Netspot: a simple Intrusion Detection System with statistical learning

Alban Siffer
Amossys, Univ. Rennes, CNRS, IRISA
alban.siffer@amossys.fr

Pierre-Alain Fouque
Univ. Rennes, CNRS, IRISA
pierre-alain.fouque@inria.fr

Alexandre Termier
Univ. Rennes, Inria, CNRS, IRISA
alexandre.termier@irisa.fr

Christine Largouet
AgroCampus Ouest, Inria, CNRS, IRISA
christine.largouet@irisa.fr

***Abstract*—Machine learning is nowadays increasingly used in cyber-security. While intrusion detection was mainly based on human expertise in the 1990s, learning models to predict attacks are now built from data. However, a large part of the developed learning algorithms hitherto has missed real-world issues, making them unpractical. Indeed, many supervised algorithms described in the literature have been trained and tuned only on the KDD99 dataset. Besides, these algorithms are often static and are unable to automatically adapt for detecting attacks depending on the network traffic. Consequently, we are far from detecting zero-day or more general Advanced Persistent Threats (APT) since only pre-registered and well-characterized attacks can be catched. Some recent systems use unsupervised ML algorithms, but the resulting tools are overly complex: many ML components are stacked with various tuning parameters, usually making the results hard to interpret. And finally, a strong ML/DM expertise is required to set up these systems on real networks.**

We present `netspot`, a very simple network intrusion detection system (NIDS) powered by SPOT, a recent streaming statistical anomaly detector. This statistical test uses Extreme Value Theory, which is a powerful method for detecting anomalies. Unlike all the previous works, it is not an end-to-end solution aimed to detect all cyber-attacks with packet resolution. It is rather a module providing a behavioral information which can be integrated in a more general monitoring system. `netspot` is simple: it has few (simple) parameters, it adapts along time to the monitored network and it is as fast as current rule-based methods. But most importantly, it is able to detect real-world cyber-attacks, making it a credible practical anomaly-based NIDS.

***Index Terms*—Network security; Intrusion detection systems; Statistical Learning.**

I. Introduction

Intrusion detection systems (IDS) are based on the following approaches: they use either static rules or they search for anomalies. Today, rule-based solutions are very efficient, more mature, and are the most widely deployed IDS (like `Snort`[1], `Zeek`[2] or `Suricata`[3]). They however require for each attack to find a signature that can be easily detected by analyzing network packets. Such approach has several weaknesses: building signatures is lengthy and laborious, the payload is likely to be inspected although it is an expensive task which will soon become deprecated with the wide use of end-to-end encryption and eventually, these rule-based methods are static thus new attacks cannot be detected.

Anomaly-based techniques aim at building a model of *normality* (in a supervised or unsupervised way) in order to discriminate abnormal observations (often considered as attacks). Such kind of *behavioral detection*, often using machine learning techniques, is often disregarded by security experts. The first reason explaining why AI failed in cyber-security while it revolutionizes other fields is the context specificity. Solving the GO game, recognizing human faces or making a car autonomous are real feats but detecting zero-day attacks seems far harder. Designing algorithms to detect attacks meets the pitfalls of:

1) All the networks are different, so adaptable techniques are paramount;
2) To detect new attacks, algorithms must not be stuck to training observations, so supervised methods are not suitable;
3) As the context is constantly evolving, dynamic models are preferred;
4) Online detection is required to prevent systems from being damaged as soon as possible.

Much work has been done to develop anomaly detectors with the help of cutting edge algorithms, and smarter IDS have been proposed [1]. However, current approaches aim to solve all the intrusion detection issues by providing a single complex end-to-end solution. This is maybe a too ambitious task in the light of the latest real progresses. Moreover, this search for complexity leads to impractical detection pipelines: many modules must generally be interfaced, multiplying the number of parameters to set, and making difficult general understanding as well as integration in different systems.

Related work on unsupervised anomaly-based IDS. Here we present two anomaly-based IDS which represent, up to our knowledge, the best unsupervised IDSs. These approaches differ from other works insofar as their goal is to build real-world IDS (unsupervised algorithm, real-world experiments). Even if these works mark out the real progress in this field, the proposed algorithms still have some drawbacks.

UNADA [2] performs anomaly detection in network flows

[1]https://www.snort.org/
[2]Formerly known as Bro, https://www.zeek.org/
[3]https://suricata-ids.org/

through aggregated statistics monitoring and sub-space clustering. Initial data are basic network flows (like Cisco NetFlow[4]). For all the incoming flows, UNADA computes statistics (batch aggregation) at different network levels (source and destination with masks from /8 to /32). All these statistics are monitored along time (multiple time series) and an algorithm $\mathcal{F}$ detects some changes. When a change occurs, the initial flows are re-aggregated at IP level only (source and destination), building then a matrix X which contains n observations (number of different IP) of p features (aggregated statistics). Clustering algorithms (DBSCAN [3]) are applied in all 2-dimensional sub-spaces. Each of them computes a score for all the observations. For all the observations, the sub-spaces clustering scores are weighted to finally output a final score. Eventually, a threshold is used to decide which observations are really abnormal among the initial data slots. From the ML/DM point of view, the whole approach is both logical and cumbersome. Every steps are coherent but the pipeline requests many operations: aggregations at many levels, anomaly detection for all the features for each aggregation level and sub-spaces clustering. Its implementation is then not easy as it needs to interface several different modules. Besides some points are not clarified: what change detection algorithm $\mathcal{F}$ to use? Such an algorithm is also very likely to request parameters. What threshold to use in the end of the pipeline? A static threshold can suffer from a lack of adaptability. Authors suggest to use an *elbow* method on the scores but the way to compute such a threshold is not given. The parameter values of DBSCAN also needs to be tuned to avoid incoherent results.

