symmetry

MDPI

Article

AppCon: Mitigating Evasion Attacks to ML Cyber Detectors

Giovanni Apruzzese *, Mauro Andreolini, Mirco Marchetti, Vincenzo Giuseppe Colacino and Giacomo Russo

Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia, 41125 Modena, Italy; mauro.andreolini@unimore.it (M.A.); mirco.marchetti@unimore.it (M.M.); vincenzogiuseppe.colacino@unimore.it (V.G.C.); giacomorusso@protonmail.com (G.R.)

* Correspondence: giovanni.apruzzese@unimore.it

Received: 24 March 2020; Accepted: 8 April 2020; Published: 21 April 2020

Abstract: Adversarial attacks represent a critical issue that prevents the reliable integration of machine learning methods into cyber defense systems. Past work has shown that even proficient detectors are highly affected just by small perturbations to malicious samples, and that existing countermeasures are immature. We address this problem by presenting AppCon, an original approach to harden intrusion detectors against adversarial evasion attacks. Our proposal leverages the integration of ensemble learning to realistic network environments, by combining layers of detectors devoted to monitor the behavior of the applications employed by the organization. Our proposal is validated through extensive experiments performed in heterogeneous network settings simulating botnet detection scenarios, and consider detectors based on distinct machine- and deep-learning algorithms. The results demonstrate the effectiveness of AppCon in mitigating the dangerous threat of adversarial attacks in over 75% of the considered evasion attempts, while not being affected by the limitations of existing countermeasures, such as performance degradation in non-adversarial settings. For these reasons, our proposal represents a valuable contribution to the development of more secure cyber defense platforms.

Keywords: adversarial attacks; network intrusion detection; evasion attacks; cyber security; machine learning

1. Introduction

Adversarial attacks represent a dangerous menace for real world implementations of machine learning (ML) algorithms [1–4]. This threat involves the production of specific samples that induce the machine learning model to generate an output that is beneficial to an attacker. Literature has identified two categories of adversarial attacks [2]: those occurring at training-time (also known as *poisoning* attacks [5]), and those occurring at test-time (often referred to as *evasion* attacks [6]). Our paper focuses on this latter category due to its relevance for cyber detection scenarios.

The topic of adversarial machine learning has been thoroughly studied by computer vision literature [7–9]. However, surprisingly, proper analyses and efficient solutions to this menace are scarce in the cybersecurity domain. The field of network intrusion detection is poorly investigated [10,11], while multiple works exist in the areas of malware, phishing, and spam detection [6,12–17]. In particular, although several studies have shown the effectiveness of adversarial evasion attacks against botnet detectors [10,11,18,19], there is a lack of proposals to counter this menace that are feasible for real world environments. Some defensive strategies have been proposed and evaluated by existing literature, but they are affected by critical limitations, such as reduced performance in non-adversarial scenarios [19,20], or high maintenance and deployment costs [2,21].

In this paper, we propose AppCon, an original approach that is focused at mitigating the impact of adversarial evasion attacks against ML-based network intrusion detection systems (NIDS), while preserving detection performance in the absence of adversarial attacks. Furthermore, our proposal is specifically addressed to real-world environments, thus favoring its integration into existing defensive schemes. Our solution is based on the idea of restricting the range of samples that an adversary may create to evade the detector, and also leverages the adoption of ensemble models that are shown to produce more robust classifiers [22]. As a practical implementation, we integrate our solution in botnet detectors, due to the high rate of botnet activities in modern organizations [23]; these detectors are based on different supervised machine and deep-learning algorithms. An extensive experimental campaign conducted on a large and public dataset of millions of labeled network flows [24] is used to evaluate AppCon. The results confirm the efficacy of our solution, which is able to decrease the rate of successful evasion attempts against state-of-the-art botnet detectors by nearly 50%, while retaining similar performance in the absence of adversarial attacks. This symmetric benefit, paired with its simple integration into existing defensive schemes, highlight that the proposed approach represents a valid contribution towards the development of more secure cyber detectors.

The remainder of this paper is structured as follows. Section 2 compares this paper against related work. Section 3 presents the proposed countermeasure and describes the evaluation methodology. Section 4 discusses the experimental results. Section 5 concludes the paper with final remarks and possible extensions for future work.

2. Related Work

It is important to provide some pieces of background knowledge on the machine learning methods employed in this work, and on adversarial attacks. Then, we compare our proposal against related work on countermeasures against evasion attacks.

2.1. Machine Learning for Cyber Detection

Machine learning methods are becoming increasingly popular in several domains, such as image and speech processing, social media marketing, healthcare, and also in cybersecurity [25–27]. These techniques can be separated into two main categories: *supervised* algorithms must undergo a training phase with a proper labeled dataset, where each sample is associated with a specific label (or class); on the other hand, *unsupervised* algorithms do not require a labeled dataset. These characteristics make unsupervised methods more suitable for data clustering and rule mining, whereas supervised techniques can be adopted for actual classification tasks [25,26]. In the context of cybersecurity, supervised algorithms can be readily employed as cyber detectors, where the (trained) model is used to determine whether a sample is benign or malicious, thus resembling a binary classification problem [28].

Among the dozens of existing supervised algorithms, this paper focuses on those classifiers that have been found to be particularly effective for scenarios of Network Intrusion Detection [1,3,28]: *Decision Tree* (DT), *Random Forest* (RF), *AdaBoost* (AB), and *Multi-Layer Perceptron* (MLP); we also consider a fifth method based on deep learning and proposed by Google, *Wide and Deep* (WnD). A brief description of each of these methods is provided below, alongside some notable examples in cybersecurity:

- *Decision Tree*: these algorithms are conditional classifiers composed of several nodes. The tree is inspected from top to bottom, where a given condition is checked at each node by analyzing the features of the input sample, leading to the following node [1,3,27,28].
- *Random Forest*: they are *ensemble methods* consisting of several Decision Trees, in which the output is computed after evaluating the prediction of each individual tree composing the "forest" [1,3,27–29].

- *AdaBoost*: similar to Random Forests, these algorithms are able to improve their final performance by putting more emphasis on the "errors" committed during their training phase [30].
- *Multi-Layer Perceptron*: also known as "neural network", they are based on sets of processing units (the neurons) organized in multiple layers and that communicate with each other. The output is provided in the last layer of the architecture [1,3,27,28].
- *Wide and Deep*: this technique is a combination of a linear "wide" model, and a "deep" neural network. The idea is to jointly train these two models and foster the effectiveness of both.

To the best of our knowledge, WnD has not been tested for Cyber Detection yet, but its promising results in other fields motivate our decision to include this deep learning technique into our experiments [31].

2.2. Adversarial Attacks

Security operators often adopt machine learning techniques [3,32], which allow for detecting anomalies and may even reveal novel attacks that are not easily recognizable through traditional signature-based approaches [33,34], thus increasing protection against advanced threats. However, the integration of these methods into cyber defence platforms still presents several issues [28]: among these, one of the most critical problems is that that of *adversarial attacks*.

Adversarial attacks leverage the generation of specific samples that induce a machine learning model to generate an output that is favorable to the attacker, who exploits the high sensitivity of machine learning models to their internal properties [3,35,36]. Early examples of adversarial attacks are the ones proposed in [37,38]: these papers studied the problem in the context of spam filtering, where linear classifiers could be easily tricked by few carefully crafted changes in the text of spam emails without significantly affecting the readability of the spam message. Another typical example of adversarial attack is the one proposed in [39] which targets neural networks classifiers. Here, the authors apply imperceptible perturbations to test images, which are then submitted to the classifiers: the results highlight that it is possible to change the algorithm's predictions arbitrarily.

Adversarial perturbations affect all integrations of machine learning, but in the cyber security sphere the problem is further aggravated due to several intrisic characteristics of this domain. Among these, we cite: the constantly evolving arms race between attackers and defenders; and the continuous modifications that affect the system and network behavior of an organization [28]. These unavoidable and unpredictable changes are known as "concept drift" [40], which is often responsible for decreasing the performance of anomaly detection models. Possible mitigations involve periodic re-calibrations and adjustment processes that can identify behavioral modifications and recent related threats. However, performing such operations is a costly and challenging task in itself [28], and it also facilitates the execution of adversarial attacks [41].

The authors of [4] propose a taxonomy of adversarial attacks that has now become widely accepted by the scientific community [2]. It considers the following properties:

- **Influence**, which denotes whether an attack is performed at training-time or test-time.
  - *Training-time*: it is possible to thwart the algorithm by manipulating the training-set before its training phase, for example by the insertion or removal of critical samples (also known as data *poisoning*).
  - *Test-time*: here, the model has been deployed and the goal is subverting its behavior during its normal operation.
- **Violation**, which identifies the security violation, targeting the availability or integrity of the system.
  - *Integrity*: these attacks have the goal of increasing the model's rate of false negatives. In cybersecurity, this involves having malicious samples being classified as benign, and these attempts are known as *evasion* attacks.

