DarkVec: Automatic Analysis of Darknet Traffic with Word Embeddings

Luca Gioacchini, Luca Vassio, Marco Mellia
Politecnico di Torino
Italy
first.last@polito.it

Idilio Drago
Università degli Studi di Torino
Italy
idilio.drago@unito.it

Zied Ben Houidi, Dario Rossi
Huawei Technologies Co. Ltd
France
zied.ben.houidi,dario.rossi@huawei.com

ABSTRACT

Darknets are passive probes listening to traffic reaching IP addresses that host no services. Traffic reaching them is unsolicited by nature and often induced by scanners, malicious senders and misconfigured hosts. Its peculiar nature makes it a valuable source of information to learn about malicious activities. However, the massive amount of packets and sources that reach darknets makes it hard to extract meaningful insights. In particular, multiple senders contact the darknet while performing similar and coordinated tasks, which are often commanded by common controllers (botnets, crawlers, etc.). How to automatically identify and group those senders that share similar behaviors remains an open problem.

We here introduce DarkVec, a methodology to identify clusters of senders (i.e., IP addresses) engaged in similar activities on darknets. DarkVec leverages word embedding techniques (e.g., Word2Vec) to capture the co-occurrence patterns of sources hitting the darknets. We extensively test DarkVec and explore its design space in a case study using one month of darknet data. We show that with a proper definition of *service*, the generated embeddings can be easily used to (i) associate unknown senders' IP addresses to the correct known labels (more than 96% accuracy), and (ii) identify new attack and scan groups of previously unknown senders. We contribute DarkVec source code and datasets to the community also to stimulate the use of word embeddings to automatically learn patterns on generic traffic traces.

CCS CONCEPTS

• **Networks** → **Network monitoring**; **Network management**;
• **Security and privacy** → **Network security**.

ACM Reference Format:
Luca Gioacchini, Luca Vassio, Marco Mellia, Idilio Drago, and Zied Ben Houidi, Dario Rossi. 2021. DarkVec: Automatic Analysis of Darknet Traffic with Word Embeddings. In *The 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '21), December 7–10, 2021, Virtual Event, Germany.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3485983.3494863

1 INTRODUCTION

Darknets are sensors that observe traffic received by networks that are announced on the Internet but hosting neither production services nor client hosts [24]. Such unsolicited packets represent a privileged source of information for network security and debugging activities [23, 26], exposing threats like scans, brute-force attempts, and misconfigured hosts [17].

Even small darknets receive traffic from hundreds of thousands of senders targeting practically all TCP/UDP ports. Extracting meaningful insights from such a large amount of data remains a challenge. Each received packet can be mapped into three dimensions: (i) the target service, coarsely represented by the used transport protocol and destination port; (ii) the time of arrival; and (iii) the space, represented by the sender source IP address. All dimensions show a highly variable picture, with new senders arriving over time, disappearing and reappearing, targeting different ports and changing their sending rates. These senders are often part of different efforts to scan the Internet, with *group of senders* that take part to the same action. These include botnets looking for vulnerable machines, scanners from security companies (and researchers) that build maps of the IPv4 Internet, and misconfigured hosts that contact the darknet in the search for a non-existent or moved server. Often, senders are victims of attacks that reply to spoofed IP addresses. The darknet traffic is a superposition of the diverse patterns that each group of senders generates, making it hard to identify relevant events in such an aggregate. The analysis of darknet traffic would clearly benefit from automatically (i) grouping senders that produce similar patterns and (ii) associating new senders to previously known groups of senders.

This paper presents our efforts in addressing these two points. We introduce DarkVec, a methodology to process darknet traffic and automatically extract complex patterns from raw traces. We borrow techniques from Natural Language Processing (NLP), making the parallel between words that co-occur in sentences and sources that hit the darknet following a given pattern. We rely on word embedding, which is a recognized method to associate rich features to words in a language. By exploiting the mere co-occurrence of words in a context, word embedding techniques can project such *categorical variables* to a latent space, in which the words are arranged in an interesting syntactic and semantic way that is easy to infer and exploit by algorithms [29]. For example, it is possible to retrieve the relationship between countries and their capital cities through simple algebraic operations on the word vectors, or map words to their translation in another language with almost no supervision [30].