Kitsune [1] is a more recent network intrusion detection system (NIDS). It maxinly uses auto-encoders (AE) to perform anomaly detection, which are a type of artificial neural network used to learn efficient data codings in an unsupervised manner. Their goal is to learn a representation for a dataset, for dimension reduction, by training network and removing noise. The first phase of autoencoders consists of neural network to compress the signal into a small number of neurons (encoding), and the second phase attempts to reconstruct the instance features and compute the reconstruction error in terms of root mean squared errors (RMSE). In Kitsune, the RMSEs are forwarded to an output autoencoder, which acts as a non-linear voting mechanism for the whole system. Autoencoders need to be trained before any execution. The first issue is that data are assumed as normal during the training phase, which is a very strong assumption in the cyber-security. In practice, a clean network cannot be ensured. This is obviously a key assumption in the Kitsune design and there is no clear idea about its behavior in case of data contamination during the training step. Furthermore, unlike UNADA, Kitsune learns once and then executes without updating its knowledge. Actually, once the AE are learned, the feature mapper cannot change otherwise AE must be re-trained (with normal data). Kitsune is static and that may be a real problem if a constantly evolving traffic is monitored. Auto-encoders are very powerful but they also suffers from their lack of interpretability. When a packet is declared as abnormal, it is hard to understand the reason. The pipeline can also be difficult to understand: using the RMSE of initial auto-encoders to feed another AE is rather uncommon. The goal of the authors was to avoid a single "high" auto-encoder which is expensive to train. Besides, some of their best results have been made with one hidden neuron meaning that an AE is used to monitor every feature. In this context, all these AE basically contain 3 neurons: one input, one hidden and one output. We can naturally question the relevance of this architecture. Moreover, like UNADA, many choices are let to the users which does not ease its setup: statistics to compute, values of the decay factor, value of the AE compression rate, the maximum height of an AE, and the decision threshold. These two examples show that modern IDS are actually really difficult to set up on real networks. `netspot` relies on an opposite approach with a simple pipeline.

[4]https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html

Contributions. In this paper, we take an opposite approach to anomaly detection and show that "simple can be beautiful" for the detection of network attacks. We propose `netspot`, a novel network IDS built on top of the powerful anomaly detector SPOT [4]. SPOT is a recent statistical anomaly detector that brings several key properties for network monitoring. First, it has only one important parameter to set, which is easy to understand and can be directly correlated to the number of alarms raised per day. Second, it does not make any assumption on the underlying distribution of the data and can handle concept-drift, making it well suited for unknown traffic. And last, it is computationally efficient and can handle high throughput data streams. `netspot` computes statistics over the network traffic information, and uses SPOT to quickly and flexibly detect anomalies in these statistics. This NIDS works at the network access, internet and transport layers of the communication model. This is an important feature to use such information in order to take into account increasing end-to-end encrypted communications as 80% of the web traffic is nowadays encrypted [5]. Experiments on real-world network datasets with realistic attacks show that `netspot` detects quickly and accurately attacks, while being able to reach a higher throughput than state-of-the-art approaches. In particular, we reuse the dataset provided by [1] and we are able to detect all the attacks present in the Kitsune dataset. There are 4 types of attacks (Botnet Malware with Mirai), Denial of Service (SSDP Flood, SYN DoS and SSL Renegotiation), Man in the Middle (Video Injection, ARP MitM and Active Wiretap), and Reconnaissance (OS Scan and Fuzzing) [1] table 3 page 11. We present here one in each category with graphs, but they are all detected.

`netspot` is thus relevant as an early warning component of a larger Security Information Management (SIM) solution, which can be easily deployed, can quickly detect suspicious behaviours, and let more complex tools to pursue the analysis on the found suspicious traffic.

Organization of the paper. The SPOT algorithm, which is the core of `netspot` is introduced in section II. In section III, the architecture of `netspot` is explained and in the last section, we present some real-world experiments and compare this tool with other systems.

II. The SPOT algorithm

We introduce the anomality detector SPOT which is the statistical test on which relies our IDS tool.

A. Motivation

The common point of the vast majority of anomaly detectors is that they rely on scores. Every algorithm computes a normality or abnormality score s_i for all the observations x_i. But the whole anomaly detection process is made in two parts: *scoring* and *thresholding*.

The scoring part is generally the most interesting operation and the scores must describe the "outlierness" of an observation. Then a threshold z is often used in back-end to make the final decision (thresholding part): if $s_i > z$ then x_i is flagged as an anomaly otherwise it is considered as normal (s_i can be seen as abnormality score). There is little interest in the thresholding stage. In many works only the scoring part is detailled: the ROC curves and AUC are used to assess the performance of the anomaly detector [6]–[8]. When the whole detector is presented, decision thresholds are usually set after a fine-tuning step to get the best results on some datasets but either no formal procedure exists for setting the thresholds or it may require some labels to set it in practice [9].