– *Availability*: the aim is to generate excessive amounts of false alarms that prevent or limit the use of the target model.

This work focuses on adversarial attacks performed at test-time that generate integrity violations, that is, *evasion attacks*.

2.3. Existing Defenses

Despite the proven effectiveness of evasion attacks against cyber detectors [1,10,11,18,42,43], proposals involving countermeasures that are suitable for realistic environments are scarce. The authors of [44] propose an approach exclusive to neural networks that hardens these classifiers against evasion attacks, but detectors based on these algorithms are under-performing in cyber detection scenarios [3,10,19,21,28,45]; furthermore, the authors of [46] showed that the method in [44] can be thwarted by skilled adversaries, while the results in [20] show an increased rate of false positives in the baseline performance of the "hardened" classifier. Other methods to defend against evasion attacks may involve *adversarial training* [21,47,48]; the problem is that such strategies require to (continuously) enrich the training dataset with (a sufficient amount of) samples representing all the possible variations of attacks that may be conceived, which is an unfeasible task for real organizations. A known line of defense leverages the adoption of an *altered feature set* [2,49], but these approaches are likely to cause significant performance drops when the modifications involve features that are highly important for the decision-making process of the considered model [11,21]. Solutions based on *game theory* are difficult to deploy [50] and evaluate in practice [2,51], and their true efficacy in real contexts is yet to be proven [52]. Finally, defenses conforming to the *security-by-obscurity* principle [2,19] are not reliable by definition, as the attacker is required to learn the underlying mechanism to thwart them [53].

Within this landscape, we propose a countermeasure that (i) can be applied to any supervised ML algorithm, (ii) does not cause a performance drop in non-adversarial scenarios, (iii) can be easily integrated into existing defensive systems, and (iv) does not rely on security-by-obscurity.

3. Materials and Methods

This section presents the considered threat model and the proposed solution, AppCon; then, it describes the experimental settings adopted for the evaluation.

3.1. Threat Model

To propose a valid countermeasure against adversarial attacks, it is necessary to define a threat model that states the characteristics of both the target system, and of the considered attacker. Our solution assumes a threat model similar to the one in [11], which we briefly summarize.

The defensive model, represented in Figure 1, consists of a large enterprise environment, whose network traffic is inspected by a flow-based [54] NIDS that leverages machine learning classifiers to identify botnet activities.

On the offensive side, an attacker has already established a foothold in the internal network by compromising some hosts and deploying botnet malware communicating with a Command and Control (CnC) infrastructure. The adversary is described by following the notation [2,4] of modeling its *goal*, *knowledge*, *capabilities*, and *strategy*.

- *Goal*: the main goal of the attacker is to evade detection so that he can maintain his access to the network, compromise more machines, or exfiltrate data.
- *Knowledge*: the attacker knows that network communications are monitored by an ML-based NIDS. However, he does not have any information on the model integrated in the detector, but he (rightly) assumes that this model is trained over a dataset containing samples generated by the same or a similar malware variant deployed on the infected machines. Additionally, since he knows that the data-type used by the detector is related to network traffic, he knows some of the basic features adopted by the machine learning model.

- *Capabilities*: we assume that the attacker can issue commands to the bot through the CnC infrastructure; however, he cannot interact with the detector.
- *Strategy*: the strategy to avoid detection is through a *targeted exploratory integrity attack* [6] performed by inserting tiny modifications in the communications between the bot and its CnC server (e.g., [55]).

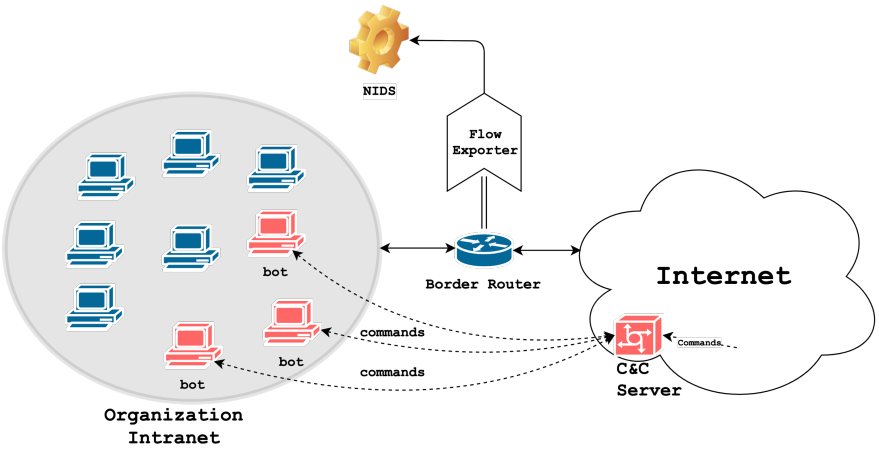

Figure 1. Model of the target network.

We thus consider a "gray-box" threat model, where the adversary has (very) limited knowledge of the defenses. Our assumption portrays a realistic scenario of modern attacks [56]: 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 a few selected secured hosts.

3.2. Proposed Countermeasure

We now describe AppCon, short for "application constraints", which is the main proposal of this paper. We recall that the attacker plans to evade detection by modifying the network communications between the bots and CnC server, without changing the logic of the adopted malware variant. Our solution is based on the idea of restricting the freedom in which an attacker can create malicious adversarial samples that evade detection. The intuition is that an attacker that is subject to additional constraints when devising his samples is less likely to evade detection.

Since our threat model assumes that the attacker cannot change the underlying functionality of the botnet, he is already limited in the amount of alterations that he can perform; however, past literature has shown that even small modifications (like extending the duration of network communications by few seconds, or adding few bytes of junk data to the transmitted packets) can lead to successful evasion [10,11,55]. We aim to further restrict the attacker's range of possibilities by having the detection mechanism to focus only on a (set of) specific web-application(s). This approach allows the detector to monitor only the network traffic with characteristics that are similar to those of the considered applications.

To apply our solution to existing detectors based on supervised ML algorithms, AppCon leverages the paradigm of *ensemble learning*, which has already been adopted to devise countermeasures against adversarial attacks [22]. In particular, the idea is to transform the "initial" detector into an ensemble of detectors, each devoted to a specific web-application. Formally, let D be the considered detector; let A be the set of web-applications employed by an enterprise, and let $a \in A$ be a specific web-application within A. Then, we split D into $|A|$ sub-instances (where $|\cdot|$ is the *cardinality* operator), each devoted to a specific web-application a denoted with D_a. The union of all the sub-instances will then represent our "hardened" detector, denoted as D' (thus, $\bigcup_a D_a = D'$). Therefore, D' will only accept as input those samples that conform to at least one of the network communications generated by a specific

web-application a. Conversely, those network flows that do not fall within the accepted ranges will be either blocked (e.g., through a firewall) or analyzed by other defensive mechanisms (which are out of the scope of this paper). We illustrate the entire workflow of AppCon in Figure 2: the generated network flows are first checked to determine their compatibility with the flows of the accepted web-applications A, and those that are compliant (with at least one) are then forwarded to the ensemble of detectors (represented by $[D_{a_1}, D_{a_2}, \ldots, D_{a_n}]$) composing D'.

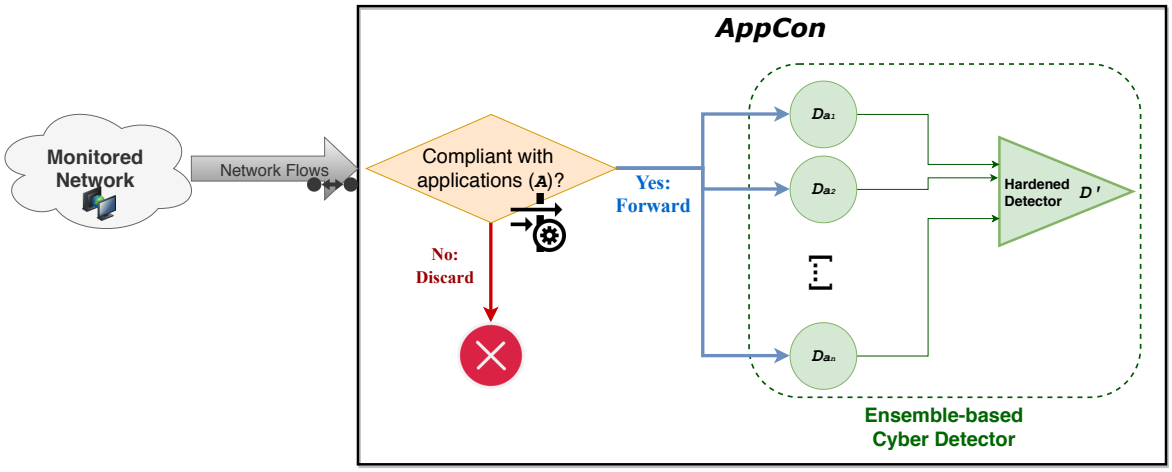

Figure 2. Overview of the proposed solution.