Similarly here, we set out to project the sources reaching the darknet in a latent space in which senders that share common traits are easy to detect and group using classic machine learning techniques. DarkVec uses darknet traffic to define *sentences* where each *sender IP address* is a word, and temporal *sequences* of source IP addresses form sentences. We consider packets going to different sets of transport-layer ports, which we call *services*, as belonging to different sequences. Hence, we have many sequences of sender IP addresses. For instance, the source IP addresses reaching port 23 are words for one service, which is different from the services built using port 80/443/8080 packets. Given a time interval, each sequence of words of the same service defines a sequence. The set of all sequences in all services forms our *corpus*, which we then process to create our embedding with NLP techniques.

We systematically explore the design space of DarkVec and show its capabilities with a comprehensive set of experiments on a one-month long darknet trace. We find that the embedding produced by DarkVec map senders performing the same activity into the same latent space regions. Using a labeled dataset, we show that DarkVec correctly assigns sources to activity groups with 96% of accuracy. Parameter tuning is not critical, thanks to the robust embedding produced by the definition of proper services.

Given the promising results on labeled data, we apply unsupervised approaches to automatically identify clusters of senders from the generated embedding. We identify new scan and attack patterns and extend our understanding of groups taking part in previously known activities. In a nutshell, DarkVec exposes several novel groups of senders that were not present in security databases.

Overall, we show how word embedding sheds light on noisy darknet traces. Beyond the darknet traffic use case, we hope our results and methodological insights can inspire the application of DarkVec to the analysis of other network traffic traces too. For that, we release DarkVec source code and an anonymized version of the dataset used in the paper.[1]

After discussing related work (Section 2) and the properties of observed darknet traffic (Section 3), we motivate DarkVec (Section 4) and describe DarkVec service and embedding definition (Section 5). We then show how representative the embedding is (Section 6) and how to extract clusters and new patterns (Section 7), summarizing our findings (Section 8).

2 RELATED WORK

The literature is rife with studies that analyze darknet traffic for various purposes. Data coming from darknets helped profiling attack strategies [23, 26, 32, 40], detecting and characterizing Internet scans [21, 22, 35] and studying malware spread [41]. Such prior studies relied on ad-hoc algorithms to characterize darknet traffic and have repeatedly proved the value of such networks. Here we propose a methodology to automate such an analysis, applying word embedding to simplify the understanding of senders contacting darknets.

Darknet traffic has been modelled using complex networks. Authors of [27, 28] adopt such an approach, modelling the traffic as a graph to detect transport-layer ports co-targeted by scanners. Authors of [39] build a bipartite graph for representing darknet traffic and then apply community detection on it, obtaining clusters of autonomous systems characterized by similar behavior. These approaches are complementary to DarkVec, as they focus on particular features of the traffic. We propose an approach that puts together multiple dimensions, e.g., finding patterns characterized not only by the type of activity performed by senders, but also by the time and sequence such activities occur.

The closest to our work are DANTE [20] and IP2VEC [37]. Both apply Word2Vec [29, 31], to extract features from traffic traces. DANTE is even closer since it also considers darknets as a traffic source, whereas IP2VEC is a more generic flow-level traffic analysis methodology.

DANTE aims at exploiting the sequence of ports that each sender targets. For this, the authors treat the sequence of the ports reached by senders as sequence of words in a NLP problem, and they represent each sender by the sequence of ports it targets within an observation window. The authors then train a separate Word2Vec embedding for each port. Finally, each sender IP address is associated to a vector by averaging the embeddings of the ports that the IP has contacted. The outcome is later analyzed with standard clustering algorithms. In a similar yet more generic fashion, IP2VEC embeds IP addresses (and also ports and protocols) by building Word2Vec models that consider as words the sequence of several flow-level variables, such as destination IP addresses, port numbers and the used transmission protocols. We report more details about DANTE and IP2VEC in Appendix A.2.