This gap may lead to an obvious practical issue, that can be notably observed on the two aforementioned IDS: given some data and a scoring algorithm, how to set the decision threshold? Such a problem particularly arises when the context changes (other data sources, concept drift etc.). Another problem is the lack of interpretability: scores have generally no meaning (except that they smartly rank the observations) so the final threshold-based decision is likely to be hard to explain. The SPOT algorithm [4] described below tackles these issues by computing and updating a meaningful threshold (quantile) over streaming data.

B. SPOT

SPOT [4] is a streaming univariate anomaly detector based on Extreme Value Theory (EVT). As a statistical method, its final task is to build a decision threshold z_q which is actually a quantile. It computes z_q such that $\mathbb{P}(X > z_q) = q$ where X represents the monitored data and q is the main user-defined parameter. In practice q is very low (10^{-5} for instance), meaning that observing X higher than z_q is very improbable, so abnormal.

If we know the distribution of X, computing a quantile is rather easy. The main strength of SPOT is its ability to work almost on any stream without distribution assumption. This is possible thanks to EVT, which builds a local model on the tails on the distribution of X. Extreme Value Theory is built upon the Fisher-Tippett-Gnedenko theorem [10], [11] which says that the distribution of the maximum of n iid random variables converges in distribution to a certain class of distributions. It is very analogous to the central limit theorem which provides such a result for the mean.

In practice, it uses the Pickands-Balkema-de Haan theorem [12], [13] which gives a model for data above a "high" threshold:

$$\mathbb{P}\left(X - t > y \mid X > t\right) \underset{t \to \tau}{\simeq} \left(1 + \frac{\gamma\, y}{\sigma}\right)^{-\frac{1}{\gamma}},$$

where τ represents the limit of the support of X (it could be $+\infty$). This result merely means that the tail of the distribution of X can be approximated by a Generalized Pareto Distribution (GPD) with parameters $\gamma \in \mathbb{R}, \sigma > 0$ to estimated (see figure 1).

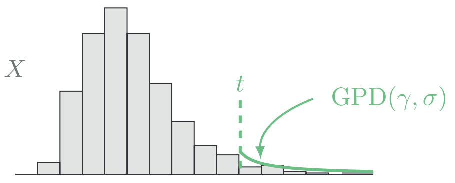

Fig. 1. Model for the distribution tail

Based on a sample $X_1, \ldots X_n$ a maximum likelihood estimation is performed in [4]. Once these parameters are estimated, the desired quantile z_q can be computed as follows:

$$z_q = t + \frac{\sigma}{\gamma}\left(\left(\frac{q\, n}{N_t}\right)^{-\gamma} - 1\right), \tag{1}$$

where q is the probability $\mathbb{P}(X > z_q)$ set by the user and N_t is the number of observations above t. The power of this method is the ability to compute z_q accurately with q as low as desired, i.e. without observations in the expected region of z_q.

How does SPOT adapt this method to streams? The figure 2 describes the stages: it takes an initial batch of data, selects the value of t, performs a GPD fit with the data above t and then computes z_q with equation (1). This threshold z_q is then used to flag anomalies (data above z_q) in the stream. The model is updated when data between t and z_q are observed (*normal* data in the tail).

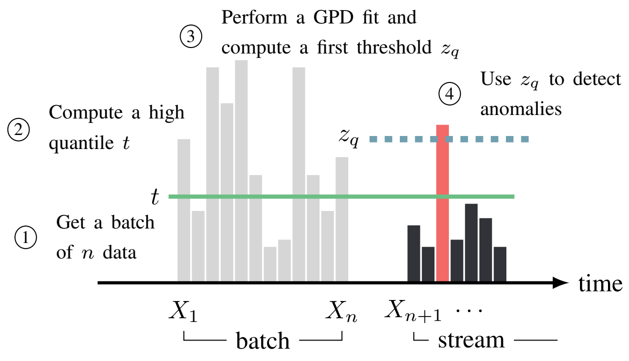
③ Perform a GPD fit and compute a first threshold z_q
② Compute a high quantile t
④ Use z_q to detect anomalies
z_q
t
① Get a batch of n data
time
X_1
X_n
X_{n+1} ···
batch
stream

Fig. 2. Anomaly detection overview (calibration)

C. Parameters and algorithm

We should naturally discuss about the parameters of SPOT. The most important parameter is q which estimates what is an abnormal event using probability. Given this probabilistic meaning, it should be set according to the user needs. In the cyber-security context one can imagine analyzing $N = 1000000$ events a day. Setting $q = 5.10^{-5}$ should flag around $q \times N = 50$ anomalies per day, which is quite reasonable for a security analyst to check. SPOT has two other parameters t and n that are less important in practice once some simple rules are given. The greater t is (the closer to τ), the more faithful the approximation is. Thus t should be taken as high as possible. However, as several observations are needed to estimate γ, σ (to reduce variance), t should not be too high. In practice, we do not set t but we use a high quantile $p_t = \mathbb{P}(X > t) = 98\%$ or 99% to compute it from the first n observations (e.g. 1000 or 2000).