Let us provide an example to facilitate the comprehension of our proposal. Assume an organization adopting a NIDS that uses some machine learning to detect malicious network flows, and assume that this organization employs the web-applications a_1 and a_2: the network flows generated by a_1 have *durations* that vary between 1 and 5 s, whereas those generated by a_2 vary between 10 and 30 s. Let us assume an attacker that has infiltrated the organization network and controls some machines through a botnet variant whose communications with the CnC server generate flows lasting 3 s. In such a scenario, if the attacker plans to evade detection by increasing the length of network communications through small latencies, he will only be able to apply increments of either $+[2, 3]$ s (to fall within the a_1 range) or $+[7, 27]$ s (to fall within the a_2 range), thus considerably limiting his options.

We highlight that the proposed method is suited to modern enterprise networks that generate network data through a finite set of web-application: in such a scenario, a potential attacker cannot apply his perturbations arbitrarily because, if an adversarial sample does not conform to the traffic generated by the considered applications, then it will automatically trigger other defensive mechanisms or be completely blocked.

3.3. Experimental Settings

We present the dataset adopted for our experiments; the development procedures of the detectors; the formulation of appropriate adversarial attacks; the definition of the application constraints that represent the core of our proposal; and the performance metrics to evaluate the cyber detectors.

3.3.1. Dataset

Our experimental campaign is based on the `CTU-13` dataset [24], which consists of a large collection of labeled network traffic data, in the format of network flows, containing both benign and malicious samples belonging to seven different botnet families. Overall, this dataset contains over 15M network flows generated in a network of hundreds of hosts over multiple days. These important characteristics make the `CTU-13` a valid representation of a realistic and modern enterprise network, and many studies have adopted it for their experiments [57,58]. To facilitate the understanding of our testbed, we now describe the main characteristics of the `CTU-13` dataset.

The CTU-13 includes network data captured at the Czech Technical University in Prague, and contains labeled network traffic generated by various botnet variants and mixed with normal and background traffic. It contains 13 distinct data collections (called *scenarios*) of different botnet activity.

The network traffic of each scenario is contained in a specific packet-capture (PCAP) file, which has been converted by the authors into *network flows* [54]. A network flow (or *netflow*) is essentially a sequence of records, each one summarizing a connection between two endpoints (that is, IP addresses). A typical representation of netflow data is given in Table 1. The inspection of network flows allows administrators to easily pinpoint important information between two endpoints (e.g., source and destination of traffic, the class of service, and the size of transmitted data). Network flows provide several advantages over traditional full packet captures, such as: reduced amount of required storage space; faster computation; and reduced privacy issues due to the lack of content payloads [59].

Table 1. Example of network flows.

Date Flow Start	Duration	Proto	Src IP Addr:Port	Dst IP Addr:Port	Packets	*Bytes*
2020-09-01 00:00:00.459	2.346	TCP	192.168.0.142:24920	192.168.0.1:22126	1	46
2020-09-01 00:00:00.763	1.286	UDP	192.168.0.145:22126	192.168.0.254:24920	1	80

To convert the raw network packets into network flows, the authors of the `CTU-13` rely on Argus [60], a network audit system with a client–server architecture: the processing of the packets is performed by the server, and the output of this computation is a detailed status report of all the netflows which is provided to the final clients. After inspecting the `CTU-13`, we can safely assume that the client used by the authors to generate the netflows from each individual PCAP file is `ra`, which has been invoked with the following Command:

Command 1: `CTU-13` netflow generation through Argus.

```
ra -L 0 -c , -s saddr daddr sport dport stime ltime flgs dur proto
stos dtos pkts sbytes dbytes dir state -r inputFile > outputFile.csv
```

where the `-L` option prints headers once, `-c` specifies the field separator, `-s` chooses the fields to extract, and `-r` specifies the file to read the data from. The output is redirected to a CSV file. After this conversion process, the authors proceeded to manually label each individual network flow. Indeed, the `CTU-13` is made available as a collection of 13 CSV files (one for each scenario) presenting the fields specified in Command 1 alongside the added "Label" field, which separates legitimate from illegitimate flows. In particular, benign flows correspond to the *normal* and *background* labels; whereas the *botnet* and *CnC-channel* labels denote malicious samples.

Table 2 shows the meaningful metrics of each scenario in the `CTU-13`. This table also shows the botnet-specific piece of malware used to create the capture, alongside 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, the `Sogou` botnet is not considered because of the limited amount of its malicious samples.

Table 2. Meaningful metrics of the CTU-13 datasets, Source: [24].

Scenario	Duration (h)	Size (GB)	Packets	Netflows	Malicious Flows	Benign Flows	Botnet	# 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	1808	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	4630	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	6126	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	8164	99,087	Rbot	3
12	1.21	8	13,212,268	325,471	2168	323,303	NSIS.ay	3
13	16.36	34	50,888,256	1,925,149	39,993	1,885,156	Virut	1

3.3.2. Developing the Baseline Detectors

As described in Section 2.1, our experiments involve botnet detectors based on five different machine and deep learning algorithms that have been shown by related literature to perform well for botnet detection tasks [3,11,28,45,61–63]: Random Forest (RF), Multi-Layer Perceptron (MLP), Decision Tree (DT), AdaBoost (AB), alongside the recent "Wide and Deep" (WnD) technique proposed by Google [31]. Each detector presents multiple instances, each focused on identifying a specific botnet variant within the adopted dataset—in our case, each detector has six instances (we do not consider the Sogou malware variant due its low amount of available samples). This design idea is motivated by the fact that ML detectors show superior performance when they are used as ensembles instead of "catch-all" solutions, in which each instance addresses a specific problem [2,18,62,63].

Each model is trained through sets of features adopted by related work on flow-based classifiers [11,64,65], reported in Table 3. To compose the training and test datasets for each instance, we rely on the common 80:20 split (in terms of overall malicious samples); benign samples are randomly chosen to compose sets with a benign-to-malicious samples ratio of 90:10.

Table 3. Features of the ML models, Source: [18].

#	Feature Name	Type
1, 2	src/dst IP address type	Bool
3, 4	src/dst port	Num
5	flow direction	Bool
6	connection state	Cat
7	duration (seconds)	Num
8	protocol	Cat
9, 10	src/dst ToS	Num
11, 12	src/dst bytes	Num
13	exchanged packets	Num
14	exchanged bytes	Num
15, 16	src/dst port type	Cat
17	bytes per second	Num
18	bytes per packet	Num
19	packets per second	Num
20	ratio of src/dst bytes	Num

Each classifier is fine-tuned through multiple grid-search operations and validate them through 3-fold cross validation.

The entire procedure followed to prepare the data used to train each detector is displayed in Figure 3. First, the `CTU-13` dataset is pre-processed to compute the derived features (such as the *bytes_per_packet*). Next, we merge the collections pertaining to the same botnet family, and finally create the pool of benign flows by extracting all the non-botnet traffic from each collection and including

them in a dedicated collection. At the end of these operations, we thus obtain eight datasets: seven containing malicious flows of each botnet family (We do not consider the Sogou malware in the evaluation) denoted as X^m, and 1 containing only legitimate flows, denoted as X^l.

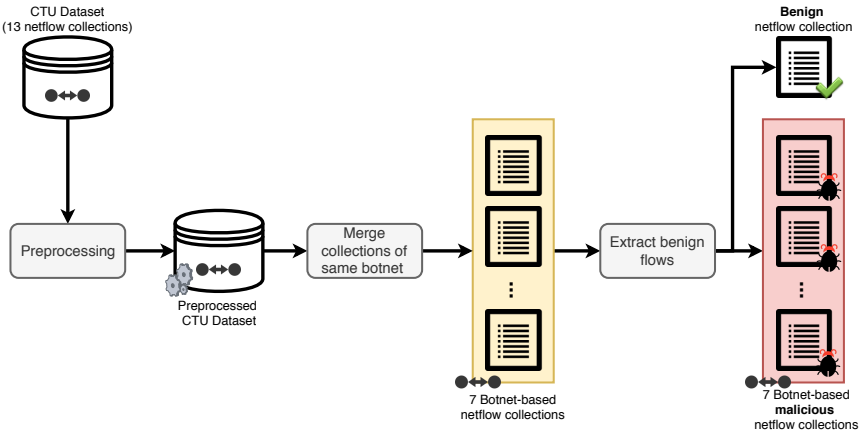

Figure 3. Data preparation workflow.

3.3.3. Design and Implementation of AppCon

Our proposed countermeasure is implemented by devising our detectors on the basis of five different web applications that are widely used by modern enterprises [66–69]: WhatsApp (WhatsApp: https://www.whatsapp.com/), Teams (Microsoft Teams: https://teams.microsoft.com/), Skype (Skype: https://www.skype.com/), OneNote (Microsoft OneNote: https://www.onenote.com/), OneDrive (Microsoft OneDrive: https://office.live.com/). We consider this specific set of applications because their popularity makes them a suitable example for a practical use-case: our approach can be easily expanded by considering more network services. To this purpose, each application is deployed on several dedicated machines which have their network behavior monitored over the course of several days; we also distinguish between *active* and *passive* use of each application. This allows us to identify the samples of network flows in the `CTU-13` dataset that are "compliant" with each of these applications, which are then used to train (and test) our hardened detectors by developing application-specific sub-instances.

