b)$ that depend on $g$;
  **return** $\mathcal{A}_s^g(x^b)$

### 4.2.6 Evaluation results

The experimental evaluation has a twofold objective:

1. confirm that the proposed detectors exhibit appreciable performance in non-adversarial settings;

2. evaluate the impact of our evasion attacks by testing these detectors against the adversarial samples.

We now present the results of our large experimental campaign by addressing both of these points. Finally, we also provide a more in-depth study on the effectiveness of adversarial perturbations against the detectors based on Random Forests trained on the `CTU-13` dataset.

**Performance in non-adversarial settings** We begin by determining which detectors reach a performance that complies with real-world requirements. Indeed, it would be unfair to show that certain adversarial attacks are effective against detectors that perform poorly even in non-adversarial scenarios. Thus, we train and test

TABLE 4.7: Performance in non-adversarial settings.

| Dataset | *F1-score* (std. dev.) | *Precision* (std. dev.) | *Recall* (std. dev.) |
|---|---|---|---|
| CTU-13 | 0.957 (0.029) | 0.958 (0.031) | 0.956 (0.028) |
| IDS2017 | 0.996 (0.002) | 0.999 (0.001) | 0.993 (0.003) |
| CIC-IDS2018 | 0.999 ($< 0.001$) | 0.999 ($< 0.001$) | 0.999 ($< 0.001$) |
| UNB-CA Botnet | 0.991 (0.017) | 0.992 (0.021) | 0.991 (0.017) |
| Average | 0.986 (0.011) | 0.987 (0.012) | 0.985 (0.011) |

the 12 machine learning approaches considered in this work against the malware families included in the 4 reference datasets, as described in Section 4.2.3.

To avoid a negative bias caused by ML approaches that are not suitable for the detection of a given botnet family or that do not perform well on a given dataset, results of Table 4.7 only consider the subset of all possible detectors that achieve appreciable performance for the considered detection task[7]. By applying this inclusion score we selected a total of 145 different detectors: 54 detectors for the CTU-13 dataset, 8 for IDS2017, 8 for CIC-IDS2018 and 75 for UNB-CA Botnet. We can observe that several (≈20%) detectors do not achieve suitable performance scores. This is motivated by the fact that some classifiers may not be appropriate for the given network environment.

Aggregated experimental results obtained by the 145 selected detectors are outlined in Table 4.7.

In this table, rows indicate a specific dataset, while columns represent the value of F1-Score, Precision and Recall metrics. Each cell contains the average value of a given metric, together with its standard deviation enclosed in parentheses. We can see that the selected detectors achieve good detection performance, comparable to state of the art solutions and consistent with results achieved in related work [174], [196].

Besides average and standard deviation, we also provide a graphical depiction of the considered performance metrics through the boxplot diagrams shown in Figure 4.2, representing the distribution of the results for each dataset. This figure includes four different diagrams, one for each dataset. Each diagram shows three boxplots, representing the results for F1-Score, Precision and Recall, respectively. Boxplots show that the performance of all algorithms are comparable and consistently

[7]The inclusion criteria is for all performance metrics (F1-Score, Precision and Recall) to be equal or above 0.9.

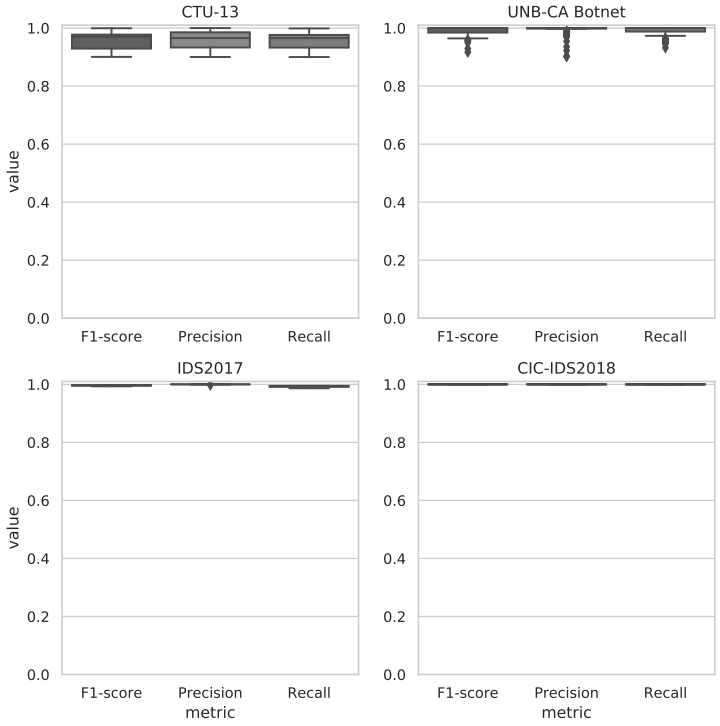



FIGURE 4.2: Distribution of F1-Score, Precision and Recall for all detectors and all datasets.

good across all datasets. Results are particularly promising for the `CIC-IDS2018` dataset, where all the detectors achieve a F1-score that is very near to 1. We highlight that in previous work similar results led authors to conclude that ML-based detectors can be successfully applied to real network environments [174], [196], but that these previous work did not account for adversarial attacks.

The 145 detectors included in this evaluation represent our baseline for the subsequent experiments, and thus will be subject to the adversarial attacks and defenses based on feature removal.

**Adversarial Attacks evaluation** We now evaluate the performance of our baseline detectors in the considered adversarial setting. We generate the adversarial datasets and test all the 145 detectors against adversarial attacks by following the procedure explained in Section 4.2.5. Aggregated experimental results are outlined in Table 4.8. This table compares, for each dataset, the average (and standard deviation) Recall of the baseline detectors with the Recall obtained on the adversarial samples. We focus on the Recall metric since it reflects the number of malicious samples that the detector is able to identify. The last column of Table 4.8 shows the Attack Severity (see

Equation 4.1), which expresses the effectiveness of adversarial attacks in reducing the Recall of a detector.

TABLE 4.8: Effects of the adversarial attacks.

| **Dataset** | ***Recall*** **baseline** (std. dev) | ***Recall*** **adversarial** (std. dev) | ***Attack*** textittextbfSeverity (std. dev) |
|---|---|---|---|
| CTU-13 | 0.956 (0.028) | 0.372 (0.112) | 0.609 (0.110) |
| IDS2017 | 0.993 (0.003) | 0.656 (0.102) | 0.327 (0.103) |
| CIC-IDS2018 | 0.999 ($< 0.001$) | 0.564 (0.112) | 0.436 (0.112) |
| UNB-CA Botnet | 0.991 (0.017) | 0.588 (0.218) | 0.328 (0.212) |
| Average | 0.985 (0.011) | 0.545 (0.136) | 0.425 (0.134) |

From Table 4.8, it is clear that even the simple but realistic adversarial samples considered in our experimental evaluation manage to cause a significant drop in the Recall of ML-based cyber detectors. A more reflective comparison of the effects of adversarial perturbations on the Recall for all classifiers of each dataset is proposed in Figure 4.3. For each dataset, the first boxplot represents the distribution of the Re-

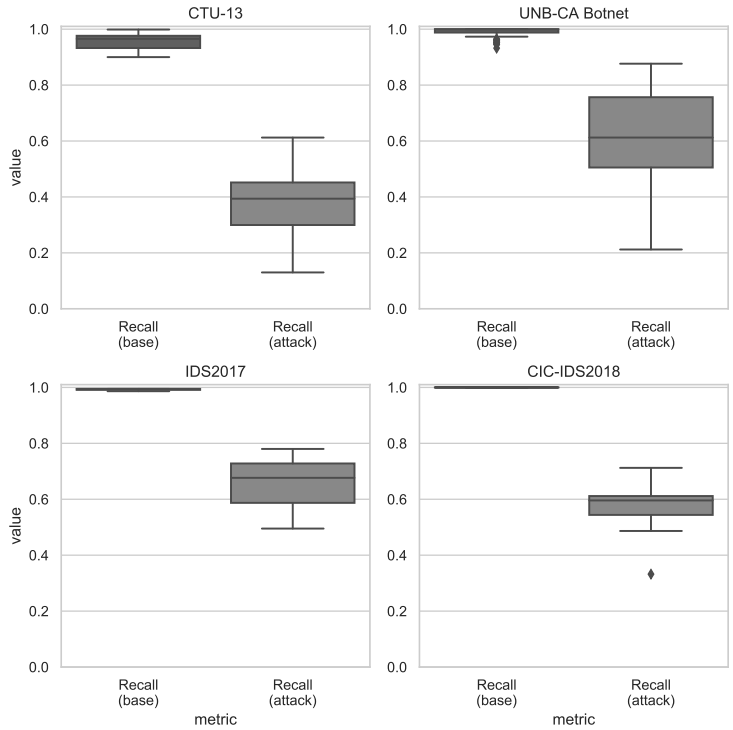



FIGURE 4.3: Comparison between the distributions of the Recall metric in the non-adversarial (baseline) and adversarial (attack) scenarios for all datasets.

call across all detectors in non-adversarial settings, while the second boxplot shows

results for the same metric in case of adversarial attacks. From these boxplots we can see that the detrimental effects of adversarial samples change significantly for different datasets and detectors. As an example, for the `UNB-CA Botnet` dataset the Recall of the more resilient ML-detector (based on the FNN algorithm) falls from 0.932 to 0.597, while the Recall for the most impacted ML-detector (based on SVM) for the same dataset falls from 0.914 to 0.198. The Attack Severity is even worse for the `CTU-13` dataset: in this case the Recall for the less affected ML-detection algorithm (based on RF) drops from 0.967 to 0.439. The effects of adversarial attacks against different datasets is summarized in Figure 4.4, which compares the Attack Severities for all detectors across all datasets.

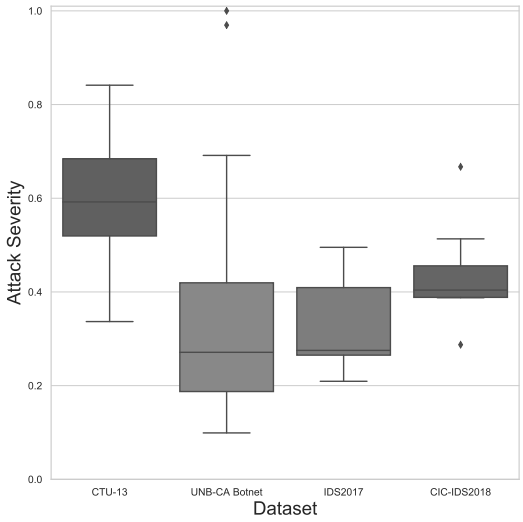



FIGURE 4.4: Comparison of the distribution of Attack Severity for all the detectors among the 4 different datasets.

We remark that none of the tested 145 detectors that exhibited good performance in the non-adversarial settings is able to maintain F1-Score, Precision and Recall above 0.9 while analyzing adversarial samples. Hence despite good classification results achieved in previous papers, it is clear that adversarial samples represent a huge menace for real-world applications of ML-detectors to the problem of botnet detection.

We conclude this section by presenting the results obtained by the 5 best detectors for each dataset in non adversarial settings, which are provided in Tables 4.9a through 4.9d.

The purpose of these tables, which show the average performance of each considered algorithm throughout our evaluation, is to highlight how even algorithms

TABLE 4.9: Results of the Top 5 algorithms on each individual datasets.

| | Baseline | | | Attack | |
|---|---|---|---|---|---|
| **Algorithm** | *F1-score* | *Precision* | *Recall* | *Recall* | *Attack Severity* |
| RF | 0.9694 | 0.9722 | 0.9668 | 0.4390 | 0.5461 |
| AB | 0.9722 | 0.9748 | 0.9696 | 0.4074 | 0.5803 |
| FNN | 0.9458 | 0.9454 | 0.9462 | 0.3141 | 0.7261 |
| KNN | 0.9296 | 0.9273 | 0.9320 | 0.2982 | 0.6806 |
| Bag | 0.9745 | 0.9799 | 0.9693 | 0.4007 | 0.5869 |

(A) `CTU-13` Top 5 algorithms results.

| | Baseline | | | Attack | |
|---|---|---|---|---|---|
| **Algorithm** | *F1-score* | *Precision* | *Recall* | *Recall* | *Attack Severity* |
| AB | 0.9972 | 1 | 0.9945 | 0.7455 | 0.2504 |
| FNN | 0.9959 | 0.9972 | 0.9945 | 0.5991 | 0.3975 |
| KNN | 0.9959 | 1 | 0.9918 | 0.5512 | 0.4442 |
| ET | 0.9972 | 1 | 0.9945 | 0.7333 | 0.2626 |
| GB | 0.9945 | 1 | 0.9891 | 0.7221 | 0.2699 |

(B) `IDS2017` Top 5 algorithms results.

| | Baseline | | | Attack | |
|---|---|---|---|---|---|
| **Algorithm** | *F1-score* | *Precision* | *Recall* | *Recall* | *Attack Severity* |
| RF | 0.9999 | 0.9999 | 0.9999 | 0.5965 | 0.4034 |
| AB | 0.9997 | 0.9999 | 0.9996 | 0.5632 | 0.4365 |
| FNN | 0.9997 | 0.9999 | 0.9995 | 0.7123 | 0.2873 |
| KNN | 0.9998 | 0.9999 | 0.9998 | 0.4866 | 0.5132 |
| ET | 0.9999 | 0.9999 | 0.9999 | 0.6023 | 0.3976 |

(C) `CIC-IDS2018` Top 5 algorithms results.

| | Baseline | | | Attack | |
|---|---|---|---|---|---|
| **Algorithm** | *F1-score* | *Precision* | *Recall* | *Recall* | *Attack Severity* |
| RF | 0.9974 | 0.9997 | 0.9951 | 0.6856 | 0.3110 |
| KNN | 0.9496 | 0.9479 | 0.9516 | 0.6167 | 0.3507 |
| ET | 0.9993 | 0.9999 | 0.9987 | 0.6831 | 0.3160 |
| FNN | 0.9215 | 0.9113 | 0.9321 | 0.5978 | 0.2756 |
| AB | 0.9955 | 0.9971 | 0.9939 | 0.6840 | 0.3118 |

(D) `UNB-CA Botnet` Top 5 algorithms results.

that achieve near-perfect results in non-adversarial settings can be heavily affected by such simple evasion mechanisms.

**In-depth analysis of the Random Forest detectors on the `CTU-13` dataset** To provide a more in-depth analysis of our study, we present the results of the Random Forest-based detector on the `CTU-13` dataset. We begin by showing in Table 4.10 the baseline performance of each instance of the detector in non-adversarial settings. These results are obtained when each botnet-specific instance of the detector is tested against a the malicious flows generated by its specific malware variant, along with legitimate network flows. We integrate the chosen performance metrics with the rate of false positives ($FP$) and false negatives ($FN$).

TABLE 4.10: Baseline performance for each instance of the RF detector on the `CTU-13` dataset.

| Malware | *FP rate* | *FN rate* | *Precision* | *Recall* | *F1-score* |
|---|---|---|---|---|---|
| `Neris` | 0.0014 | 0.0472 | 0.9624 | 0.9528 | 0.9575 |
| `Rbot` | $< 0.0001$ | 0.0015 | 0.9999 | 0.9985 | 0.9992 |
| `Virut` | 0.0003 | 0.0525 | 0.9871 | 0.9475 | 0..9669 |
| `Menti` | 0 | 0.0015 | 1 | 0.9967 | 0.9983 |
| `Murlo` | 0 | 0.0162 | 1 | 0.9838 | 0.9918 |
| `NSIS.ay` | $< 0.0001$ | 0.1557 | 0.9872 | 0.8443 | 0.9102 |

These results denote that our botnet detector obtains appreciable performance under "normal" circumstances, as each classifier exhibits low $FP$ and $FN$ rates, while retaining high Precision and Recall scores.

Next, we evaluate the botnet detector with the crafted adversarial samples. We remark that each instance of the classifier is tested only with the adversarial samples of its corresponding malware variant. Since we are interested in determining how many adversarial samples are classified as negatives, we base our evaluation on the *Recall* (or *detection rate)*. The results of the evaluation are presented in Tables 4.11, where each table reports the detection rates obtained by a specific instance of the classifier. In every table, columns designate the *group of altered features* (described in Table 4.5), whereas rows represent the *increment steps* (described in Table 4.6). Detection rates below 50% are denoted with a gray background, while those below 10% are written in bold. The baseline detection rate is provided in the caption of each table.

Consider as an example Table 4.11b, which refers to the detection rate for the `Rbot` botnet variant. The baseline detection rate is 0.9985, however the result of

column 1d and row I shows that the attacker only has to increase the duration of network communications by just 1 second to cause a drop of the detection rate from 0.9985 to a clearly unacceptable 0.1922. To reduce the detection rate below 1% an attacker only has to add 128 bytes of useless data and generate additional 20 network packets, as shown by the value 0.0094 in column 2e and row VI.

As expected, we observe that the performance tends to decrease for higher increment steps and for groups of multiple altered features. Intuitively, higher increments of multiple features imply larger modifications to the adversarial samples with respect to the original samples, which negatively impact the performance of the botnet detector. However, we highlight that even small perturbations cause a severe drop to the detection rate: for example, the results in the second row of column 3b in all Tables 4.11 show that by simply increasing the *Duration*, *Src_Bytes* and *Tot_Pkts* by 2 units, it is possible to evade all instances with a good confidence (~60% to ~98%).

To provide a more intuitive representation of these results, we report in Figures 4.5 the results obtained by a single instance of the classifier for three different increment steps. In every figure, histograms represent the detection rates for each group of altered features, and the dotted horizontal line represents the baseline detection rate. Without loss of generality, we represent results of the classifier for the Neris botnet. This choice is motivated by its higher number of malicious samples with respect to all other bot variants, thus leading to a more representative training dataset. We consider the I, IV and IX steps. From Figure 4.5a we notice that even the very small perturbations of the I increment step (combinations of: one second, one byte, one packet) reduce the detection rate of more than 20%, and up to 50% for some feature groups. On the other hand, the greater (but still realistic) perturbations reported in Figure 4.5b and Figure 4.5c cause almost all adversarial samples to be classified as benign flows, with detection rates always below 60% and even below 10% in most cases.

These results demonstrate the fragility of random forest classifiers to adversarial examples, and evidence the great problem posed by adversarial attacks against these types of security technologies.

TABLE 4.11: Detection rates on the adversarial datasets obtained by each instance of the classifier.

| Neris | 1a | 1b | 1c | 1d | 2a | 2b | 2c | 2d | 2e | 2f | 3a | 3b | 3c | 3d | 4a |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I | 0.7368 | 0.6664 | 0.6471 | 0.5927 | 0.4717 | 0.4454 | 0.6504 | 0.6070 | 0.5190 | 0.5090 | 0.4091 | 0.4875 | 0.4941 | 0.4715 | 0.4617 |
| II | 0.5792 | 0.6220 | 0.6396 | 0.4532 | 0.2877 | 0.2612 | 0.2243 | 0.5884 | 0.3466 | 0.3558 | 0.2413 | 0.1433 | 0.1564 | 0.3220 | 0.1436 |
| III | 0.4864 | 0.5036 | 0.5788 | 0.3192 | 0.1733 | 0.1996 | 0.1775 | 0.5106 | 0.1348 | 0.1723 | 0.1674 | 0.0788 | 0.0989 | 0.1299 | 0.0620 |
| IV | 0.4773 | 0.4495 | 0.5739 | 0.2534 | 0.1556 | 0.1760 | 0.1281 | 0.4728 | 0.0971 | 0.1253 | 0.1550 | 0.0541 | 0.0417 | 0.0945 | 0.0415 |
| V | 0.4749 | 0.2251 | 0.5712 | 0.2497 | 0.0614 | 0.1743 | 0.1209 | 0.2238 | 0.0571 | 0.1475 | 0.0680 | 0.0394 | 0.0416 | 0.0506 | 0.0385 |
| VI | 0.4695 | 0.1407 | 0.5584 | 0.2447 | 0.0332 | 0.1767 | 0.1220 | 0.1312 | 0.0502 | 0.1179 | 0.0293 | 0.0364 | 0.0404 | 0.0425 | 0.0345 |
| VII | 0.4685 | 0.1009 | 0.5173 | 0.2409 | 0.0586 | 0.2002 | 0.1184 | 0.1579 | 0.0381 | 0.0993 | 0.0538 | 0.0333 | 0.0354 | 0.0363 | 0.0325 |
| VIII | 0.4656 | 0.0825 | 0.4057 | 0.2346 | 0.0481 | 0.1631 | 0.1142 | 0.0911 | 0.0332 | 0.0824 | 0.0239 | 0.0726 | 0.0321 | 0.0315 | 0.0309 |
| IX | 0.4650 | 0.0611 | 0.3265 | 0.1899 | 0.0199 | 0.1119 | 0.1061 | 0.0768 | 0.0272 | 0.0726 | 0.0211 | 0.0223 | 0.0261 | 0.0276 | 0.0253 |

(A) Detection rates of the `Neris` instance of the classifier (Baseline $DR$: 0.9528).

| Rbot | 1a | 1b | 1c | 1d | 2a | 2b | 2c | 2d | 2e | 2f | 3a | 3b | 3c | 3d | 4a |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I | 0.9918 | 0.8456 | 0.8457 | 0.1922 | 0.8208 | 0.8208 | 0.1867 | 0.8418 | 0.1751 | 0.1754 | 0.8197 | 0.0203 | 0.0204 | 0.1751 | 0.0202 |
| II | 0.9917 | 0.8410 | 0.8420 | 0.1899 | 0.8182 | 0.8191 | 0.1846 | 0.8379 | 0.1538 | 0.1546 | 0.8157 | 0.0182 | 0.0185 | 0.0253 | 0.0182 |
| III | 0.9846 | 0.8080 | 0.8157 | 0.1896 | 0.8033 | 0.8139 | 0.1845 | 0.8079 | 0.0188 | 0.0192 | 0.8031 | 0.0175 | 0.0175 | 0.0178 | 0.0173 |
| IV | 0.9848 | 0.8082 | 0.8131 | 0.1896 | 0.8028 | 0.8030 | 0.1840 | 0.8059 | 0.0175 | 0.0179 | 0.8026 | 0.0169 | 0.0168 | 0.0172 | 0.0056 |
| V | 0.9852 | 0.8049 | 0.8116 | 0.1892 | 0.8118 | 0.7898 | 0.1831 | 0.7911 | 0.0168 | 0.0166 | 0.2391 | 0.0169 | 0.0028 | 0.0160 | 0.0046 |
| VI | 0.9853 | 0.8027 | 0.7979 | 0.1892 | 0.8004 | 0.2392 | 0.1835 | 0.7902 | 0.0094 | 0.0141 | 0.2386 | 0.0054 | 0.0015 | 0.0146 | 0.0036 |
| VII | 0.9850 | 0.8024 | 0.7944 | 0.1892 | 0.2419 | 0.2388 | 0.1834 | 0.7896 | 0.0073 | 0.0139 | 0.2377 | 0.0050 | 0.0015 | 0.0121 | 0.0014 |
| VIII | 0.9850 | 0.8023 | 0.7904 | 0.1891 | 0.8003 | 0.2479 | 0.1834 | 0.7852 | 0.0025 | 0.0136 | 0.2373 | 0.0098 | 0.0031 | 0.0049 | 0.0013 |
| IX | 0.9847 | 0.8022 | 0.7856 | 0.1888 | 0.8003 | 0.2377 | 0.1834 | 0.7856 | 0.0017 | 0.0045 | 0.2380 | 0.0024 | 0.0013 | 0.0047 | 0.0012 |

(B) Detection rates of the `Rbot` instance of the classifier (Baseline $DR$: 0.9985).

| Virut | 1a | 1b | 1c | 1d | 2a | 2b | 2c | 2d | 2e | 2f | 3a | 3b | 3c | 3d | 4a |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I | 0.6412 | 0.7175 | 0.7433 | 0.7326 | 0.6095 | 0.6122 | 0.2400 | 0.7072 | 0.6334 | 0.1488 | 0.2556 | 0.0789 | 0.0747 | 0.1376 | 0.0740 |
| II | 0.6115 | 0.7000 | 0.7275 | 0.6825 | 0.3345 | 0.3321 | 0.5607 | 0.1938 | 0.0922 | 0.0874 | 0.0700 | 0.0516 | 0.0549 | 0.0740 | 0.0500 |
| III | 0.6048 | 0.1319 | 0.1876 | 0.5815 | 0.0600 | 0.0530 | 0.5458 | 0.1215 | 0.4844 | 0.0664 | 0.0541 | 0.0397 | 0.0401 | 0.0528 | 0.0371 |
| IV | 0.6009 | 0.1097 | 0.1824 | 0.5628 | 0.0449 | 0.0506 | 0.5410 | 0.1099 | 0.0915 | 0.0492 | 0.0463 | 0.0353 | 0.0361 | 0.0476 | 0.0353 |
| V | 0.5898 | 0.0696 | 0.1616 | 0.5560 | 0.0364 | 0.0430 | 0.5398 | 0.0692 | 0.0392 | 0.0421 | 0.0343 | 0.0332 | 0.0329 | 0.0367 | 0.0316 |
| VI | 0.5834 | 0.0546 | 0.1612 | 0.5519 | 0.0297 | 0.0339 | 0.5340 | 0.0552 | 0.0348 | 0.0376 | 0.0313 | 0.0295 | 0.0278 | 0.0317 | 0.0261 |
| VII | 0.5704 | 0.0498 | 0.1407 | 0.5488 | 0.0263 | 0.0269 | 0.5315 | 0.0500 | 0.0297 | 0.0347 | 0.0333 | 0.0229 | 0.0245 | 0.0283 | 0.0222 |
| VIII | 0.7052 | 0.0365 | 0.0772 | 0.5418 | 0.0210 | 0.0248 | 0.5285 | 0.0376 | 0.0211 | 0.0267 | 0.0204 | 0.0143 | 0.0167 | 0.0235 | 0.0178 |
| IX | 0.6999 | 0.0316 | 0.0563 | 0.5294 | 0.0155 | 0.0156 | 0.5199 | 0.0304 | 0.0128 | 0.0171 | 0.0161 | 0.0098 | 0.0101 | 0.0173 | 0.0118 |

(C) Detection rates of the `Virut` instance of the classifier (Baseline $DR$: 0.9475).

| Menti | 1a | 1b | 1c | 1d | 2a | 2b | 2c | 2d | 2e | 2f | 3a | 3b | 3c | 3d | 4a |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I | 0.5852 | 0.0445 | 0.0434 | 0.8903 | 0.0358 | 0.0380 | 0.4300 | 0.0434 | 0.8219 | 0.7872 | 0.0380 | 0.4235 | 0.4278 | 0.4289 | 0.0347 |
| II | 0.9870 | 0.0445 | 0.0434 | 0.7524 | 0.0337 | 0.0380 | 0.8284 | 0.0380 | 0.4267 | 0.3985 | 0.0315 | 0.4311 | 0.4365 | 0.4278 | 0.4311 |
| III | 0.9870 | 0.0380 | 0.0380 | 0.7524 | 0.0054 | 0.0293 | 0.7904 | 0.0380 | 0.3540 | 0.3985 | 0.0054 | 0.3985 | 0.0597 | 0.3540 | 0.0597 |
| IV | 0.9870 | 0.0380 | 0.0380 | 0.7524 | 0.0054 | 0.0228 | 0.4517 | 0.0271 | 0.3540 | 0.3985 | 0.0033 | 0.0597 | 0.0000 | 0.3540 | 0.0000 |
| V | 0.9870 | 0.0000 | 0.0380 | 0.7524 | 0.0000 | 0.0000 | 0.4517 | 0.0000 | 0.3540 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| VI | 0.9870 | 0.0000 | 0.0098 | 0.7524 | 0.0000 | 0.0000 | 0.4517 | 0.0000 | 0.3540 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| VII | 0.9870 | 0.0000 | 0.0000 | 0.7524 | 0.0000 | 0.0000 | 0.4517 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| VIII | 0.9870 | 0.0000 | 0.0000 | 0.7524 | 0.0000 | 0.0000 | 0.4517 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| IX | 0.9870 | 0.0000 | 0.0000 | 0.7524 | 0.0000 | 0.0000 | 0.4517 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |

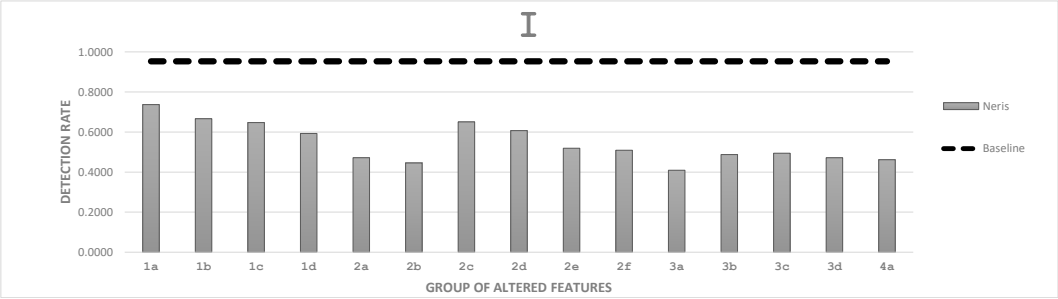
(D) Detection rates of the `Menti` instance of the classifier (Baseline $DR$: 0.9967).