This algorithm deals with stationary streams but it can be adapted to drifting ones with the computation of a local model (a moving average for example): DSPOT algorithm [4]. In this case, SPOT merely flags peaks relative to the current model and this feature requires one additional parameter d (the model depth). As SPOT corresponds to the case $d = 0$ so we will use the term SPOT to denote either SPOT or DSPOT.

III. Architecture of Netspot

We present a new NIDS we developed around the SPOT algorithm, called `netspot`. Unlike previous approaches which provide complex end-to-end solutions, we opt for a simple design which can be integrated in a more general detection system. Our goal was to build a simple IDS with a statistical learning core (SPOT) able to provide a relevant behavioral information about the network traffic. `netspot` does not aim to replace rule-based IDS but to propose a credible anomaly-based IDS, which by design tries to fill in the gaps of the legacy methods.

`netspot` is implemented in `Go`, for performance and concurrency reasons. Keeping in mind the practical stakes, we made `netspot` available on Github[5]. In addition to the source code, it is distributed through `debian` packages and `docker` images Here we use the latest available version (1.3.1) but a new version with better performances in under active development (2.0a).

Strategy. Network anomaly detection is a hard task since the outliers are not clearly known. It means that there is not an obvious framework to catch all of them. The first part is the choice of the features of interest. One may naturally try to feed algorithm with hundreds of them to widen the detection range but it implies to use complex algorithms able to tackle high-dimensional data (like neural networks). Their training cost (data and time) is generally too high to embed them on live detection solutions.

[5]https://github.com/asiffer/netspot

One solution would be to aggregate these features by computing a distance in the feature space for instance. Unfortunately, this kind of computation really suffers from the curse of dimensionality. By considering a dimension reduction technique one may struggle against this issue. Henceforth, the main problem is that all the interpretability is lost: when an anomaly occurs, the latter can only be described in the reduced feature space, making it inexplicable from the operator's point of view. It is undoubtedly the first criticism the security experts make towards ML/DM techniques.

Thus, one reasonable strategy is then to use a small number of features and avoid too complex operations between them. The choice of the features should also be led by the network security expertise to cover some categories of attacks.

One critical part of the anomaly detection pipeline is the thresholding part. Given some time series, we can easily find a constant threshold able to discriminate what would look like an anomaly. Now, let us consider the monitoring of several statistics along time: we must manually define a threshold for each of them. Actually, we also know that these statistics are likely to drift, so the normal/abnormal frontier. Hard-coded thresholds are definitely not a maintainable solution. Why not using a $\mu \pm 3\sigma$ rule? In many case, it is a reasonable solution however these thresholds have a meaning only if the monitored statistics are gaussian distributed (quantile at 95%). Furthermore such a choice can become totally inefficient (too many or too few alerts raised) once the real distribution is not close enough to the normal one.

Eventually, our strategy is the following: using the SPOT algorithm to automatically compute/update a meaningful threshold (quantile) on several expert-defined network statistics. It clearly circumvents all the issues mentioned above. Obviously it would be nice to extend SPOT to multivariate distributions since cyber attacks are usually described using 2 or 3 features as most attacks can be identified by a combination of a small number of alarms [2]. However, this is a well-known open problems in statistics which is very difficult. Consequently, we rely on the more simple task of identifying these alarms individually. Actually, we will see that this choice is sufficient to detect real-world cyber-attacks.

Design. In a nutshell, `netspot` monitors network statistics with the SPOT algorithm. At the basis, `netspot` increments some network counters once a packet is received. This operation is light as the counters mainly perform "atomic" operations (e.g. looking at flag, retrieving an IP address, computing the packet size etc.) This part of the program is called the MINER and uses the `GoPacket`[6] library, wrapping around the `libpcap`[7] library, to parse network packets. This also allows to process packets from network capture files (e.g. `.pcap`) and network interfaces without any distinction.

Figure 3 illustrates the MINER's task. When a packet is incoming, the MINER extracts some layers if they exist: Ethernet, IP, ICMP, TCP and UDP. Every layer will be useful only for

[6]https://github.com/google/gopacket
[7]https://www.tcpdump.org/

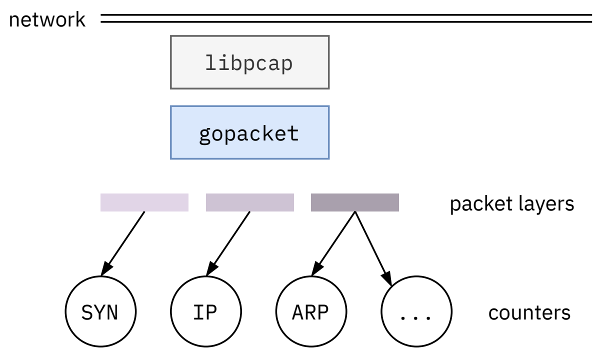

Fig. 3. The MINER component. It parses network packets, extracting layers and dispatching them so as to increment the counters

specific counters, so the MINER dispatches them according to the counter needs. For example, the `IP` counter can only receive IP layers and it increments an internal accumulator every time it receives such a layer. But the `SYN` counter, which accepts only TCP layers, increments itself once the received TCP layer has a SYN flag set to 1.