As an example, consider the case of the application Teams. After monitoring its network behavior, we determine that, when it is *actively used*, it generates flows transmitting between 71 and 1488 bytes per packet, whereas it transmits 50–1050 bytes per packet during its *passive use*. We then take these two ranges (alongside the accepted ranges of other features used by our detectors—see Table 3) and use them to filter all the flows in the `CTU-13` dataset: only those flows (both malicious and benign) compatible with these ranges will be used to train each (instance of the) detector. Thus, by taking the Random Forest detector as an example, its hardened version will be composed of six instances (each corresponding to a specific malware variant); each of these instances is, in turn, split into five sub-instances, each devoted to monitor a specific application. A schematic representation of this implementation is provided in Figure 4, where the initial ensemble(s) of application-specific detectors (Denoted with $(D_{WhatsApp}, D_{Teams}, D_{Skype}, D_{OneNote}, D_{OneDrive})$) is used as input to another "layer" of botnet-specific classifiers (Denoted with $(D_A^{b_1}, \ldots, D_A^{b_n})$, with $n = 6$ in our case, and b is a specific botnet family), whose output is then combined to generate the final detection.

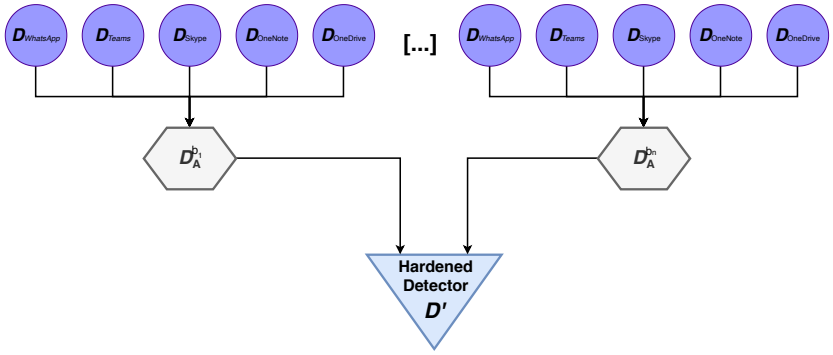

Figure 4. Implementation of the proposed method.

3.3.4. Generation of Adversarial Samples

The attack scenario considered in this paper is reproduced by inserting small perturbations in the malicious network flow samples contained in the `CTU-13` dataset. These perturbations are obtained by adopting the same procedure described in [18], which involve altering (through increments) combinations of the (malicious) flow-based features by small amounts (the corresponding derived features are also updated). This procedure allows for generating multiple adversarial datasets, each corresponding to a specific malware variant, a specific set of altered features, and a specific increment value(s). We anticipate that the generated adversarial samples will be used to evaluate our solution, hence only those botnet flows included in the datasets used for testing the detectors are altered, which allows us to avoid performing the validation phase on samples included in the training sets. To facilitate in the reproduction of our experiments, we now provide a more detailed discussion of the adversarial samples generation process.

Attackers may try to evade detection by inserting small latencies to increase the flow duration; another possibility is appending insignificant bytes to the transmitted packets, thus increasing the number of exchanged bytes or packets. These operations can be easily achieved by botmasters who only need to modify the network communications of the controlled bots [10]; at the same time, these simple strategies comply with our assumption that the underlying application logic of the piece of botnet malware remains unchanged. To devise similar attack strategies, the adversarial samples are crafted by modifying the malicious flows contained in the `CTU-13` through manipulations of up to four features: *flow_duration*, *exchanged_packets*, *src_bytes*, *dst_bytes*. The 15 different groups of feature manipulations are reported in Table 4, which we denote as G. In practical terms, the adversarial samples in group `1b` alter only the flow *src_bytes*, while those of group `3b` include modifications to the *duration*, *src_bytes* and *tot_packets* features.

Table 4. Groups of altered features, Source: [18].

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

The alterations of these features are obtained by increasing their values through nine fixed steps, which we denote as S and which are reported in Table 5. To provide an example, samples obtained through the `V` step of the group `1c` have the values of their flow incoming bytes increased by 64. The adversarial datasets obtained through the `I` step of the group `3c` have the values of their flow duration, outgoing bytes, and exchanged packets increased by 1. We put more emphasis on small increments because not only they are easier to introduce, but they also yield samples that are more similar to their "original" variant (which is a typical characteristic of adversarial perturbations [9]). Furthermore, increasing some of these features by higher amounts may trigger external defensive mechanisms based on anomaly detection [59], or may generate incorrect flows (e.g.,: some flow collectors [70] have flow upper duration limits of 120 s [71]).

Table 5. Increment steps of each feature for generating realistic adversarial samples, Source: [18].

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 complete breakdown of the operations performed to generate our adversarial datasets is provided in Algorithm 1, in which an adversarially manipulated input is denoted through the $\mathcal{A}(\cdot)$ operator. It should be noted that some features are mutually dependent: for instance, changing the flow *duration* also requires to update other time-related features (such as *packets_per_second*): these operations are addressed in line 19 of Algorithm 1.

After applying all these procedures, we generate a total of 135 adversarial datasets for each botnet family (Given by: $15[\text{groups of altered features}] * 9[\text{increment steps}] = 135$), where each dataset represents a different type of evasion attempt. All these attack patterns are compatible with our threat model and can be easily achieved by botmasters [10,55].

3.3.5. Performance Metrics

The machine learning community usually relies on one or more of the following metrics to measure the performance of machine learning models: *Accuracy*, *Recall*, *Precision*, and *F1-score*. These metrics are based on the concept of *Confusion Matrix*. We stress that all problems pertaining to cyber detection can be identified as binary classification problems (that is, a data sample can be either malicious or benign), hence the Confusion Matrix in these contexts is represented as a 2×2 matrix, in which rows represent the output of the detector, and columns represent the true class of the input data. An example of such matrix is reported in Table 6, where TP, FP, TN, and FN denote True Positives, False Positives, True Negatives, and False Negatives, respectively. As is common in cybersecurity settings, a True Positive represents a correct prediction of a malicious sample [18].

Table 6. Example of Confusion Matrix.

		Predicted	
		Malicious	**Benign**
Actual	Malicious	*TP*	*FN*
	Benign	*FP*	*TN*

Algorithm 1: Algorithm for generating datasets of adversarial samples. Source: [72]

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 $\mathcal{A}_s^g(x^b)$

An explanation of each performance metric is provided below.

- *Accuracy*: This metric denotes the percentage of correct predictions out of all the predictions made. It is computed as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

  In Cybersecurity contexts, and most notably in Network Intrusion Detection [28], the amount of malicious samples is several orders of magnitude lower with respect of that of benign samples; that is, malicious actions can be considered as "rare events". Thus, this metric is often neglected in cybersecurity [3,73]. Consider for example a detector that is validated on a dataset with 990 benign samples and 10 malicious samples: if the detector predicts that *all* samples are benign, it will achieve almost perfect *Accuracy* despite being unable to recognize any attack.
- *Precision*: This metric denotes the percentage of correct detections out of all "positive" predictions made. It is computed as follows:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

  Models that obtain a high Precision have a low rate of false positives, which is an appreciable result in Cybersecurity. However, this metric does not tell anything about false negatives.

- *Recall*: This metric, also known as *Detection Rate* or *True Positive Rate*, denotes the percentage of correct detections with respect of all possible detections, and is computed as follows:

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

  In Cybersecurity contexts, it is particularly important due to its ability to reflect how many malicious samples were correctly identified.
- *F1-score*: it is a combination of the *Precision* and *Recall* metrics. It is computed as follows:

$$F1\text{-}score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4}$$

  It is used to summarize in a single value the *Precision* and *Recall* metrics.

To evaluate the quality of the developed detectors, we thus rely on the combination of three different metrics: *Recall*, *Precision*, and *F1-score*; the *Accuracy* metric is not considered due to the reasons provided above.

In addition, we measure the effectiveness of the considered adversarial evasion attacks through a derived metric denoted as *Attack Severity* (*AS*), which is computed as follows:

$$AS = 1 - \frac{DR(\text{after the attack})}{DR(\text{before the attack})} \tag{5}$$

This metric (which has been previously used also in [11,19]) allows for quickly determining if an attack family is effective or not by taking into account the different *Detection Rate* of the targeted detector before and after the submission of adversarial samples. It considers only the Detection Rate because our focus is on *evasion* attacks, which implies modifying *malicious* samples so that they are classified as *benign*. Higher (lower) values of *AS* denote attacks with higher (lower) amounts of evaded samples.