We also rely on Word2Vec to characterize and cluster senders. Differently from DANTE, we define services based on target ports (instead of IP addresses) and use senders' IP addresses as words (instead of ports). In our corpus we have sequences of senders ordered by their arrival time, and projected into separate services. Then, we build a single embedding. This approach prevents the averaging of different embeddings adopted by DANTE and makes the corpus much more compact, so that DarkVec complete the Word2Vec training in minutes, instead of days taken by DANTE. Differently from IP2VEC, our service definition is key to limit the negative sampling which poses significant scalability problems to IP2VEC. As we show in Section 6.1, DarkVec outperforms both DANTE and IP2VEC in the darknet analysis use case. All in all, our improvements make it practical to apply Word2Vec to darknet setups. Moreover, our thorough exploration of word embedding design space (choice of service definition, parameters, etc.) should provide enough guidelines for their application to other networking use cases where the goal is to learn from sequences of categorical variables (e.g., log events, DNS queries, HTTP requests etc.).

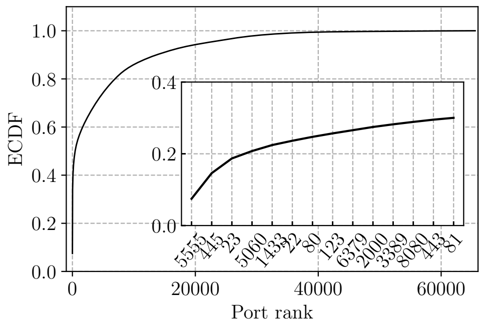
3 DATASET

We setup a /24 darknet in the range of a university campus network and use it for running experiments. For our analysis, we focus on 30 days of traffic covering the period from 2021-03-02 to 2021-03-31. In Table 1 we provide some statistics about the dataset, from which we separate the last day of our collection as a testing set.

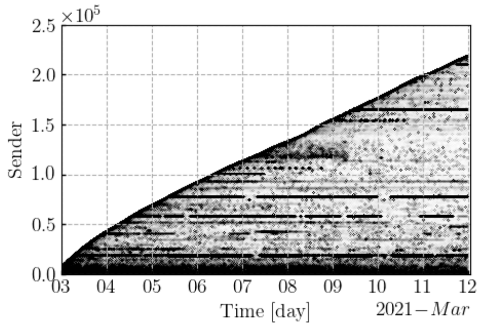
[1]https://github.com/SmartData-Polito/darkvec

Table 1: Single day and complete dataset statistics.

	Dates	Sources	Packets	Ports	Top-3 TCP ports		
					Port	Traffic [%]	Sources
30 days	[2021-03-02, 2021-03-31]	543 900	63 562 427	65 537	5555	7.43	20 844
					445	7.09	73 665
					23	4.07	209 396
Last day	2021-03-31	43 118	3 461 220	19 583	445	8.33	4 274
					5555	8.15	1 522
					23	3.54	16 102



(a) Port ranking. Zoom on top-14 ports.



(b) Sender's activity pattern.

Figure 1: Darknet traffic overview.

3.1 Darknet traffic overview

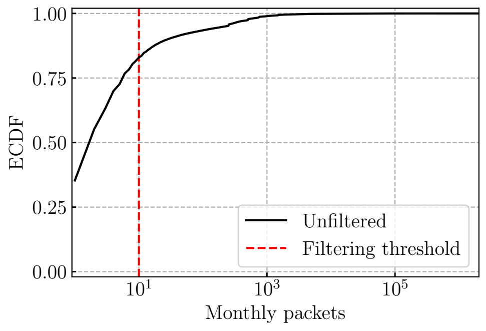
Figure 1 gives an overview of the traffic the darknet observes along the (i) service, (ii) space, and (iii) time dimensions. In details, Figure 1a reports the Empirical Cumulative Distribution Function (ECDF) of the number of packets received by each port in one month.[2] All ports get some unsolicited packets, albeit most traffic is directed to specific ports. The inset shows the top-14 ports - each easily linked to well-known services or widely exploited vulnerabilities.[3]

Figure 1b showcases the activity of each sender over time. On the y-axis, each line (y value) represents a different sender IP address, sorted by the timestamp of first appearance in the trace. The x-axis represents time. A dot is a packet received from a sender at given time. In total, we observe more than 220 000 senders, sending about 1 million packets. As expected [24], we observe a continuous growth of the number of senders over time. Some senders are persistently present (darker bottom part); some senders appear sporadically (horizontal segments); some senders are seldom visible (sparse dots).