Above the MINER, we built the ANALYZER which manages the network statistics. The statistics merely need counters to be computed. For instance, the `R_SYN` statistic (ratio of SYN packets) divides the `SYN` counter by the `IP` counter. At given intervals (e.g. every 5 seconds), the ANALYZER asks the MINER the counter values and computes the statistics. This statistic corresponds then to the ratio of SYN packets during the last 5 seconds (the ANALYZER also asks the MINER to reset its counters).

The statistics are not as basic as the counters (i.e. elements making a single computation) because they embed a SPOT instance (or more generally a DSPOT instance) which monitors what they compute in real time. Finally the ANALYZER has three outputs: the raw statistics, the SPOT thresholds and some alerts (see figure 4).

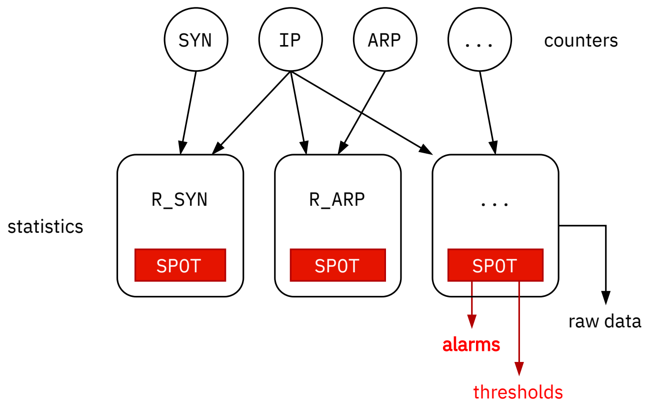

Fig. 4. The ANALYZER: statistics request periodically counter values, compute stats and feed them to an embedded SPOT instance, which can trigger alerts

Command & Control. `netspot` runs as a server (through a `systemd` service or a `docker` container) and can be managed by clients. The server can be configured through a simple file or through command line arguments. In particular, some statistics can be loaded and their SPOT instances can be individually parameterized.

We also developed a `Go` client (command line interface), called `netspotctl`, which currently uses the RPC facilities provided by `Go` to interact with the server. However, `netspot` also exposes a REST API, with an OpenAPI[8] specification, making the implementation of other HTTP clients rather easy and even “automatable”.

Integration. Currently, `netspot` outputs three JSON files, described in figure 4: the raw statistics, the thresholds computed by SPOT (one for each statistic) and the alarms (when a value is beyond a threshold). However `netspot` is also able to send these information to an `influxdb`[9] database, which is “a time series database designed to handle high write and query loads”. Such a storage eases the information aggregation. For instance Security Information and Event Management softwares (SIEM) are likely to handle real-time data from multiple heterogeneous sources. Using such a technology to store events makes `netspot` easier to integrate in a general monitoring system. At a first level, one can use a visualization technology upon `influxdb`, like `Grafana`[10], to build a dashboard showing the `netspot` events (see figure 5).

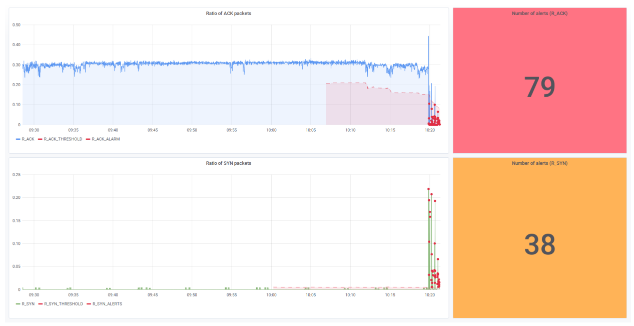

Fig. 5. Example of a `Grafana` dashboard to visualize the `netspot` output

Modularity. `netspot` can also be adapted to specific needs. Indeed, the main abstractions behind `netspot` are the counters and the statistics. A developer can easily implement its own counters and then its own statistics. Thus, one may imagine the need to monitor the activity of some particular IP addresses or sub-networks (like a server or the external network). In this example, new counters and new statistics (written in `Go`) should be added to the sources and `netspot` must be re-compiled. Especially, we provide a simple process for the developer to enrich `netspot` at his convenience.

IV. EXPERIMENTS

In this section we give some experiments performed using `netspot`. First, our IDS is confronted with various real-world

[8]https://www.openapis.org/
[9]https://www.influxdata.com/products/influxdb-overview/
[10]https://grafana.com/

attacks. We use the dataset used by Kitsune [1] and `netspot` was able to detect all the attacks present in this dataset. We will see that these attacks are likely to disrupt the network traffic and this can be detected by monitoring some relevant statistics.

Second, the performances of `netspot` in terms of packet processing will be analyzed. This experiment shows that `netspot` is comparable to a signature-based IDS and also 10x faster than the state-of-art in network anomaly detection (Kitsune).