4. Experimental Results

Our experimental campaign has the twofold objective of: (i) showing that the proposed countermeasure is effective in mitigating evasion attacks; and (ii) showing that its integration has negligible impact in non-adversarial settings. To this purpose, we conduct our evaluation by following this outline:

1. determine the performance of the "baseline" detectors in non-adversarial settings;
2. assess the effectiveness of the considered evasion attacks against the "baseline" detectors;
3. measure the performance of the "hardened" detectors in non-adversarial settings;
4. gauge the impact of the considered evasion attacks against the "hardened" detectors.

We address and discuss each of these points in the following sections; and then provide some final considerations.

4.1. Baseline Performance in Non-Adversarial Settings

It is important that the considered detectors achieve performance compliant with real-world requirements: the crafted adversarial attacks must be effective on detectors that are ready for production environments. Thus, we train and test each detector with the procedure described in Section 3.3.2, and report the results in Table 7, which shows the average (and standard deviation) values of the chosen performance metrics computed across all botnet-specific instances of the detectors. The boxplot representation of these results is provided in Figure 5.

Table 7. Performance of the baseline detectors in non-adversarial settings.

Algorithm	*F1-Score* (std. dev.)	*Precision* (std. dev.)	*Recall* (std. dev.)
RF	0.9760 (0.0167)	0.9830 (0.0177)	0.9691 (0.0157)
MLP	0.9253 (0.0246)	0.9254 (0.0252)	0.9253 (0.0240)
DT	0.9648 (0.0144)	0.9689 (0.0186)	0.9606 (0.0117)
WnD	0.9322 (0.0594)	0.9469 (0.0671)	0.9180 (0.0533)
AB	0.9679 (0.0142)	0.9700 (0.0121)	0.9658 (0.0172)
Average	0.9538 (0.0258)	0.9588 (0.0281)	0.9478 (0.0244)

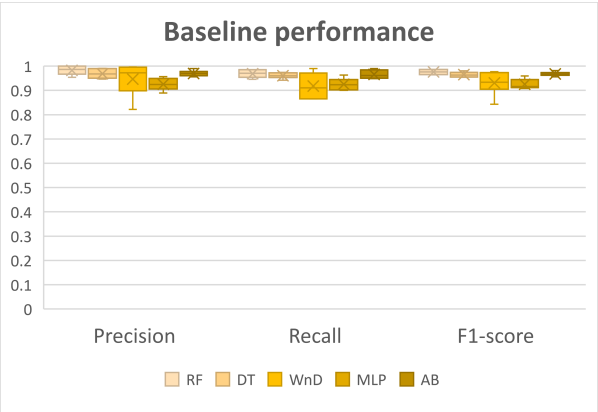

Figure 5. Performance of the baseline detectors.

We observe that all detectors achieve performance scores suitable for real-world environments [11], thus representing a valid baseline for the remaining evaluations.

4.2. Adversarial Samples against Baseline Detectors

The impact of the considered adversarial attacks against the baseline detectors is now evaluated. This step is critical because our goal is showing that our countermeasure is capable of mitigating attacks that are highly effective. Hence, we generate the adversarial samples with the procedure described in Section 3.3.4 and submit them to the baseline detectors. The results are displayed in Table 8, which reports the *Recall* obtained by all the detectors before and after the attack, alongside the *Attack Severity* metric; in particular, each cell denotes the average (and standard deviation) values achieved by all botnet-specific instances against all the generated adversarial samples; the boxplot diagrams of Table 8 are also presented in Figure 6.

Having established a solid baseline, we can now proceed to evaluate the quality of the proposed countermeasure.

Table 8. Performance of the baseline detectors under evasion attacks.

Algorithm	*Recall* (no-attack)	*Recall* (attack)	*Attack Severity*
RF	0.9691 (0.0157)	0.3142 (0.2236)	0.6760 (0.2284)
MLP	0.9253 (0.0240)	0.4623 (0.1832)	0.4979 (0.2048)
DT	0.9606 (0.0117)	0.3011 (0.2125)	0.6861 (0.2218)
WnD	0.9180 (0.0533)	0.4634 (0.1818)	0.4902 (0.2118)
AB	0.9658 (0.0172)	0.3228 (0.2221)	0.6637 (0.2345)
Average	0.9478 (0.0244)	0.3728 (0.2046)	0.6028 (0.2203)

We note that all detectors are significantly affected by our evasion attacks, which is particularly evident by comparing Figure 6a with Figure 6b. As an example, consider the results obtained by the Deep Learning algorithm (WnD): its *Detection Rate* goes from an appreciable 92% to a clearly unacceptable 46%.

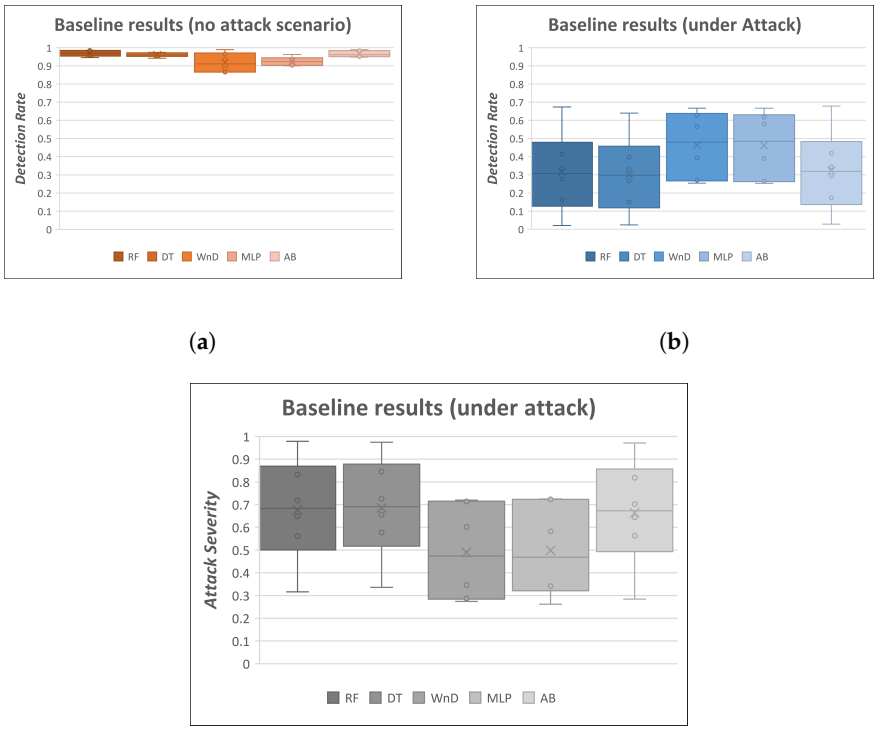

Figure 6. Effectiveness of the baseline detectors against the adversarial samples. (**a**) *Detection Rate* of the baseline detectors in non-adversarial settings; (**b**) *Detection Rate* of the baseline detectors in adversarial settings; (**c**) *Attack Severity* of the evasion attacks against the baseline detectors.

4.3. Hardened Performance in Non-Adversarial Settings

The first goal is determining if AppCon has a negative effect on the detectors when they are not subject to adversarial attacks. Thus, we devise all the application specific sub-instances of our detectors through the methodology explained in Section 3.3.3 and evaluate them on the same test dataset used

for the "baseline" detectors. The results are reported in Table 9 and their boxplot representation in Figure 7.

Table 9. Performance of the hardened detectors in non-adversarial settings.

Algorithm	*F1-Score* (std. dev.)	*Precision* (std. dev.)	*Recall* (std. dev.)
RF	0.9726 (0.0146)	0.9762 (0.0125)	0.9691 (0.0177)
MLP	0.9239 (0.0259)	0.9231 (0.0258)	0.9247 (0.0261)
DT	0.9633 (0.0128)	0.9669 (0.0156)	0.9597 (0.0108)
WnD	0.9323 (0.0590)	0.9469 (0.0669)	0.9180 (0.0528)
AB	0.9681 (0.0134)	0.9693 (0.0109)	0.9669 (0.0174)
Average	0.9520 (0.0251)	0.9565 (0.0263)	0.9477 (0.0249)

From Table 9 and Figure 7, it is evident that our solution has a negligible impact on the performance of the detectors in non-adversarial settings: indeed, its scores are very similar to the ones obtained by the baseline (see Table 7 and Figure 5). These results are critical because several existing countermeasures against adversarial attacks induce a reduced performance on samples that are not adversarially manipulated.

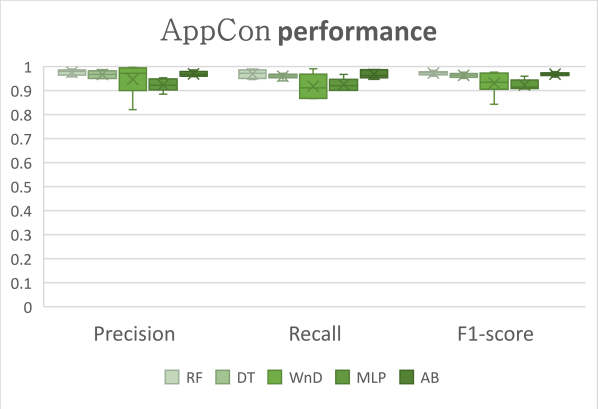

Figure 7. Performance of AppCon in non-adversarial scenarios.