| Murlo | 1a | 1b | 1c | 1d | 2a | 2b | 2c | 2d | 2e | 2f | 3a | 3b | 3c | 3d | 4a |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I | 0.2247 | 0.2221 | 0.2204 | 0.9694 | 0.2221 | 0.2247 | 0.2119 | 0.2204 | 0.2085 | 0.2094 | 0.2213 | 0.1974 | 0.1966 | 0.2051 | 0.1957 |
| II | 0.2170 | 0.2221 | 0.2204 | 0.9651 | 0.2153 | 0.2153 | 0.2085 | 0.2170 | 0.0570 | 0.2068 | 0.2077 | 0.0604 | 0.2060 | 0.0502 | 0.0545 |
| III | 0.2034 | 0.2170 | 0.2196 | 0.9302 | 0.1787 | 0.1779 | 0.0289 | 0.2119 | 0.0340 | 0.1660 | 0.1864 | 0.0221 | 0.0306 | 0.0315 | 0.0332 |
| IV | 0.0536 | 0.2085 | 0.2136 | 0.9115 | 0.0289 | 0.0264 | 0.0196 | 0.2043 | 0.0196 | 0.1583 | 0.0315 | 0.0196 | 0.0196 | 0.0196 | 0.0196 |
| V | 0.0511 | 0.2017 | 0.2043 | 0.9106 | 0.0187 | 0.0196 | 0.0187 | 0.1957 | 0.0187 | 0.1685 | 0.0187 | 0.0187 | 0.0187 | 0.0187 | 0.0187 |
| VI | 0.0434 | 0.1923 | 0.1966 | 0.9106 | 0.0187 | 0.0187 | 0.0187 | 0.0340 | 0.0187 | 0.1779 | 0.0187 | 0.0187 | 0.0281 | 0.0187 | 0.0187 |
| VII | 0.0434 | 0.0357 | 0.0357 | 0.9098 | 0.0187 | 0.0187 | 0.0179 | 0.0196 | 0.0179 | 0.1702 | 0.0187 | 0.0179 | 0.0179 | 0.0179 | 0.0179 |
| VIII | 0.0417 | 0.0187 | 0.0298 | 0.9064 | 0.0179 | 0.0289 | 0.0128 | 0.0289 | 0.0162 | 0.1660 | 0.0289 | 0.0153 | 0.0153 | 0.0170 | 0.0170 |
| IX | 0.0409 | 0.0179 | 0.0315 | 0.9047 | 0.0128 | 0.0289 | 0.0077 | 0.0281 | 0.0128 | 0.0136 | 0.0213 | 0.0077 | 0.0077 | 0.0136 | 0.0085 |

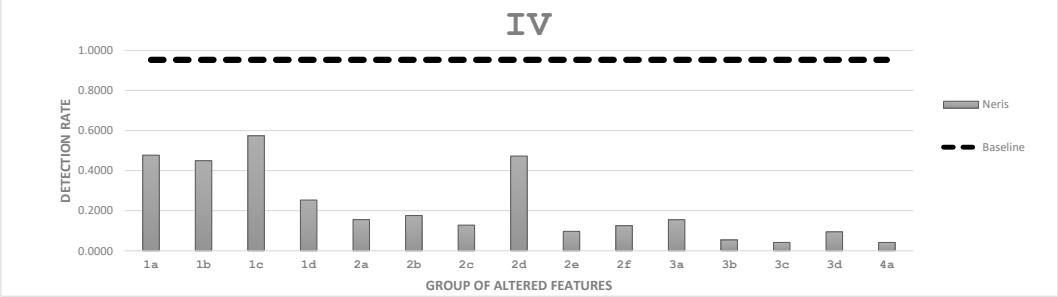
(E) Detection rates of the `Murlo` instance of the classifier (Baseline $DR$: 0.9838).

| NSIS.ay | 1a | 1b | 1c | 1d | 2a | 2b | 2c | 2d | 2e | 2f | 3a | 3b | 3c | 3d | 4a |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I | 0.8004 | 0.8026 | 0.8333 | 0.5768 | 0.7785 | 0.8026 | 0.6228 | 0.8114 | 0.5263 | 0.5482 | 0.7873 | 0.5921 | 0.6096 | 0.5329 | 0.6031 |
| II | 0.7610 | 0.7632 | 0.8246 | 0.5307 | 0.6601 | 0.7171 | 0.4298 | 0.7237 | 0.4561 | 0.4934 | 0.5570 | 0.3399 | 0.3750 | 0.4605 | 0.3465 |
| III | 0.7061 | 0.5504 | 0.7456 | 0.4364 | 0.4561 | 0.6382 | 0.2895 | 0.5132 | 0.3377 | 0.4013 | 0.4715 | 0.2259 | 0.2544 | 0.3421 | 0.2346 |
| IV | 0.6842 | 0.5241 | 0.7149 | 0.3640 | 0.3838 | 0.5417 | 0.2303 | 0.4715 | 0.2632 | 0.3070 | 0.3904 | 0.2237 | 0.2215 | 0.2303 | 0.2281 |
| V | 0.6689 | 0.4342 | 0.6184 | 0.3465 | 0.3575 | 0.5088 | 0.2193 | 0.4539 | 0.2346 | 0.2346 | 0.3333 | 0.2346 | 0.2061 | 0.2039 | 0.2193 |
| VI | 0.6272 | 0.3662 | 0.5614 | 0.3311 | 0.2873 | 0.3618 | 0.1886 | 0.3947 | 0.2149 | 0.2083 | 0.3136 | 0.1908 | 0.1623 | 0.1732 | 0.1667 |
| VII | 0.6118 | 0.3289 | 0.4956 | 0.3268 | 0.2675 | 0.3816 | 0.1513 | 0.3004 | 0.2018 | 0.1864 | 0.3092 | 0.1579 | 0.1272 | 0.1535 | 0.1382 |
| VIII | 0.6053 | 0.3048 | 0.4232 | 0.2917 | 0.2719 | 0.3443 | 0.1228 | 0.2741 | 0.1776 | 0.1425 | 0.2325 | 0.1294 | 0.0833 | 0.0877 | 0.0614 |
| IX | 0.5724 | 0.2895 | 0.2873 | 0.2763 | 0.2610 | 0.1974 | 0.0811 | 0.2478 | 0.1557 | 0.1294 | 0.1886 | 0.0570 | 0.0373 | 0.0702 | 0.0351 |

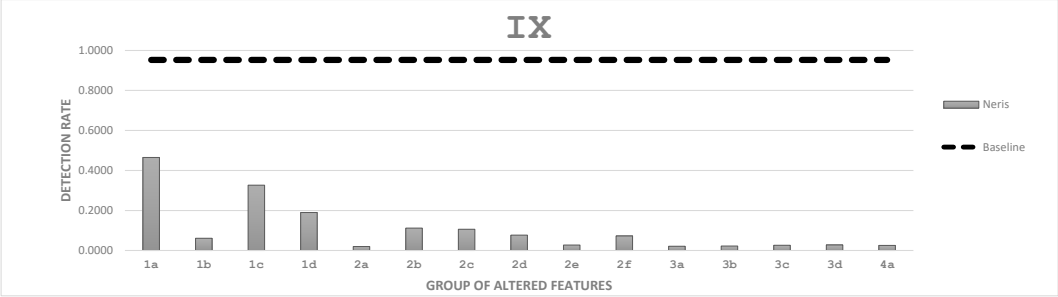
(F) Detection rates of the `NSIS.ay` instance of the classifier (Baseline $DR$: 0.8443).




(A) Results on the `I` increment step.




(B) Results on the `IV` increment step.




(C) Results on the `IX` increment step.

FIGURE 4.5: Detection rates of the `Neris` instance on the adversarial datasets obtained with three steps of every group of altered features.

# Chapter 5

# Countermeasures against Adversarial Attacks

After having introduced and explained the problem of adversarial attacks from a cybersecurity perspective in the previous chapter (see Section 4.1), and after having shown the high effectiveness of this threat (in Section 4.2), we devote this chapter to the countermeasures against this recent menace. We begin by providing an overview of existing countermeasures that have been proposed by the literature, which is corroborated with novel experiments that aim to assess the maturity of these techniques for protecting cyber detectors. Indeed, we will show that known defensive approaches against adversarial attacks present several drawbacks that limit their applicability in cybersecurity scenarios, therefore increasing the need for novel solutions that can be used as stepping stone for more resilient cyber defence systems. We address this demand in the second part of this chapter, where we propose novel methods that are oriented at countering both attacks at training- and at test-time; in addition, they can be adopted in a wide array of cyber security settings, such as network intrusion detection, and phishing detection. As a prerogative of the proposed solutions, the presented approaches have either negligible, zero or even positive impact on the detectors when they are not being targeted by adversarial attacks; furthermore, their integration into existing systems is provided at little- or no-cost (in terms of operations required). These characteristics make them suitable for realistic scenarios – which is a crucial aspect when evaluating the quality of security analytics methods. Moreover, all the proposed countermeasures are experimentally validated by extensive experimental campaigns. From the appreciable results obtained, alongside the evident room for further improvements, it is safe to assume that these proposals represent a valid step towards more secure security analytics

platforms.

The remainder of this chapter is structured as follows. Section 5.1 presents the current known defensive methods against adversarial attacks, paired with an original evaluation of some these techniques. Then, we present several innovative solutions against adversarial attacks: Section 5.2 describes a method to harden random forest-based botnet detectors against evasion attacks; Section 5.3 proposes an approach to counter poisoning attacks; and Section 5.4 presents an original solution to thwart evasion attempts against phishing detectors.

## 5.1 Existing Defences against Adversarial Attacks

Devising effective solutions against adversarial attacks is a challenging task. In this section, we present existing methods proposed in the literature that aim to mitigate these critical threats at both training- and test-time. Countermeasures can be divided into two groups: those conforming to the *security-by-design* paradigm that are effective against perfect-knowledge attacks; and *security-by-obscurity* methods that are only effective against partial- or zero-knowledge attacks. As an additional contribution, we perform original experiments that aim to assess the effectiveness of these methods in cybersecurity contexts. To this purpose, we conduct a thorough evaluation that involves adversarial attacks against machine learning-based NIDS. The experiments involve publicly available and labelled datasets representing the network behavior of medium- or large-sized enterprises, consisting of hundreds of hosts. Hence, we consider our evaluation to be a valid representation of the quality of existing and known techniques against adversarial perturbations. We anticipate that one of the main limitations of most solutions against adversarial attacks is that they may worsen the performance of the cyber detector in the absence of adversarial attacks, typically causing higher false positive rates (e.g., [45], [161], [183], [205]).

### 5.1.1 Defences against attacks at test-time

We present existing countermeasures against attacks performed at test-time. We recall from Section 4.1 (see Table 4.1) that there are no known examples of availability attacks at test-time, Hence, we focus on defences against attacks targeting the integrity of the system.

These threats involve the creation of specific samples that evade the detection mechanism. For example, an opponent can alter a malicious sample to induce its classification as a benign sample. The security-by-design countermeasures aim to improve the machine learning system capabilities to detect even adversarially manipulated samples.

- **Adversarial training**. These solutions train the model on datasets that include samples of possible adversarial attacks [206]. A recent proposal [45] suggests the adoption of a generative adversarial network (GAN) to automatically generate a similar dataset, achieving promising results. However, these approaches are not a final solution, because it is simply unfeasible to obtain a dataset that contains all possible variations of realistic adversarial samples.

- **Robust optimisation**. The authors in [207] and [208] propose techniques aimed at smoothing the decision boundaries of the machine learning algorithm, thus reducing the effects of adversarial samples. Similar solutions can help to mitigate some attacks, but expert opponents are still able to craft malicious samples that look like licit activities.

- **Feature selection**. Other proposals (e.g., [39], [183]) suggest training the detection model by considering only the subset of features that cannot be manipulated by an attacker. While this method can prevent certain types of evasion attacks, feature removal reduces the detection rates in non-adversarial scenarios [183].

- **Game theory**. These approaches represent the problem of adversarial attacks as a zero-sum game between the attacker and the defender, and work under several assumptions. They require a model of the attacker knowledge and capabilities that must be integrated into the machine learning algorithm. The optimal defence course against the modelled attacker is found when the system reaches an equilibrium. An example of application to spam detection is described in [208]. The main limitation of these strategies is that they are only able to counter attacks that strictly conform to the considered attacker's model, because even small deviations nullify their effectiveness. Since the cybersecurity world is intrinsically unpredictable and fuzzy, most of these solutions are not applicable to real contexts.

- **Ensemble methods**. The paper by Biggio et al. [209] shows that it is possible to counter evasion attacks at test-time by devising systems composed by multiple classifiers. However, each classifier represents a weak link in the security chain because the misconfiguration of even one component can lead to poor results, as shown in [166].

Most black- and gray-box evasion attacks involve a *probing* step, in which the adversary aims to gather information on the detector by submitting specific inputs to the system and observing the subsequent response. Thus, existing defences address these malicious exploratory activities by providing misleading information to the attacker. For example, the authors in [209] suggest mechanisms that are difficult to reverse-engineer or propose a randomization of the detector output. The problem of these solutions is that they tend to work against attackers with limited time or skill that adopt automated tools. Expert opponents can detect such deception activities and bypass them.

### 5.1.2 Defences against attacks at training-time

Attacks performed at training-time alter the decision process of the machine learning algorithm by modifying the configuration of the model before the training phases, that is, by manipulating the training dataset(s). Existing solutions focus on protecting the training dataset with the objective of minimizing the effects of adversarial perturbations. We identify the following two groups of security-by-design defences.

- **Data sanitization**. Poisoning attacks are countered through a data sanitization process that aims to detect and remove poisoned samples introduced in the training data [210]. The problem is that some assumptions of these approaches are not always applicable to the cybersecurity field. For example, the work in [211] assumes that each poisoning sample significantly affects the training process. This assumption is not valid in many situations in which an attacker introduces few samples just to avoid some specific detections of his interest. Other solutions [212] leverage the machine unlearning concept that allows the effects of poisoned data to be cancelled without the need to retrain the machine learning model. The main limitation of this approach is that it needs to know which (poisoned) data to unlearn, that is, it requires the knowledge of

which poisoned data samples have been introduced by the attacker. This is an unrealistic assumption in real cybersecurity contexts.

- **Ensemble methods**. The adoption of multiple-classifier systems can also be effective against attacks at training-time [211]. These solutions present the same advantages and problems characterizing their test-time version, that is, a misconfiguration of even one component can damage the results of the entire detection mechanism.

Defences against partial- or zero-knowledge attacks include the collection of training data from randomized sources [213] with the goal of making it harder for the attacker to devise effective adversarial samples; and the application of strategies to prevent the attacker from controlling the actual training dataset [213].

### 5.1.3 Evaluation results

To further emphasize the need for novel solutions against adversarial attacks, we present an experimental evaluation of two existing countermeasures against evasion attacks at test-time against botnet detectors: we begin by assessing the effectiveness of defenses based on *feature removal*, and then evaluate methods based on *adversarial retraining*. We anticipate that these evaluations are based on the same threat model described in Section 4.2.2, and are performed on the same testbeds described in Section 4.2.3. Regardless, we provide a brief description of the adopted experimental settings.

These experiments involve integrity violations performed at test-time. We consider an attacker that has already established a foothold within the enterprise's internal network by compromising one or more machines with botnet malware; these bots communicate with an external Command and Control server. The attacker model is based on the following three assumptions: his goal is to evade detection in order to expand his control of the internal network [75]; he knows that the organization adopts a botnet detector based on machine learning, which is trained on malware samples that are similar to the variant used by the bots; he can interact with the controlled bots, but he cannot access the botnet detector. To achieve his goal, the attacker plans to slightly modify the network communications performed by the bots (e.g., small increments in the amount of exchanged data and in the communications duration [193]) so that these small perturbations can induce misclassifications

of botnet flows. We simulate this realistic attack scenario by altering the following flow-based features: *sent_bytes*, *received_bytes*, *duration*, *total_packets*. This process is repeated for all the samples of each botnet variants (see Section 4.2.5 for the complete details of the generation of adversarial samples).

**Effectiveness of Feature Removal strategies** We begin by assessing the effectiveness of defensive strategies based on feature removal. These approaches have the appreciable benefit of being able to completely nullify the effectiveness of adversarial attacks. However, this advantage comes with a critical issue, which is a reduced performance of the detector in contexts that are not subject to adversarial attacks – for example by increasing the rate of false positives or false negatives. The magnitude of this performance drop is usually dependant on the importance of the features that have been excluded from the training set. To this purpose, after training and testing the ML-detectors described in Section 4.2.4 (we remark that their appreciable baseline results are presented in Table 4.7), we train them again by only considering features that are not affected by the considered evasion attacks. That is, we exclude the following features from the training dataset: *sent_bytes*, *received_bytes*, *duration*, *total_packets*. The "updated" detectors are then tested on the very same testing datasets used to determine the baseline performance of the original detectors. Experimental results are summarized in Table 5.1, which reports the performance metrics of interest (F1-score, Precision, Recall – see Eq. 2.1 through 2.3) for each considered dataset, averaged for all the chosen algorithms (after training them with the removed features).

TABLE 5.1: Detection results for ML detectors with feature removal.

| **Dataset** | ***F1-score*** (std. dev.) | ***Precision*** (std. dev.) | ***Recall*** (std. dev.) |
|---|---|---|---|
| CTU-13 | 0.803 (0.092) | 0.810 (0.089) | 0.799 (0.101) |
| IDS2017 | 0.503 (0.304) | 0.777 (0.388) | 0.596 (0.306) |
| CIC-IDS2018 | 0.859 (0.164) | 0.814 (0.212) | 0.942 (0.128) |
| UNB-CA Botnet | 0.691 (0.276) | 0.645 (0.285) | 0.808 (0.209) |
| Average | 0.714 (0.209) | 0.761 (0.2235) | 0.786 (0.186) |

By comparing Table 4.7 against Table 5.1 it is clear that feature removal causes a

considerable reduction in the average detection performance, which is now well below acceptable standards for real world cyber detectors. To also compare the distribution of F1-Score, Precision and Recall of the different detectors across all datasets we propose the boxplot diagrams of Figure 5.1, which can be directly compared with Figure 4.2.

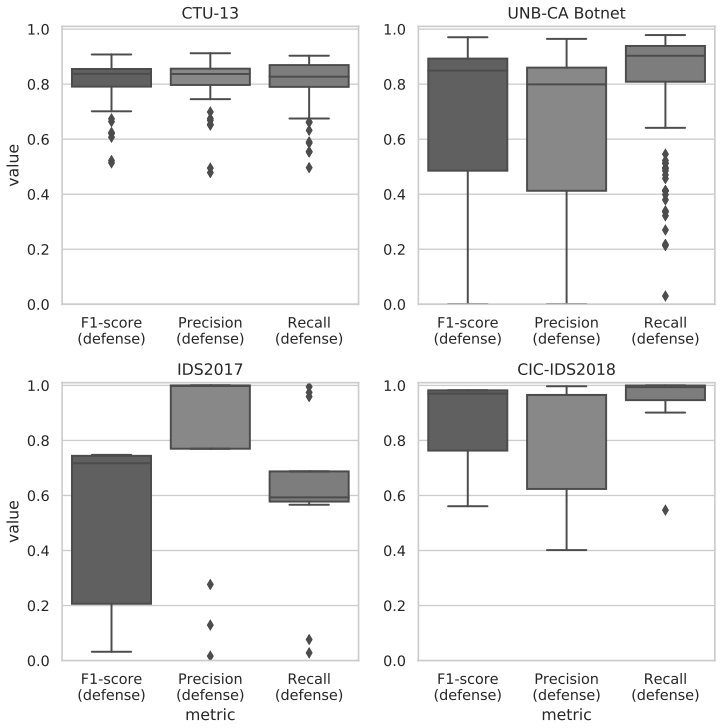



FIGURE 5.1: Distribution of F1-Score, Precision and Recall using feature removal for all detectors and all datasets.

We can observe a significant decrease for all the considered metrics. In particular, we highlight that the lower Precision leads to a relevant number of false positives, which are extremely undesirable for modern cyber defence platforms. This reduction in quality is explained by the fact that the removed features have a meaningful impact to the underlying mechanisms of the baseline detectors; thus, their exclusion causes an important performance drop, with most detectors obtaining scores that are well below acceptable in real contexts. However, we remark that by testing these detectors on the adversarial datasets, the detection rates achieved did not decrease (and the corresponding *Attack Severity* was 0): indeed, the main advantage of feature removal methods is to make certain kinds of adversarial perturbations completely ineffective. Our takeaway is that despite its ability to nullify the considered attack

TABLE 5.2: Results of the Top 5 algorithms on each individual dataset.

| | Baseline | | | Attack | | Defense | | |
|---|---|---|---|---|---|---|---|---|
| **Algorithm** | *F1-score* | *Precision* | *Recall* | *Recall* | *Attack Severity* | *F1-score* | *Precision* | *Recall* |
| RF | 0.9694 | 0.9722 | 0.9668 | 0.4390 | 0.5461 | 0.8564 | 0.8498 | 0.8641 |
| AB | 0.9722 | 0.9748 | 0.9696 | 0.4074 | 0.5803 | 0.8446 | 0.8487 | 0.8410 |
| FNN | 0.9458 | 0.9454 | 0.9462 | 0.3141 | 0.7261 | 0.7235 | 0.7734 | 0.6886 |
| KNN | 0.9296 | 0.9273 | 0.9320 | 0.2982 | 0.6806 | 0.6992 | 0.7265 | 0.6767 |
| Bag | 0.9745 | 0.9799 | 0.9693 | 0.4007 | 0.5869 | 0.8477 | 0.8516 | 0.8442 |

(A) `CTU-13` Top 5 algorithms results.

| | Baseline | | | Attack | | Defense | | |
|---|---|---|---|---|---|---|---|---|
| **Algorithm** | *F1-score* | *Precision* | *Recall* | *Recall* | *Attack Severity* | *F1-score* | *Precision* | *Recall* |
| AB | 0.9972 | 1 | 0.9945 | 0.7455 | 0.2504 | 0.7172 | 0.9779 | 0.5663 |
| FNN | 0.9959 | 0.9972 | 0.9945 | 0.5991 | 0.3975 | 0.7169 | 0.9344 | 0.5816 |
| KNN | 0.9959 | 1 | 0.9918 | 0.5512 | 0.4442 | 0.4292 | 0.2764 | 0.9591 |
| ET | 0.9972 | 1 | 0.9945 | 0.7333 | 0.2626 | 0.7456 | 1 | 0.5943 |
| GB | 0.9945 | 1 | 0.9891 | 0.7221 | 0.2699 | 0.7476 | 1 | 0.5969 |

(B) `IDS2017` Top 5 algorithms results.

| | Baseline | | | Attack | | Defense | | |
|---|---|---|---|---|---|---|---|---|
| **Algorithm** | *F1-score* | *Precision* | *Recall* | *Recall* | *Attack Severity* | *F1-score* | *Precision* | *Recall* |
| RF | 0.9999 | 0.9999 | 0.9999 | 0.5965 | 0.4034 | 0.9822 | 0.9653 | 0.9996 |
| AB | 0.9997 | 0.9999 | 0.9996 | 0.5632 | 0.4365 | 0.9709 | 0.9969 | 0.9463 |
| FNN | 0.9997 | 0.9999 | 0.9995 | 0.7123 | 0.2873 | 0.9696 | 0.9939 | 0.9465 |
| KNN | 0.9998 | 0.9999 | 0.9998 | 0.4866 | 0.5132 | 0.8225 | 0.7564 | 0.9012 |
| ET | 0.9999 | 0.9999 | 0.9999 | 0.6023 | 0.3976 | 0.9822 | 0.9653 | 0.9996 |

(C) `CIC-IDS2018` Top 5 algorithms results.

| | Baseline | | | Attack | | Defense | | |
|---|---|---|---|---|---|---|---|---|
| **Algorithm** | *F1-score* | *Precision* | *Recall* | *Recall* | *Attack Severity* | *F1-score* | *Precision* | *Recall* |
| RF | 0.9974 | 0.9997 | 0.9951 | 0.6856 | 0.3110 | 0.8912 | 0.8584 | 0.9283 |
| KNN | 0.9496 | 0.9479 | 0.9516 | 0.6167 | 0.3507 | 0.8144 | 0.7555 | 0.8871 |
| ET | 0.9993 | 0.9999 | 0.9987 | 0.6831 | 0.3160 | 0.8897 | 0.8544 | 0.9294 |
| FNN | 0.9215 | 0.9113 | 0.9321 | 0.5978 | 0.2756 | 0.7393 | 0.6779 | 0.8325 |
| AB | 0.9955 | 0.9971 | 0.9939 | 0.6840 | 0.3118 | 0.8926 | 0.8595 | 0.9303 |

(D) `UNB-CA Botnet` Top 5 algorithms results.

scenario, similar defensive strategies are inefficient and their application is discouraged in actual production environments. These results further motivate the need for novel defensive approaches that are both effective and applicable to modern network scenarios.

We conclude this evaluation by presenting the results obtained by the 5 best detection algorithms for each dataset in non adversarial settings, provided in Tables 5.2a through 5.2d, which are the natural expansion of Tables 4.9 (presented in Section 4.2.6). All these tables show the average performance of each considered algorithm throughout all the steps performed in our experimental evaluation (baseline, attack and defense).

The purpose of these tables is to highlight how even algorithms that achieve near-perfect results in non-adversarial settings can be heavily affected by such simple evasion mechanisms. Moreover, after the application of feature removal, only three algorithms (RF, AB and FNN) are able to achieve acceptable values of F1-Score, Precision and Recall, but only when applied to the `CIC-IDS2018` dataset.

**Effectiveness of Adversarial Retraining strategies** We now explore the efficacy of methods based on adversarial retraining. Without loss of generality, these experiments are based on detectors leveraging three popular machine learning algorithms (Random Forest, Feedforward Deep Neural Network, K-Nearest Neighbor) which are evaluated on the `CTU-13` dataset. To assess the quality of adversarial retraining countermeasures, we first train each detector and determine its baseline performance (through measuring their Precision, Recall, F1-score), and then verify the effectiveness of the considered evasion attacks by testing it on the generated adversarial datasets and measuring its Recall and Attack Severity. Then, we harden the detectors by inserting some of the adversarial samples that we manually crafted into their training sets (with the appropriate malicious label), we repeat the training process, and we test the classifiers again on the respective adversarial datasets.

The baseline results (that is, in non-adversarial settings) of each detector are presented in Table 5.11, from which we observe an appreciable performance of these algorithms that is compatible with real world requirements.

TABLE 5.3: Baseline performance of the classifiers.

| Algorithm | *Precision* | *Recall* | *F1-score* |
|---|---|---|---|
| **RF** | 0.9774 | 0.9684 | 0.9729 |
| **FNN** | 0.9616 | 0.9438 | 0.9526 |
| **KNN** | 0.9558 | 0.9375 | 0.9466 |

Next, we report in Table 5.4 the average results for the three detectors considered when are subject to the evasion attacks. We highlight that all these algorithms are severely affected by the adversarial samples: the detection rate in the second column is about one-third of the original rate.

Finally, we present the effectiveness of the implemented adversarial retraining strategy in Table 5.5, which compares the severity of the attacks before and after retraining. The decreased severity of the attack after retraining shows the validity of a similar approach. However, it should be observed that this technique does not guarantee detection against other types of adversarial perturbations: in other words,

TABLE 5.4: Effects of the evasion attack on each classifier.

| Algorithm | *Recall* (before the attack) | *Recall* (after the attack) | *Attack Severity* |
|---|---|---|---|
| **RF** | 0.9684 | 0.3429 | **0.6459** |
| **FNN** | 0.9438 | 0.3012 | **0.6809** |
| **KNN** | 0.9375 | 0.3121 | **0.6671** |

these kinds of approaches are only effective against the types of attacks that can be anticipated, and for which an augmented training set can be produced; conversely, an attacker that could manipulate the malicious samples by altering different features (for example, the *network ports*) would be not affected by a similar countermeasure.

TABLE 5.5: Evaluation of the countermeasure based on adversarial retraining.

| Algorithm | *Attack Severity* | *Attack Severity* (after Retraining) |
|---|---|---|
| **RF** | 0.6459 | **0.3842** |
| **FNN** | 0.6809 | **0.4089** |
| **KNN** | 0.6671 | **0.4772** |

We can conclude that existing defences, while providing some appreciable results in some contexts, are still immature and present critical issues that prevent their complete adoption in real scenarios. These results highlight the importance of developing novel countermeasures that not only improve the resiliency of machine learning cyber detectors against adversarial attacks, but that also do not cause significant performance drops in non-adversarial settings. The following sections will provide original contributions to mitigate the effectiveness of adversarial attacks against cyber detectors based on machine learning.