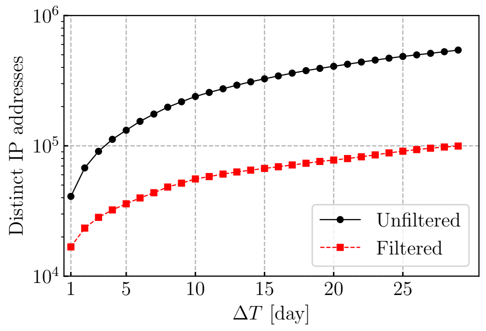
To complete the overview, Figure 2a reports the ECDF of the total number of packets received from each sender. The large majority of senders hits the darknet with few packets – 36% are seen just once in a month. These senders are likely victims of attacks with spoofed addresses – i.e., we see the so-called backscatter phenomenon [26, 36]. Yet, there exist many senders that are quite active, which are the focus of our analysis. Here we discard the occasional senders, filtering those addresses sending less than 10 packets to the darknet in the considered period. The rationale is to go after senders for which we have enough evidence to perform a reliable analysis. The remaining 20% of *active* senders (i.e., sending 10 or more packets) are responsible for the majority of the darknet traffic.

At last, Figure 2b shows the count of distinct IP addresses seen over an increasing period of time. Focusing on the first day, we observe about 40 000 distinct senders. This figure quickly grows over time so that after 30 days we observe more than 500 000 unique senders. About 20% of them are active, i.e., after 30 days we collect enough data to characterize 100 000 active senders.

[2]UDP and TCP ports are summed together for simplicity.
[3]Port 5555 is often scanned in the search for Android Debug Bridge (ADB) service.



(a) Amount of packets per sender in 1 month.



(b) Cumulative number of senders over time.

Figure 2: Senders characterization and filtering criteria definition.

Table 2: Ground truth classes present in the last day of the collection and active in the 30 day dataset.

Label	Source	Senders	Packets	Ports	Top-5 Ports (% Traffic)	Top-5 Ports Traffic [%]
GT1	Mirai-like [25]	7 351	88 192	75	23/TCP (89.6%), 2323/TCP(3.9%) , 5555/TCP(1.7%), 26/TCP(1.3%), 9530/TCP(0.84%)	97.34
GT2	Censys [4]	336	233 004	11 118	5060/TCP(3.4%), 2000/TCP(2.9%), 443/TCP(0.4%), 445/TCP(0.4%), 5432/TCP(0.4%)	7.5
GT3	Stretchoid [15]	104	57 144	91	22/TCP(3.5%), 443/TCP(3.5%), 21/TCP(2.7%), 9200/TCP(2.7%), 139/TCP(1.8%)	14.2
GT4	Internet Census [8]	103	9 396	231	5060/TCP(10.4%), 161/UDP(9.8%), 2000/TCP(7.7%), 443/TCP(6.5%), 53/UDP(2.9%)	37.3
GT5	BinaryEdge [3]	101	7 646	21	15/TCP(10%), 3000/TCP(9.6%), 4222/TCP(6.7%), 587/TCP(6.6%), 9100/TCP(5.8%)	38.7
GT6	Sharashka [12]	50	5 436	485	5986/TCP(0.48%), 2103/TCP(0.48%), 2052/TCP(0.44%), 3005/TCP(0.44%), 2087/TCP(0.44%)	2.28
GT7	Ipip [2]	49	17 342	41	5060/TCP(41.5%), ICMP(10.9%), 8000/TCP(2.3%), 8888/TCP(2.1%), , 22/TCP(2.1%)	58.9
GT8	Shodan [13]	23	13 566	349	443/TCP(0.9%), 80/TCP(0.9%), 2222/TCP(0.9%), 2000/TCP(0.7%), 2087/TCP(0.7%)	4.1
GT9	Engin-Umich [9]	10	506	1	53/UDP(100%)	100
Unknown	–	14 272	2 971 687	10 520	445/TCP(9.4%), 5555/TCP(9.4%), 1433/TCP(1.8%), 123/UDP(1.6%), 6379/TCP(1.5%)	23.7
Total		22 399	3 403 959	19 882	445/TCP(8.3%), 5555/TCP(8%), 23/TCP(3.5%), 1433/TCP(1.6%), 123/UDP(1.4%)	22.8