As machine learning-based detectors require undergoing a training phase, we report in Table 10 the length (in minutes) of the training operations for the baseline and hardened detectors, which are performed on an Intel Core i5-4670 CPU with 16 GB of RAM, 512 GB SSD and Nvidia GTX1070 GPU. The implementation of AppCon is computationally more demanding to develop because of the additional layer of classifiers integrated in the detectors. However, these operations need to be executed only periodically.

Table 10. Training Times (in minutes) of the baseline and hardened detectors.

Detector	RF	MLP	DT	WnD	AB	avg
Baseline	5:34	9:58	4:12	16:15	5:58	8:23
Hardened	9:48	16:31	6:37	24:22	10:09	13:29

4.4. Countermeasure Effectiveness

Finally, we measure the effectiveness of the proposed solution at mitigating evasion attacks. Hence, we submit all the generated adversarial samples to the hardened detectors.

We first measure how many of the crafted adversarial samples are immediately blocked by AppCon. Indeed, it is safe to assume that all samples that do not conform to the network traffic generated by our set of applications are not able to evade detection as they are not classified as accepted traffic for any of the known applications. In the ideal case in which the set of applications A covers all the web applications used by the protected enterprise, these samples will be blocked. These results are outlined in Table 11, which displays the percentage of adversarial samples that are immediately blocked, alongside the amount of samples that will be forwarded to the hardened detectors.

Table 11. Samples blocked by AppCon.

Application	Blocked Samples	Forwarded Samples
Teams	77.84%	22.15%
OneDrive	61.79%	38.21%
OneNote	77.83%	22.16%
Skype	69.33%	30.66%
WhatsApp	90.21%	9.79%
Average	75.41%	24.59%

From this table, we appreciate that the simple integration of our method allows for blocking over 75% of the generated adversarial samples, which is a promising result.

Next, the performance of the hardened detectors on the remaining adversarial samples is reported in Table 12; as usual, this table is paired with its corresponding boxplots displayed in Figure 8.

Table 12. Performance of the hardened detectors on the forwarded adversarial samples.

Algorithm	*Recall* (no-attack)	*Recall* (attack)	*Attack Severity*
RF	0.9691 (0.0177)	0.4418 (0.2186)	0.5456 (0.2216)
MLP	0.9247 (0.0261)	0.5644 (0.1801)	0.3871 (0.2035)
DT	0.9597 (0.0108)	0.4435 (0.2128)	0.5376 (0.2222)
WnD	0.9180 (0.0528)	0.5893 (0.1830)	0.3533 (0.2150)
AB	0.9669 (0.0174)	0.4742 (0.2233)	0.5071 (0.2383)
Average	0.9477 (0.0249)	0.5026 (0.2036)	0.4659 (0.2201)

By comparing the values in Table 12 with those obtained by the baselines in Table 8, we observe that AppCon allows for devising detectors that are less affected by the considered evasion attacks. For example, let us inspect the performance of the RF classifier: its adversarial *Recall* goes from $\sim$0.31 to $\sim$0.44, which is an improvement of nearly 50%. We stress that the complete quality of our solution is shown through both Tables 11 and 12: in the considered scenario [11], AppCon can immediately prevent about 75% of adversarial samples, and it improves the *Detection Rate* of detectors based on different supervised ML algorithms on the remaining (about 25%) attack samples.

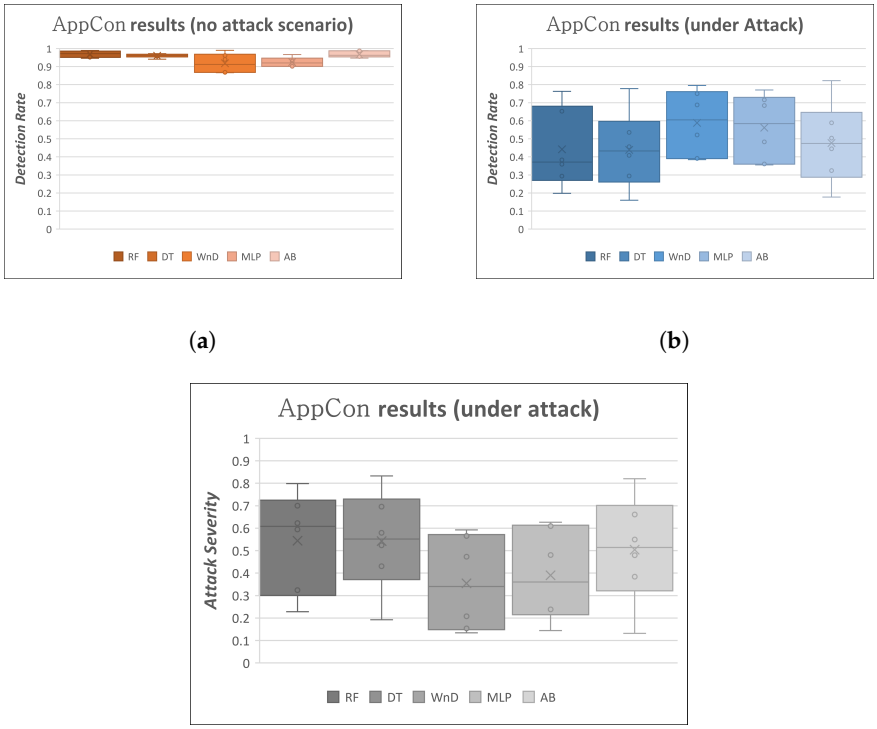

Figure 8. Effectiveness of AppCon against the adversarial samples. (**a**) *Detection Rate* of AppCon in non-adversarial settings; (**b**) *Detection Rate* of AppCon in adversarial settings; (**c**) *Attack Severity* of the evasion attacks against AppCon.

4.5. Considerations

We provide some considerations on the effectiveness of the proposed solution, with regard to (i) its improvement over existing defensive techniques; and to (ii) possible strategies that attackers may adopt to evade our system.

The authors of [44] propose a method to harden neural network classifiers, but the application of this approach to neural network-based malware detectors (described in [20]) increased the false positive rate in non-adversarial contexts by nearly 2%, which is not acceptable in modern Network Intrusion Detection scenarios in which NIDS analyze thousands of events every second [59]. Moreover, we highlight the study performed in [11], which considers a large array of ML-based botnet detectors tested also on the `CTU-13` dataset against adversarial samples: their results show that techniques such as *feature removal* may cause drops in *Precision* from 96% to a worrying 81%. Another possible defense against adversarial evasion attacks relies on re-training the algorithm on perturbed samples [21,47,74]: however, the authors in [47,74] do not evaluate the efficacy of such strategies in non-adversarial settings, while the detection improvements in [21] are very small (only 2%). In contrast to all these past efforts, the values of *Precision* and *F1-score* achieved by AppCon in non-adversarial settings only differ by less than 1% from our baseline, while significantly improving the performance in adversarial circumstances.

A skilled adversary may still be able to thwart AppCon. To do this, the following three Conditions must be met:

1. the attacker must know (fully or partially) the set of web-applications A considered by AppCon. Let us call this set $\overline{A}$.
2. the attacker must know the characteristics of the traffic that $\overline{A}$ generate in the targeted organization. We denote this piece of knowledge with $C_{\overline{A}}$.
3. the attacker must be able to modify its malicious botnet communications so as to conform to $C_{\overline{A}}$.

An attacker that meets all three of these conditions does not conform to the threat model considered in this paper and that is used to devise AppCon (see Section 3.1). Regardless, we stress that, while it may be feasible for a persistent attacker to learn $\overline{A}$ (Condition #1), obtaining $C_{\overline{A}}$ (Condition #2) would require far more effort as the attacker would need to gain access to systems that monitor the behavior of the entire network to acquire such information. Finally, concerning Condition #3, the attacker may be able to modify the malicious CnC communications to comply with $C_{\overline{A}}$, but this may raise alarms by other detection mechanisms [59]. We conclude by highlighting that, as evidenced by our experiments (see Table 11), AppCon allows protection against over 75% of the considered evasion attempts—regardless of the attacker's capabilities.