## 5.2 Countering Evasion Attacks on Random Forest Detectors

In this section, we propose a novel approach for hardening cyber detectors based on machine learning. We focus on the random forest algorithm due to its proven effectiveness for intrusion detection [27], [48], [49], [51], [136]; but also because recent studies (alongside this thesis) highlight its vulnerability to adversarial perturbations [182], [185], [214]. Our solution is based on the observation that existing machine learning cyber detectors rely on excessively rigid classification criteria: they

are typically trained through *class labels* that separate samples in disjointed categories where each sample may be either malicious or benign. A similar approach cannot work in the cyber domain where each sample may present more vague attributes. For this reason, we leverage the idea of introducing some degree of flexibility in the training data set by using *probability labels*. The intuition is that a model that uses probability labels instead of hard class labels can be more resilient to adversarial perturbations, and can achieve comparable or even superior results even in the absence of attacks. Our methodology has several applications in all fuzzy scenarios characterizing cybersecurity that involve classifiers based on random forests. As a first test case, we adopt it for devising botnet detectors based on network flows analyzers.

We validate our approach through a large set of experiments, performed on a set of publicly available and labelled traffic traces containing over 20 million network flows with benign and malicious samples of different malware families. These data sets capture the network behavior of medium-large enterprises and represent an appropriate setting for a realistic evaluation. The results of our evaluations demonstrate the effectiveness of the proposed method, which achieves a twofold advantage over the state of the art: in scenarios subject to adversarially manipulated inputs, it improves the detection rate up to 250%; in scenarios that are not subject to adversarial attacks, it achieves a similar or superior accuracy than existing techniques. This latter achievement is of particular importance because existing approaches that aim to counter adversarial attacks are often subject to a reduced performance in non-adversarial settings. Moreover, despite these promising results, our method presents room for further improvements. The proposed approach represents an original contribution to design robust detectors with high detection rates and strong enough against adversarial attacks. Hence, we are confident that this contribution represents a meaningful step towards more robust cyber defensive platforms based on machine learning against adversarial attacks.

### 5.2.1 Related work

Although the threats posed by adversarial inputs are clear, the few existing solutions are not immediately applicable to real contexts. For example, [45] and [206] propose to harden the classifier through multiple re-training steps based on adversarial samples. This is an interesting theoretic solution with practical limitations because it

requires the creation and continuous management of datasets with realistic adversarial samples. Moreover, [39] suggests to improve the robustness against evasion attacks by not considering the features that can be manipulated by an attacker. The problem of this approach is that it reduces accuracy in normal scenarios as shown in this thesis (see Section 5.1.3) but also in [150], [215]. On the other hand, our proposal is immediately applicable to real contexts as demonstrated by multiple experimental settings.

A more recent defensive method is the *defensive distillation*, which has been proposed within the image processing domain. The first use of distillation has been compression [216]: its aim was the reduction of the computational complexity of training a (deep) neural network architecture by transferring knowledge from a larger one to a smaller one. This allows the deployment of deep learning in low-powered devices which cannot rely on specialized processors to perform heavy computations. Then, the work by Papernot et al. [161] leveraged this *transferability* property to harden classifiers. While defensive distillation may work in mitigating adversarial perturbations against image classification, this technique is built and evaluated only on neural network algorithms [217]. Although cyber detectors based on this algorithm exist and can be hardened through the original distillation proposal [218], in cybersecurity scenarios detectors based on random forests outperform those relying on neural networks and other supervised methods [27], [48]–[51], [185], [215]. More recently, [196] evaluates different classifiers for the specific problem of botnet detection and confirms that random forest yields the best results. Finally, [219] proposes a NIDS that inspects network flows through a random forest classifier to identify botnets and obtains outstanding results with detection rates close to 99%. For this reason, we devise an original formulation of the distillation technique that is specifically aimed at hardening random forest detectors, thus allowing to devise robust defensive schemes for cyber detection based on machine learning. A recent work [147] shows that it is possible to evade the defensive distillation, however we observe that the considered threat model is unrealistic because it assumes an attacker with complete control of the detector: with similar privileges, attackers can (and most likely will) adopt measures much more invasive and disruptive than those based on adversarial perturbations. Other proposals on defenses against adversarial samples [164], [181] consider just SVM classifiers applied to malware analysis, which is out of the scope of this work. We are not aware of other

defensive mechanisms against evasion adversarial attacks that are applicable to random forest algorithms for network intrusion detection. Hence, we can conclude that the topic considered in this work is a promising research theme, which we address through a novel approach that hardens random forest-based detectors through an original defensive distillation method.

### 5.2.2 Proposed method

We propose a novel method that hardens machine learning detectors based on random forest against adversarial attacks. The idea comes from the observation that the excessively rigid classification criteria learned by machine learning algorithms in the training phase are vulnerable to subtle adversarial perturbations. Indeed, existing detectors are trained through class labels that separate samples in disjointed categories where each sample may be either malicious or benign but not both. On the other hand, the cyber domain is more fuzzy, and a sample may present characteristics belonging to different categories. Any rigid classification produced by *hard class labels* may represent an exploitable weakness of cyber detectors in adversarial settings. For this reason, we aim to introduce some degree of flexibility and uncertainty in the training process by using *probability labels* that allow the algorithm to capture additional information between classes such as similarity. The intuition is that a model that uses probability labels instead of hard class labels can be more resilient to adversarial samples, and can achieve comparable or superior results even in the absence of attacks. The main difficulty of a similar approach is that probability labels are not readily available in the cyber domain; hence we devise an original solution built upon the two following phases:

1. generation of probability labels from hard class labels;
2. deployment of a supervised model trained with the generated probability labels to perform the cyber detection.

Figure 5.2 shows that this approach considers as its input a *dataset* and its *class labels*. Then, it computes the corresponding *probability labels* (represented in the leftmost box), and uses them to train a *supervised model* that will be integrated in the detector. We apply this method to the random forest machine learning algorithm by leveraging the foundations [220] of the defensive distillation for neural networks [161]. By using the information encoded in the probability labels in the

form of probability vectors, generated after training an initial model, it is possible to develop a second "distilled" model that is more robust against adversarial attacks. The entire workflow applied to the random forest algorithm is illustrated in Figure 5.3 where each step is denoted by a circled number that is explained in the following subsections. Unlike the original defensive distillation technique, the generation of probability labels and their use for detection is performed through random forest-based models instead of neural networks.

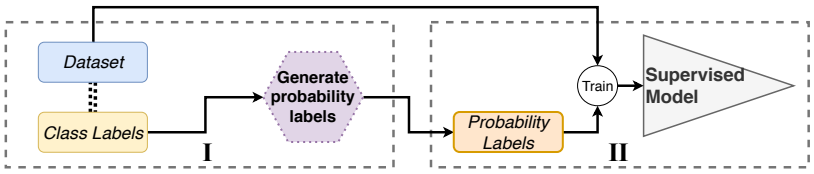



FIGURE 5.2: The two phases of the cyber detector.

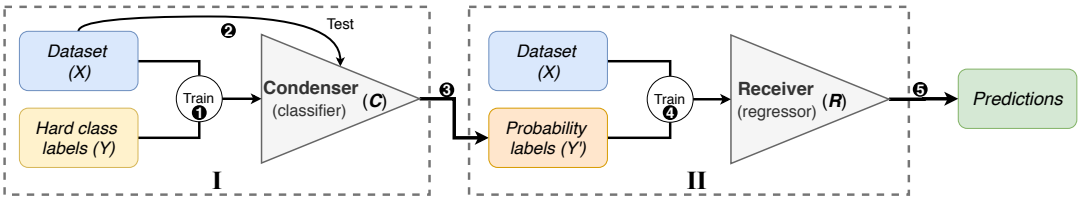



FIGURE 5.3: Workflow of the proposal: distillation is applied to the random forest algorithm.

**Generation of the probability labels** The initial phase is performed through a random forest classifier, the **Condenser**, denoted by $\mathcal{C}$. We first train this classifier (step ❶ in Figure 5.3). Then, we leverage the intrinsic property of the random forest algorithm of being an ensemble method, that is, a composition of several decision trees (or estimators), where the final output is generated after evaluating the response of each individual tree. This characteristic allows us to produce the desired probability vectors by considering the percentage of estimators that predicted a specific result (step ❷ in Figure 5.3). Formally, let $X$ be a dataset, $|X| \in \mathbb{N}$ the number of samples that constitute $X$, and $x_i \in X (0 \leq i \leq |X|)$ a sample within this dataset; let $Y$ be the set of hard class labels (in the form of indicator vectors) associated to dataset $X$, and $y_i \in Y$ the label associated to $x_i$. If $\mathcal{C}$ is a random forest classifier, then $|\mathcal{C}| \in \mathbb{N}$ is the number of estimators that compose $\mathcal{C}$, and $t_j \in \mathcal{C} (0 \leq j \leq |\mathcal{C}|)$ is a tree of classifier $\mathcal{C}$. After training $\mathcal{C}$ by means of $X$ (as training dataset) and of $Y$ (as labels), the set of probability labels $Y'$ that can be obtained from $X$ through $\mathcal{C}$ is:

$$Y' = \left\{ y'_i \mid y'_i = \frac{\sum_{j=1}^{|\mathcal{C}|} t_j^i}{|\mathcal{C}|} \right\}, \tag{5.1}$$

where $y'_i$ is the probability vector corresponding to sample $x_i$, and $t_j^i$ denotes the output of tree $t_j$ for sample $x_i$, which is an indicator vector. As an example, let us consider a random forest classifier consisting of 100 estimators that are trained to solve a binary classification problem (either 0 or 1). Now, let us assume that, for a given sample, 31 estimators predict 0 and produce the indicator vector $(1, 0)$, while the remaining 69 predict 1 and produce the indicator vector $(0, 1)$. In this case, although the final output of the classifier is the indicator vector $(0, 1)$, we generate the binary probability vector (0.31, 0.69) which encodes the output produced by each individual tree. On the other hand, if 69 estimators predict 0 and 31 estimators predict 1, we would obtain the probability vector (0.69, 0.31).

It should be noted that the objective of the Condenser is to generate accurate probability labels but it does not perform detection. As the focus is on the prediction of every individual estimator, and not on the classification results of the whole random forest classifier, the concept of "misclassification" does not strictly apply to this phase. For example, let us consider a binary classification scenario where we train the Condenser and then test it to generate the probability labels: it may be possible that, for a sample associated to the label 1, 69% of the estimators of the Condenser predict a 0. This event cannot be considered a misclassification because the output of the Condenser is a probability (e.g., the probability vector (0.69, 0.31)). However, such occurrences may have a detrimental effect in the next phase. To minimize similar risks, we utilize the entire available dataset to both train and test $\mathcal{C}$: this approach would yield the best results as it ensures that each sample is associated to a probability label with the highest degree of confidence.

**Model deployment** In the second phase, the probability vectors generated by the Condenser (step ❸ in Figure 5.3) are used as training labels for a random forest *regressor* that uses those probabilities as its training input (step ❹ in Figure 5.3). We define this model as the **Receiver** denoted by $\mathcal{R}$. Since this model performs the actual detection tasks (step ❺ in Figure 5.3), we evaluate it against the adversarial inputs. Hence, it is important that this model is trained by following the best practices (as in [155]) to avoid the risk of overfitting. For example, the training and validation sets should be chosen through appropriate splits of the available dataset.

We remark that the Receiver can be seen as a complex multi-output regressor with the challenging task of multi-target regression [221]. However, for the specific scenarios related to cyber detection, it is possible to devise a simpler regressor because the main goal is to analyze network traffic and to identify illegitimate activities. Hence, we can model the case as a binary classification instead of a multi-class problem, in which the algorithm is required to determine only whether a given sample of traffic is malicious or not. To this purpose, for each data sample, the Condenser needs to generate a single probability value (denoting the likelihood of being a malicious sample) instead of a multi-dimensional probability vector. By considering the binary classification example described earlier, the 31 estimators of the Condenser that predicted a 0 would give the value 0, while the remaining 69 estimators would produce the value 1. Thus, the corresponding probability value for the analyzed sample is 0.69. These probability values are then used as the labels for the Receiver, whose output is another probability value that can be converted into a discrete number through a rounding operation:

$$P(x_i) = \lfloor \mathcal{R}_{x_i} \rceil, \tag{5.2}$$

where $\mathcal{R}_{x_i}$ is the output of the Receiver $\mathcal{R}$ for the sample $x_i$, and $P(x_i) \in [0, 1]$ denotes the final prediction of the distilled model.

### 5.2.3 Experimental methodology

We now describe the details of the experimental evaluation performed in this work. We begin by outlining the considered application scenario, and then propose the implementation details of the considered detectors.

**Application scenario for the detector** A realistic scenario where the proposed detector can be applied successfully is the one presented in Section 4.2.2 (see Figure 4.1). Briefly, it consists in a large enterprise network with many internal hosts, whose network traffic is inspected by a NIDS based on machine learning that aims to identify botnet activities by leveraging the random forest algorithm. We assume that an attacker has already established a foothold in the internal network by compromising one or more machines and deploying botnet malware that communicate with a Command and Control infrastructure. The main goal of the attacker is to evade detection. He knows that network communications are monitored by a NIDS based

on machine learning. We assume that the attacker can issue commands to the bot through the CnC infrastructure, but he cannot interact with the detector. Although the attacker does not know the specific machine learning algorithm (alongside its parameters and features) used by the NIDS, he can easily guess that the detector is trained over a dataset containing malicious flows generated by the same or a similar malware variant deployed on the infected machines. The strategy to avoid detection is through a *targeted exploratory integrity attack* [145] that is performed by inserting tiny modifications in the communications between the bot and its CnC server. We stress that the considered use-case closely resembles a realistic scenario of modern attacks [74]: machines that do not present administrative privileges are more vulnerable to botnet malware, and skilled attackers can easily assume control of them. On the other hand, the NIDS is usually one of the most protected devices in an enterprise network, and can be accessed only through few selected secured hosts.

**Characteristics of the detector** We anticipate that the experimental evaluation is conducted on the `CTU-13` dataset, which is described in Section 4.2.3. Moreover, we generate the adversarial samples by following the same procedure described in Section 4.2.5.

The experimental campaigns considers the following detectors based on random forest:

- The **Undistilled** detector, which presents characteristics similar to the random forest classifier model proposed in [174], is used as the baseline for the experiments; a graphical representation of its architecture is provided in Figure 5.4.
- The **Distilled** detector represents the main proposal of this work. It consists of the *Condenser* for generating the probability labels, and of the *Receiver* to perform the detection tasks. This detector is evaluated against the Undistilled detector in adversarial and non-adversarial settings.

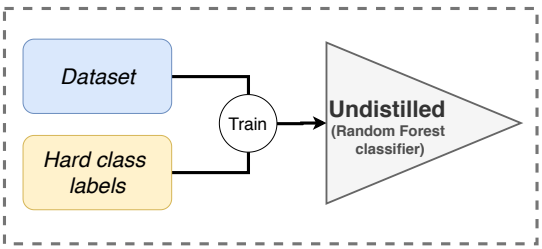



FIGURE 5.4: Architecture of the Undistilled detector.

Each detector has 6 instances, each one focusing on recognizing a specific malware family of the dataset. The development of the detectors follows the same procedure described in Section 4.2.4. The instances of the Receiver are trained with 80% of the botnet flows generated by each malware variant included in the `CTU-13` dataset, and validated on the remaining 20%. On the other hand, the instances of the Condenser, which generate the probability labels, are trained and tested on the same dataset containing all the malicious flows of the related botnet family. Other details are presented in Section 5.2.2. Moreover, we report in Table 5.6 the meaningful parameter settings of each model, which are chosen through extensive grid search operations. The $F$ parameter denotes the number of features in input, and MSE is the Mean Squared Error.

TABLE 5.6: Parameters of the random forest models.

| | Parameter name | Value |
|---|---|---|
| **Undistilled** | Number of estimators | 763 |
| | Quality Function | Gini |
| | Features for best split | $\sqrt{F}$ |
| | Bootstrap | Yes |
| **Condenser** | Number of estimators | 894 |
| | Quality Function | Gini |
| | Features for best split | $\sqrt{F}$ |
| | Bootstrap | Yes |
| **Receiver** | Number of estimators | 1352 |
| | Quality Function | MSE |
| | Features for best split | $F/2$ |
| | Bootstrap | Yes |

The performance of these detectors will be measured through the typical machine learning metrics: *Precision* (see Eq. 2.1), *Detection Rate* (see Eq. 2.2), *F1-score* (see Eq. 2.3).

### 5.2.4 Evaluation results

We present the results of a large set of experiments with the aim of demonstrating that (i) the proposed distilled random forest detector achieves comparable or better detection performance than state of the art algorithms in scenarios that are not subject to adversarial inputs; and that (ii) it significantly improves the robustness of machine learning models against adversarial attacks. Achieving both results is an important outcome for cybersecurity contexts where we cannot anticipate when a machine learning detector is being targeted by evasion attempts.

We evaluate and compare the performance of Distilled and Undistilled detectors

in scenarios where samples are not adversarially modified. Then, we assess the effectiveness of the distilled random forest model against adversarial perturbations. Finally, we compare the result of the proposed method against two existing defensive strategies that can be applied to any supervised machine learning algorithm.

**Evaluation in normal scenarios** We initially generate the probability labels for the Distilled detector by training and testing its Condenser model. Then, we train both the Distilled (through the Receiver) and Undistilled detectors on the same training set (but with appropriate labels), and proceed to evaluate them on the same test set. The results are shown in Table 5.7, where the columns report the chosen evaluation metrics, and the rows denote the botnet-specific instances of the Undistilled and Distilled detectors; the last row summarizes the results of each detector, which are averaged among all instances. From this table, we observe that the Distilled detector achieves the best results as it obtains higher Precision and F1-scores, and superior detection rates. We stress that the performance of the Distilled is similar to that obtained by state of the art random forest-based botnet detectors [174], [196]. Furthermore, we highlight that our proposal also outperforms the initial defensive distillation technique applied to neural networks in non-adversarial settings, because the distilled neural network model presents a reduced accuracy of ~1.5% when compared to a not-distilled neural network model [161]; this performance drop also affects distilled neural networks for malware classification scenarios [218], which exhibit an increased rate of false alarms. It is important to note that the unusual perfect *Precision* scores achieved by both models for the `Murlo` botnet and by the Undistilled model for the `Menti` botnet can be motivated as follows: the large majority of the network flows generated by these botnet variants are significantly different from benign traffic, hence the models are able to recognize their malicious samples without generating false positives; however, some instances are still able to evade detection as indicated by the imperfect Recall value. These experiments show that, in the absence of adversarial attacks, our version of the distillation technique applied to random forests yields a detector with similar or superior performance than those that do not adopt a distillation technique. These results are crucial because they refer to a large set of scenarios and demonstrate that random forest-based detectors integrated with distillation are effective even in the absence of adversarial inputs.

TABLE 5.7: Baseline vs. Distilled model performance.

| Botnet | Detector | *F1-Score* | *Precision* | *Recall* |
|---|---|---|---|---|
| Neris | Undistilled | 0.9577 | 0.9615 | 0.9540 |
| | Distilled | 0.9651 | 0.9671 | 0.9632 |
| Virut | Undistilled | 0.9682 | 0.9876 | 0.9496 |
| | Distilled | 0.9753 | 0.9876 | 0.9633 |
| Murlo | Undistilled | 0.9932 | 1 | 0.9866 |
| | Distilled | 0.9968 | 1 | 0.9937 |
| Rbot | Undistilled | 0.9994 | 0.9999 | 0.9999 |
| | Distilled | 0.9995 | 0.9999 | 0.9990 |
| Menti | Undistilled | 0.9984 | 1 | 0.9969 |
| | Distilled | 0.9979 | 0.9997 | 0.9969 |
| NSIS.ay | Undistilled | 0.9213 | 0.9925 | 0.8596 |
| | Distilled | 0.9273 | 0.9784 | 0.8812 |
| Average | Undistilled | 0.9729 | 0.9774 | 0.9684 |
| | Distilled | 0.9777 | 0.9804 | 0.9751 |

Since supervised machine learning methods for cyber defense need periodic retrainings (as explained in Section 2.2, it is important to evaluate the computational cost of the proposed solution. Thus, we measure and report the training times of the considered detectors in Table 5.8, which compares the time (in seconds) required for training the baseline Undistilled detector (composed of a single random forest classifier) with those required by our method; as the proposed Distilled detector includes both the Condenser and the Receiver, we report the combined training time of these components. Computations are performed on a machine with the following hardware: CPU Intel Core i7-7700HQ, RAM 32GB, and SSD 512GB. We observe that training the Distilled detector requires more effort, because it is composed of two models and, in addition, training a random forest regressor (that is, the Receiver) is more demanding than training a classifier. However, we stress that these operations needs to be executed only periodically. Moreover, by performing the training computations on machines with dedicated hardware it is possible to decrease the absolute training time difference to negligible amounts.

**Evaluation in adversarial settings** It must be determined whether and to which extent the proposed method is able to address issues related to adversarial attacks. To this purpose, we test the Distilled and the Undistilled detectors against the generated adversarial datasets, and compare their performance. The detection rate is the metric of interest for these analyses. We anticipate that this evaluation highlights a twofold improvement of our proposal: a significant increase in the detection rate; a more stable behavior against different adversarial samples of the same botnet family.

TABLE 5.8: Training time of each instance of the detectors.

| Botnet | Detector | Time (s) |
|---|---|---|
| Neris | Undistilled | 75.8 |
| | Distilled | 212.5 |
| Virut | Undistilled | 16.7 |
| | Distilled | 42.7 |
| Murlo | Undistilled | 19.8 |
| | Distilled | 53.9 |
| Rbot | Undistilled | 77.1 |
| | Distilled | 210.4 |
| Menti | Undistilled | 2.8 |
| | Distilled | 8.5 |
| NSIS.ay | Undistilled | 1.6 |
| | Distilled | 5.7 |
| Average | Undistilled | 32.3 |
| | Distilled | 87.0 |

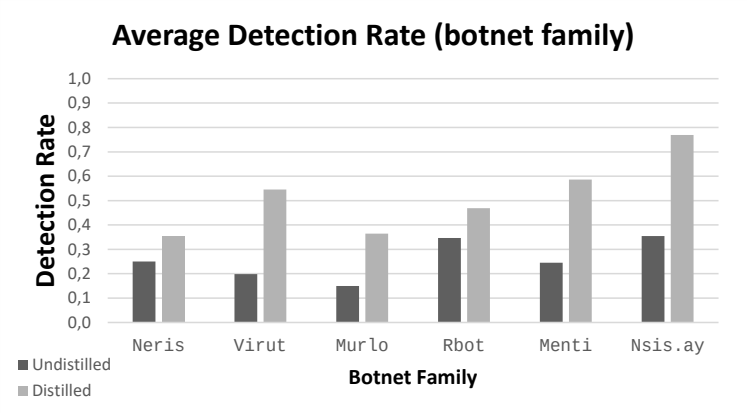



FIGURE 5.5: Comparison of the average detection rates on each malware family.

Among the considered 810 adversarial datasets[1], the Distilled detector clearly outperforms the baseline Undistilled in 759 cases; for the remaining 51 datasets, the results of the two detectors are close. A comprehensive overview of the effectiveness of the two detectors is presented in Figure 5.5, where the black and gray histograms report the detection rates of the Undistilled and Distilled detectors, respectively. Each histogram denotes the average performance of the models applied to each botnet family. There is no doubt that the Distilled is significantly superior to the Undistilled detector, with improvements ranging from 50% to 250%.

We provide a more detailed comparison of the two detectors by considering the impact on detection rates of different altered features. The results are reported in

[1]Given by 15(groups of altered features) $\times$ 9(increment steps) $\times$ 6(botnet families in the CTU-13)

Figure 5.6, where the x-axis denotes the group of altered features, and every histogram is generated by averaging the detection rates achieved by each instance of the detectors for all increment steps. From this figure, we can observe that the Distilled achieves superior detection rates for all the groups. The improvements for the groups `2a`, `2b` and `3a` are the most significant, as they allow the Distilled to retain a detection rate that is much higher than that of the Undistilled model. Moreover, the results for group `1a` show that the Distilled detector is almost unaffected by alterations of the flow duration. On the other hand, adversarial alterations involving multiple features have a high impact on the performance of both detectors, as these modifications cause the malicious test samples to be considerably different than those used to train each model. Nevertheless, it is appreciable that, even in these tough circumstances, the Distilled is able to correctly identify more than twice the amount of malicious flows with respect to the Undistilled detector.

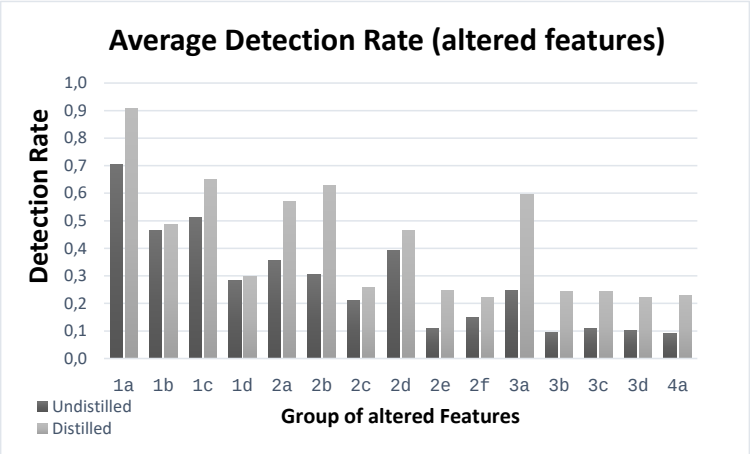



FIGURE 5.6: Comparison of the average detection rates for each group of altered features.

We also evaluate the detection rates of the two detectors for variable increment steps. The results are presented in Figure 5.7, where the x-axis represents the increment steps and the histograms are generated by averaging the performance over all groups of altered features.

We note that not only the Distilled outperforms the Undistilled model, but that it is much more resilient against samples that greatly differ from their original malicious version. Indeed, the detection rates for the `VIII` and `IX` steps are close to 50%; whereas the 15% detection rate of the Undistilled model is unacceptably low. This figure also shows that the Distilled presents a more stable behavior against adversarial samples that are obtained through different increment steps: its detection

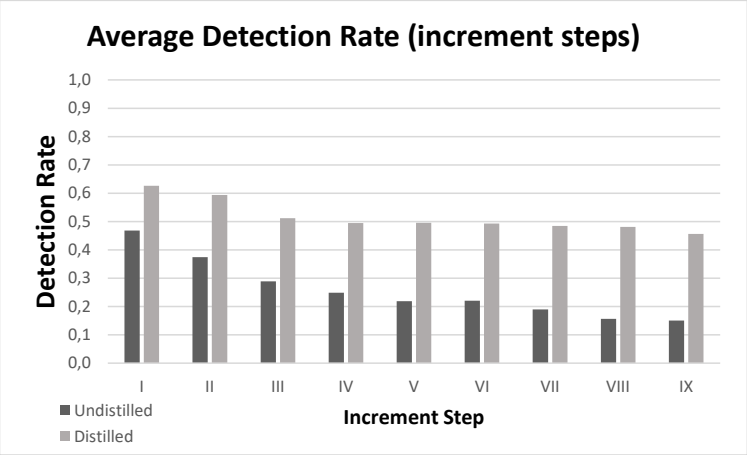
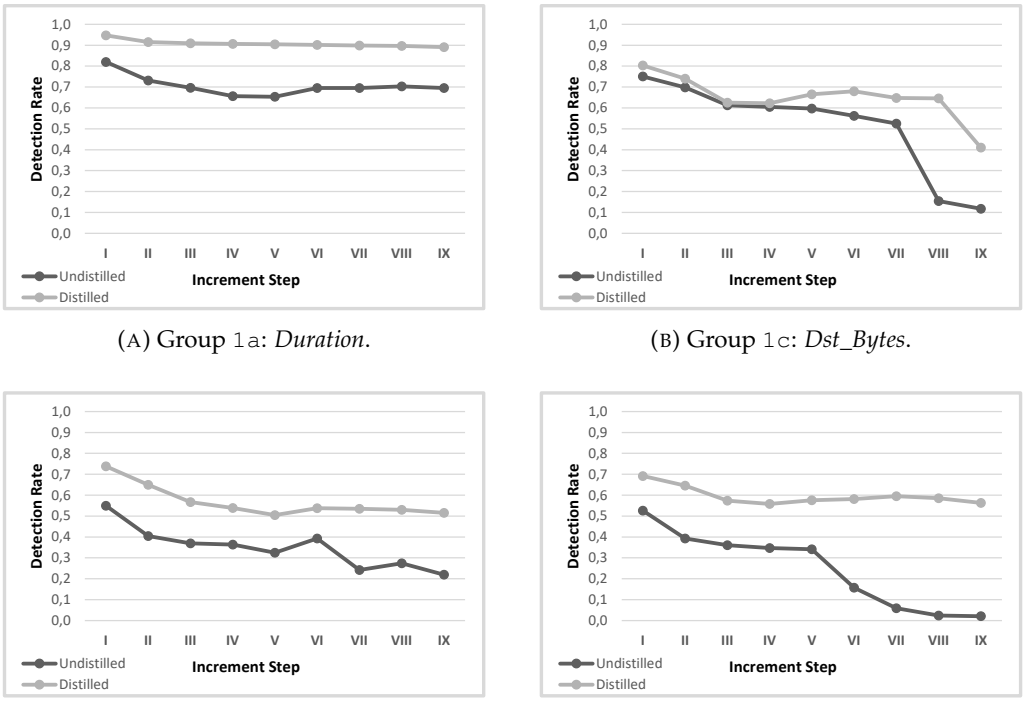



FIGURE 5.7: Comparison of the average detection rates for each increment step.

rates are between 46% and 61%, against the much broader 11% to 45% range of the Undistilled model. From Figure 5.7 and Figure 5.6, we observe that greater perturbations correspond to the lowest detection rates; however, we remark that such modifications may generate alerts from other defensive mechanisms (as explained in Section 5.2.3). Furthermore, we highlight that adversarial attacks are more effective and more difficult to detect when they are carried out through adversarial samples that are as close as possible to original samples.