3.2 Getting a labeled dataset

One of the main difficulties for automating the analysis of darknet traffic is the lack of ground truth for evaluating the results. DarkVec aims to group senders that perform similar activity over time. It is thus fundamental to gather samples of such groups to gauge the performance of DarkVec. For labeling, we exploit two sources of data: (i) the presence of the widely known Mirai-like malware(s) fingerprint [19, 25] in packets; (ii) our knowledge about popular security search engines and research projects such as Shodan [13] and Sonar [10] that make publicly available the IP addresses they use.

As said, in our 30-day long dataset, we observe about 100 000 unique IP addresses corresponding to active senders. The manual evaluation and search for publicly known coordination of all such sources is unfeasible. We thus focus on the most active senders seen in the last day of our collection, which we use as labeled dataset for testing. Here, we observe 22 399 active senders in total. Among these senders, we identify nine ground truth (GT) classes, summarized in Table 2. We identify senders part of the Mirai-like botnet(s) with more than 7 300 hosts targeting a limited number of ports and services, i.e., Telnet (23/TCP and 2323/TCP) or ADB (5555/TCP). Next, we identify senders that are part of well-known projects performing Internet scans. The largest group includes 336 active senders of the Censys project [4] that target more than 11 000 unique destination ports. The smallest groups include 10 senders of the Engin-Umich project [9] that performs scans focusing on DNS (port 53/UDP) only. As expected, about 2/3 of the active senders remain *Unknown*. These senders may belong to other classes or even be part of some of the known classes, which however we could not identify.

4 BASELINE

To motivate DarkVec, we test simpler approaches that leverage traffic-related information to cluster groups of senders. We consider simple features, like top-destination ports, numbers of packets and others. Intuitively, one could argue that grouping senders by such features could already lead to the identification of coordinated groups.

To check whether this intuition is correct, Figure 3 shows the fraction of daily packets sent by senders of each ground truth class

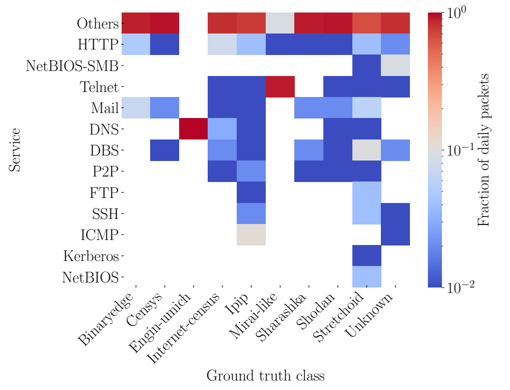


Figure 3: Fraction of daily packets sent to generic services, normalized by columns.

to generic services. Here we identify a service with the group of ports typically used by the service, i.e., port 25, 110, 143, 587, ..., belong to the 'Mail' service. The heatmap clearly shows that a naive port-based approach could work only for the cases where a single class dominates the traffic observed for a service, like for the Engin-umich group, which dominates the DNS traffic. For the other classes, the traffic is scattered among different services, calling for algorithms able to identify hidden patterns on the data.

To check whether this would be possible with basic service features, we build a supervised classifier that uses as features the fraction of traffic each sender generates to top destination ports. We take the last day of traffic and we label senders according to the 10 GT classes in Table 2, i.e., the 9 GT classes and the 'Unknown' class. For each class, we extract its top-5 ports in terms of packets. We then merge all top-5 port sets to compose our final feature set, with the percentage of traffic of the senders to each of the selected top ports.[4]

We use a k-Nearest-Neighbor (k-NN) classifier to assign a label to each sender according to the labels of the majority of its k neighbors.

[4]We select the top-5 ports for each class to intentionally create a biased feature set that would favour the 9 classes in our ground truth.