5. Conclusions

The application of machine learning algorithms to cybersecurity must face the problem posed by adversarial attacks. In this paper, we propose AppCon, a novel approach that aims to improve the resilience of cyber detectors against evasion attacks. Our solution is particularly suited to strengthen machine learning-based network intrusion detection systems deployed in realistic environments. The proposal combines the effectiveness of ensemble learning with the intuition that modern network environments generate traffic from a finite set of applications; the goal is limiting the options that an attacker can use to craft his malicious adversarial samples by tailoring the NIDS for the set of applications used in the protected network. We evaluate the quality of AppCon through an extensive experimental campaign in a botnet detection scenario. The results provide evidence that our solution achieves the symmetric quality of mitigating evasion attacks while not affecting the detection performance in non-adversarial settings, and that it is effective on multiple supervised ML algorithms. These improvements represent a meaningful step towards the development of more secure cyber detectors relying on machine learning. The present work presents margins for future improvements: an enticing idea consists of evaluating the synergy of the proposed AppCon approach with other defensive strategies, with the goal of further improving the detection rate against evasion attacks.

Author Contributions: Conceptualization: G.A. and G.R.; methodology, G.A., M.A., and G.R.; software, G.R. and V.G.C.; validation, G.A., M.M., and V.G.C.; formal analysis, G.A., M.M.; investigation, G.A. and G.R.; resources, M.M.; data curation, G.A. and G.R.; writing—original draft preparation, G.A.; writing—review and editing, G.A., M.M., V.G.C., and M.A.; visualization, G.A., V.G.C., and G.R.; supervision, M.A.; project administration, M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Gardiner, J.; Nagaraja, S. On the security of machine learning in malware c&c detection: A survey. *ACM Comput. Surv.* **2016**, *49*, 59.
2. Biggio, B.; Roli, F. Wild patterns: Ten years after the rise of adversarial machine learning. *Elsevier Pattern Recogn.* **2018**, *84*, 317–331. [CrossRef]
3. Buczak, A.L.; Guven, E. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 1153–1176. [CrossRef]
4. Huang, L.; Joseph, A.D.; Nelson, B.; Rubinstein, B.I.; Tygar, J. Adversarial machine learning. In Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, Chicago, IL, USA, 21 October 2011; pp. 43–58.
5. Biggio, B.; Nelson, B.; Laskov, P. Poisoning attacks against support vector machines. In Proceedings of the 29th International Coference on International Conference on Machine, Edinburgh, UK, 26 June–1 July 2012; pp. 1467–1474.

6. Biggio, B.; Corona, I.; Maiorca, D.; Nelson, B.; Šrndić, N.; Laskov, P.; Giacinto, G.; Roli, F. Evasion attacks against machine learning at test time. In Proceedings of the 2013th European Conference on Machine Learning and Knowledge Discovery in Databases, Prague, Czech Republic, 23–27 September 2013; pp. 387–402.
7. Papernot, N.; McDaniel, P.; Sinha, A.; Wellman, M. SoK: Security and Privacy in Machine Learning. In Proceedings of the 2018 IEEE European Symposium on Security and Privacy (EuroS&P), London, UK, 24–26 April 2018; pp. 399–414.
8. Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z.B.; Swami, A. The limitations of deep learning in adversarial settings. In Proceedings of the 2016 IEEE European Symposium on Security and Privacy (EuroS&P), Saarbrucken, Germany, 21–24 March 2016; pp. 372–387.
9. Su, J.; Vargas, D.V.; Sakurai, K. One pixel attack for fooling deep neural networks. *IEEE Trans. Evol. Comput.* **2019**. [CrossRef]
10. Wu, D.; Fang, B.; Wang, J.; Liu, Q.; Cui, X. Evading Machine Learning Botnet Detection Models via Deep Reinforcement Learning. In Proceedings of the 2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; pp. 1–6.
11. Apruzzese, G.; Colajanni, M.; Marchetti, M. Evaluating the effectiveness of Adversarial Attacks against Botnet Detectors. In Proceedings of the 2019 IEEE 18th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, USA, 26–28 September 2019; pp. 1–8.
12. Laskov, P. Practical evasion of a learning-based classifier: A case study. In Proceedings of the 2014 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 18–21 May 2014; pp. 197–211.
13. Demontis, A.; Russu, P.; Biggio, B.; Fumera, G.; Roli, F. On security and sparsity of linear classifiers for adversarial settings. In Proceedings of the Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), Mérida, Mexico, 29 November–2 December 2016; pp. 322–332.
14. Demontis, A.; Melis, M.; Biggio, B.; Maiorca, D.; Arp, D.; Rieck, K.; Corona, I.; Giacinto, G.; Roli, F. Yes, machine learning can be more secure! A case study on android malware detection. *IEEE Trans. Dependable Secur. Comput.* **2017**. [CrossRef]
15. Corona, I.; Biggio, B.; Contini, M.; Piras, L.; Corda, R.; Mereu, M.; Mureddu, G.; Ariu, D.; Roli, F. Deltaphish: Detecting phishing webpages in compromised websites. In Proceedings of the ESORICS 2017—22nd European Symposium on Research in Computer Security, Oslo, Norway, 11–15 September 2017; pp. 370–388.
16. Liang, B.; Su, M.; You, W.; Shi, W.; Yang, G. Cracking classifiers for evasion: A case study on the google's phishing pages filter. In Proceedings of the 25th International World Wide Web Conference (WWW 2016), Montréal, QC, Canada, 11–15 April 2016; pp. 345–356.
17. Muñoz-González, L.; Biggio, B.; Demontis, A.; Paudice, A.; Wongrassamee, V.; Lupu, E.C.; Roli, F. Towards poisoning of deep learning algorithms with back-gradient optimization. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 27–38.
18. Apruzzese, G.; Colajanni, M. Evading botnet detectors based on flows and Random Forest with adversarial samples. In Proceedings of the 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, USA, 1–3 November 2018; pp. 1–8.
19. Apruzzese, G.; Colajanni, M.; Ferretti, L.; Marchetti, M. Addressing Adversarial Attacks against Security Systems based on Machine Learning. In Proceedings of the 2019 11th International Conference on Cyber Conflict (CyCon), Tallinn, Estonia, 28–31 May 2019; pp. 1–18.
20. Grosse, K.; Papernot, N.; Manoharan, P.; Backes, M.; McDaniel, P. Adversarial perturbations against deep neural networks for malware classification. *arXiv* **2016**, arXiv:1606.04435 .
21. Calzavara, S.; Lucchese, C.; Tolomei, G. Adversarial Training of Gradient-Boosted Decision Trees. In Proceedings of the 28th ACM Inter-national Conference on Information and Knowledge Management (CIKM'19), Beijing, China, 3–7 November 2019; pp. 2429–2432.
22. Biggio, B.; Corona, I.; He, Z.M.; Chan, P.P.; Giacinto, G.; Yeung, D.S.; Roli, F. One-and-a-half-class multiple classifier systems for secure learning against evasion attacks at test time. In Proceedings of the 12th International Workshop, MCS 2015, Günzburg, Germany, 29 June–1 July 2015; pp. 168–180.
23. Kettani, H.; Wainwright, P. On the Top Threats to Cyber Systems. In Proceedings of the 2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT), Kahului, HI, USA, 14–17 March 2019; pp. 175–179.