We investigate the increased stability of our proposal through the fine grained comparisons in Figures 5.8, where the lines denote the detection rate (averaged for all botnet families) of the two models for four fixed groups of altered features (reported on top of each figure) and variable increment steps. The x-axis denotes the increment steps, and the y-axis the detection rate. The black and the gray line refers to the Undistilled and the Distilled model, respectively.

In order to appreciate the improved stability of the performance, we include in Figures 5.9 the boxplots related to the results of Figures 5.8. These boxplots highlight that the Distilled detector is not affected by sudden performance drops, thus indicating that it is able to maintain its performance even against adversarial inputs that are different from the scenarios considered in this work. The increased resilience of the Distilled detector is motivated by the fact that its Receiver model adopts a more robust set of feature importances when compared to the Undistilled model. In other words, a random forest model makes a prediction by comparing the features of a sample with the feature importances learned during its training phase: the probability labels used to train the Receiver produce a random forest model with a set

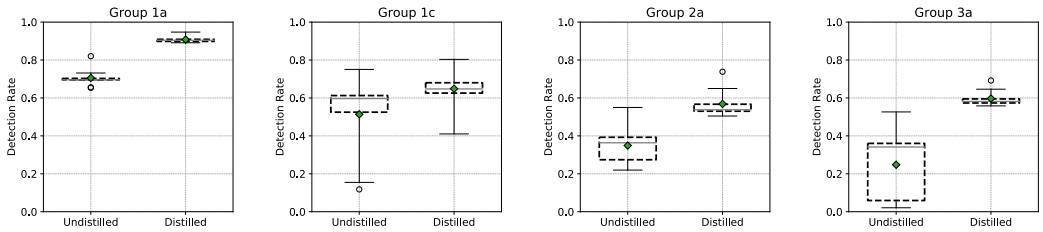



(A) Group 1a: *Duration*. (B) Group 1c: *Dst_Bytes*.

(C) Group 2a: *Duration & Src_Bytes*. (D) Group 3a: *Duration & Src_Bytes & Dst_Bytes*.

FIGURE 5.8: Comparison of the detection rates on the adversarial datasets generated by all malware families.

of feature importances having a higher degree of flexibility than that of the Undistilled classifier, which adopts hard class labels. As a consequence, an adversary can significantly alter the detection results of the Undistilled model through tiny alterations of the features, while the Distilled detector is capable of withstanding even perturbations of high magnitude.



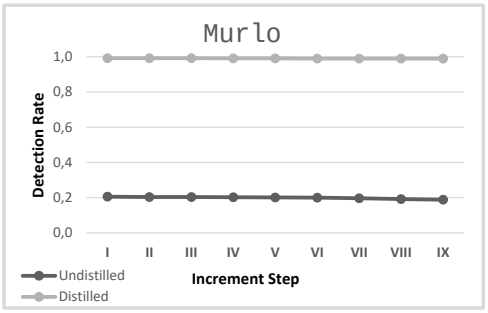

(A) Boxplot for Fig. 5.8a. (B) Boxplot for Fig. 5.8b. (C) Boxplot for Fig. 5.8c. (D) Boxplot for Fig. 5.8d.

FIGURE 5.9: Boxplot visualization of the results in Figures 5.8.

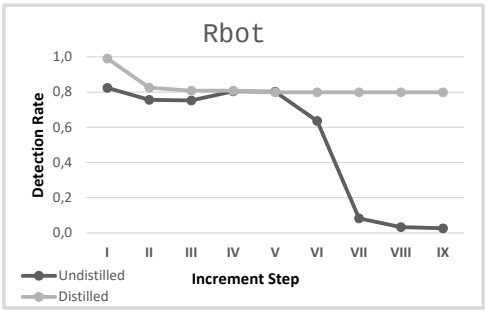
For example, let us consider two cases: in Figure 5.8b the adversary modifies only one feature (*Dst_Bytes*); in Figure 5.8d the adversary changes three features (*Duration*, *Src_Bytes* and *Dst_Bytes*). In the former case, the two detectors have comparable performance for the first increment steps because the manipulated feature (*Dst_Bytes*) has high and similar importance for both models. In the latter instance,

when alterations concern even incoming bytes and flow duration, the detection rates of the Undistilled model are unacceptably low (below 15%).

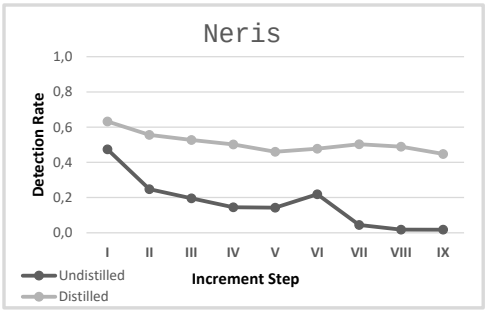
The improved resilience of our method is confirmed by comparing the detection rates of the two detectors for fixed botnet families. The results and corresponding boxplots are presented in Figure 5.10 and Figure 5.11, respectively. The name of the considered botnet family is reported on top of each figure.



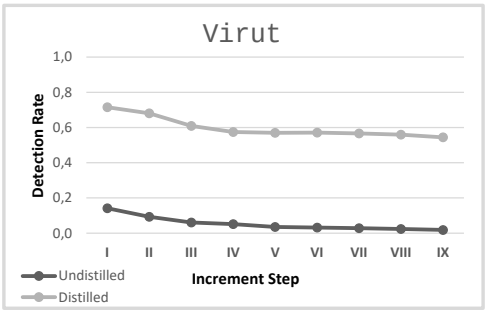

(A) Group 1a: *Duration*




(B) Group 2b: *Duration & Dst_Bytes*



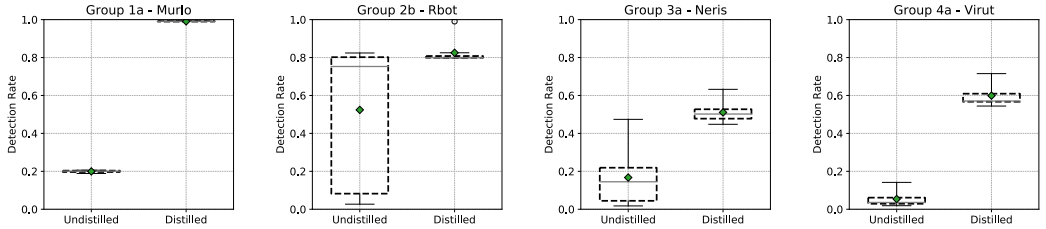

(C) Group 3a: *Duration & Src_Bytes & Dst_Bytes*




(D) Group 4a: *Duration & Src_Bytes & Dst_Bytes & Tot_Pkts*

FIGURE 5.10: Comparison of the detection rates on the adversarial samples generated by specific malware families.

Overall, these figures confirm the superior detection capabilities and improved stability of the Distilled model.




(A) Boxplot for Fig. 5.10a.(B) Boxplot for Fig. 5.10b.(C) Boxplot for Fig. 5.10c.(D) Boxplot for Fig. 5.10d.

FIGURE 5.11: Boxplot visualization of the results in Figures 5.10.

**Comparison with existing defensive strategies** We compare the effectiveness of our proposal against two known countermeasures against evasion adversarial attacks that have been proposed in the literature [39], [45], [183], and that can be applied to any supervised machine learning algorithm: *adversarial retraining* and *feature removal*. To this purpose, we perform our experiments by following the same procedures described in Section 4.2, due to the common characteristics shared by the considered adversarial scenarios and employed datasets. Hence, for the case of *adversarial retraining* we generate a "hardened" Undistilled detector by re-training it after introducing a small (10%) portion of the generated adversarial samples into the corresponding training sets, and then measure its detection rate on the same adversarial datasets used in our previous experiments for both the normal and adversarial scenarios. The results of this evaluation are presented in Table 5.9 which shows the (averaged) Recall obtained by the re-trained Undistilled detector, the proposed Distilled detector, and the baseline Undistilled detector that we include for completeness.

TABLE 5.9: Comparison with adversarial retraining.

| **Detector Type** | ***Recall*** (normal) | ***Recall*** (adversarial) |
|---|---|---|
| Undistilled (retrained) | 0.9695 | 0.4987 |
| Undistilled (baseline) | 0.9684 | 0.2573 |
| Distilled | 0.9751 | 0.5152 |

With regards to *feature removal*, we develop a different Undistilled detector by training it on the same dataset used in our previous experiments but without considering the features that we modified to generate our adversarial samples (that is, *Tot_Pkts*, *Duration*, *Dst_Bytes*, *Src_Bytes*), and then test it on the datasets used to determine the baseline performance in non adversarial settings; this is motivated by the fact that feature removal countermeasures, despite being resilient against adversarial attacks targeting the removed features, are known to generate excessive false alarms. The evaluation results are shown in Table 5.10, which compares the (average) Precision, Recall and F1-score of the Undistilled detector (after excluding the features) with those obtained by the Distilled and the baseline Undistilled detector.

By observing Table 5.9, we note that our proposal exhibits a higher detection rate in both scenarios. At the same time, concerning Table 5.10, we appreciate that the Distilled detector achieves significantly better results. Indeed, we highlight that the proposed distillation method is not affected by the issues that characterize similar

TABLE 5.10: Comparison with feature removal.

| Detector Type | *F1-Score* | *Precision* | *Recall* |
|---|---|---|---|
| Undistilled (feature removal) | 0.8728 | 0.8497 | 0.8974 |
| Undistilled (baseline) | 0.9729 | 0.9774 | 0.9684 |
| Distilled | 0.9777 | 0.9804 | 0.9751 |

countermeasures: *feature removal* strategies generate unacceptable rates of false positives, whereas *adversarial retraining* requires to constantly update the training set with all the possible variations of samples that can be modified by the attacker (as explained in Section 5.2.1).

By taking into account all these analyses and evaluations, we conclude that the proposed variation of the defensive distillation technique can be used to devise random forest detectors that: achieve same or better detection performance than existing algorithms in scenarios that are not subject to adversarial inputs; exhibit improved robustness and stability against adversarial attacks; are not affected by the limitations of existing countermeasures.

## 5.3 Countering Poisoning Attacks against Cyber Detectors

We now focus our attention to adversarial attacks performed at training-time, and propose an original methodology to counter this critical menace to cyber detectors. In particular, we focus on *poisoning* attacks due to their relevance in this domain [172], [222]–[224]. Our method leverages the *security-by-obscurity* principle discussed in Section 5.1.2, and is based on the idea of computing the actual training dataset only at the moment of training the machine learning model.

This approach provides three important benefits: (i) it can be adopted for a wide array of applications including, but not limited to, cybersecurity; (ii) it can be applied to harden any supervised machine learning algorithm; and (iii) it does not cause any changes in the performance of the detectors in non-adversarial scenarios. Moreover, our technique can be further improved by combining it with other existing defensive strategies for countering poisoning attacks (such as those in [210]–[212]). All these reasons make our novel method suitable for many real world scenarios, despite the known limitations of security-by-obscurity approaches.

As a practical implementation, we experimentally validate its effectiveness to strengthen flow-based botnet detectors using supervised algorithms. The results of

this experimental campaign show that the proposed approach can be successfully adopted to increase the resilience against poisoning attacks of different machine learning algorithms.

### 5.3.1 Proposed method

The proposed approach is based on the idea of generating the actual training set only at training-time. To this purpose, we introduce data transformation procedures on the training dataset. In this way, even if an adversary manages to poison the stored dataset by injecting malicious samples that are labelled as benign, the data transformation step ensures that the model is not trained on those exact poisoned samples. The expected result is that these samples will have a significantly smaller impact on the detector. The complete description of this solution is as follows.

We assume an organization that adopts a cyber detector relying on a supervised algorithm, which is periodically retrained. The training is based on a dataset $\chi'$ that is stored on a dedicated database server. Let $T$ be an invertible function with domain $K$ so that:

$$T^{-1}(T(k)) = T^{-1}(k') = k, \forall k \in K \tag{5.3}$$

The organization employs the transformation defined by $T$. More specifically, each time a new piece of data $\varkappa$ is added to the dataset $\chi'$ it is transformed as $T(\varkappa) = \varkappa'$. When it is necessary to retrain the detector, the dataset $\chi'$ is retrieved and is inversely transformed through $T^{-1}$, providing the original training dataset, $\chi$.

Now, let us assume that an attacker obtains full access to the database server containing $\chi'$. The attacker attempts a poisoning attack by introducing some samples $\overline{\varkappa}$ in $\chi'$ that are labelled as benign, and that represent malicious actions. (For example, the underlying code or network behaviour of a piece of malware, or a spam email). As the attacker is unaware of the data transformation, he does not try to infer the existence of a similar function by analysing the dataset and does not apply the data transformation $T$ to the $\overline{\varkappa}$ samples. When the detector is retrained, these samples $\overline{\varkappa}$ will undergo the transformation $T^{-1}$, resulting in samples $\overline{\varkappa}^{-1}$ with different characteristics than those of the malicious actions that the attacker wanted to evade detection. This results in poisoning samples whose effect on the detector will be different from that desired by the attacker. We report the entire workflow of the proposed approach in Figure 5.12 and Figure 5.13.

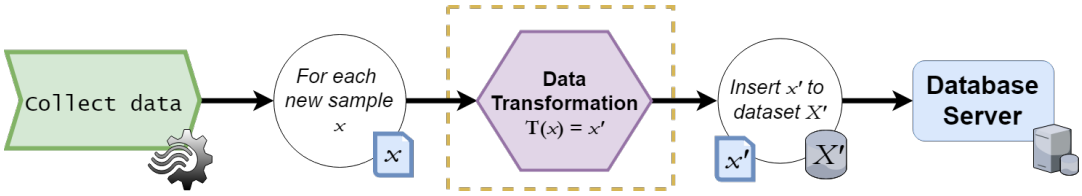



FIGURE 5.12: Workflow of the proposed poisoning countermeasure: operations performed before the (re)training.

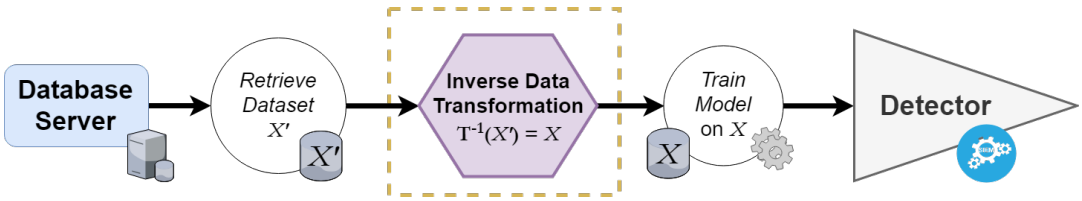



FIGURE 5.13: Workflow of the proposed poisoning countermeasure: operations performed at (re)training-time.

To facilitate the understanding of the proposed method, we present the following example. Consider an organization adopting a classifier $C$ that analyses network flows [46] to distinguish between malicious and benign traffic; let $\hat{\chi}$ be the dataset of network flows used to train the classifier, and let $\hat{T}$ be a transformation that modifies a flow sample by multiplying the *flow_duration* by $d \in \mathbb{R}$, and dividing the *flow_exchanged_bytes* by $b \in \mathbb{R}$; conversely, $\hat{T}^{-1}$ modifies a flow sample by dividing its *flow_duration* by $d$ and multiplying its *flow_exchanged_bytes* by $b$. With these assumptions, the dataset $\hat{\chi}$ is stored in the organization database as $\hat{\chi}'$. That is, every flow sample $\hat{x} \in \hat{\chi}$ is modified into $\hat{x}'$ by having the values of its *flow_duration* multiplied by $d$, and the values of its *flow_exchanged_bytes* divided by $b$. Therefore, every time the dataset $\hat{\chi}'$ is updated with a new set of flows, the flows are subject to the transformation denoted by $\hat{T}$. Consequently, whenever the classifier $C$ undergoes a retraining process, each flow $\hat{x}' \in \hat{\chi}'$ will be inversely transformed by $\hat{T}^{-1}$ into its original version, $\hat{x}$.

Now, if an unaware attacker attempts to poison the stored dataset $\hat{\chi}'$ by inserting some adversarial samples $\hat{\tilde{x}}$ that are wrongly labelled, he will not perform the transformation defined by $\hat{T}$, that is, the adversarial flows will not have their *flow_duration* and *flow_exchanged_bytes* modified. Hence, when the classifier, $C$ is retrained, the adversarial samples $\hat{\tilde{x}}$ will be transformed by $\hat{T}^{-1}$ into $\hat{\tilde{x}}^{-1}$. As a practical example, if $b = 10$ and $d = 2$, and if an attacker introduces in $\hat{\chi}'$ the adversarial sample $\hat{\tilde{x}}$ having *flow_duration* $= 2$ and *flow_exchanged_bytes* $= 240$, then $\hat{T}^{-1}$ will modify it into $\hat{\tilde{x}}^{-1}$ having *flow_duration* $= 1$ and *flow_exchanged_bytes* $= 2400$. Thus, this sample will

have different effects on the retraining process of classifier $C$ than the ones intended by the attacker.

We conclude this description by observing that our method does not involve any changes to the *actual* training dataset. Hence, the application of our countermeasure will not cause any modification to the baseline performance (that is, in non-adversarial settings) of the considered cyber detector. This point is important, because as it was explained in Section 5.1, one of the main limitations of existing techniques to counter adversarial attacks is a reduced performance on samples that have not been adversarially modified.

### 5.3.2 Experimental methodology

The experiments consider integrity attacks performed at training-time against network intrusion detection systems based on three supervised machine learning algorithms that achieve appreciable detection performance [4]: Random Forest (RF), Feedforward Deep Neural Network (FNN), K-Nearest Neighbour (KNN).

We assume a typical context, shown in Figure 5.14, where the network of a large enterprise is monitored by an NIDS based on a machine learning classifier that inspects the network flows of the border router [46]. The NIDS is periodically retrained with updated data stored on a dedicated database server.

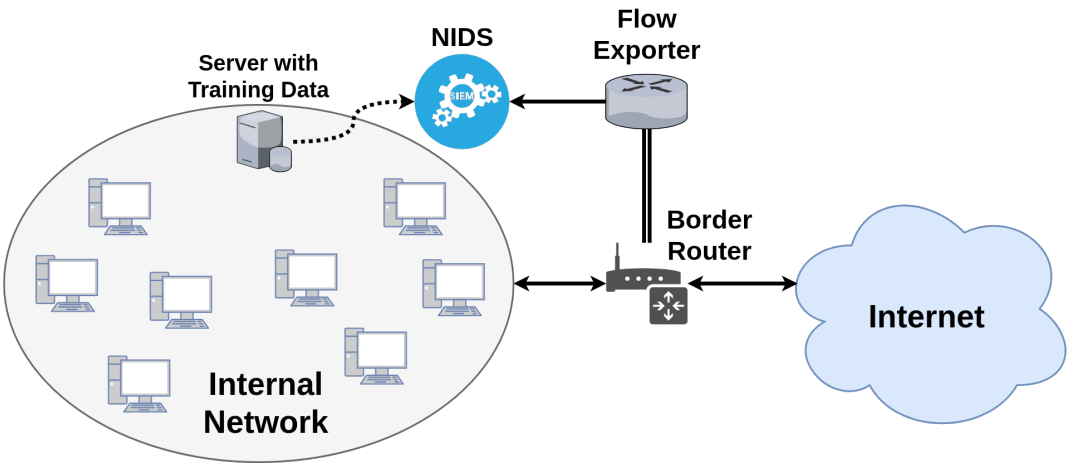



FIGURE 5.14: Scenario adopted for the experiments.

The attacker model considers an opponent who has compromised the targeted network and plans to infect other hosts with novel malware. He is aware that the network is monitored by a NIDS based on some supervised machine learning algorithms, and he also knows that this detector is periodically retrained. His goal is to

ensure that the deployed new malware variants evade detection mechanisms. The attacker has full access to the server that contains the training dataset, but he cannot interact with the detector. To reach his goal, the attacker plans to poison the training dataset through malicious samples (labeled as benign) representing the behaviour of the deployed malware variant.

The experimental campaign is based on the `CTU-13` dataset, described in 4.2.3, which is used to both develop the considered botnet detectors (based on three different ML algorithms); and to generate the required adversarial samples by following the procedure detailed in Section 4.2.5. Without loss of generality, we consider a smaller subset of increment steps (see Table 4.6) for altering the *duration* and *total_packets* features, of $+[1, 5]$ and $+[1, 10]$ respectively. As is common in our experiments, we split each detector into several instances, each devoted to one botnet family. Each instance is trained on a training set containing 80% of the malicious samples of the related botnet family, while the remaining 20% is used in the test-set. We use a fixed $85 : 15$ ratio of legitimate-to-illegitimate samples for each training- and test-set. The quality of each detector is measured through the traditional performance indicators *Precision* (see Eq. 2.1), *Recall* (or *Detection Rate*, *DR* – see Eq. 2.2), *F1-score* (see Eq. 2.3), whereas we measure the effectiveness of the considered adversarial attacks through the *Attack Severity* (*AS* – see Eq. 4.1) metric; we remark that attacks with higher (respectively, lower) magnitude will obtain *AS* scores that are closer to 1 (respectively, 0).

### 5.3.3 Evaluation results

We organize the experimental evaluation as follows. We initially assess the baseline performance of the considered detectors, because it is important to show that the considered attacks (and proposed countermeasure) are effective on detectors that obtain appreciable performance in non-adversarial settings. Then, we measure the effectiveness of the implemented poisoning attacks on these detectors. Finally, we analyze the quality of our solution.

The values presented in Table 5.11 represent the average of the performance metrics obtained by each detector. We observe that these detectors achieve results that are comparable to the state of the art [4], [174].

Next, we analyse the effects of the considered poisoning attacks that implement

TABLE 5.11: Baseline performance of the classifiers.

| Algorithm | *Precision* | *Recall* | *F1-score* |
|---|---|---|---|
| **RF** | 0.9774 | 0.9684 | 0.9729 |
| **FNN** | 0.9616 | 0.9438 | 0.9526 |
| **KNN** | 0.9558 | 0.9375 | 0.9466 |

integrity violations. To this purpose, we inject some of the crafted adversarial samples into each training dataset. Then, we measure the effectiveness of similar poisoning attempts by comparing the performance of the detectors on the poisoned samples before and after the poisoned retraining phase. The results, shown in Table 5.12, highlight that, before the poisoning attempt, the classifiers were able to identify the novel attack samples with detection rates comparable to other proposals against zero-day malware [151]. The performance of the same algorithms suffered a significant drop after a retraining phase with the poisoned data. The high $AS$ score gives a clear idea of the severity of the effect.

TABLE 5.12: Effects of the poisoning attack on each cyber detector.

| Algorithm | *Recall* (before the attack) | *Recall* (after the attack) | *Attack Severity* |
|---|---|---|---|
| **RF** | 0.8834 | 0.2636 | **0.7016** |
| **FNN** | 0.8674 | 0.2777 | **0.6798** |
| **KNN** | 0.8611 | 0.2391 | **0.7223** |

Finally, we evaluate the proposed original methodology to counter poisoning integrity attacks. Hence, we introduce a custom data transformation procedure on the training set, and then replicate the poisoning attack. For the sake of clarity, we consider a simple function $\hat{T}$ that multiplies the *flow duration* by $d \in \mathbb{R}$, and divides the *flow exchanged_bytes* by $b \in \mathbb{R}$. In this way, the poisoned samples introduced by the attacker are (inversely) transformed into samples that are different from the flows generated by the malware variant, because they have durations of $+[\frac{1}{d}, \frac{5}{d}]$ seconds (instead of $+[1,5]$) in which the hosts exchange $+[1 * b, 1024 * b]$ bytes (instead of $+[1, 1024]$). In Table 5.13, we compare the attack severity of the poisoning attempt before and after the application of the countermeasure, from which we can deduce that the proposed approach can significantly mitigate the effects of a poisoning attack. Furthermore, we stress that the proposed mechanism does not cause any alteration to the detection performance in non-adversarial settings, as explained in Section 5.3.1.

TABLE 5.13: Evaluation of the proposed defensive method. These results are obtained by setting $d = 2$ and $m = 5$.

| Algorithm | *Attack Severity* | *Attack Severity* (after Retraining) |
|---|---|---|
| **RF** | 0.7016 | **0.1587** |
| **FNN** | 0.6798 | **0.1741** |
| **KNN** | 0.7223 | **0.2830** |

## 5.4 Countering Evasion Attacks against Phishing Detectors

Machine learning algorithms are being increasingly used in a wide array of cybersecurity applications, including malware detection [225], intrusion detection [4], insider threat detection [226], spam detection [227], as well as the detection of malicious or phishing websites [228].

Phishing attacks are one of the most popular types of offences. For instance, in its 2019 "State of the Phish" report, ProofPoint[2] states that over 1.5 million phishing URLs are created *every month* and that 76% of businesses reported being a victim of a phishing attempt in 2017. Simply put, phishing represents one of the easiest ways for malicious hackers to penetrate an enterprise. In order to address this problem, there has been considerable work on the detection of phishing websites [43], [228]–[237]. In addition to blacklists maintained by corporations such as Google, there are also publicly available resources from sites such as `PhishTank`[3]. However, these blacklists end up becoming out of date frequently as malicious hackers move their phishing URLs from site to site in order to evade detection. Rule-based systems were therefore developed by several researchers. For instance, [238] develops a set of 8 rules to capture phishing webpages, as well as approaches that analyze the content of a URL [239]. Other researchers examined the use of a discriminative set of features associated with phishing URLs and then checked to see whether a given URL was similar to a known phishing URL based on associated feature vectors [240]. [241], [242] pioneered the idea of using visual similarity between a legitimate webpage (e.g. a bank website) and another website in order to check if the latter might be related to phishing. Over the past decade or so, the use of machine learning for phishing detection has now become commonplace. Early efforts in this direction include [243]–[249]. Despite even more recent research efforts proposing increasingly sophisticated machine learning solutions to counter this threat [43], [228]–[236], there is no

[2] `https://www.proofpoint.com/us/resources/threat-reports/state-of-phish`
[3] `https://www.phishtank.org/`

doubt that phishing websites still represent a dangerous menace [11].

One reason for this is that machine learning classifiers are trained on datasets from which they learn a model that separates benign samples from malicious ones. However, adversaries (i.e. the malicious hackers) are continuously adapting to phishing detectors (PDs for short) and often times, thwarting the defensive mechanism is very simple, allowing their phishing URLs to evade existing PDs with relative ease. Most work on adversarial machine learning in cybersecurity contexts deal with two extremes: white box attacks, in which the adversary has full knowledge of the defenses used (e.g. classifier used, list of features used); or black box attacks, in which the adversary has no knowledge. However, real world attackers are unlikely to represent one of these extremes: their knowledge is somewhere in the "gray area" between full knowledge (white box) and no knowledge (black box) of the defenses. We use the term *gray box* to refer to attacks where the attacker's knowledge can lie between these two extremes.

Our goal in this section is to propose a more robust phishing website detector that is capable of withstanding a large class of gray box attacks. Our notion of gray box is powerful enough to capture both white box and black box attacks as a special case. In particular, we make the following contributions.

- We define the notion of an *operation chain* or OC for short. OCs transform the training data into a new feature space through the iterative application of certain simple operators. Even if the adversary knew the original feature set, it is unlikely that he guesses the new features or feature space. We propose the *Phishing Operation Chain* (or POC) algorithm.