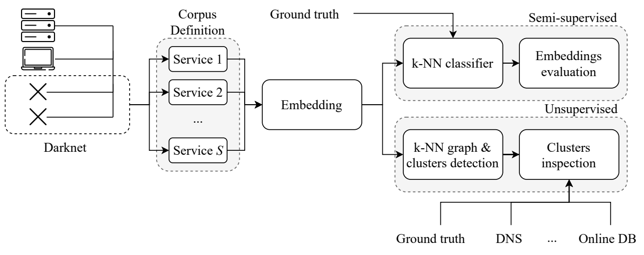


Figure 4: DarkVec overview.

We compute the cosine similarity to identify the k nearest neighbors. Using a Leave-One-Out approach, for each sender we compare the k-NN classifier prediction with the original label and evaluate the accuracy of the classifier. We test values of $k \in \{1, 3, 5, 7, 9, 11\}$, with best performance with $k = 7$. Results are quite poor and details are reported in Table 6 in the Appendix.

These results strengthen the intuition that we need a more advanced approach that exploits also the temporal information present in the data.

5 DARKVEC

We now present DarkVec. We assume the reader is familiar with Word2Vec and provide context about it in Appendix A.1.

5.1 DarkVec in a nutshell

To process the aggregate traffic received by a darknet, DarkVec leverages word embeddings. Ubiquitously used in modern NLP tasks, word embeddings leverage the frequency and – most important – the co-occurrence of words in sentences to project them in a high-dimensional space, associating a rich feature vector to each word. Although they were built exploiting the co-occurrence of words, these vectors end up encoding well semantics and syntactical properties of words. They are used as input to other ML algorithms to perform various NLP tasks. We build on the same idea to embed senders' IP addresses into a latent space, thus mapping each each sender IP address to a vector in the embedding space.

Figure 4 provides the high-level overview of DarkVec. From the left, we collect all packets received from the darknet. Given packets observed during a time period, we extract separate sequences of senders considering the *services* they target. We identify senders by the source IP address, and we define services based on destination ports they target. Next, we use such sequences to train a *single* Word2Vec model. We map the categorical IP addresses into a multidimensional space using a traditional one-hot encoding to create independent input vectors. Sequences of senders thus become sequences of vectors that define the corpus of the Word2Vec input to create the embedding, which maps the original sender IP address into a compact low-dimensional space. The resulting embedding can be analysed using semi-supervised and unsupervised machine learning algorithms to exploit the correlation the Word2Vec mapping creates in the process.

In the semi-supervised case, we assume to have external knowledge about the classes of some IP addresses, e.g., specific botnets and known scan projects. This ground truth allows us to extend the classification for other IP addresses using the concept of distance in the embedded space. It also allows us to evaluate the goodness of the embedding, i.e., setting up validation experiments with IP addresses belonging to the ground truth.

In the unsupervised case, we rely on clustering or community detection techniques to identify groups of IP addresses that are nearby in the embedded space. The extraction of groups dramatically simplifies the characterization of the activity performed by senders, reducing the amount of manual work during the investigation.

We detail the key steps of the methodology in the following, starting from the definition of the services for the aggregation and embedding creation. We detail the results of the semi-supervised and unsupervised analyses in Section 6 and Section 7.

5.2 Service definition

We aim at finding similarities among senders' activity considering packets they send to a darknet. We consider each source IP address associated to an incoming packet to be a *word* w. We then create ordered sequence of words to build embeddings with Word2Vec [29, 31].

We leverage the definition of *services* to coarsely separate senders into different semantic groups. Note that the definition of the services is helpful to guide the training of the embedding and to let it scale to large datasets.

As previously done in the baseline classifier model, we exploit domain knowledge to define different services. Given a destination port p, with $p \in \{0, \ldots, 65\,535\}$, we characterize a service $s \in S$ by the set of ports used by common-purpose services $P^s = \{p_1, \ldots, p_n\}$. For example, port 23/TCP defines the Telnet service, whereas ports 80/TCP, 8080/TCP and 443/TCP define the HTTP service.