A. *Netspot facing real-world attacks*

First we test `netpot` on several real-world scenarios provided in [1]. Our results justify the "simplicity" of `netspot`: monitoring single statistics is enough to detect cyber-attacks. There are 4 types of attacks (Botnet Malware with Mirai), Denial of Service (SSDP Flood, SYN DoS and SSL Renegotiation), Man in the Middle (Video Injection, ARP MitM and Active Wiretap), and Reconnaissance (OS Scan and Fuzzing) [1]. We present here one in each category with graphs, but they are all detected.

SYN DoS. In this first scenario, an attacker "disables a camera's video stream by overloading its web server". This is a classical deny-of-service (DoS) attack which floods a system by sending a huge amount of SYN packets. The ratio of SYN packets (number of TCP packets with SYN flag over the number of IP packets) is then a natural statistic to catch such attack. The figure 6 shows the monitoring by `netspot`. One may observe that the threshold learned (dashed line) on the first value allows to flag abnormal events occurring in the end (red spots).

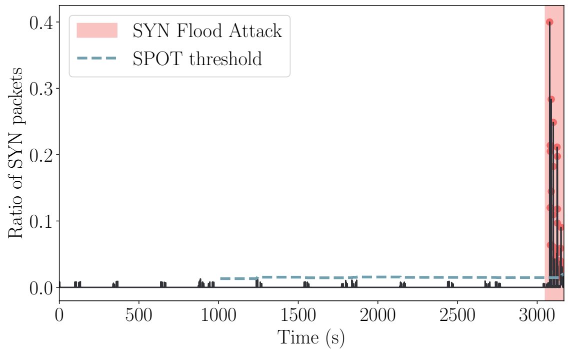

Fig. 6. Monitoring of the ratio of SYN packets (500ms time window) to detect DoS attack

Besides, this attack also impacts the ratio of ACK packets. When a user sends a SYN packet, the server responds with SYN-ACK packet and waits for a last ACK packet from the user (3-way TCP handshake). In this scenario, the attacker sends the first SYN packet but he ensures that the last ACK packet will not be sent (spoofed or rogue IP can be used). So the server waits for the ACK packet. If many SYN packets have been sent, the server will use all its resources waiting for ACK packets. A legitimate user who sends a new SYN packet will not have the SYN-ACK reply (denial-of-service). At this moment, the number of ACK packets intuitively falls. The figure 7 particularly illustrates this phenomenon during the attack. By computing a lower threshold (dashed line), `netspot` can easily flags the fall of the ratio of ACK packets.

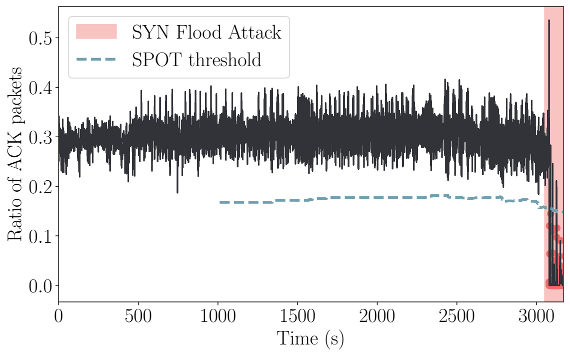

Fig. 7. Monitoring of the ratio of ACK packets (500ms time window) to detect DoS attack

SSL renegotiation. This attack is similar to the previous one insofar as it leads to a denial of service. This scenario exploits the vulnerability CVE-2009-3555[11] which affects TLS and SSLv3 protocols. Once a client has established a secure connection (SSL/TLS) with a server, he can renegotiate a connection (and start a new handshake) if the SSL-renegotiation is activated on the server. However, a SSL/TLS handshake requires at least 10 times more processing power on the server than on the client. Therefore a malicious client can request many re-negotiations to overload the server.

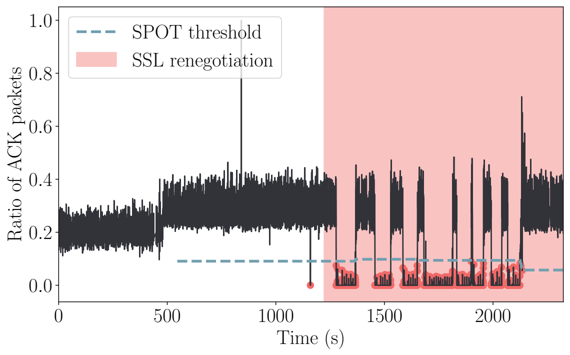

Fig. 8. Monitoring of the ratio of ACK packets (1s time window) to detect SSL renegotiation attack

In [1], this attack is performed on a camera (to disable its video stream) and the figure 8 particularly shows how the ratio of ACK packets behaves. Like in the SYN flood example, many drops occur during the attack, highlighting a denial of

[11]https://access.redhat.com/security/cve/cve-2009-3555

service from the camera. We can check that the lower threshold computed by SPOT is able to detect these abnormal events.

Mirai. Mirai is a malware aiming at infecting Linux devices (notably IoT devices). Once several systems are infected, the created botnet can then be used to launch Distributed Denial-of-Service attacks (DDoS). Among highly publicized Mirai attacks, one can mention the attack on OVH[12] (French web host) in September 2016 and the attack on Dyn[13] (DNS provider) in October 2016.