- *Gray Box Attack Scenarios.* We consider and devise the following 4 attack types:

  - GBA-1 attacks are ones where the attacker knows that the PD uses information about features in the source HTML of a phishing URL webpage.

  - GBA-2 attacks are ones where the attacker knows that the PD uses features related to the actual URL string itself.

  - GBA-3 attacks are hybrids of the previous two attacks.

  - GBA-4 attacks are ones where the attacker has knowledge of some percentage $\Delta$ of the features used by the defender. When $\Delta = 100\%$, we have a white box attack, while when $\Delta = 0\%$, we have a black box attack.

- We show that phishing detectors based on **13** recent works [43], [166], [228], [230]–[233], [235], [236], [250]–[253] during the 2014 to 2019 time period are susceptible to these attacks. Specifically, we formally define the *Impact* of an attack on a phishing detector and show that all 13 well known classifiers (both shallow- and deep-learning based) experience a significant impact under these three kinds of attacks. The impact of these gray box attacks is shown on 3 well-known datasets (DeltaPhish [254], Mendeley [255] and UCI [256]) as well as a new (fourth) DSAIL dataset that we produced during this work. This is important, because most past work only considers one type of attack, against one or few classifiers, and adopts only one dataset.

- We show that POC is significantly more robust to these attacks than the 13 existing PDs that we compare against on the same 4 datasets mentioned above. It is important to note that these past works use a wide range of classification techniques and features — so POC is robust when used on top of many different classification algorithms and feature sets. We verify this claim by computing the statistical significance through both a Student's t-test and the Mann-Whitney U-test, obtaining p-values of less than 0.00001 (which denotes a 99.999% chance that POC outperforms past work).

In summary, we develop an original countermeasure to evasion attacks through the POC algorithm. Then, we propose and evaluate 4 gray box attacks on existing phishing detectors. We formally define the *Impact* of an attack on a dataset and classifier, and show that these attacks are highly effective against existing PDs. Finally, we show that POC is much more robust to the considered attacks than past classifiers. Our claims are based on experiments involving 13 classifiers on 4 datasets, including new classifiers such as Google's *Deep & Wide* that have not been used previously for phishing detectors to the best of our knowledge. For these reasons, we are confident that this work represents a valid and effective method to counter the effectiveness of adversarial evasion attacks against machine learning-based phishing detectors.

### 5.4.1 Related work

We divide related work into 3 parts: (i) work on detection of phishing websites; (ii) general work on vulnerability of ML algorithms to adversarial perturbations and

existing countermeasures; (iii) related work on adversarial attacks against phishing detectors (PDs for short).

**Detection of Phishing Websites** Though rule based methods (e.g. [229]) formed the initial corpus of work on phishing URL detection, machine learning (ML) approaches have been common of late. Some proposals [230], [232] identify phishing domains by analysing hundreds of features extracted from the corresponding URLs. Other studies [43], [232], [233], [250], [253] devise classifiers that use a reduced number of URL-based features. Some papers combine URL- with HTML-based features to improve performance [228], [231], [234], [236], [251]. The authors of [252] and [235] also consider information provided by external reputation sources (such as DNS records). More recently, [166] leverage image processing with HTML inspection to detect phishing content in compromised websites. The research in [237] develop methods to predict how Twitter is used to lure victims to phishing URL sites. Despite all these efforts, phishing websites still represent a widespread menace [11].

**Adversarial Machine Learning** We have already explained how small perturbations to the input can lead to huge errors by ML-based classifiers. These findings led to the development of recent important work on adversarial ML also for cybersecurity [180], [257]–[260]. However, these research efforts make very strong assumptions about the adversary's knowledge on the defensive system. White box models assume the attacker has complete knowledge of the defense including the algorithm used and all the features used [172], [218], [261], [262]. Conversely, other researchers assume a black box model [263]–[267] in which the adversary knows absolutely nothing about the defenses used by the target. Both of these are extreme cases — in the real-world, defenders might use custom classifiers (e.g. ensembles) with novel features and feature selection and late fusion [268] or custom combinations of supervised and unsupervised learning [32] which would be almost impossible for an adversary to guess correctly. Some recent work considers more realistic scenarios [45], [150], [185], but assume the classifier used by the PD is known which is problematic [39], [269]. Moreover, [161], [218] propose methods to harden detectors based on Neural Networks, while [206] proposes an approach to improve the robustness of tree-based mechanisms [200], [270], [271] focus on Support Vector Machine-based techniques.

Our POC algorithm improves upon past work in the following ways: (i) we are the first to propose mapping feature vectors into a new feature space for purposes of increasing robustness to adversarial attacks against phishing detectors[4]; (ii) POC is experimentally shown to be robust against attacks on 13 different classifiers as opposed to just one; (iii) POC is robust to a family of gray box attacks; and (iv) we test and validate the performance of our algorithms on 4 different datasets — not just 1 as most past research.

**Adversarial Attacks Against Phishing Detectors** Most work on adversarial ML in Cybersecurity has focused on malware detection [225], spam detection [158], [227], [270], [271] and network intrusions [4], [185]. An important recent work by [167] reverse engineers and subverts the phishing detector used by the Google Chrome web browser. Another recent important paper [166] devises a phishing detector by combining the analysis of the webpage HTML and its image data in a white box setting where the attacker has complete control of the entire URL domain. Both these works, though very important, make strong assumptions. First, [167] attacks just *one* phishing detector (albeit a very important one). Second, [166] makes a white box assumption and only considers one classifier (a linear SVM). In contrast, our POC method can harden multiple base classifiers used by a defender and can protect against multiple types of attack models, not just white box attacks. Furthermore, POC is robust to attacks against 13 different classifiers as opposed to just one classifier as was considered in most previous works. Finally, as mentioned earlier, POC's performance has been tested on 4 datasets, not just one, with different features.

### 5.4.2 Proposed method

We introduce the concept of *operation chains* or OCs and the POC algorithm. The basic idea behind POC is to randomly select a given number of features from the total feature-set. This makes it hard for the attacker to determine exactly what features are used in the defense. The second idea is to use OCs which consist of operators that transform the randomly selected features into a new set of features. We assume the features are numeric (real valued) and our framework assumes (without loss of generality) that categorical feature values will be replaced by values from a discrete domain.

[4]Note that mapping feature spaces into new feature spaces is not new in other domains — for example, kernel tricks used in Support Vector Machines — use a similar trick.

A *unary (resp. binary) feature mapping* operator $\alpha$ (resp. $\beta$) is a mapping from $\mathbb{R}$ (resp. $\mathbb{R} \times \mathbb{R}$) to $\mathbb{R}$. We assume the existence of a set Fmop of unary and binary feature mapping operators. We use Fmop $= \{\log,\sin,\cos,\tan, exp^i, +, -, *, /\}$ as our feature mapping operators in our implementation, where $\{\log, \sin, \cos, \tan\}$ are unary operators, $exp^i$ is not one but a family of numeric unary operators which take an input value $x$ and return $i^x$ for $i \in \{-3, -2, \ldots, 2, 3\}$. The arithmetic operators $+, -, *, /$ are binary feature mapping operators[5]. Note that our definitions below apply to virtually any choice of unary and binary operators in Fmop — we are not limited in any way to the specific operators chosen in our implementation — new ones and the definition of operation chains below can be seamlessly incorporated into our framework. Given a set $\mathbb{F}$ of (original) features, we can recursively define *operation chains* based on a (randomly chosen) subset $\mathbb{F}' \subseteq \mathbb{F}$ as follows:

1. Every feature $f \in \mathbb{F}'$ is an operator chain of size 0.
2. For each unary operator $\alpha \in$ Fmop and for each operator chain $oc$ of size $s$, $\alpha(oc)$ is an operator chain of size $s + 1$.
3. For each binary operator $\beta \in$ Fmop and for each pair of operator chains $oc_1, oc_2$ of sizes $s_1, s_2$ respectively, $\beta(oc_1, oc_2)$ is an operator chain of size $s_1 + s_2 + 1$.

Suppose $\mathbb{F}$ is a list of features and suppose $\mathbb{F}'$ consists of any two of these feature, i.e. let $\mathbb{F}' = \{f_1, f_2\}$. Then examples of operation chains based on $\mathbb{F}'$ include:

1. $f_1$ and $f_2$ are both operation chains of size 0.
2. $\sin(f_1), \cos(f_2)$ are OCs that create new features by taking the sine and cosine, respectively of values of features $f_1, f_2$ respectively. They each have size 1.
3. $\sin(f_1) + \cos(f_2)$ is an OC that generates a new feature that creates feature values by summing up the sine of the value of feature $f_1$ and the cosine of the value of feature $f_2$. The size of this OC is 3.
4. $exp^i(\sin(f_1) + \cos(f_2))$ creates a new feature whose value is $i^{\sin(f_1)+\cos(f_2)}$. The size of this OC is 4.

Thus, in our POC framework, we develop operation chains as follows.

[5]The theory allows $i$ to range over any set of integers.

1. (Random Selection) First, we randomly choose a subset $\mathbb{F}'$ of $\mathbb{F}$. These are the features that we will use – directly or indirectly – in our mapped feature space.

2. (MaxSize Selection) We choose *MaxSize* which is an integer greater than 0.

3. (Operation Chain Transformations) We replace the features in $\mathbb{F}$ with the set of all new features generated by operation chains of size *MaxSize* or less that are based on $\mathbb{F}'$.

4. (Random Selection) We select a random subset of a given cardinality from the set of all operation chains thus generated.

The POC (Protective Operation Chain) algorithm that formally captures the informal process described above is shown in Algorithm 3.

### 5.4.3 Experimental methodology

We now described the adopted dataset and the considered evasion attacks that will be launched against our phishing detectors.

**The DSAIL dataset** The POC method proposed in this work is tested and validated on 4 datasets. Three are well-known datasets in the literature: `Mendeley` [255], `DeltaPhish` [254], `UCI` [256]. In addition, we create a new dataset called DSAIL [6]. With over 23 000 entries, DSAIL is one of the biggest labeled datasets for phishing detection. The only bigger labeled dataset we are aware of for PD is the `Ebbu` dataset[7] — but as shown in Table 5.14, this dataset does not contain many features included in our DSAIL dataset.

Table 5.15 describes the features used in the DSAIL dataset, which includes HTML, URL and Reputation-based features which have been also employed by prior work [43], [231]–[233], [250].

**Proposed gray box attacks on phishing detectors** We now describe four families of gray box attacks on phishing detectors.

1. GBA-1: The attacker assumes that the PD uses features related to the HTML-code [228], but he may not know exactly which features. Hence, he tries to

[6] DSAIL: short for "Dartmouth Security (and) AI Lab", where the dataset was created
[7] `https://github.com/ebubekirbbr/pdd`

**Algorithm 3:** Algorithm for generating a feature mapping.

**Input:** List of features $\mathbb{F}'$ included in a given dataset $d$; $\phi$, number of new features that will compose the new set of *mapped_features*; *MaxSize* maximum size of an operation chain; $\mathsf{Fmop}_\alpha$ set of unary feature mapping operators; $\mathsf{Fmop}_\beta$ set of binary feature mapping operators

**Output:** The *mapped_features* that generate the new feature space $\Phi$.

```
mapped_features ← emptyList();
for h ← 0 to φ by 1 do
    new_feature ← computeNewFeature(F', MaxSize);
    Insert new_feature in mapped_features;
return mapped_features
// Procedure that generates a mapped_feature
Function computeNewFeature(F', MaxSize)
    feature_block ← chooseFeatures(F', MaxSize);
    L ← len(feature_block);
    for h ← 0 to L by 1 do
        f' ← unaryOperation(feature_block[h]);
        Replace feature_block[h] with f';
    for h ← 1 to L by 1 do
        f' ← binaryOperation(feature_block[0],feature_block[h]);
        Replace feature_block[0] with f';
    new_feature ← feature_block[0];
    return new_feature
// Procedure that determines how many and which features from F' are considered to
generate a mapped_feature
Function chooseFeatures(F', MaxSize)
    feature_block ←emptyList();
    for h ← 0 to randomChoice(MaxSize) by 1 do
        Insert randomChoice(F') in feature_block;
    return feature_block
// Procedure that chooses and applies an unary operation among Fmop_α on a given
feature f that composes a given new_feature.
Function unaryOperation(f)
    f' ← Apply randomChoice(Fmop_α) to f;
    return f'
// Procedure that chooses and applies a binary operation among Fmop_β on a given pair
of features (f_1, f_2) that composes a given new_feature.
Function binaryOperation(f_1, f_2)
    f' ← Apply randomChoice(Fmop_β) to f_1 and f_2;
    return f'
```

TABLE 5.14: Comparison of existing static datasets for PDs.

| Name | Released | Phishing samples | Legit samples | URL data | HTML data | Reputation data | Screenshot data | Features |
|---|---|---|---|---|---|---|---|---|
| PhishLoad | 2012 | 3 510 | 8 190 | ✓ | ✓ | ✗ | ✗ | ✗ |
| UCI | 2015 | 6 050 | 3 950 | ✗ | ✗ | ✗ | ✗ | ✓ |
| DeltaPhish | 2017 | 1 200 | 4 800 | ✓ | ✓ | ✗ | ✓ | ✗ |
| Ebbu | 2017 | 37 175 | 36 400 | ✓ | ✗ | ✗ | ✗ | ✗ |
| Mendeley | 2018 | 5 000 | 5 000 | ✗ | ✗ | ✗ | ✗ | ✓ |
| DSAIL | 2019 | 7 861 | 15 773 | ✓ | ✓ | ✓ | ✓ | ✓ |

TABLE 5.15: List of features included the DSAIL dataset.

| *URL-features* | *REP-features* | *HTML-features* |
|---|---|---|
| IP address | SSL final state | SFH |
| '@' (at) symbol | URL/DNS mismatch | Anchors |
| '-' (dash) symbol | DNS Record | Favicon |
| Dots number | Domain Age | iFrame |
| Fake HTTPS | PageRank | MailForm |
| URL Length | PortStatus | Pop-Up |
| Redirect | Redirections | RightClick |
| Shortener | | Objects |
| dataURI | | StatusBar |
| | | Meta-Scripts |
| | | CSS |

alter some aspects of the underlying HTML code. For example, the attacker might assume that some PDs consider the ratio of internal-to-external links. Hence, in this attack, the adversary adds some fictitious "internal" links to his webpage that might thwart classifiers that use this feature. We inserted such internal links to a host of resources such as images, favicons, CSS snippets, videos, audio, as well as the usual "textual" links. Figure 5.15 shows how such an attack might be accomplished. The original HTML code is shown on the left. The modified HTML source is shown on the right with one inserted "fake" internal link shown inside the red box.

2. GBA-2: Here, the attacker guesses that the PD uses information about the length of the URL to predict whether it belongs to a phishing website or not [232], [233], [251]. Hence it is reasonable to assume that an attacker may try to circumvent such mechanisms. Often times, phishing URLs are longer than benign URLs in order to confuse and trick users into clicking on the link. For this reason, existing PDs are usually trained on malicious samples characterized by longer URLs. Thus, an attacker may try to evade detection by adopting shorter URLs: a possible way to accomplish this is by using a URL shortening service such as `bit.ly` or `tinyurl.com`.

3. GBA-3: This attack is a combination of one or more instances of the GBA-1 and GBA-2 attacks.

4. GBA-4: The most important attack we propose is the GBA-4 attack. In this attack, the adversary has *variable knowledge* of the features used by the defender. Suppose $F_d$ is the set of features used by the defender and $F_a$ is the set of features that the attacker thinks the defender is using. In the GBA-4 attack, we

vary $\Delta$, the percentage of features actually used by the defender that the attacker guessed correctly, i.e.

$$\Delta = \frac{|F_d \cap F_a|}{|F_d|}.$$

In the GBA-4 attack, we vary $\Delta$ and randomly select features for $F$. Note that when $\Delta = 0$, we have a black box attack, and when $\Delta = 1$, we have a white box attack.

In this work, we consider attacks on the following 13 well-known classifiers. 11 are based on shallow algorithms: Random Forest (RF), K-Nearest Neighbor (KNN), Decision Tree (DT), Logistic Regression (LR), Naive Bayes (NB), Support Vector Machines (SVM), Extra Trees (ET), Stochastic Gradient Descent (SGD), AdaBoost (AB), Gradient Boost (GB), Bagging (Bag); while the remaining 2 are based on deep learning techniques: Feedforward Deep Neural Network (FNN), and Google's recent "Deep and Wide" (DnW) method [272].

Thus, we note that our gray box attacks assume that the PD is using any one of these 13 classifiers.

We define the *Impact* of an attack $Att_i^d$ of type $i$ on dataset $d$ and classifier $Clf_j$ on a performance metric $\mu$ as:

$$Impact(Att_i^d, Clf_j, \mu) = \frac{\mu(Clf_j|\neg Att_i^d) - \mu(Clf_j|Att_i^d)}{\mu(Clf_j|\neg Att_i^d)}$$

where $\mu(Clf_j|Cond)$ denotes the value of the performance metric $\mu$ of classifier $Clf_j$ when condition $Cond$ is true; in the remainder, we will typically consider two conditions, namely if an attack is present or absent. In the above formulation, $\mu$ can be any measure of classifier performance (e.g. AUC, F1-score, Accuracy, FPR, etc).

For instance, suppose the AUC of a given classifier (say Random Forest) is 80% when no attack is performed, and 60% when an attack is launched on it. In this case, $Impact(Att_i^d, RF, AUC) = \frac{0.8-0.6}{0.8} = 0.25$, i.e. there is a 25% reduction in performance as measured using AUC.

Even though GBA-1 and GBA-2 are relatively simple attacks, there is considerable evidence that defenders use the HTML content in phishing websites and the structure of the URLs of phishing websites to build PDs [166], [252]. It is therefore

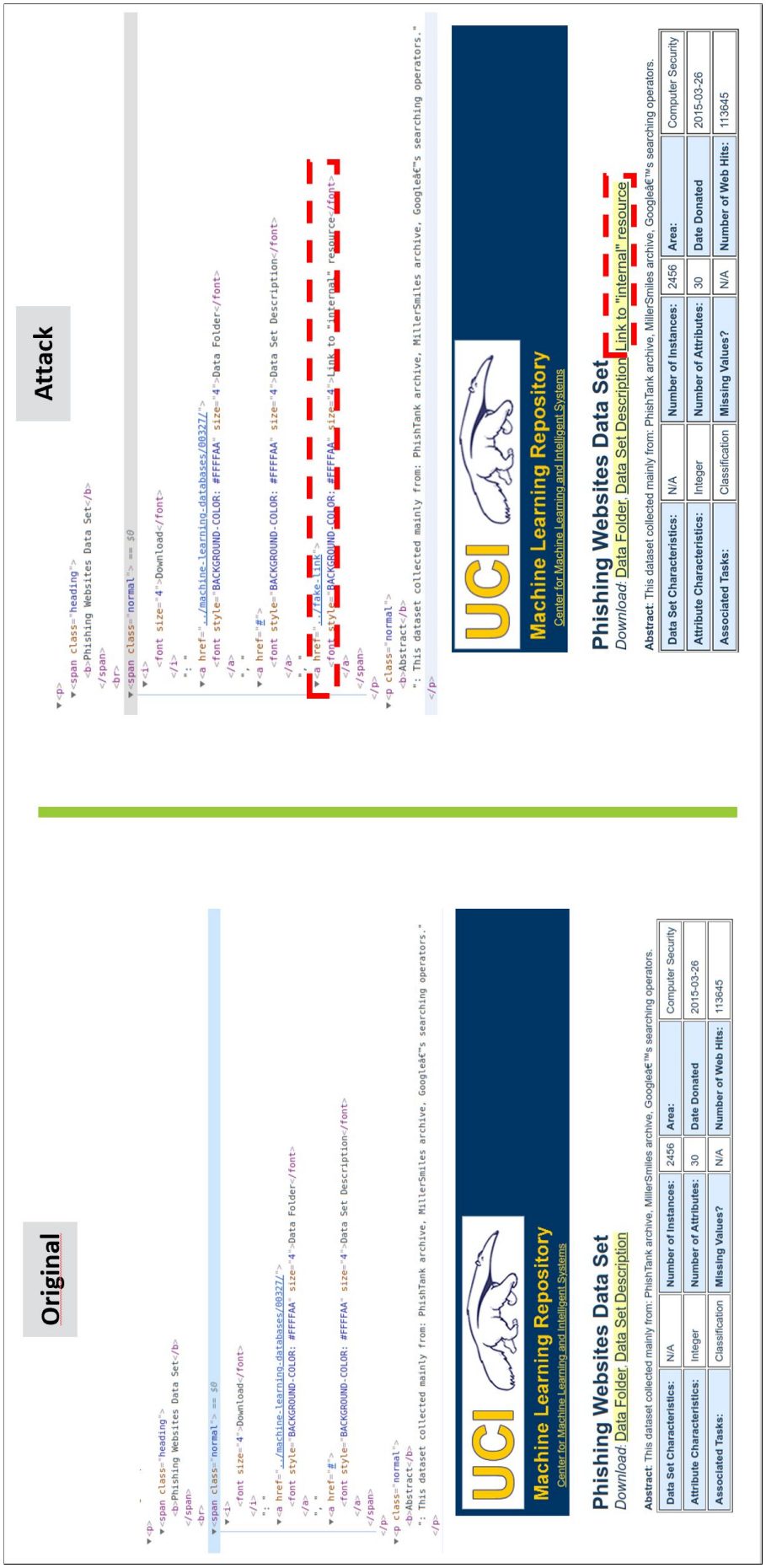



FIGURE 5.15: An example of the GBA-1 Attack.

reasonable to assume that attackers will try to use attacks GBA-1 and GBA-2. Nevertheless, we will show that all the gray box attacks have a significant impact on classification algorithms used in the literature [43], [166], [231]–[233], [251]–[253].

### 5.4.4 Evaluation results

We implemented algorithms to extract all the features shown in Table 5.15, as well as all the gray box attacks and the POC algorithm by leveraging the classification algorithms in Scikit-Learn [203]. We use the implementation to: (i) quantify the *Impact* (as defined earlier in Section 5.4.3) of the GBA-1-GBA-4 attacks on the performance of existing phishing detectors, and (ii) to assess the efficacy of the POC algorithm in mitigating these attacks. We address each of these below.

**Efficacy of GBA-1–GBA-3 on existing PDs** Past works on phishing website detection tend to use shallow classifiers tested mostly on the `UCI` dataset or with custom-made datasets (whose malicious samples are collected from PhishTank or OpenPhish, and benign samples from Amazon or Google) – thus, the features these past works use are determined by the features provided by the adopted dataset. Table 5.16 shows what classifiers and datasets (and hence feature sets) different past works use. As a consequence, when we compare the *Impact* of attacks GBA-1–GBA-4 and the efficacy of POC, we show the results for 13 different classifiers on the 3 publicly available datasets considered in this evaluation (as well as on the DSAIL dataset) — these results include those of past work as well as some combinations that have not been tried before. For instance, Deep & Wide is a relatively new deep learning classifier that we have not previously seen used for phishing URL prediction and so all combinations involving this classifier are novel.

Tables 5.17 show the *Impact* of the GBA-1–GBA-3 attack on each of the 4 datasets and 13 classifiers (used in existing phishing detectors). We see, for instance, that the *Impact* of GBA-1 on the DSAIL dataset using RF leads to a 12.4% drop, but the drop is 96.6% on the `DeltaPhish` dataset. On average (last rows of subtables in Tables 5.17), the GBA-1–GBA-3 attacks lead to significant drops on all datasets (varying from 11.7% to 90.2%) and irrespective of the classifier used. All results are statistically significant via a Mann-Whitney U-test with $p < 0.00001$.

TABLE 5.16: Classifiers and Dataset considered by existing PDs.

| *Reference* | *Classifier* | *Dataset* |
|---|---|---|
| [250] | FNN | Custom |
| [230] | RF, NB, FNN, LR, SVM | Custom |
| [43] | Association Rules | Custom |
| [273] | RF, SVM, NB, FNN, LR | Custom |
| [232] | RF, NB, LR | Custom |
| [233] | Association Rules | Custom |
| [231] | RF, SVM, AB | UCI |
| [228] | RF, KNN, SVM, FNN, NB | UCI |
| [236] | FNN, SVM, KNN, NB, RF | UCI |
| [166] | SVM | DeltaPhish |
| [251] | LR, SVM | UCI |
| [252] | RF | Custom |
| [235] | RF, KNN, AB | Custom |
| [253] | RF | Ebbu |

TABLE 5.17: *Impact* of GBA-1 to GBA-3 on every classifier for each dataset.

| Classifier | DSAIL | DeltaPhish | Mendeley | UCI |
|---|---|---|---|---|
| RF | 0.124 | 0.966 | 0.305 | 0.750 |
| SVM | 0.107 | 0.784 | 0.397 | 0.730 |
| KNN | 0.066 | 0.436 | 0.017 | 0.189 |
| SGD | 0.121 | 0.422 | 0.341 | 0.730 |
| DT | 0.126 | 0.942 | 0.813 | 0.755 |
| LR | 0.003 | 0.080 | 0.484 | 0.746 |
| NB | 0.267 | 0.911 | 0.096 | 0.506 |
| FNN | 0.137 | 0.696 | 0.293 | 0.712 |
| AB | 0.114 | 0.994 | 0.252 | 0.748 |
| ET | 0.190 | 0.965 | 0.619 | 0.696 |
| GB | 0.132 | 0.984 | 0.373 | 0.616 |
| DnW | 0.007 | 0.610 | 0.249 | 0.269 |
| Bag | 0.124 | 0.980 | 0.675 | 0.723 |
| average | 0.117 | 0.751 | 0.378 | 0.628 |

(A) *Impact* of GBA-1.

| Classifier | DSAIL | DeltaPhish | Mendeley | UCI |
|---|---|---|---|---|
| RF | 0.290 | 0.099 | 0.161 | 0.112 |
| SVM | 0.437 | 0.103 | 0.070 | 0.200 |
| KNN | 0.567 | 0.081 | 0.970 | 0.361 |
| SGD | 0.672 | 0.573 | 0.954 | 0.157 |
| DT | 0.097 | 0.078 | 0.121 | 0.159 |
| LR | 0.022 | 0.270 | 0.280 | 0.200 |
| NB | 0.364 | 1.000 | −0.056 | 0.502 |
| FNN | 0.048 | 0.134 | 0.123 | 0.272 |
| AB | 0.075 | 0.126 | 0.128 | 0.104 |
| ET | 0.189 | 0.139 | 0.081 | 0.090 |
| GB | 0.172 | 0.062 | 0.080 | 0.455 |
| DnW | 0.102 | 0.004 | 0.301 | 0.739 |
| Bag | 0.145 | 0.119 | 0.732 | 0.162 |
| average | 0.244 | 0.214 | 0.303 | 0.270 |

(B) *Impact* of GBA-2.

| Classifier | DSAIL | DeltaPhish | Mendeley | UCI |
|---|---|---|---|---|
| RF | 0.311 | 1.000 | 0.669 | 1.000 |
| SVM | 0.540 | 0.972 | 0.611 | 1.000 |
| KNN | 0.654 | 0.709 | 0.972 | 0.673 |
| SGD | 0.697 | 0.926 | 0.998 | 1.000 |
| DT | 0.287 | 0.942 | 0.823 | 1.000 |
| LR | 0.015 | 0.603 | 0.831 | 1.000 |
| NB | 0.460 | 1.000 | 0.062 | 1.000 |
| FNN | 0.171 | 0.930 | 0.452 | 0.998 |
| AB | 0.132 | 1.000 | 0.481 | 1.000 |
| ET | 0.198 | 1.000 | 0.628 | 1.000 |
| GB | 0.185 | 1.000 | 0.510 | 1.000 |
| DnW | 0.106 | 0.652 | 0.469 | 0.999 |
| Bag | 0.285 | 1.000 | 0.951 | 0.940 |
| average | 0.310 | 0.902 | 0.650 | 0.893 |

(C) *Impact* of GBA-3.

**Efficacy of GBA-4 on existing PDs** We recall that the GBA-4 attack proceeds under the assumption that the attacker knows $\Delta$% of the features used by the phishing detector. We vary $\Delta$ from $10 - 70$% in steps of 10%. Tables 5.18 show the *Impact* of this attack on existing phishing detectors on all considered datasets. Stastistical significance of all results is computed through Mann-Whitney U-test obtaining $p < 0.00001$. Unsurprisingly, as $\Delta$ increases, the attacks have a greater impact on average (the last line in the subtables of Tables 5.18 shows steady increases from left to right). Moreover, the attacks have a significant impact — for instance, if the attacker knows 30% of the features used by the defender, the *Impact* is 15.7% to 43.1% which is very substantial.