We split incoming packets into multiple sequences, one for each service. DarkVec employs a fixed time window of duration ΔT to split the stream of packets into separate sequences. In details, taking the sequence of sender IP addresses appearing in a given time interval of duration ΔT, the final sequence $W^s(t)$ of the service s is the sequence of IP addresses sending packets to P^s in $[t, t+\Delta T)$. The choice of ΔT indirectly limits the length of the sequences, whose size depends on the traffic rate during ΔT. Sequences thus have variable size.

Next, for each service s, we build a *corpus* C^s of sequences by considering all non-overlapping time windows in $[t_0, t_f]$:

$$C^s = \cup_{i \in \{0,1,\ldots,N-1\}} W^s(t_0 + i\Delta T)$$

where t_0 and t_f are the initial and final observation times, and $N = \lceil (t_f - t_0)/\Delta T \rceil$ is the number of observation windows. The final corpus C is the union of the $|S|$ per-service corpora.

$$C = \cup_{s \in S} C^s$$

The final corpus C is made of all sequences of sender IP addresses. We use it to train a single Word2Vec embedding.

As we will see later, the definition of services is key to guide the construction of a good embedding. Here we consider three alternatives:

- Single service: all ports belong to a single service.

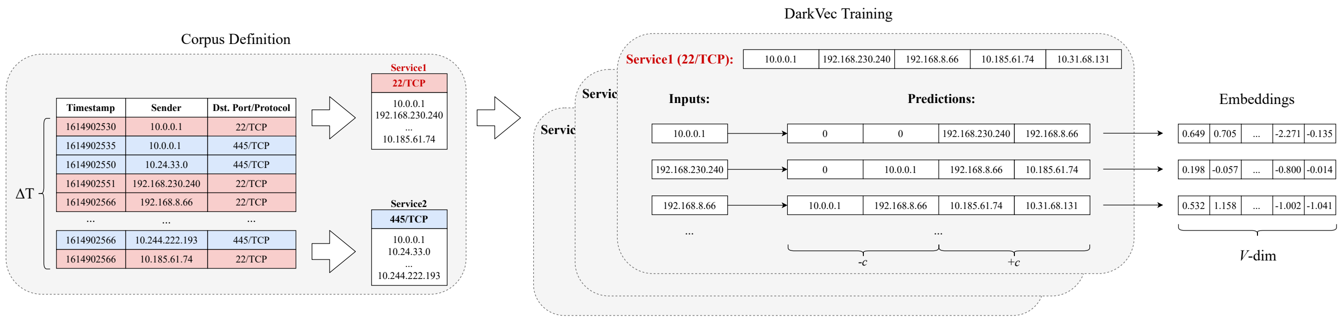


Figure 5: DarkVec training: On the left, the sequence definition by services. On the right, the skip-gram construction used to build the embedding. In output, each IP address is mapped to a point in a V-dimension space.

- Auto-defined services: we take the top-n popular ports, and create a specific service for each of them. All remaining ports form the $(n+1)$th service.
- Domain knowledge: we manually assign ports to services based on domain knowledge. Each service groups ports that are commonly used for popular applications. In total we build 15 services as detailed in Table 7.

Figure 5 gives an example of the corpus definition. On the left, we have the sequence of packets observed in a time window. There are only two services, 22/TCP and 445/TCP. IP addresses of senders targeting those ports form sequences. Notice that the same sender IP address may appear in different services, as "10.0.0.1" in the example.

In the following we set $n = 10$ for the auto-defined service scenario, $\Delta T = 1$ hour.[5]

5.3 Embedding definition

Given the final corpus C, we train our embedding. We employ the skip-gram model (see Appendix A.1), which provides excellent results when looking for embeddings that efficiently predict the next word in a sentence.

Given a sequence $W = [w_1, w_2, w_3, \ldots, w_n]$ and a context window of size c, for each w_i, we build a skip-gram $S(w_i)$ as the sub-sequence of the c previous and c following words of w_i:

$$S(w_i) = [w_{i-c}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+c}]$$

We pad the beginning and the end of each sequences with a special *NULL* word. The central part of Figure 5 represents the construction of skip-grams starting from one sequence. For instance, the skip-gram of w_i="10.185.61.74" is $S(w_i)$=["192.168.230.240", "192.168.8.66", "10.31.68.131", "NULL"] if $c = 2$.