Once Mirai has infected a system, it tries to infect others. For that purpose, it scans the network so as to discover potential targets. A simple solution to scan the neighborhood is to broadcast ARP (Address Resolution Protocol) probe packets and wait for responses. Indeed, ARP is basically used to find MAC address associated to a given IP address. When a host receives an ARP probe, it sends back its IP and MAC addresses.

The experiment proposed in [1] considers an IoT Wi-Fi network (3 PC and 9 IoT devices) with a security camera infected by a real sample of the Mirai malware. The corresponding network capture notably records this attack discovery stage. The figure 9 shows the detection by `netspot` of the Mirai scan.

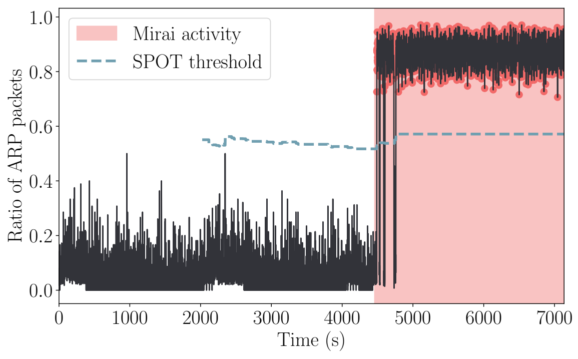

Fig. 9. Monitoring of the ratio of ARP packets (1s time window) to detect Mirai activity

Active Wiretap. In this last scenario, an attacker "intercepts all LAN traffic via active wiretap (network bridge) covertly installed on an exposed cable". Practically, a Raspberry Pi (in promiscuous mode) is plugged on a switch and eavesdrops the local network. To detect such an attack, we may wonder what happens when an Ethernet cable is plugged. Naturally, it also starts with a discovery phase where the new system asks the LAN who is connected at some IP addresses. Possibly other requests can occur according to the system (network time, name resolution etc.).

Like Mirai, many ARP packets are likely to be broadcast on the network. On figure 10, we can observe the behavior of the ratio of ARP packets. This rate is rather low but two high peaks are noticeable. If we investigate deeper, these peaks are not the most interesting. However, the zoom on the plot (bottom figure) highlights small peaks flagged by SPOT (red circles). The latter are especially created by the Raspberry Pi activity (the wiretap). It means that `netspot` is able to detect low amplitude anomalies that is obviously a paramount feature for an IDS.

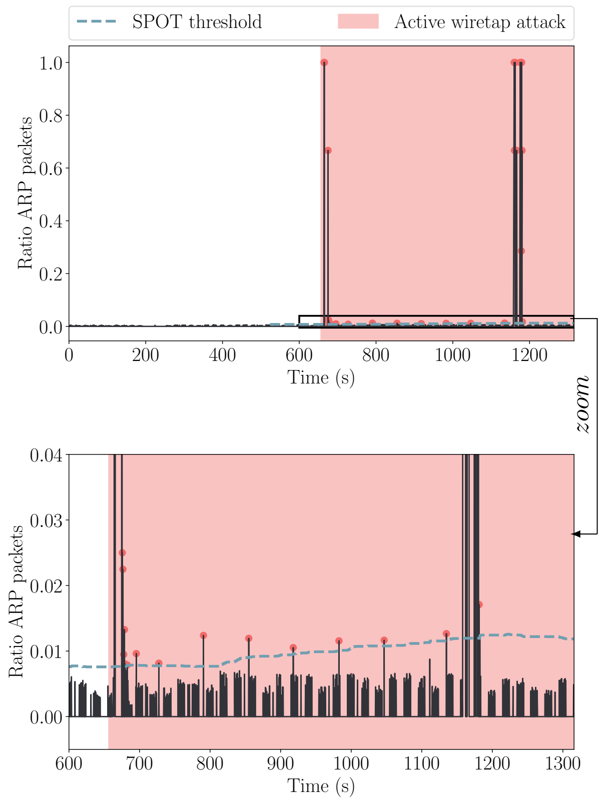

Fig. 10. Monitoring of the ratio of ARP packets (250ms time window) to detect active wiretap

B. Performances

The simplicity of `netspot` makes it fast. To check its performances, we feed a big capture file (2 278 689 packets) to `netspot`, varying the number of statistics to monitor. When `netspot` processes the file, the current packet parsing rate can be retrieved. We have noticed that `netspot` is not slower once it is calibrated, so the performances take all into account: the cold start (calibration) and the cruising regime (flag/update). Figure 11 shows the performances of `netspot` on two systems: a classical desktop computer (8 Intel cores and 8GB RAM) running `KUbuntu`, and a Raspberry Pi 3B+ (4 cores ARMv7 and 1GB RAM) running `Raspbian`.

On the desktop, `netspot` is roughly able to process 500 000 packets a second (with some peaks higher than 1M packets/s). On the Raspberry, `netspot` is about ten times slower with about 50 000 packets/s.