**No Attack Case: Performance of Baselines vs. POC** Table 5.19 shows the result of using 13 classifiers on the four datasets as done by past work, while Table 5.20 shows the result of using the POC approach using all the features in the dataset. All results are proven to be statistically significant via a Mann-Whitney U-test with $p < 0.00001$. Verifying the efficacy of POC in the no attack case is crucial because the random selection of features can lead to a drop in performance (e.g. if important features

TABLE 5.18: *Impact* of the GBA-4 attacks on the baseline versions of each classifier for every dataset.

| **DSAIL** | **Features Modified** | | | | | | |
|---|---|---|---|---|---|---|---|
| Classifier | *10%* | *20%* | *30%* | *40%* | *50%* | *60%* | *70%* |
| RF | 0.001 | 0.044 | 0.084 | 0.145 | 0.173 | 0.178 | 0.137 |
| SVM | 0.072 | 0.243 | 0.481 | 0.496 | 0.577 | 0.681 | 0.686 |
| KNN | 0.001 | 0.013 | 0.022 | 0.026 | 0.043 | 0.043 | 0.034 |
| SGD | 0.258 | 0.278 | 0.307 | 0.360 | 0.425 | 0.496 | 0.612 |
| DT | −0.002 | −0.001 | 0.099 | 0.099 | 0.205 | 0.204 | 0.204 |
| LR | 0.082 | 0.089 | 0.239 | 0.325 | 0.368 | 0.387 | 0.441 |
| NB | 0.068 | 0.084 | 0.125 | 0.199 | 0.340 | 0.519 | 0.639 |
| FNN | 0.102 | 0.113 | 0.193 | 0.375 | 0.450 | 0.742 | 0.650 |
| AB | 0.005 | 0.005 | 0.101 | −0.000 | −0.000 | −0.003 | −0.002 |
| ET | −0.002 | 0.005 | 0.011 | 0.047 | 0.090 | 0.087 | 0.087 |
| GB | 0.045 | 0.049 | 0.153 | 0.344 | 0.369 | 0.478 | 0.530 |
| DnW | 0.012 | 0.010 | 0.009 | 0.056 | 0.009 | 0.089 | 0.264 |
| Bag | 0.003 | 0.008 | 0.219 | 0.271 | 0.277 | 0.277 | 0.337 |
| average | 0.049 | 0.072 | 0.157 | 0.211 | 0.255 | 0.321 | 0.356 |

(A) *Impact* for the DSAIL dataset.

| **DeltaPhish** | **Features Modified** | | | | | | |
|---|---|---|---|---|---|---|---|
| Classifier | *10%* | *20%* | *30%* | *40%* | *50%* | *60%* | *70%* |
| RF | 0.038 | 0.198 | 0.356 | 0.389 | 0.678 | 0.599 | 0.567 |
| SVM | 0.049 | 0.221 | 0.037 | 0.046 | −0.123 | −0.082 | −0.405 |
| KNN | 0.002 | 0.157 | 0.327 | 0.491 | 0.568 | 0.699 | 0.557 |
| SGD | −0.064 | 0.106 | 0.267 | 0.111 | 0.302 | 0.484 | 0.112 |
| DT | −0.001 | 0.080 | 0.110 | 0.191 | 0.189 | 0.270 | 0.269 |
| LR | 0.135 | 0.327 | 0.522 | 0.543 | 0.625 | 0.659 | 0.523 |
| NB | 0.037 | 0.105 | 0.254 | 0.474 | 0.631 | 0.640 | 0.693 |
| FNN | 0.081 | 0.211 | 0.317 | 0.389 | 0.528 | 0.626 | 0.763 |
| AB | −0.023 | 0.167 | 0.224 | 0.223 | 0.176 | 0.599 | 0.637 |
| ET | 0.010 | 0.096 | 0.235 | 0.254 | 0.503 | 0.583 | 0.687 |
| GB | 0.018 | 0.080 | 0.138 | 0.220 | 0.292 | 0.376 | 0.415 |
| DnW | 0.012 | 0.010 | 0.009 | 0.011 | 0.009 | 0.098 | 0.265 |
| Bag | 0.087 | 0.150 | 0.298 | 0.319 | 0.351 | 0.393 | 0.400 |
| average | 0.030 | 0.147 | 0.238 | 0.282 | 0.363 | 0.458 | 0.421 |

(B) *Impact* for the DeltaPhish dataset.

| **Mendeley** | **Features Modified** | | | | | | |
|---|---|---|---|---|---|---|---|
| Classifier | *10%* | *20%* | *30%* | *40%* | *50%* | *60%* | *70%* |
| RF | 0.033 | 0.235 | 0.293 | 0.474 | 0.618 | 0.593 | 0.596 |
| SVM | 0.189 | 0.317 | 0.345 | 0.299 | 0.467 | 0.596 | 0.643 |
| KNN | 0.132 | 0.240 | 0.408 | 0.631 | 0.653 | 0.706 | 0.700 |
| SGD | 0.041 | 0.212 | 0.218 | 0.184 | 0.245 | 0.320 | 0.336 |
| DT | 0.095 | 0.247 | 0.371 | 0.461 | 0.517 | 0.585 | 0.522 |
| LR | 0.082 | 0.117 | 0.213 | 0.397 | 0.433 | 0.393 | 0.461 |
| NB | −0.072 | −0.148 | −0.233 | −0.227 | −0.202 | −0.169 | −0.105 |
| FNN | 0.118 | 0.141 | 0.148 | 0.221 | 0.239 | 0.300 | 0.374 |
| AB | 0.049 | 0.156 | 0.226 | 0.375 | 0.450 | 0.548 | 0.453 |
| ET | 0.196 | 0.345 | 0.564 | 0.659 | 0.758 | 0.672 | 0.783 |
| GB | 0.034 | 0.116 | 0.261 | 0.323 | 0.406 | 0.431 | 0.614 |
| DnW | 0.037 | 0.081 | 0.185 | 0.271 | 0.456 | 0.568 | 0.631 |
| Bag | 0.129 | 0.444 | 0.570 | 0.580 | 0.746 | 0.805 | 0.886 |
| average | 0.083 | 0.193 | 0.275 | 0.358 | 0.445 | 0.488 | 0.531 |

(C) *Impact* for the Mendeley dataset.

| **UCI** | **Features Modified** | | | | | | |
|---|---|---|---|---|---|---|---|
| Classifier | *10%* | *20%* | *30%* | *40%* | *50%* | *60%* | *70%* |
| RF | 0.022 | 0.304 | 0.554 | 0.481 | 0.553 | 0.329 | 0.038 |
| SVM | 0.135 | 0.337 | 0.501 | 0.577 | 0.537 | 0.352 | 0.338 |
| KNN | 0.163 | 0.374 | 0.481 | 0.557 | 0.800 | 0.864 | 0.814 |
| SGD | 0.201 | 0.298 | 0.371 | 0.532 | 0.657 | 0.601 | 0.400 |
| DT | 0.300 | 0.390 | 0.403 | 0.389 | 0.435 | 0.480 | 1.000 |
| LR | 0.199 | 0.429 | 0.435 | 0.604 | 0.445 | 0.316 | 0.220 |
| NB | 0.400 | 0.350 | 0.237 | 0.085 | −0.221 | −0.422 | −0.735 |
| FNN | 0.254 | 0.467 | 0.715 | 0.639 | 0.695 | 0.331 | 0.261 |
| AB | 0.200 | 0.310 | 0.401 | 0.389 | 0.454 | 0.312 | 0.334 |
| ET | 0.110 | 0.212 | 0.348 | 0.607 | 0.110 | 0.085 | 0.086 |
| GB | 0.123 | 0.270 | 0.364 | 0.612 | 0.730 | 0.565 | 0.464 |
| DnW | 0.205 | 0.250 | 0.384 | 0.491 | 0.472 | 0.586 | 0.627 |
| Bag | 0.056 | 0.192 | 0.409 | 0.517 | 0.616 | 0.666 | 0.695 |
| average | 0.182 | 0.322 | 0.431 | 0.498 | 0.483 | 0.390 | 0.350 |

(D) *Impact* for the UCI dataset.

TABLE 5.19: No attack case: Baseline results for each dataset (using all available features).

| | DSAIL | | | | DeltaPhish | | | | Mendeley | | | | UCI | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Classifier | *F1-score* | *ROCAUC* | *TPR* | *Acc* | *F1-score* | *ROCAUC* | *TPR* | *Acc* | *F1-score* | *ROCAUC* | *TPR* | *Acc* | *F1-score* | *ROCAUC* | *TPR* | *Acc* |
| RF | 0.973 | 0.980 | 0.972 | 0.982 | 0.959 | 0.963 | 0.926 | 0.989 | 0.978 | 0.974 | 0.989 | 0.975 | 0.973 | 0.973 | 0.975 | 0.973 |
| SVM | 0.983 | 0.985 | 0.974 | 0.989 | 0.450 | 0.650 | 0.319 | 0.876 | 0.927 | 0.928 | 0.943 | 0.928 | 0.943 | 0.934 | 0.958 | 0.936 |
| KNN | 0.996 | 0.997 | 0.997 | 0.998 | 0.971 | 0.983 | 0.971 | 0.991 | 0.974 | 0.975 | 0.984 | 0.975 | 0.984 | 0.981 | 0.997 | 0.983 |
| SGD | 0.985 | 0.987 | 0.977 | 0.990 | 0.458 | 0.655 | 0.333 | 0.874 | 0.931 | 0.933 | 0.943 | 0.932 | 0.939 | 0.930 | 0.951 | 0.932 |
| DT | 0.986 | 0.989 | 0.985 | 0.991 | 0.988 | 0.992 | 0.985 | 0.996 | 0.946 | 0.948 | 0.948 | 0.948 | 0.990 | 0.989 | 0.991 | 0.989 |
| LR | 0.985 | 0.987 | 0.977 | 0.990 | 0.521 | 0.771 | 0.765 | 0.776 | 0.937 | 0.939 | 0.946 | 0.938 | 0.942 | 0.933 | 0.957 | 0.935 |
| NB | 0.955 | 0.961 | 0.932 | 0.971 | 0.714 | 0.814 | 0.667 | 0.915 | 0.802 | 0.828 | 0.700 | 0.832 | 0.651 | 0.740 | 0.485 | 0.715 |
| FNN | 0.990 | 0.991 | 0.983 | 0.994 | 0.929 | 0.943 | 0.892 | 0.978 | 0.976 | 0.976 | 0.975 | 0.976 | 0.984 | 0.982 | 0.985 | 0.983 |
| AB | 0.986 | 0.987 | 0.977 | 0.991 | 0.872 | 0.909 | 0.833 | 0.961 | 0.949 | 0.951 | 0.952 | 0.951 | 0.947 | 0.939 | 0.962 | 0.941 |
| ET | 0.999 | 0.999 | 0.999 | 0.999 | 0.988 | 0.990 | 0.980 | 0.996 | 0.991 | 0.991 | 0.988 | 0.991 | 0.991 | 0.990 | 0.992 | 0.990 |
| GB | 0.999 | 0.999 | 0.999 | 0.999 | 0.995 | 0.995 | 0.990 | 0.998 | 0.990 | 0.990 | 0.989 | 0.991 | 0.990 | 0.988 | 0.993 | 0.988 |
| DnW | 0.988 | 0.989 | 0.980 | 0.992 | 0.929 | 0.941 | 0.886 | 0.980 | 0.969 | 0.970 | 0.965 | 0.970 | 0.974 | 0.969 | 0.981 | 0.970 |
| Bag | 0.987 | 0.989 | 0.980 | 0.992 | 0.983 | 0.989 | 0.980 | 0.995 | 0.986 | 0.987 | 0.984 | 0.987 | 0.989 | 0.987 | 0.991 | 0.988 |
| best | 0.999 | 0.999 | 0.999 | 0.999 | 0.995 | 0.995 | 0.990 | 0.998 | 0.991 | 0.991 | 0.989 | 0.991 | 0.991 | 0.990 | 0.997 | 0.990 |

TABLE 5.20: No attack case: Performance of POC on each dataset (using all available features).

| | DSAIL | | | | DeltaPhish | | | | Mendeley | | | | UCI | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Classifier | *F1-score* | *ROCAUC* | *TPR* | *Acc* | *F1-score* | *ROCAUC* | *TPR* | *Acc* | *F1-score* | *ROCAUC* | *TPR* | *Acc* | *F1-score* | *ROCAUC* | *TPR* | *Acc* |
| RF | 0.997 | 0.998 | 0.997 | 0.998 | 0.990 | 0.990 | 0.980 | 0.997 | 0.965 | 0.963 | 0.963 | 0.963 | 0.988 | 0.987 | 0.994 | 0.987 |
| SVM | 0.957 | 0.963 | 0.935 | 0.972 | 0.306 | 0.590 | 0.195 | 0.866 | 0.682 | 0.642 | 0.715 | 0.646 | 0.898 | 0.886 | 0.913 | 0.888 |
| KNN | 0.997 | 0.998 | 0.997 | 0.998 | 0.985 | 0.991 | 0.985 | 0.995 | 0.933 | 0.922 | 0.983 | 0.925 | 0.986 | 0.984 | 0.993 | 0.985 |
| SGD | 0.842 | 0.892 | 0.904 | 0.888 | 0.409 | 0.645 | 0.365 | 0.841 | 0.815 | 0.798 | 0.831 | 0.800 | 0.902 | 0.891 | 0.915 | 0.892 |
| DT | 0.989 | 0.991 | 0.986 | 0.993 | 0.990 | 0.990 | 0.980 | 0.997 | 0.881 | 0.875 | 0.869 | 0.875 | 0.987 | 0.985 | 0.991 | 0.986 |
| LR | 0.942 | 0.948 | 0.904 | 0.963 | 0.382 | 0.667 | 0.635 | 0.689 | 0.781 | 0.789 | 0.723 | 0.785 | 0.895 | 0.882 | 0.909 | 0.884 |
| NB | 0.882 | 0.916 | 0.901 | 0.921 | 0.723 | 0.880 | 0.850 | 0.902 | 0.734 | 0.623 | 0.927 | 0.642 | 0.891 | 0.876 | 0.913 | 0.879 |
| FNN | 0.990 | 0.990 | 0.982 | 0.993 | 0.940 | 0.963 | 0.935 | 0.982 | 0.883 | 0.864 | 0.933 | 0.868 | 0.980 | 0.978 | 0.989 | 0.979 |
| AB | 0.987 | 0.990 | 0.984 | 0.991 | 0.921 | 0.945 | 0.900 | 0.977 | 0.870 | 0.865 | 0.853 | 0.864 | 0.923 | 0.913 | 0.940 | 0.915 |
| ET | 0.998 | 0.998 | 0.997 | 0.998 | 0.987 | 0.988 | 0.975 | 0.996 | 0.962 | 0.960 | 0.954 | 0.960 | 0.988 | 0.987 | 0.994 | 0.987 |
| GB | 0.998 | 0.998 | 0.998 | 0.998 | 0.985 | 0.987 | 0.975 | 0.995 | 0.961 | 0.958 | 0.962 | 0.958 | 0.991 | 0.990 | 0.996 | 0.990 |
| DnW | 0.987 | 0.989 | 0.983 | 0.992 | 0.901 | 0.918 | 0.841 | 0.970 | 0.901 | 0.897 | 0.913 | 0.912 | 0.968 | 0.963 | 0.985 | 0.964 |
| Bag | 0.988 | 0.990 | 0.983 | 0.992 | 0.990 | 0.990 | 0.980 | 0.997 | 0.956 | 0.954 | 0.956 | 0.954 | 0.987 | 0.986 | 0.993 | 0.986 |
| best | 0.998 | 0.998 | 0.998 | 0.998 | 0.990 | 0.991 | 0.980 | 0.997 | 0.962 | 0.960 | 0.983 | 0.963 | 0.991 | 0.990 | 0.996 | 0.990 |

are left out). We observe that the notion of operation chains used by POC does lead to a small reduction in performance of the best classifier in each case (shown in the last row of Tables 5.19 and 5.20), however we note that this drop is very small. Moreover, as we will show in the next few experiments, POC performs much better than current work when the adversary carries out any of the attacks GBA-1–GBA-4 and hence this almost-negligible reduction in performance is ably compensated by POC's robustness against attacks GBA-1–GBA-4.

**Comparison of Baseline and POC Algorithms under Attacks GBA-1-GBA-3** We now turn to the value of POC in protecting against the GBA-1–GBA-3 attacks. Tables 5.21 show the *difference* between the *Impact* of each these three attacks using an off the shelf classifier and when using POC. A positive difference (shown in bold) means that POC was more resilient to the attack than the baselines, while a negative number means POC was less resilient; higher values are highlighted with a darker background. We verify the statistical significance of these results with a Mann-Whitney U-test with $p < 0.00001$. Tables 5.21 consist mostly of bold entries, showing that POC is more resilient than past work for almost all combinations of

dataset and classifier used. Additionally, we can see from the last rows that on average POC exhibited superior performance for each of the datasets considered — the *Impact* of the GBA-1–GBA-3 attacks on the baselines are 1% to 53.5% higher than for POC(last row of the subtables in Tables 5.21).

TABLE 5.21: Difference of the *Impact* of GBA-1 to GBA-3 between the Baseline and the POC variations of every classifier for each dataset.

| Classifier | DSAIL | DeltaPhish | Mendeley | UCI |
|---|---|---|---|---|
| RF | **0.102** | **0.061** | **0.284** | **0.075** |
| SVM | **0.115** | **0.791** | **0.372** | **0.080** |
| KNN | **0.044** | **0.157** | −0.030 | **0.013** |
| SGD | **0.179** | **0.001** | **0.349** | **0.027** |
| DT | **0.130** | **0.110** | **0.807** | **0.030** |
| LR | **0.028** | **0.076** | **0.477** | **0.025** |
| NB | **0.133** | **0.348** | **0.096** | −0.152 |
| FNN | **0.124** | **0.273** | **0.284** | −0.073 |
| AB | **0.089** | **0.068** | **0.252** | **0.235** |
| ET | **0.171** | **0.115** | **0.544** | **0.086** |
| GB | **0.115** | **0.084** | **0.342** | **0.106** |
| DnW | −0.001 | −0.015 | **0.050** | **0.030** |
| Bag | **0.120** | **0.083** | **0.633** | **0.225** |
| average | **0.103** | **0.166** | **0.344** | **0.055** |

(A) *Impact* difference for GBA-1.

| Classifier | DSAIL | DeltaPhish | Mendeley | UCI |
|---|---|---|---|---|
| RF | **0.082** | −0.365 | **0.058** | **0.057** |
| SVM | **0.426** | **2.026** | **0.377** | **0.191** |
| KNN | **0.048** | −0.876 | **0.903** | **0.256** |
| SGD | **0.763** | −0.223 | **0.682** | **0.137** |
| DT | −0.001 | −0.210 | **0.032** | **0.009** |
| LR | **0.101** | **0.274** | **0.290** | **0.197** |
| NB | **0.270** | **1.077** | −0.125 | **0.540** |
| FNN | −0.106 | **0.054** | **0.058** | **0.079** |
| AB | −0.019 | −0.408 | **0.067** | **0.116** |
| ET | −0.011 | −0.215 | −0.061 | **0.034** |
| GB | −0.002 | −0.398 | −0.023 | **0.223** |
| DnW | −0.03 | −0.219 | **0.288** | **0.130** |
| Bag | −0.059 | −0.386 | **0.591** | **0.058** |
| average | **0.112** | **0.010** | **0.241** | **0.156** |

(B) *Impact* difference for GBA-2.

| Classifier | DSAIL | DeltaPhish | Mendeley | UCI |
|---|---|---|---|---|
| RF | −0.005 | **0.108** | **0.348** | **0.242** |
| SVM | **0.590** | **4.080** | **0.415** | **0.845** |
| KNN | **0.040** | **0.538** | **0.905** | **0.420** |
| SGD | **0.809** | **0.511** | **1.038** | **0.806** |
| DT | **0.152** | **0.223** | **0.569** | **0.484** |
| LR | **0.132** | **0.859** | **0.958** | **0.831** |
| NB | **0.038** | **0.067** | **0.127** | **0.987** |
| FNN | −0.059 | **1.198** | **0.092** | **0.508** |
| AB | **0.025** | **0.046** | **0.248** | **0.550** |
| ET | −0.081 | **0.185** | **0.284** | **0.450** |
| GB | **0.021** | **0.075** | **0.161** | **0.237** |
| DnW | −0.011 | −0.019 | **0.155** | **0.356** |
| Bag | −0.036 | **0.116** | **0.625** | **0.233** |
| average | **0.124** | **0.614** | **0.456** | **0.535** |

(C) *Impact* difference for GBA-3.

**Comparison of Baseline and POC Algorithms under Attack GBA-4** We now turn to the quality of POC in protecting against the GBA-4 attacks. Tables 5.22 show the *difference* between the *Impact* obtained by these three attacks using an off the shelf classifier and when using POC. A positive difference (denoted in bold) means that POC is more resilient to the attack than the baselines, while a negative number means POC is less resilient; higher values are highlighted with a darker background. We can see that most entries in the table are in boldface, suggesting that POC is more resilient to attack GBA-4 irrespective of the dataset and classifier used. As usual, all values are statistically significant due to a Mann-Whitney U-test with $p < 0.00001$. As can be seen from the last rows ("average"), POC exhibits superior performance for 27 out of 28 combinations of dataset and classifier (the one exception is the UCI dataset with $\Delta = 60\%$ where the performance of the baseline is very slightly better than that of POC).

TABLE 5.22: Differences between the *Impact* of the GBA-4 attack on the baseline and on POC

| DSAIL | Features Modified | | | | | | |
|---|---|---|---|---|---|---|---|
| Classifier | *10%* | *20%* | *30%* | *40%* | *50%* | *60%* | *70%* |
| RF | **0.003** | **0.004** | −0.003 | −0.111 | −0.197 | −0.315 | −0.408 |
| SVM | **0.048** | **0.107** | **0.196** | **0.210** | **0.150** | **0.146** | **0.084** |
| KNN | **0.004** | **0.004** | **0.005** | **0.003** | **0.005** | **0.008** | **0.002** |
| SGD | **0.183** | **0.179** | **0.209** | **0.161** | **0.144** | **0.104** | **0.070** |
| DT | **0.000** | **0.001** | **0.006** | −0.029 | −0.034 | −0.237 | −0.133 |
| LR | **0.054** | **0.017** | **0.091** | **0.118** | **0.096** | **0.062** | **0.069** |
| NB | **0.067** | **0.076** | **0.098** | **0.116** | **0.115** | **0.056** | **0.063** |
| FNN | **0.001** | **0.105** | **0.093** | **0.090** | **0.247** | **0.499** | **0.559** |
| AB | **0.008** | **0.008** | −0.027 | −0.136 | −0.166 | −0.370 | −0.368 |
| ET | **0.002** | −0.001 | −0.000 | **0.005** | **0.006** | −0.075 | −0.188 |
| GB | **0.013** | **0.013** | **0.011** | **0.107** | **0.053** | **0.171** | **0.189** |
| DnW | −0.015 | −0.004 | −0.012 | **0.03** | −0.004 | **0.005** | **0.025** |
| Bag | **0.003** | **0.003** | **0.014** | **0.104** | **0.209** | **0.209** | **0.305** |
| average | **0.029** | **0.039** | **0.053** | **0.049** | **0.048** | **0.021** | **0.0210** |

(A) *Impact* differences for the DSAIL dataset.

| DeltaPhish | Features Modified | | | | | | |
|---|---|---|---|---|---|---|---|
| Classifier | *10%* | *20%* | *30%* | *40%* | *50%* | *60%* | *70%* |
| RF | −0.032 | **0.072** | **0.161** | **0.127** | **0.260** | **0.157** | **0.054** |
| SVM | **0.311** | **0.283** | **0.425** | **0.532** | **1.053** | **1.964** | **2.508** |
| KNN | **0.239** | **0.169** | **0.197** | **0.369** | **0.391** | **0.696** | **0.614** |
| SGD | **2.188** | **1.064** | **4.088** | **2.633** | **3.741** | **5.915** | **8.476** |
| DT | −0.084 | −0.141 | −0.238 | −0.211 | −0.220 | −0.231 | −0.219 |
| LR | **0.041** | **0.070** | **0.191** | **0.086** | **0.173** | **0.127** | **0.243** |
| NB | −0.058 | −0.067 | −0.027 | **0.114** | **0.131** | **0.154** | −0.051 |
| FNN | −0.130 | −0.174 | −0.188 | −0.140 | **0.022** | **0.219** | **0.602** |
| AB | −0.023 | −0.022 | −0.080 | −0.216 | −0.405 | −0.093 | −0.027 |
| ET | −0.051 | −0.045 | −0.044 | −0.094 | **0.011** | −0.037 | **0.033** |
| GB | −0.014 | −0.019 | −0.077 | −0.074 | −0.039 | −0.042 | −0.026 |
| DnW | −0.058 | −0.003 | **0.0310** | **0.037** | −0.010 | **0.032** | **0.037** |
| Bag | **0.050** | **0.050** | **0.104** | **0.096** | **0.136** | **0.157** | **0.090** |
| average | **0.183** | **0.095** | **0.348** | **0.252** | **0.403** | **0.694** | **0.949** |

(B) *Impact* differences for the DeltaPhish dataset.

| Mendeley | Features Modified | | | | | | |
|---|---|---|---|---|---|---|---|
| Classifier | *10%* | *20%* | *30%* | *40%* | *50%* | *60%* | *70%* |
| RF | −0.034 | −0.003 | **0.011** | **0.099** | **0.113** | −0.002 | −0.052 |
| SVM | **0.282** | **0.258** | **0.142** | **0.464** | −0.158 | **0.040** | **0.200** |
| KNN | **0.078** | **0.112** | **0.150** | **0.292** | **0.264** | **0.288** | −0.037 |
| SGD | **0.052** | **0.175** | **0.114** | **0.089** | **0.075** | **0.128** | **0.265** |
| DT | −0.063 | **0.027** | −0.086 | −0.016 | **0.084** | **0.129** | **0.174** |
| LR | **0.097** | **0.126** | **0.127** | **0.234** | **0.232** | **0.125** | −0.106 |
| NB | −0.073 | −0.135 | −0.196 | −0.183 | −0.143 | −0.106 | −0.037 |
| FNN | −0.008 | −0.028 | −0.114 | −0.182 | −0.003 | **0.116** | −0.056 |
| AB | **0.023** | **0.049** | **0.114** | **0.067** | **0.066** | **0.319** | **0.162** |
| ET | **0.159** | **0.073** | **0.158** | **0.245** | **0.242** | **0.194** | **0.138** |
| GB | −0.044 | **0.059** | **0.007** | **0.037** | **0.058** | −0.060 | **0.076** |
| DnW | **0.068** | **0.073** | **0.055** | **0.001** | **0.124** | **0.111** | **0.049** |
| Bag | −0.034 | **0.173** | **0.155** | **0.062** | **0.115** | **0.096** | **0.242** |
| average | **0.039** | **0.074** | **0.049** | **0.093** | **0.083** | **0.106** | **0.079** |

(C) *Impact* differences for the Mendeley dataset.

| UCI | Features Modified | | | | | | |
|---|---|---|---|---|---|---|---|
| Classifier | *10%* | *20%* | *30%* | *40%* | *50%* | *60%* | *70%* |
| RF | −0.034 | **0.002** | **0.140** | −0.199 | **0.102** | −0.019 | −0.067 |
| SVM | −0.070 | **0.040** | **0.068** | −0.016 | **0.113** | **0.009** | **0.105** |
| KNN | **0.072** | **0.134** | **0.096** | **0.018** | **0.062** | **0.133** | **0.308** |
| SGD | −0.161 | −0.176 | −0.027 | **0.221** | **0.430** | **0.327** | **0.305** |
| DT | **0.060** | −0.007 | −0.078 | −0.172 | −0.063 | **0.100** | **0.795** |
| LR | **0.116** | **0.093** | −0.021 | **0.015** | −0.115 | **0.001** | **0.174** |
| NB | **0.122** | −0.141 | −0.220 | −0.264 | −0.586 | −0.343 | −0.652 |
| FNN | **0.010** | −0.032 | −0.039 | −0.189 | −0.235 | −0.568 | −0.097 |
| AB | −0.007 | **0.113** | **0.028** | **0.004** | **0.137** | −0.035 | **0.158** |
| ET | **0.026** | **0.008** | **0.173** | **0.553** | **0.045** | **0.048** | −0.136 |
| GB | −0.017 | **0.016** | **0.006** | **0.010** | **0.174** | **0.115** | **0.401** |
| DnW | **0.015** | **0.052** | −0.014 | **0.048** | **0.016** | **0.043** | **0.020** |
| Bag | −0.017 | −0.015 | −0.035 | **0.004** | −0.009 | **0.101** | **0.294** |
| average | **0.010** | **0.007** | **0.007** | **0.003** | **0.006** | −0.005 | **0.124** |

(D) *Impact* differences for the UCI dataset.