24. Garcia, S.; Grill, M.; Stiborek, J.; Zunino, A. An empirical comparison of botnet detection methods. *Elsevier Comput. Secur.* **2014**, *45*, 100–123. [CrossRef]
25. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436. [CrossRef]
26. Jordan, M.I.; Mitchell, T.M. Machine Learning: Trends, Perspectives, and Prospects. *Science* **2015**, *349*, 255–260. [CrossRef]
27. Truong, T.C.; Diep, Q.B.; Zelinka, I. Artificial Intelligence in the Cyber Domain: Offense and Defense. *Symmetry* **2020**, *12*, 410. [CrossRef]
28. Apruzzese, G.; Colajanni, M.; Ferretti, L.; Guido, A.; Marchetti, M. On the Effectiveness of Machine and Deep Learning for Cybersecurity. In Proceedings of the 2018 10th International Conference on Cyber Conflict (CyCon), Tallinn, Estonia, 29 May–1 June 2018; pp. 371–390.
29. Xu, R.; Cheng, J.; Wang, F.; Tang, X.; Xu, J. A DRDoS detection and defense method based on deep forest in the big data environment. *Symmetry* **2019**, *11*, 78. [CrossRef]
30. Yavanoglu, O.; Aydos, M. A review on cyber security datasets for machine learning algorithms. In Proceedings of the 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 11–14 December 2017; pp. 2186–2193.
31. Cheng, H.T.; Koc, L.; Harmsen, J.; Shaked, T.; Chandra, T.; Aradhye, H.; Anderson, G.; Corrado, G.; Chai, W.; Ispir, M.; et al. Wide & deep learning for recommender systems. In Proceedings of the Workshop on Deep Learning for Recommender Systems, Boston, MA, USA, 15–19 September 2016; pp. 7–10.
32. Blanzieri, E.; Bryl, A. A survey of learning-based techniques of email spam filtering. *Artif. Intell. Rev.* **2008**, *29*, 63–92. [CrossRef]
33. Sommer, R.; Paxson, V. Outside the closed world: On using machine learning for network intrusion detection. In Proceedings of the 2010 IEEE Symposium on Security and Privacy, Berkeley/Oakland, CA, USA, 16–19 May 2010; pp. 305–316.
34. Alazab, M.; Venkatraman, S.; Watters, P.; Alazab, M. Zero-day malware detection based on supervised learning algorithms of API call signatures. In Proceedings of the Ninth Australasian Data Mining Conference, December 2011, Ballarat, Australia, 1–2 December 2011; Volume 121, pp. 171–182.
35. Mannino, M.; Yang, Y.; Ryu, Y. Classification algorithm sensitivity to training data with non representative attribute noise. *Elsevier Decis. Support Syst.* **2009**, *46*, 743–751. [CrossRef]
36. Witten, I.H.; Frank, E.; Hall, M.A.; Pal, C.J. *Data Mining: Practical Machine Learning Tools and Techniques*; Morgan Kaufmann: Burlington, MA, USA, 2016.
37. Dalvi, N.; Domingos, P.; Sanghai, S.; Verma, D. Adversarial classification. In Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, WA, USA, 22–25 August 2004; pp. 99–108.
38. Lowd, D.; Meek, C. Adversarial learning. In Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, IL, USA, 21–24 August 2005; pp. 641–647.
39. Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; Fergus, R. Intriguing properties of neural networks. *arXiv* **2013**, arXiv:1312.6199 .
40. Gama, J.; Žliobaitė, I.; Bifet, A.; Pechenizkiy, M.; Bouchachia, A. A survey on concept drift adaptation. *ACM Comput. Surv.* **2014**, *46*, 44. [CrossRef]
41. Kantchelian, A.; Afroz, S.; Huang, L.; Islam, A.C.; Miller, B.; Tschantz, M.C.; Greenstadt, R.; Joseph, A.D.; Tygar, J. Approaches to adversarial drift. In Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security, Berlin, Germany, 4 November 2013; pp. 99–110.
42. Xu, W.; Qi, Y.; Evans, D. Automatically evading classifiers. In Proceedings of the Network and Distributed Systems Symposium 2016, San Diego, CA, USA, 21–24 February 2016; pp. 21–24.
43. Ibitoye, O.; Shafiq, O.; Matrawy, A. Analyzing adversarial attacks against deep learning for intrusion detection in IoT networks. In Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA, 9–13 December 2019.
44. Papernot, N.; McDaniel, P.; Wu, X.; Jha, S.; Swami, A. Distillation as a defense to adversarial perturbations against deep neural networks. In Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2016; pp. 582–597.
45. Fajana, O.; Owenson, G.; Cocea, M. TorBot Stalker: Detecting Tor Botnets through Intelligent Circuit Data Analysis. In Proceedings of the 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, USA, 1–3 November 2018; pp. 1–8.

46. Carlini, N.; Wagner, D. Towards evaluating the robustness of neural networks. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2017; pp. 39–57.
47. Anderson, H.S.; Woodbridge, J.; Filar, B. DeepDGA: Adversarially-tuned domain generation and detection. In Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security, Vienna, Austria, 28 October 2016; pp. 13–21.
48. Kantchelian, A.; Tygar, J.D.; Joseph, A. Evasion and hardening of tree ensemble classifiers. In Proceedings of the 33rd International Conference on Machine Learning (ICML), New York, NY, USA, 19–24 June 2016; pp. 2387–2396.
49. Zhang, F.; Chan, P.P.; Biggio, B.; Yeung, D.S.; Roli, F. Adversarial feature selection against evasion attacks. *IEEE Trans Cybern.* **2016**, *46*, 766–777. [CrossRef]
50. Gourdeau, P.; Kanade, V.; Kwiatkowska, M.; Worrell, J. On the hardness of robust classification. In Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, BC, Canada, 8–14 December 2019; pp. 7444–7453.
51. Do, C.T.; Tran, N.H.; Hong, C.; Kamhoua, C.A.; Kwiat, K.A.; Blasch, E.; Ren, S.; Pissinou, N.; Iyengar, S.S. Game theory for cyber security and privacy. *ACM Comput. Surv.* **2017**, *50*, 30. [CrossRef]
52. Wooldridge, M. Does game theory work? *IEEE Intell. Syst.* **2012**, *27*, 76–80. [CrossRef]
53. Pavlovic, D. Gaming security by obscurity. In Proceedings of the 2011 New Security Paradigms Workshop, Marin County, CA, USA, 12–15 September 2011; pp. 125–140.
54. Cisco IOS NetFlow. Available online: https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/ (accessed on 14 April 2020).
55. Xiang, C.; Binxing, F.; Lihua, Y.; Xiaoyi, L.; Tianning, Z. Andbot: Towards advanced mobile botnets. In Proceedings of the 4th USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET '11, Boston, MA, USA, 29 March 2011; p. 11.
56. Marchetti, M.; Pierazzi, F.; Guido, A.; Colajanni, M. Countering Advanced Persistent Threats through security intelligence and big data analytics. In Proceedings of the 2016 8th International Conference on Cyber Conflict (CyCon), Tallinn, Estonia, 31 May–3 June 2016; pp. 243–261.
57. Bridges, R.A.; Glass-Vanderlan, T.R.; Iannacone, M.D.; Vincent, M.S.; Chen, Q.G. A Survey of Intrusion Detection Systems Leveraging Host Data. *ACM Comput. Surv.* **2019**, *52*, 128. [CrossRef]
58. Berman, D.S.; Buczak, A.L.; Chavis, J.S.; Corbett, C.L. A survey of deep learning methods for cyber security. *Information* **2019**, *10*, 122. [CrossRef]
59. Pierazzi, F.; Apruzzese, G.; Colajanni, M.; Guido, A.; Marchetti, M. Scalable architecture for online prioritisation of cyber threats. In Proceedings of the 2017 9th International Conference on Cyber Conflict (CyCon), Tallinn, Estonia, 30 May–2 June 2017; pp. 1–18.
60. OpenArgus. Available online: https://qosient.com/argus/ (accessed on 14 April 2020).
61. Stevanovic, M.; Pedersen, J.M. An analysis of network traffic classification for botnet detection. In Proceedings of the 2015 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), London, UK, 8–9 June 2015; pp. 1–8.
62. Abraham, B.; Mandya, A.; Bapat, R.; Alali, F.; Brown, D.E.; Veeraraghavan, M. A Comparison of Machine Learning Approaches to Detect Botnet Traffic. In Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 8–13 July 2018; pp. 1–8.
63. Alejandre, F.V.; Cortés, N.C.; Anaya, E.A. Feature selection to detect botnets using machine learning algorithms. In Proceedings of the 2017 International Conference on Electronics, Communications and Computers (CONIELECOMP), Cholula, Mexico, 22–24 February 2017; pp. 1–7.
64. Pektaş, A.; Acarman, T. Deep learning to detect botnet via network flow summaries. *Neural Comput. Appl.* **2019**, *31*, 8021–8033. [CrossRef]
65. Stevanovic, M.; Pedersen, J.M. An efficient flow-based botnet detection using supervised machine learning. In Proceedings of the 2014 International Conference on Computing, Networking and Communications (ICNC), Honolulu, HI, USA, 3–6 February 2014; pp. 797–801.
66. WhatsApp Customer Stories. 2019. Available online: https://www.whatsapp.com/business/customer-stories (accessed on 14 April 2020).
67. OneDrive Customer Stories. 2019. Available online: https://products.office.com/en-us/onedrive-for-business/customer-stories (accessed on 14 April 2020).

68. General Electric Uses Teams. 2019. Available online: https://products.office.com/en-us/business/customer-stories/726925-general-electric-microsoft-teams (accessed on 14 April 2020).
69. OneNote Testimonials. 2019. Available online: https://products.office.com/en/business/office-365-customer-stories-office-testimonials (accessed on 14 April 2020).
70. nProbe: An Extensible NetFlow v5/v9/IPFIX Probe for IPv4/v6. 2015. Available online: http://www.ntop.org/products/netflow/nprobe/ (accessed on 14 April 2020).
71. Apruzzese, G.; Pierazzi, F.; Colajanni, M.; Marchetti, M. Detection and Threat Prioritization of Pivoting Attacks in Large Networks. *IEEE Trans. Emerg. Top. Comput.* **2017**. [CrossRef]
72. Apruzzese, G.; Andreolini, M.; Colajanni, M.; Marchetti, M. Hardening Random Forest Cyber Detectors against Adversarial Attacks. *IEEE Trans. Emerg. Top. Comput. Intell.* **2019**. [CrossRef]
73. Xin, Y.; Kong, L.; Liu, Z.; Chen, Y.; Li, Y.; Zhu, H.; Gao, M.; Hou, H.; Wang, C. Machine learning and deep learning methods for cybersecurity. *IEEE Access* **2018**, *6*, 35365–35381. [CrossRef]
74. Usama, M.; Asim, M.; Latif, S.; Qadir, J. Generative Adversarial Networks for Launching and Thwarting Adversarial Attacks on Network Intrusion Detection Systems. In Proceedings of the 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC), Tangier, Morocco, 24–28 June 2019; pp. 78–83.