To compare with Kitsune (table I), the latter is able to process about 40 000 packets/s on a desktop computer and

[12]https://www.ovh.com/world/
[13]https://dyn.com/

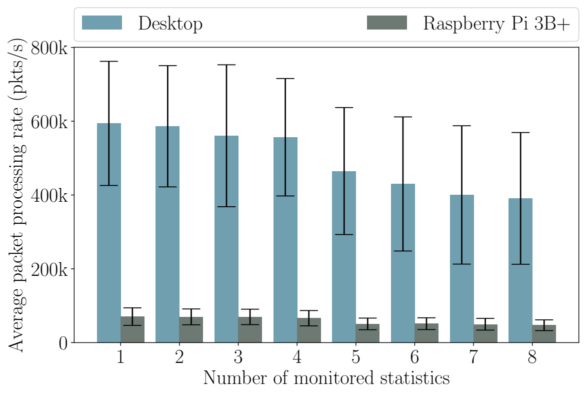

Fig. 11. Performances of `netspot` on two systems according to the number of monitored statistics

about 5 000 packets/s on a Raspberry Pi (performances are taken from their experiments as their own `C++` implementation is not publicly available). On rather equivalent configurations `netspot` is therefore 10× faster than Kitsune. To compare with a signature-base IDS, we also run `Suricata` on the same capture file with default rules[14] (20434 signatures). On the desktop computer (there is no release of `Suricata` targeting a Raspberry Pi), `Suricata` analyzes the whole capture ($\simeq$2M packets) in about 5s, so it leads to an average packet rate of 400 000 packets/s.

Platform	Suricata	Kitsune	**Netspot**
Desktop	400 000	40 000	**500 000**
Raspberry Pi 3B+	n/a	5 000	**50 000**

TABLE I
AVERAGE NUMBER OF PACKETS PROCESSED PER SECOND

In a word, `netspot` is about ten times faster than the state-of-the-art in network anomaly detection while comparable with well-proven methods (signature-based IDS). This efficiency has to be balanced with the accuracy of the tool. While UNADA and Kitsune are able to detect individually attack packets, `netspot` is more efficient but only flags a window containing possible attacks. An analysis is required in a second time to identify alarm packets. We think that this efficiency/information tradeoff is very important. Finally, the accuracy of the probability parameter of SPOT makes `netspot` more convenient to install and configure. The fact that it is automatically adapted to the network traffic also makes it more easier to deploy and integrate into an existant information system. It does not need to be retrained for each new printer we install.

V. Conclusion

Applying ML/DM techniques to intrusion detection is a very hard task since goals are not clear and relevant data are lacking. `netspot` is a new simple IDS embedding statistical learning. Its approach is completely different from the previous works. It has a simple design with a powerful detection engine based on both expertise (network statistics) and anomaly detection (SPOT). Indeed, through some experiments we show that it is able to detect real-world cyber-attacks. This is obviously not the ultimate solution to detect all the zero-day attacks but a new component, providing behavioral information, which can be integrated in a more general monitoring system.

`netspot` is still under active development as many improvements are possible: on the performance side one may have a look to more efficient ways to parse packet metadata (like PF_RING[15] or XDP [14]). On the monitoring side, one may naturally implement other counters and statistics. We hope that it could be a modest but credible example of practical anomaly-based NIDS.

References

[1] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, “Kitsune: An ensemble of autoencoders for online network intrusion detection,” *NDSS*, vol. 5, p. 2, 2018.

[2] P. Casas, J. Mazel, and P. Owezarski, “Unada: Unsupervised network anomaly detection using sub-space outliers ranking,” in *International Conference on Research in Networking*. Springer, 2011, pp. 40–51.

[3] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, “A density-based algorithm for discovering clusters in large spatial databases with noise.” in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[4] A. Siffer, P.-A. Fouque, A. Termier, and C. Largouët, “Anomaly detection in streams with extreme value theory,” in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 1067–1075.

[5] J. A. Halderman, “How let’s encrypt doubled the internet’s percentage of secure websites,” 2019, available at https://news.umich.edu/how-lets-encrypt-doubled-the-internets-percentage-of-secure-websites\-in-four-years/.

[6] I. Golan and R. El-Yaniv, “Deep anomaly detection using geometric transformations,” in *Advances in Neural Information Processing Systems*, 2018, pp. 9758–9769.

[7] E. Manzoor, H. Lamba, and L. Akoglu, “xstream: Outlier detection in feature-evolving data streams,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 1963–1972.

[8] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang, “Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2672–2681.

[9] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, “Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 387–395.

[10] R. A. Fisher and L. H. C. Tippett, “Limiting forms of the frequency distribution of the largest or smallest member of a sample,” in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 24, no. 2. Cambridge University Press, 1928, pp. 180–190.

[11] B. Gnedenko, “Sur la distribution limite du terme maximum d’une serie aleatoire,” *Annals of mathematics*, pp. 423–453, 1943.

[12] J. Pickands III *et al.*, “Statistical inference using extreme order statistics,” *the Annals of Statistics*, vol. 3, no. 1, pp. 119–131, 1975.

[13] A. A. Balkema and L. De Haan, “Residual life time at great age,” *The Annals of probability*, pp. 792–804, 1974.

[14] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, “The express data path: Fast programmable packet processing in the operating system kernel,” in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, 2018, pp. 54–66.

[14]A `python` tool allows to update the rules, see https://github.com/OISF/suricata-update

[15]https://www.ntop.org/products/packet-capture/pf_ring/