# Chapter 6

# Conclusions

The defense of large information systems is characterized by two major problems. On one hand, attackers are capable of performing attacks spanning over long periods of time and employing advanced techniques, allowing them to avoid detection; on the other hand, security analysts are overwhelmed by the huge volume of logs generated daily by network traffic. This situation increases the demand for novel solutions that automate the manual triaging process of thousands of alarms and that are capable of detecting even advanced threats. Modern proposals based on security analytics are increasingly starting to leverage algorithms from the machine learning domain, which are becoming pervasive in multiple fields. However, the application of these techniques to cyber security must still be evaluated with the due diligence.

The first contribution of this thesis is the development of innovative detection approaches based on security analytics. We propose novel methods that aim to provide actionable insights to the human operators through prioritized lists of few suspicious hosts. Moreover, we also devise the first algorithm to detect instances of pivoting activities through the analysis of network flows. The second contribution is devoted to adversarial attacks against cyber detectors. We devise original approaches to mitigate these threats. These solutions improve the state of the art because they not only allow to mitigate the impact of adversarial attacks at both training- and test-time, but also do not suffer the drawbacks of existing countermeasures proposed in the literature, such as reduced detection performance in non-adversarial scenarios.

We initially study the maturity of machine and deep learning algorithms to three relevant detection problems: intrusion, malware and spam. We propose a taxonomy of the most used algorithms for cybersecurity, and evaluate them. Our results evidence that these techniques still present several shortcomings that must be known.

To address the problem of manually triaging thousands of alarms in large organizations, we propose a novel architecture based on ranking and prioritization. We consider an innovative perspective in which we start by analyzing *individual host* behaviors, and then post-correlate outputs to compute various indicators corresponding to different attacker activities. A prioritized list of likely compromised hosts is passed to human analysts, who can focus their attention only on the most suspicious hosts and activities. Experimental evaluations and use-case examples in real-world enterprise networks demonstrate the feasibility and scalability of the proposed approach for online autonomous triage of different attack scenarios.
Then, we devise a novel approach that integrates detection and threat prioritization of pivoting attacks. We formalize pivoting as a temporal graph problem, and then devise the first algorithm for detecting all pivoting paths occurring within the network by analyzing the internal network flows. The reduction of false alarms related to benign pivoting paths is achieved through a novel threat prioritization algorithm that considers different threat indicators typical of malicious pivoting activities. Extensive experimental evaluations on a real and large dataset show that the proposed approach is able to effectively detect and prioritize malicious pivoting activities even against stealthy attackers.
Finally, we propose an innovative method for automatically identifying malicious periodic communications with external hosts. The output is a manageable graylist of external hosts characterized by a considerably higher likelihood of being malicious compared to the entire set of contacted hosts, allowing security analysts to focus only on a limited amount of targets. Extensive evaluation on real traffic data of a large organization validated through external sources demonstrates the efficacy of our proposal, which is capable of timely identification of even malicious hosts that do not raise any NIDS alarm.

In the second part of this thesis we focus on the problems caused by adversarial attacks against cyber detectors. One of the main limitations of machine learning algorithms is their inherent susceptibility to adversarial samples, which are used by attackers to avoid detection by introducing tiny modifications in their malicious data. We perform an extensive analysis of adversarial perturbations against cyber detectors; moreover, we study the effects of existing defensive strategies based on *feature removal* and on *adversarial retraining* when applied to harden the considered

detectors. Our evaluations show that attackers can easily thwart state of the art detection schemes by simply applying slight modification to malicious samples. To further aggravate this problem, the results also highlight some critical limitations of existing countermeasures. All these findings emphasize a critical vulnerability of machine learning applications in cybersecurity scenarios.
To address this issue, we propose innovative countermeasures against attacks performed at both training- and test-time. We present an original approach that aims to mitigate poisoning integrity attacks through transformations of the training dataset. Furthermore, we also devise a novel countermeasure against evasion attempts against botnet detectors by means of the *defensive distillation* technique. Finally, we consider evasion attacks against phishing detectors. We propose and evaluate innovative evasion attacks on existing ML-based classifiers for phishing detection. Then we develop an original countermeasure to these attacks through an algorithm that uses a mix of randomization and feature mapping. All these solutions are experimentally validated in realistic settings and are shown to outperform state of the art methods: the approaches increase the robustness of the detector against the considered adversarial attacks, while retaining its accuracy in non-adversarial settings.

We highlight that most experiments conducted in this thesis are performed on known and publicly available datasets, and are hence reproducible by following the methodologies explained in the experimental sections of this thesis, as we have provided the implementation details to replicate the evaluations performed. Moreover, we have released part of the data (which has been anonymized for privacy reasons) used for the evaluations performed on the network traffic collected in the major department of our institution, so as to induce other researchers to experiment on this data. Finally, we are currently planning to also release the custom dataset that we created for evaluating the countermeasure against adversarial attacks on phishing detectors, which we believe will greatly benefit the scientific community in developing more advanced defensive techniques against the critical threat posed by phishing webpages.

As a concluding remark, the research efforts discussed in this work resulted in the publications of several papers in Conference Proceedings or International Journals: [274]–[279].

Possible extensions of this thesis includes the development of cyber detection

schemes that integrate all our original security analytics solutions. This requires devising methods to correlate the output of our proposals into more high-level reports for the security analysts. Furthermore, with regards to our innovative countermeasures against adversarial attacks, future work involves the evaluation of combinations of the proposed approaches to further increase the resilience of ML-based cyber detectors against adversarial attacks.

# Bibliography


[1] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects", *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[2] R. Ramprasad, R. Batra, G. Pilania, A. Mannodi-Kanakkithodi, and C. Kim, "Machine learning in materials informatics: Recent applications and prospects", *NPJ Computational Materials*, vol. 3, no. 1, p. 54, 2017.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *Nature*, vol. 521, no. 7553, p. 436, 2015.

[4] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection", *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1153–1176, 2016.

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

[6] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection", in *Proc. IEEE Symp. Secur. Privacy*, 2010, pp. 305–316.

[7] P. Torres, C. Catania, S. Garcia, and C. G. Garino, "An analysis of recurrent neural networks for botnet detection behavior", in *Proc. IEEE Biennal Congress of Argentina*, Jun. 2016, pp. 1–6.

[8] J. Kim, J. Kim, H. L. T. Thu, and H. Kim, "Long short term memory recurrent neural network classifier for intrusion detection", in *Proc. IEEE Int. Conf. Platform Techn. Service*, 2016, pp. 1–5.

[9] *Darktrace Industrial Uses Machine Learning to Identify Cyber Campaigns Targeting Critical Infrastructure*, `https://www.darktrace.com/en/press/2017/204/`, Aug. 2019.

[10] *Pluribus one - Attack Prophecy*, `https://www.pluribus-one.it/products/attack-prophecy`, Dec. 2019.

[11] H. Kettani and P. Wainwright, "On the top threats to cyber systems", in *Proc. IEEE Int. Conf. Inf. Comp. Tech.*, Mar. 2019, pp. 175–179.

[12] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers", *Machine learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[13] S. K. Harms and J. S. Deogun, "Sequential association rule mining with time lags", *Journal of Intelligent Information Systems*, vol. 22, no. 1, pp. 7–22, 2004.

[14] A. O. Ali, A. Saleh, and T. Ramdan, "Multilayer perceptrons networks for an intelligent adaptive intrusion detection system", *International Journal of Computer Science and Network Security*, vol. 10, no. 2, p. 275, 2010.

[15] J. Esmaily, R. Moradinezhad, and J. Ghasemi, "Intrusion detection system based on multi-layer perceptron neural networks and decision tree", in *Proc. IEEE Conf. Inf. Knowledge Tech.*, 2015, pp. 1–5.

[16] Z. Zhang, J. Li, C. Manikopoulos, J. Jorgenson, and J. Ucles, "Hide: A hierarchical network intrusion detection system using statistical preprocessing and neural network classification", in *Proc. IEEE Workshop Inf. Assurance Secur.*, 2001, pp. 85–90.

[17] A. Chawla, B. Lee, S. Fallon, and P. Jacob, "Host based intrusion detection system with combined cnn/rnn model", in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2018, pp. 149–158.

[18] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks", in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, IEEE, 2013, pp. 3422–3426.

[19] G. D. Hill and X. J. Bellekens, "Deep learning based cryptographic primitive classification", *arXiv:1709.08385*, 2017.

[20] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks", in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, IEEE, 2015, pp. 1916–1920.

[21] M. Z. Alom, V. Bontupalli, and T. M. Taha, "Intrusion detection using deep belief networks", in *Proc. IEEE National Conf. Aerospace Electr.*, IEEE, 2015, pp. 339–344.

[22] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system", in *Proc. ICST EAI Int. Conf. Bio-inspired Inf. Commun. Techn.*, ICST (Institute for Computer Sciences, Social-Informatics and ..., 2016, pp. 21–26.

[23] Y. Li, R. Ma, and R. Jiao, "A hybrid malicious code detection method based on deep learning", *International Journal of Security and Its Applications*, vol. 9, no. 5, pp. 205–216, 2015.

[24] W. Hardy, L. Chen, S. Hou, Y. Ye, and X. Li, "Dl4md: A deep learning framework for intelligent malware detection", in *Proc. Int. Conf. Data Mining*, The Steering Committee of The World Congress in Computer Science, Computer, 2016, p. 61.

[25] G. Tzortzis and A. Likas, "Deep belief networks for spam filtering", in *Proc. IEEE Int. Conf. Tools Artif. Int.*, IEEE, vol. 2, 2007, pp. 306–309.

[26] G. Mi, Y. Gao, and Y. Tan, "Apply stacked auto-encoder to spam detection", in *Proc. Springer Int. Conf. Swarm Intell.*, Springer, 2015, pp. 3–15.

[27] M. Stevanovic and J. M. Pedersen, "An efficient flow-based botnet detection using supervised machine learning", in *Proc. IEEE Int. Conf. Comput., Netw. and Commun.*, Feb. 2014, pp. 797–801.

[28] S. Ranjan, *Machine learning based botnet detection using real-time extracted traffic features*, US Patent 8,682,812, 2014.

[29] B. Rahbarinia, R. Perdisci, A. Lanzi, and K. Li, "Peerrush: Mining for unwanted p2p traffic", in *Proc. Springer Int. Conf. Detection of Intrusions and Malware and Vuln. Assessment*, Springer, 2013, pp. 62–82.

[30] A. Feizollah, N. B. Anuar, R. Salleh, F. Amalina, S. Shamshirband, *et al.*, "A study of machine learning classifiers for anomaly-based mobile botnet detection", *Malaysian Journal of Computer Science*, vol. 26, no. 4, pp. 251–265, 2013.

[31] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon, "From throw-away traffic to bots: Detecting the rise of dga-based malware", in *USENIX Secur. Symp.*, 2012, pp. 491–506.

[32] T. Chakraborty, F. Pierazzi, and V. Subrahmanian, "Ec2: Ensemble clustering and classification for predicting android malware families", *IEEE Trans. Depend. Sec. Comput.*, 2017.

[33] C. Annachhatre, T. H. Austin, and M. Stamp, "Hidden markov models for malware classification", *Springer Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 2, pp. 59–73, 2015.

[34] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters", *ACM Comput. Arch. News*, vol. 41, no. 3, pp. 559–570, 2013.

[35] S. Abu-Nimeh, D. Nappa, X. Wang, and S. Nair, "A comparison of machine learning techniques for phishing detection", in *Proc. anti-phishing working groups: 2nd annual eCrime researchers summit*, ACM, 2007, pp. 60–69.

[36] G. Xiang, J. Hong, C. P. Rose, and L. Cranor, "Cantina+: A feature-rich machine learning framework for detecting phishing web sites", *ACM T. Inf. Syst. Secur.*, vol. 14, no. 2, p. 21, 2011.

[37] S. Roshan, Y. Miche, A. Akusok, and A. Lendasse, "Adaptive and online network intrusion detection system using clustering and extreme learning machines", *Journal of the Franklin Institute*, vol. 355, no. 4, pp. 1752–1779, 2018.

[38] F. S. Tsai, "Network intrusion detection using association rules", *International Journal of Recent Trends in Engineering*, vol. 2, no. 2, p. 202, 2009.

[39] J. Gardiner and S. Nagaraja, "On the security of machine learning in malware c&c detection: A survey", *ACM Comput. Surv.*, vol. 49, no. 3, p. 59, 2016.

[40] F. Bisio, S. Saeli, P. Lombardo, D. Bernardi, A. Perotti, and D. Massa, "Real-time behavioral dga detection through machine learning", in *Proc. IEEE Int. Conf. Secur. Techn.*, IEEE, 2017, pp. 1–6.

[41] Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang, "An intelligent pe-malware detection system based on association mining", *Journal in computer virology*, vol. 4, no. 4, pp. 323–334, 2008.

[42] W.-F. Hsiao and T.-M. Chang, "An incremental cluster-based approach to spam filtering", *Elsevier Expert Syst. Appl.*, vol. 34, no. 3, pp. 1599–1608, 2008.

[43] N. Abdelhamid, A. Ayesh, and F. Thabtah, "Phishing detection based associative classification data mining", *Elsevier Expert Syst. Appl.*, vol. 41, no. 13, pp. 5948–5959, 2014.

[44] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning", *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

[45] H. S. Anderson, J. Woodbridge, and B. Filar, "Deepdga: Adversarially-tuned domain generation and detection", in *Proc. ACM Workshop Artif. Intell. Secur.*, Oct. 2016, pp. 13–21.

[46] *Cisco IOS NetFlow.* `https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/`, Aug. 2019.

[47] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection", *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41–50, 2018.

[48] P. A. A. Resende and A. C. Drummond, "A survey of random forest based methods for intrusion detection systems", *ACM Comput. Surv.*, vol. 51, no. 3, p. 48, 2018.

[49] R. A. R. Ashfaq, X.-Z. Wang, J. Z. Huang, H. Abbas, and Y.-L. He, "Fuzziness based semi-supervised learning approach for intrusion detection system", *Elsevier Inf. Sci.*, vol. 378, pp. 484–497, 2017.

[50] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning", in *Proc. IEEE Int. Conf. Inf. Netw.*, IEEE, 2017, pp. 712–717.

[51] S. Choudhury and A. Bhowal, "Comparative analysis of machine learning algorithms along with classifiers for network intrusion detection", in *Proc. IEEE Int. Conf. Smart Tech. and Manag. Comp., Commun., Controls, Energy and Materials*, May 2015, pp. 89–95.

[52] B. Hentschel, P. J. Haas, and Y. Tian, "Temporally-biased sampling schemes for online model management", *arXiv preprint 1906.05677*, 2019.

[53] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey", *ACM Comput. Surv.*, vol. 41, no. 3, p. 15, 2009.

[54] E. S. Pilli, R. C. Joshi, and R. Niyogi, "Network forensic frameworks: Survey and research challenges", *Elsevier Digit. Invest.*, vol. 7, no. 1-2, pp. 14–27, 2010.

[55] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, "Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks", in *Proc. ACM Annual Comput. Secur. Appl. Conf.*, ACM, Dec. 2013, pp. 199–208.

[56] M. Andreolini, M. Colajanni, and M. Marchetti, "A collaborative framework for intrusion detection in mobile networks", *Elsevier Inform. Sciences*, vol. 321, pp. 179–192, 2015.

[57] W. Wang and T. E. Daniels, "A graph based approach toward network forensics analysis", *ACM T. Inform. Syst. Secur.*, vol. 12, no. 1, p. 4, 2008.

[58] S. J. Stolfo, "Worm and attack early warning: Piercing stealthy reconnaissance", *IEEE Secur. Privacy*, vol. 2, no. 3, pp. 73–75, 2004.

[59] M. P. Collins and M. K. Reiter, "Hit-list worm detection and bot identification in large networks using protocol graphs", in *Proc. Springer Int. Workshop Recent Advances Intrusion Detection*, Springer, Sep. 2007, pp. 276–295.

[60] M. Feily, A. Shahrestani, and S. Ramadass, "A survey of botnet and botnet detection", in *Proc. IEEE Int. Conf. Emerging Secur. Inform., Syst., Techn.*, IEEE, Jun. 2009, pp. 268–273.

[61] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: Detecting botnet command and control servers through large-scale netflow analysis", in *Proc. ACM Annual Conf. Comput. Secur. Appl.*, Dec. 2012, pp. 129–138.

[62] G. Gu, P. A. Porras, V. Yegneswaran, M. W. Fong, and W. Lee, "Bothunter: Detecting malware infection through ids-driven dialog correlation.", in *USENIX Secur. Symp.*, 2007.

[63] G. Gu, R. Perdisci, J. Zhang, W. Lee, *et al.*, "Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection.", in *USENIX Secur. Symp.*, 2008.

[64] R. Brewer, "Advanced persistent threats: Minimising the damage", *Elsevier Netw. Secur.*, vol. 2014, no. 4, pp. 5–9, 2014.

[65] M. Colajanni, D. Gozzi, and M. Marchetti, "Enhancing interoperability and stateful analysis of cooperative network intrusion detection systems", in *Proc. ACM/IEEE Symp. Arch. Netw. Commun. Syst.*, ACM, Dec. 2007, pp. 165–174.

[66] M. Marchetti, M. Colajanni, and F. Manganiello, "Framework and models for multistep attack detection", *International Journal of Security and Its Applications*, vol. 5, no. 4, pp. 73–90, Oct. 2011.

[67] S. J. Yang, A. Stotz, J. Holsopple, M. Sudit, and M. Kuhl, "High level information fusion for tracking and projection of multistage cyber attacks", *Information Fusion*, vol. 10, no. 1, pp. 107–121, 2009.

[68] K. Ruan, J. Carthy, and T. Kechadi, "Survey on cloud forensics and critical criteria for cloud forensic capability: A preliminary analysis", in *Proc. Conf. Digit. Forensics, Secur., Law*, Association of Digital Forensics, Security and Law, 2011, p. 55.

[69] J. Grover, "Android forensics: Automated data collection and reporting from a mobile device", *Elsevier Digit. Invest.*, vol. 10, S12–S20, 2013.

[70] P. Bhatt, E. Toshiro Yano, and P. M. Gustavsson, "Towards a framework to detect multi-stage advanced persistent threats attacks", in *Proc. IEEE Int. Symp. Servic. Oriented Syst. Eng.*, IEEE, 2014, pp. 390–395.

[71] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains", *Leading Issues in Information Warfare & Security Research*, vol. 1, no. 1, p. 80, 2011.

[72] I. Jeun, Y. Lee, and D. Won, "A practical study on advanced persistent threats", in *Springer Comput. Appl. Secur., Control, Syst. Eng.* Springer, 2012, pp. 144–152.

[73] N. Virvilis and D. Gritzalis, "The big four-what we did wrong in advanced persistent threat detection?", in *Proc. IEEE Int. Conf. Availability, Reliability, Secur.*, IEEE, 2013, pp. 248–254.

[74] M. Marchetti, F. Pierazzi, A. Guido, and M. Colajanni, "Countering advanced persistent threats through security intelligence and big data analytics", in *Proc. IEEE Int. Conf. Cyber Conflict*, May 2016, pp. 243–261.

[75] M. Marchetti, F. Pierazzi, M. Colajanni, and A. Guido, "Analysis of high volumes of network traffic for advanced persistent threat detection", *Elsevier Comput. Netw.*, vol. 109, pp. 127–141, 2016.

[76] J. McPherson, K.-L. Ma, P. Krystosk, T. Bartoletti, and M. Christensen, "Portvis: A tool for port-based detection of security events", in *Proc. ACM Workshop Visual. Data Mining Comput. Secur.*, ACM, Oct. 2004, pp. 73–81.

[77] V. Sekar, N. G. Duffield, O. Spatscheck, J. E. van der Merwe, and H. Zhang, "Lads: Large-scale automated ddos detection system.", in *USENIX Annual Tech. Conf.*, 2006, pp. 171–184.

[78] M. H. Bhuyan, D. Bhattacharyya, and J. Kalita, "Surveying port scans and their detection methodologies", *The Computer Journal*, vol. 54, no. 10, pp. 1565–1581, 2011.

[79] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle, "Grids-a graph based intrusion detection system for large networks", in *Proc. Baltimore National Conf. Inf. Syst. Secur.*, Baltimore, vol. 1, 1996, pp. 361–370.

[80] S. Casolari, S. Tosi, and F. L. Presti, "An adaptive model for online detection of relevant state changes in internet-based systems", *Elsevier Perf. Eval.*, vol. 69, no. 5, pp. 206–226, 2012.

[81] F. Pierazzi, S. Casolari, M. Colajanni, and M. Marchetti, "Exploratory security analytics for anomaly detection", *Elsevier Comput. Secur.*, vol. 56, pp. 28–49, 2016.

[82] *nProbe: An Extensible NetFlow v5/v9/IPFIX Probe for IPv4/v6.* `http://www.ntop.org/products/netflow/nprobe/`, Aug. 2015.

[83] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of ip flow-based intrusion detection", *IEEE Commun. Surveys Tuts.*, vol. 12, no. 3, pp. 343–356, 2010.

[84] P. Goyal, S. Batra, and A. Singh, "A literature review of security attack in mobile ad-hoc networks", *Citeseer Int. J. Comput. Appl.*, vol. 9, no. 12, pp. 11–15, 2010.

[85] M. Newman, *Networks*. Oxford university press, 2018.

[86] V. Ramachandran and S. Nandi, "Detecting arp spoofing: An active technique", in *Proc. Springer Int. Conf. Inform. Syst. Secur.*, Springer, Dec. 2005, pp. 239–250.

[87] U Steinhoff, A Wiesmaier, and R Araújo, "The state of the art in dns spoofing", in *Proc. Intl. Conf. Applied Cryptography and Netw. Secur.*, 2006.

[88] *Pivoting.* `https://offensive-security.com/metasploit-unleashed/pivoting/`, Aug. 2019.

[89] *Cisco Annual Security Report (2014).* `http://www.cisco.com/web/offer/gist_ty2_asset/Cisco_2014_ASR.pdf`, Jun. 2017.

[90] C. Tankard, "Advanced persistent threats and how to monitor and deter them", *Elsevier Netw. Secur.*, vol. 2011, no. 8, pp. 16–19, 2011.

[91] *McAfee Technical Report on Night Dragon Operation.* `https://www.mcafee.com/hk/resources/white-papers/wp-global-energy-cyberattacks-night-dragon.pdf`, Jun. 2017.

[92] *Analysis of the Cyber Attack on the Ukrainian Power Grid.* `http://www.nerc.com/pa/CI/ESISAC/Documents/E-ISAC_SANS_Ukraine_DUC_18Mar2016.pdf`, Jun. 2017.

[93] L. Ayala, "Active Medical Device Cyber-Attacks", in *Cybersecurity for Hospitals and Healthcare Facilities*, Springer, 2016, pp. 19–37.

[94] *Wikileaks vault7: Archimedes documentation.* `https://wikileaks.org/vault7`, Jun. 2017.

[95] L. Hellemons, L. Hendriks, R. Hofstede, A. Sperotto, R. Sadre, and A. Pras, "Sshcure: A flow-based ssh intrusion detection system", in *Proc. Springer Int. Conf. Autonomous Infrastructure, Manag., Secur*, Springer, 2012, pp. 86–97.

[96] A. Sperotto, R. Sadre, P.-T. de Boer, and A. Pras, "Hidden markov model modeling of ssh brute-force attacks", in *Proc. Springer Int. Workshop Distrib. Syst. Operat. Manag.*, Springer, 2009, pp. 164–176.

[97] F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer, "Comprehensive approach to intrusion detection alert correlation", *IEEE Trans. Depend. Sec. Comput.*, vol. 1, no. 3, pp. 146–169, 2004.

[98] P. Li, M. Salour, and X. Su, "A survey of internet worm detection and containment", *IEEE Commun. Surveys Tuts.*, vol. 10, no. 1, 2008.

[99] D. R. Ellis, J. G. Aiken, K. S. Attwood, and S. D. Tenaglia, "A behavioral approach to worm detection", in *Proc. ACM Workshop Rapid Malcode*, ACM, 2004, pp. 43–53.

[100] B. Li, J. Springer, G. Bebis, and M. H. Gunes, "A survey of network flow applications", *Elsevier J. Netw. Comput. Appl.*, vol. 36, no. 2, pp. 567–581, 2013.

[101] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks", *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 2046–2069, 2013.

[102] A. Liu, M. Dong, K. Ota, and J. Long, "Phack: An efficient scheme for selective forwarding attack detection in wsns", *Sensors*, vol. 15, no. 12, pp. 30 942–30 963, 2015.

[103] J. Wu, K. Ota, M. Dong, and C. Li, "A hierarchical security framework for defending against sophisticated attacks on wireless sensor networks in smart cities", *IEEE Access*, vol. 4, pp. 416–424, 2016.

[104] A. Furtună, V.-V. Patriciu, and I. Bica, "A structured approach for implementing cyber security exercises", in *Proc. IEEE Int. Conf. Commun.*, IEEE, 2010, pp. 415–418.

[105] J. L. Obes, C. Sarraute, and G. Richarte, "Attack planning in the real world", in *Working Notes AAAI Workshop Intel. Secur.*, 2010, p. 10.

[106] M. Chapman, G. Tyson, P. McBurney, M. Luck, and S. Parsons, "Playing hide-and-seek: An abstract game for cyber security", in *Proc. ACM Int. Workshop Agents Cybersecur.*, ACM, 2014, p. 3.

[107] J. R. Johnson, E. Hogan, *et al.*, "A graph analytic metric for mitigating advanced persistent threat", in *Proc. IEEE Int. Conf. Intel. Secur. Inform.*, IEEE, 2013, pp. 129–133.

[108] M. A. Noureddine, A. Fawaz, W. H. Sanders, and T. Başar, "A game-theoretic approach to respond to attacker lateral movement", in *Proc. Springer Int. Conf. Decision Game Theory Secur.*, Springer, 2016, pp. 294–313.

[109] A. Fawaz, A. Bohara, C. Cheh, and W. H. Sanders, "Lateral movement detection using distributed data fusion", in *Proc. IEEE Symp. Reliable Distrib. Syst.*, IEEE, 2016, pp. 21–30.

[110] *Penetration Testing Software.* `http://https://www.metasploit.com/`, Aug. 2017.

[111] D. E. Denning, "An intrusion-detection model", *IEEE T. Soft. Eng.*, no. 2, pp. 222–232, 1987.

[112] *Snort users manual*, `http://manual.snort.org/`, Aug. 2017.

[113] C. Kruegel, W. Robertson, and G. Vigna, *Using alert verification to identify successful intrusion attempts*, 4. Walter de Gruyter GmbH & Co. KG, 2004, vol. 27, pp. 219–227.

[114] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks", in *Proc. ACM Conf. Comput. Commun. Secur.*, ACM, 2003, pp. 251–261.

[115] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "Exposure: Finding malicious domains using passive dns analysis.", in *Proc. Netw. Distrib. Syst. Symp.*, 2011.

[116] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, "Path problems in temporal graphs", *Proc. VLDB Endowment*, vol. 7, no. 9, pp. 721–732, 2014.

[117] A. Paranjape, A. R. Benson, and J. Leskovec, "Motifs in temporal networks", in *Proc. ACM Int. Conf. Web Search Data Mining*, ACM, 2017, pp. 601–610.

[118] A. Gupta and N. Nishimura, "Characterizing the complexity of subgraph isomorphism for graphs of bounded path-width", *STACS 96*, pp. 453–464, 1996.

[119] D. B. West *et al.*, *Introduction to graph theory*. Prentice hall Upper Saddle River, 2001, vol. 2.

[120] T. T. Soong, *Fundamentals of probability and statistics for engineers*. John Wiley & Sons, 2004.

[121] A. Arora and V. Dutt, "Cyber security: Evaluating the effects of attack strategy and base rate through instance based learning", in *Int. Conf. Cognitive Model.*, 2013, pp. 1–10.

[122] G. Werner, S. Yang, and K. McConky, "Time series forecasting of cyber attack intensity", in *Proc. ACM Conf. Cyber Inf. Secur. Research*, ACM, 2017, p. 18.

[123] L. Khan, M. Awad, and B. Thuraisingham, "A new intrusion detection system using support vector machines and hierarchical clustering", *The VLDB Journal—The International Journal on Very Large Data Bases*, vol. 16, no. 4, pp. 507–521, 2007.

[124] G. Werner, A. Okutan, S. Yang, and K. McConky, "Forecasting cyberattacks as time series with different aggregation granularity", in *Proc. IEEE Int. Symp. Techn. Homeland Secur.*, IEEE, 2018, pp. 1–7.

[125] A. Shalaginov, K. Franke, and X. Huang, "Malware beaconing detection by mining large-scale dns logs for targeted attack identification", in *Int. Conf. Comput. Intell. Secur. Inf. Syst.*, Apr. 2016.

[126] X. Hu, J. Jang, M. P. Stoecklin, T. Wang, D. L. Schales, D. Kirat, and J. R. Rao, "Baywatch: Robust beaconing detection to identify infected hosts in large-scale enterprise networks", in *IEEE DSN*, 2016.

[127] N. Ben-Asher, S. Hutchinson, and A. Oltramari, "Characterizing network behavior features using a cyber-security ontology", in *Military Communications Conference, MILCOM 2016-2016 IEEE*, IEEE, 2016, pp. 758–763.

[128] R. Perdisci, G. Giacinto, and F. Roli, "Alarm clustering for intrusion detection systems in computer networks", *Elsevier Eng. Appl. Artif. Intell.*, vol. 19, no. 4, pp. 429–438, 2006.

[129] K. Julisch, "Clustering intrusion detection alarms to support root cause analysis", *ACM T. Inf. Syst. Secur.*, vol. 6, no. 4, pp. 443–471, 2003.

[130] S. Moskal, S. J. Yang, and M. E. Kuhl, "Extracting and evaluating similar and unique cyber attack strategies from intrusion alerts", in *Proc. IEEE Int. Conf. Intel. Secur. Inf.*, IEEE, 2018, pp. 49–54.

[131] F. Manganiello, M. Marchetti, and M. Colajanni, "Multistep attack detection and alert correlation in intrusion detection systems", *Information Security and Assurance*, pp. 101–110, 2011.

[132] S. Noel and S. Jajodia, "Optimal ids sensor placement and alert prioritization using attack graphs", *J. Netw. Syst. Manag.*, vol. 16, no. 3, pp. 259–275, 2008.

[133] K. Alsubhi, E. Al-Shaer, and R. Boutaba, "Alert prioritization in intrusion detection systems", in *IEEE NOMS*, 2008.

[134] J. Goebel and T. Holz, "Rishi: Identify bot contaminated hosts by irc nickname evaluation.", *HotBots*, vol. 7, pp. 8–8, 2007.

[135] W. T. Strayer, R. Walsh, C. Livadas, and D. Lapsley, "Detecting botnets with tight command and control", in *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, IEEE, 2006, pp. 195–202.

[136] O. Fajana, G. Owenson, and M. Cocea, "Torbot stalker: Detecting tor botnets through intelligent circuit data analysis", in *Proc. IEEE Int. Symp. Netw. Comput. Appl.*, Oct. 2018, pp. 1–8.

[137] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, "Botfinder: Finding bots in network traffic without deep packet inspection", in *Proc. ACM Int. Conf. Emerging Netw. Exp. Techn.*, ACM, Dec. 2012, pp. 349–360.

[138] P. J. Brockwell and R. A. Davis, *Introduction to time series and forecasting*. Taylor & Francis, 2002, vol. 1.

[139] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic.", in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2008.

[140] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient", in *Noise reduction in speech processing*, Springer, 2009, pp. 1–4.

[141] *Suricata IDS*, `http://suricata-ids.org/`, Aug. 2017.

[142] *Emerging Threats.net Open rulesets.* `https://rules.emergingthreats.net/`, Aug. 2017.

[143] *VirusTotal*, `http://https://www.virustotal.com`, Aug. 2017.

[144] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar, "Adversarial machine learning", in *Proc. ACM Workshop Secur. and Artif. Intell.*, Oct. 2011, pp. 43–58.

[145] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time", in *Springer Joint Europ. Conf. Mach. Learn. and Knowl. Discov. Databases*, Sept. 2013, pp. 387–402.

[146] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Sok: Security and privacy in machine learning", in *Proc. IEEE Europ. Symp. Secur. Privacy*, Apr. 2018, pp. 399–414.

[147] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks", in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 39–57.

[148] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines", in *Proc. Int. Conf. Machin. Learning*, Omnipress, 2012, pp. 1467–1474.

[149] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. Rubinstein, U. Saini, C. A. Sutton, J. D. Tygar, and K. Xia, "Exploiting machine learning to subvert your spam filter", *Proc. of the LEET*, vol. 8, pp. 1–9, 2008.

[150] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, machine learning can be more secure! a case study on android malware detection", *IEEE Trans. Depend. Sec. Comput.*, 2017.

[151] M. Alazab, S. Venkatraman, P. Watters, and M. Alazab, "Zero-day malware detection based on supervised learning algorithms of api call signatures", in *Proc. Australasian Conf. Data Mining*, vol. 121, 2011, pp. 171–182.

[152] S. Dua and X. Du, *Data mining and machine learning in cybersecurity*. Auerbach Publications, 2016.

[153] M. Stevanovic and J. M. Pedersen, "On the use of machine learning for identifying botnet network traffic", *Journal of Cyber Security and Mobility*, vol. 4, no. 2, pp. 1–32, 2016.

[154] M. Mannino, Y. Yang, and Y. Ryu, "Classification algorithm sensitivity to training data with non representative attribute noise", *Elsevier Decis. Support Syst.*, vol. 46, no. 3, pp. 743–751, 2009.

[155] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[156] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation", *ACM Comput. Surv.*, vol. 46, no. 4, p. 44, 2014.

[157] A. Kantchelian, S. Afroz, L. Huang, A. C. Islam, B. Miller, M. C. Tschantz, R. Greenstadt, A. D. Joseph, and J. Tygar, "Approaches to adversarial drift", in *Proc. ACM Workshop Artif. Int. Secur.*, ACM, 2013, pp. 99–110.

[158] B. Biggio, I. Corona, B. Nelson, B. I. Rubinstein, D. Maiorca, G. Fumera, G. Giacinto, and F. Roli, "Security evaluation of support vector machines in adversarial environments", in *Springer Support Vector Machines Appl.* Springer, 2014, pp. 105–153.

[159] N. Dalvi, P. Domingos, S. Sanghai, D. Verma, *et al.*, "Adversarial classification", in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, Aug. 2004, pp. 99–108.

[160] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures", in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, ACM, 2015, pp. 1322–1333.

[161] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks", in *Proc. IEEE Symp. Secur. Privacy*, San Jose, CA, May 2016, pp. 582–597.

[162] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, "Adversarial attacks on neural network policies", *arXiv:1702.02284*, 2017.

[163] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic morphing: An efficient defense against statistical traffic analysis", in *Proc. Netw. Distrib. Syst. Symp.*, Citeseer, 2009.

[164] P. Laskov *et al.*, "Practical evasion of a learning-based classifier: A case study", in *Proc. IEEE Symp. Secur. Privacy*, 2014, pp. 197–211.

[165] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, "Is feature selection secure against training data poisoning?", in *Proc. Int. Conf. Machin. Learning*, Jun. 2015, pp. 1689–1698.

[166] I. Corona, B. Biggio, M. Contini, L. Piras, R. Corda, M. Mereu, G. Mureddu, D. Ariu, and F. Roli, "Deltaphish: Detecting phishing webpages in compromised websites", in *Proc. Springer Europ. Symp. Res. Comput. Secur.*, Sep. 2017, pp. 370–388.

[167] B. Liang, M. Su, W. You, W. Shi, and G. Yang, "Cracking classifiers for evasion: A case study on the google's phishing pages filter", in *Proc. Int. Conf. World Wide Web*, Apr. 2016, pp. 345–356.

[168] D. Lowd and C. Meek, "Good word attacks on statistical spam filters.", in *CEAS*, vol. 2005, 2005.

[169] J. Newsome, B. Karp, and D. Song, "Paragraph: Thwarting signature learning by training maliciously", in *Proc. Springer Int. Workshop Recent Advances Intrusion Detection*, Springer, 2006, pp. 81–105.

[170] B. Biggio, I. Pillai, S. Rota Bulò, D. Ariu, M. Pelillo, and F. Roli, "Is data clustering in adversarial settings secure?", in *Proc. ACM Workshop Artif. Intell. Secur.*, ACM, 2013, pp. 87–98.

[171] C. Liu, B. Li, Y. Vorobeychik, and A. Oprea, "Robust linear regression against training data poisoning", in *Proc. ACM Workshop Artif. Intell. Secur.*, ACM, 2017, pp. 91–102.

[172] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli, "Towards poisoning of deep learning algorithms with back-gradient optimization", in *Proc. ACM Workshop Artif. Intel. Secur.*, ACM, Nov. 2017, pp. 27–38.

[173] S. S. Silva, R. M. Silva, R. C. Pinto, and R. M. Salles, "Botnets: A survey", *Elsevier Comp. Netw.*, 2013.

[174] M. Stevanovic and J. M. Pedersen, "An analysis of network traffic classification for botnet detection", in *Proc. IEEE Int. Conf. Cyber Situat. Awar., Data Analyt., Assessment*, Jun. 2015, pp. 1–8.

[175] *Internet Security Threat Report 2018.* `https://www.symantec.com/security-center/threat-report`, Jun. 2018.

[176] R. Perdisci, I. Corona, and G. Giacinto, "Early detection of malicious flux networks via large-scale passive dns traffic analysis", *IEEE Trans. Depend. Sec. Comput.*, 2012.

[177] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou, and D. Dagon, "Detecting malware domains at the upper dns hierarchy.", in *USENIX Secur. Symp.*, vol. 11, 2011, pp. 1–16.

[178] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings", in *Proc. IEEE Europ. Symp. Secur. Privacy*, Mar. 2016, pp. 372–387.

[179] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks", *IEEE Trans. Evol. Comput.*, 2019.

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

[181] B. Biggio, G. Fumera, and F. Roli, "Pattern recognition systems under attack: Design issues and research challenges", *Int. J. Pattern Recogn. Artif. Intel.*, vol. 28, no. 07, p. 1 460 002, 2014.

[182] Z. Abaid, M. A. Kaafar, and S. Jha, "Quantifying the impact of adversarial evasion attacks on machine learning based android malware classifiers", in *Proc. IEEE Int. Symp. Netw. Comput. Appl.*, Oct. 2017, pp. 1–10.

[183] F. Zhang, P. P. Chan, B. Biggio, D. S. Yeung, and F. Roli, "Adversarial feature selection against evasion attacks", *IEEE Trans. Cybern.*, vol. 46, no. 3, pp. 766–777, 2016.

[184] A. Demontis, P. Russu, B. Biggio, G. Fumera, and F. Roli, "On security and sparsity of linear classifiers for adversarial settings", in *Proc. Springer Joint. Int. Workshops Statist. Tech. Pattern Recognit. and Struct. Syntactic Pattern Recognit.*, Nov. 2016, pp. 322–332.

[185] D. Wu, B. Fang, J. Wang, Q. Liu, and X. Cui, "Evading machine learning botnet detection models via deep reinforcement learning", in *Proc. IEEE Int. Conf. Commun.*, 2019, pp. 1–6.

[186] I. Corona, G. Giacinto, and F. Roli, "Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues", *Elsevier Inform. Sciences*, vol. 239, pp. 201–225, 2013.

[187] Z. Lin, Y. Shi, and Z. Xue, "Idsgan: Generative adversarial networks for attack generation against intrusion detection", *arXiv 1809.02077*, 2018.

[188] T. S. Sethi and M. Kantardzic, "Data driven exploratory attacks on black box classifiers in adversarial domains", *Elsevier Neurocomputing*, vol. 289, pp. 129–143, 2018.

[189] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization", in *Proc. Springer Int. Conf. Inf. Syst. Secur. Privacy*, Jan. 2018, pp. 108–116.

[190] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, "Deceiving end-to-end deep learning malware detectors using adversarial examples", *arXiv:1802.04528*, 2018.

[191] J. Clements, Y. Yang, A. Sharma, H. Hu, and Y. Lao, "Rallying adversarial techniques against deep learning for network security", *arXiv:1903.11688*, 2019.

[192] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial malware binaries: Evading deep learning for malware detection in executables", in *Proc. IEEE Europ. Conf. Sign. Proc.*, Sep. 2018, pp. 533–537.

[193] C. Xiang, F. Binxing, Y. Lihua, L. Xiaoyi, and Z. Tianning, "Andbot: Towards advanced mobile botnets", in *Proc. USENIX Conf. Large-scale Exploits and Emergent Threats*, 2011, pp. 11–11.

[194] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods", *Elsevier Comput. Secur.*, vol. 45, pp. 100–123, 2014.

[195] E. B. Beigi, H. H. Jazi, N. Stakhanova, and A. A. Ghorbani, "Towards effective feature selection in machine learning-based botnet detection approaches", in *Proc. IEEE Conf. Comm. Netw. Secur.*, Oct. 2014.

[196] B. Abraham, A. Mandya, R. Bapat, F. Alali, D. E. Brown, and M. Veeraraghavan, "A comparison of machine learning approaches to detect botnet traffic", in *Proc. IEEE Int. Joint Conf. Neur. Netw.*, Jul. 2018, pp. 1–8.

[197] R. A. Bridges, T. R. Glass-Vanderlan, M. D. Iannacone, M. S. Vincent, and Q. G. Chen, "A survey of intrusion detection systems leveraging host data", *ACM Computing Surveys*, vol. 52, no. 6, p. 128, 2019.

[198] D. S. Berman, A. L. Buczak, J. S. Chavis, and C. L. Corbett, "A survey of deep learning methods for cyber security", *Information*, vol. 10, no. 4, p. 122, 2019.

[199] F. V. Alejandre, N. C. Cortés, and E. A. Anaya, "Feature selection to detect botnets using machine learning algorithms", in *Proc. IEEE Int. Conf. Elect. Commun. Comp.*, Feb. 2017, pp. 1–7.

[200] B. Biggio, I. Corona, Z.-M. He, P. P. Chan, G. Giacinto, D. S. Yeung, and F. Roli, "One-and-a-half-class multiple classifier systems for secure learning against evasion attacks at test time", in *Proc. Springer Int. Workshop Multiple Classifier Syst.*, Jun. 2015, pp. 168–180.

[201] G Kirubavathi and R Anitha, "Botnet detection via mining of traffic flow characteristics", *Comput. Elect. Eng.*, vol. 50, pp. 91–101, 2016.

[202] A. Pektaş and T. Acarman, "Deep learning to detect botnet via network flow summaries", *Springer Neural Comput. Appl.*, pp. 1–13, 2018.

[203] *scikit-learn: Machine Learning in Python*, Available: `https://scikit-learn.org/`, Aug. 2019.

[204] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[205] B. Biggio, B. Nelson, and P. Laskov, "Support vector machines under adversarial label noise", in *Proc. Asian Conf. Machin. Learning*, 2011, pp. 97–112.

[206] A. Kantchelian, J. D. Tygar, and A. Joseph, "Evasion and hardening of tree ensemble classifiers", in *Int. Conf. Machin. Learning*, 2016, pp. 2387–2396.

[207] H. Xu, C. Caramanis, and S. Mannor, "Robustness and regularization of support vector machines", *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1485–1510, 2009.

[208] M. Brückner and T. Scheffer, "Stackelberg games for adversarial prediction problems", in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2011, pp. 547–555.

[209] B. Biggio, G. Fumera, and F. Roli, "Adversarial pattern classification using multiple classifiers and randomisation", in *Proc. Springer Joint Int. Workshops Statist. Techn. Pattern Recogn. and Structural and Syntactic Pattern Recogn.*, Springer, 2008, pp. 500–509.

[210] G. F. Cretu, A. Stavrou, M. E. Locasto, S. J. Stolfo, and A. D. Keromytis, "Casting out demons: Sanitizing training data for anomaly sensors", in *Proc. IEEE Symp. Secur. Privacy*, IEEE, 2008, pp. 81–95.

[211] B. Biggio, I. Corona, G. Fumera, G. Giacinto, and F. Roli, "Bagging classifiers for fighting poisoning attacks in adversarial classification tasks", in *Proc. Springer Int. Workshop Multiple Classifier Syst.*, Springer, 2011, pp. 350–359.

[212] Y. Cao and J. Yang, "Towards making systems forget with machine unlearning", in *Proc. IEEE Symp. Secur. Privacy*, IEEE, 2015, pp. 463–480.

[213] A. D. Joseph, P. Laskov, F. Roli, J. D. Tygar, and B. Nelson, "Machine learning methods for computer security", in *Dagstuhl Manifestos*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, vol. 3, 2013.

[214] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on gan", *arXiv:1702.05983*, 2017.

[215] S. Calzavara, C. Lucchese, and G. Tolomei, "Adversarial training of gradient-boosted decision trees", in *Proc. ACM Int. Conf. Inf. Knowledge Manag*, 2019, pp. 2429–2432.

[216] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network", in *Int. Conf. Neural Inf. Proces. Syst. Workshop*, Montreal, CAN, Dec. 2014.

[217] A. S. Ross and F. Doshi-Velez, "Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients", in *AAAI Conf. Artif. Intell.*, Apr. 2018.

[218] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification", *arXiv:1606.04435*, 2016.

[219] M. Stevanovic and J. M. Pedersen, "Detecting bots using multi-level traffic analysis", *Int. J. Cyber Situational Awareness*, vol. 1, no. 1, 2016.

[220] S. J. Pan, Q. Yang, *et al.*, "A survey on transfer learning", *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, 2010.

[221] H. Borchani, G. Varando, C. Bielza, and P. Larrañaga, "A survey on multi-output regression", *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 5, no. 5, pp. 216–233, 2015.

[222] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli, "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks", in *Proc. USENIX Secur. Symp.*, 2019, pp. 321–338.

[223] B. Biggio, K. Rieck, D. Ariu, C. Wressnegger, I. Corona, G. Giacinto, and F. Roli, "Poisoning behavioral malware clustering", in *Proc. ACM Workshop Artif. Intell. Secur.*, ACM, 2014, pp. 27–36.

[224] B. Biggio, S. R. Bulò, I. Pillai, M. Mura, E. Z. Mequanint, M. Pelillo, and F. Roli, "Poisoning complete-linkage hierarchical clustering", in *Proc. Springer Joint IAPR Int. Workshops Stat. Tech. Pattern Recogn. and Struct. Syntactic Pattern Recogn.*, Springer, 2014, pp. 42–52.

[225] D. Maiorca, B. Biggio, and G. Giacinto, "Towards robust detection of adversarial infection vectors: Lessons learned in pdf malware", *arXiv:1811.00830*, 2018.

[226] M. Mayhew, M. Atighetchi, A. Adler, and R. Greenstadt, "Use of machine learning in big data analytics for insider threat detection", in *Proc. IEEE Conf. Military Comm.*, Oct. 2015, pp. 915–922.

[227] M. Zhao, B. An, W. Gao, and T. Zhang, "Efficient label contamination attacks against black-box learning models", in *Int. Joint Conf. Artif. Intell.*, Aug. 2017, pp. 3945–3951.

[228] A. Subasi, E. Molah, F. Almkallawi, and T. J. Chaudhery, "Intelligent phishing website detection using random forest classifier", in *Proc. IEEE Int. Conf. Elec. Comput. Tech. Appl.*, Nov. 2017, pp. 1–5.

[229] T. W. Moore and R. Clayton, "The impact of public information on phishing attack and defense", *Communications and Strategies*, no. 81, pp. 45–68, 2011.

[230] R. Basnet, "Learning to detect phishing urls", *International Journal of Research in Engineering and Technology*, vol. 03, pp. 11–24, 2014.

[231] N. Abdelhamid, F. Thabtah, and H. Abdel-jaber, "Phishing detection: A recent intelligent machine learning comparison based on models content and features", in *Proc. IEEE Int. Conf. Intel. Secur. Inform.*, Jul. 2017, pp. 72–77.

[232] R. Verma and K. Dyer, "On the character of phishing urls: Accurate and robust statistical learning classifiers", in *Proc. ACM Conf. Data Appl. Secur. Privacy*, Mar. 2015, pp. 111–122.

[233] S. C. Jeeva and E. B. Rajsingh, "Intelligent phishing url detection using association rule mining", *Springer Hum-Cent. Comput. Info.*, vol. 6, no. 1, p. 10, 2016.

[234] C. L. Tan, K. L. Chiew, K. Wong, *et al.*, "Phishwho: Phishing webpage detection via identity keywords extraction and target domain name finder", *Elsevier Decis. Support Syst.*, vol. 88, pp. 18–27, 2016.

[235] A. Niakanlahiji, B.-T. Chu, and E. Al-Shaer, "Phishmon: A machine learning framework for detecting phishing webpages", in *Proc. IEEE Int. Conf. Intel. Secur. Inf.*, Nov. 2018, pp. 220–225.

[236] W. Ali, "Phishing website detection based on supervised machine learning with wrapper features selection", *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 9, pp. 72–78, 2017.

[237] E. Lancaster, T. Chakraborty, and V. Subrahmanian, "Malt$^p$: Parallel prediction of malicious tweets", *IEEE T. Computational Social Systems*, vol. 5, no. 4, pp. 1096–1108, 2018.

[238] D. L. Cook, V. K. Gurbani, and M. Daniluk, "Phishwish: A stateless phishing filter using minimal rules", in *Proc. Springer Int. Conf. Financ. Crypt. Data Secur.*, Jan. 2008, pp. 182–186.

[239] Y. Zhang, J. I. Hong, and L. F. Cranor, "Cantina: A content-based approach to detecting phishing web sites", in *Proc. ACM Int. Conf. World Wide Web*, May 2007, pp. 639–648.

[240] K.-T. Chen, J.-Y. Chen, C.-R. Huang, and C.-S. Chen, "Fighting phishing with discriminative keypoint features", *IEEE Internet Comput.*, vol. 13, no. 3, pp. 56–63, 2009.

[241] E. Medvet, E. Kirda, and C. Kruegel, "Visual-similarity-based phishing detection", in *Proc. ACM Int. Conf. Secur. Privacy Commun. Netw.*, Sep. 2008, p. 22.

[242] M. Hara, A. Yamada, and Y. Miyake, "Visual similarity-based phishing detection without victim site information", in *Proc. IEEE Symp. Comput. Intel. Cyber Secur.*, Mar. 2009, pp. 30–36.

[243] H Kim and J. Huh, "Detecting dns-poisoning-based phishing attacks from their network performance characteristics", *IET Electron. Lett.*, vol. 47, no. 11, pp. 656–658, 2011.

[244] G. Liu, B. Qiu, and L. Wenyin, "Automatic detection of phishing target from phishing webpage", in *Proc. IEEE Int. Conf. Pattern Recogn.*, Aug. 2010, pp. 4153–4156.

[245] H. Zhang, G. Liu, T. W. Chow, and W. Liu, "Textual and visual content-based anti-phishing: A bayesian approach", *IEEE Trans. Neural Netw.*, vol. 22, no. 10, pp. 1532–1546, Aug. 2011.

[246] C. Whittaker, B. Ryner, and M. Nazif, "Large-scale automatic classification of phishing pages", *Google AI Research*, 2010.

[247] L Cranor, S Egelman, J Hong, and Z. P. Phish, "An evaluation of anti-phishing toolbars", *Technical Report CMU-CyLab-06–018, Carnegie Mellon University CyLab*, vol. 13, 2006.

[248] A. Bergholz, J. De Beer, S. Glahn, M.-F. Moens, G. Paaß, and S. Strobel, "New filtering approaches for phishing email", *Journal of computer security*, vol. 18, no. 1, pp. 7–35, 2010.

[249] F. Toolan and J. Carthy, "Phishing detection using classifier ensembles", in *IEEE eCrime Researchers Summit*, 2009, pp. 1–9.

[250] R. M. Mohammad, F. Thabtah, and L. McCluskey, "Predicting phishing websites based on self-structuring neural network", *Springer Neural Comput. Appl.*, vol. 25, no. 2, pp. 443–458, 2014.

[251] M. Babagoli, M. P. Aghababa, and V. Solouk, "Heuristic nonlinear regression strategy for detecting phishing websites", *Soft Comput.*, vol. 23, no. 12, pp. 4315–4327, 2019.

[252] A. K. Jain and B. B. Gupta, "Towards detection of phishing websites on client-side using machine learning based approach", *Springer Telecom. Syst.*, vol. 68, no. 4, pp. 687–700, 2018.

[253] O. K. Sahingoz, E. Buber, O. Demir, and B. Diri, "Machine learning based phishing detection from urls", *Elsevier Expert Syst. Appl.*, vol. 117, pp. 345–357, 2019.

[254] *Deltaphish dataset*, `https://www.pluribus-one.it/research/cybersecurity/deltaphish`, Accessed: Sept. 2019.

[255] *Mendeley phishing dataset*, `https://data.mendeley.com/datasets/h3cgnj8hft/1`, Accessed: Sept. 2019.

[256] *Uci phishing websites dataset*, `https://archive.ics.uci.edu/ml/datasets/phishing+websites`, Accessed: Sept. 2019.

[257] Z. Katzir and Y. Elovici, "Quantifying the resilience of machine learning classifiers used for cyber security", *Elsevier Expert Syst. Appl.*, vol. 92, pp. 419–429, 2018.

[258] M. Kantarcioglu and B. Xi, "Adversarial data mining: Big data meets cyber security", in *Proc. ACM Conf. Comput. Comm. Secur.*, Oct. 2016, pp. 1866–1867.

[259] Y. Shi and Y. E. Sagduyu, "Evasion and causative attacks with adversarial deep learning", in *Proc. IEEE Conf. Military Comm.*, Oct. 2017, pp. 243–248.

[260] Y. Vorobeychik and M. Kantarcioglu, "Adversarial machine learning", *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 12, no. 3, pp. 1–169, 2018.

[261] A. Warzyński and G. Kołaczek, "Intrusion detection systems vulnerability on adversarial examples", in *Proc. IEEE Conf. Innovations Intell. Syst. Appl.*, Jul. 2018, pp. 1–4.

[262] S. Chen, M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu, and B. Li, "Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach", *Elsevier Comp. Secur.*, vol. 73, pp. 326–344, 2018.

[263] I. Rosenberg, A. Shabtai, L. Rokach, and Y. Elovici, "Generic black-box end-to-end attack against state of the art api call based malware classifiers", in *Proc. Springer Int. Symp. Res. Attacks, Intrusions and Defenses*, Sep. 2018, pp. 490–510.

[264] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning", in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, Apr. 2017, pp. 506–519.

[265] H. Dang, Y. Huang, and E.-C. Chang, "Evading classifiers by morphing in the dark", in *Proc. ACM Conf. Comp. Commun. Secur.*, Oct. 2017, pp. 119–133.

[266] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers", in *Proc. Symp. Netw. Distrib. Syst.*, Feb. 2016, pp. 21–24.

[267] B. Li and Y. Vorobeychik, "Feature cross-substitution in adversarial classification", in *Proc. Advances Neur. Inf. Process. Syst. Conf.*, 2014, pp. 2087–2095.

[268] C. Bai, Q. Han, G. Mezzour, F. Pierazzi, and V. Subrahmanian, "Dbank: Predictive behavioral analysis of recent android banking trojans", *IEEE T. Depend. Secur.*, 2019.

[269] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "Enabling fair ml evaluations for security", in *Proc. ACM Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 2264–2266.

[270] P. Russu, A. Demontis, B. Biggio, G. Fumera, and F. Roli, "Secure kernel machines against evasion attacks", in *Proc. ACM Workshop Artific. Intell. Secur.*, ACM, 2016, pp. 59–69.

[271] Z. He, J. Su, M. Hu, G. Wen, S. Xu, and F. Zhang, "Robust support vector machines against evasion attacks by random generated malicious samples", in *Proc. IEEE Int. Conf. Wavelet Anal. Pattern Recogn.*, Jul. 2017, pp. 243–247.

[272] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, *et al.*, "Wide & deep learning for recommender systems", in *Proc. ACM Workshop on Deep Learn. for Recommender Syst.*, Sep. 2016, pp. 7–10.

[273] R. B. Basnet and T. Doleck, "Towards developing a tool to detect phishing urls: A machine learning approach", in *Proc. IEEE Int. Conf. Comput. Intel. Commun. Techn.*, Feb. 2015, pp. 220–223.

[274] G. Apruzzese, M. Marchetti, M. Colajanni, G. G. Zoccoli, and A. Guido, "Identifying malicious hosts involved in periodic communications", in *Proc. IEEE Int. Symp. Netw. Comput. Appl.*, Oct. 2017, pp. 1–8.

[275] G. Apruzzese and M. Colajanni, "Evading botnet detectors based on flows and random forest with adversarial samples", in *Proc. IEEE Int. Symp. Netw. Comput. Appl.*, Oct. 2018, pp. 1–8.

[276] G. Apruzzese, M. Colajanni, L. Ferretti, A. Guido, and M. Marchetti, "On the effectiveness of machine and deep learning for cybersecurity", in *Proc. IEEE Int. Conf. Cyber Conflicts*, May 2018, pp. 371–390.

[277] G. Apruzzese, M. Colajanni, L. Ferretti, and M. Marchetti, "Addressing adversarial attacks against security systems based on machine learning", in *Proc. IEEE Int. Conf. Cyber Conflicts*, May 2019, pp. 1–18.

[278] G. Apruzzese, F. Pierazzi, M. Colajanni, and M. Marchetti, "Detection and threat prioritization of pivoting attacks in large networks", *IEEE Trans. Emerg. Topics Comput.*, 2017.

[279] F. Pierazzi, G. Apruzzese, M. Colajanni, A. Guido, and M. Marchetti, "Scalable architecture for online prioritisation of cyber threats", in *Proc. IEEE Int. Conf. Cyber Conflicts*, May 2017, pp. 1–18.