Given all the skip-grams, we extract the embeddings using off-the-shelf Word2Vec. Word2Vec embeds the input words into a V dimensional space, i.e., $w_i \rightarrow u_i \in \mathbb{R}^V$. The dimension V of the projection is a parameter which impacts the quality of the embedding. We will evaluate this parameter later in our experiments.

In our case, Word2Vec leverages the co-occurrence of the sender IP addresses targeting the same port/service in a given time window. The resulting embedding shall map those IP addresses that frequently appear in the same context window into the same region in the V-dimensional space, i.e., senders that perform similar patterns nearby on time are mapped into a compact region.

For our experiments, we use the skip-gram-based Word2Vec Python implementation of *Gensim* [6]. We make all our source code and anonymized datasets available to the community to allow others to reproduce results.[6]

6 TESTING DARKVEC

We now validate the embedding DarkVec creates. First, we compare DarkVec with DANTE and IP2VEC. Then we present an analysis of parameter sensitivity.

We run all experiments on a high-end server equipped with 2 Intel Xeon Gold 6130 CPUs (each with 16 physical cores at 2.10 GHz) and 256 GB of memory. We implement DarkVec and DANTE in Python using the Gensim library, and IP2VEC in Keras, with parameters suggested in the original papers. In contrast to the Gensim-based cases, IP2VEC can profit from a Tesla V100 GPU (16 GB of memory) to speedup training.[7]

6.1 Comparison with DANTE and IP2VEC

In this experiment, we compare DarkVec, DANTE and IP2VEC on the same two scenarios. We consider the traffic observed in our darknet for a period of 5 days (1st scenario) and 30 days (2nd scenario). For each scenario, we select the subset of *active* senders and create the embedding for each case. We then run the same semi-supervised test we did with the baseline classifier to check how senders in the ground truth are projected into the embedded space. Intuitively, a good embedding shall project IP addresses of the same ground truth class to compact regions in the latent space so that a k-NN classifier can recover the correct label.

Following a Leave-One-Out approach, we take each IP address IP_i for which (i) we have a label as in Table 2 and (ii) that results active in the considered dataset. We use the cosine similarity to measure the distance to other IP addresses to the target IP_i, i.e., given IP_j, we use $cosine(u_i, u_j)$ of their projection vectors u_i, u_j to

[5] ΔT has marginal impact on performance. It is mostly instrumental to create an equivalent concept of sentence from time continuous traffic traces.

[6] The repository for the code is available at https://github.com/SmartData-Polito/darkvec

[7] We migrate the original IP2VEC PyTorch-based implementation to Keras to optimize performance. Gensim does not support GPU offloading.

Table 3: Comparison between DarkVec, IP2VEC and DANTE

			5 Day dataset (coverage: 82%)			**30 Day dataset** (coverage: 100%)		
	Corpus	**Epochs**	**Skip-grams**	**ETA**	**Accuracy**	**Skip-grams**	**ETA**	**Accuracy**
DarkVec	Domain knowledge based	20	17 M	~14 min	0.93	486 M	~1.2 hours	0.96
IP2VEC	Flow-based	10	38 M	~60 min	0.67	–	>10 hours	–
DANTE	Ports per IP	10	>7 B	>10 days	–	–	–	–

measure the similarity between IP_i and IP_j. Then we extract IP_i k-NN in the embedded space. We use majority voting over the k neighbors to assign the predicted class to IP_i and compare it against the ground truth class. If the predicted and actual class match, we have a correct prediction. By repeating the procedure for all labeled IP addresses, we compute the average accuracy, i.e., the probability that an IP address gets associated to the correct class. We consider only senders that belong to some ground truth class, i.e., GT1-GT9, skipping all IP addresses of the Unknown class since we do not know if they eventually belong to any of the GT classes.

Given the large amounts of data to process, scalability is vital for practical implementations. Thus, we compare the number of skip-grams and the training time required to build the embedding. We consider DarkVec with the domain knowledge service definition, context window $c = 25$ and embedding of $V = 50$ dimensions.

We summarize results in Table 3. Consider first the 5-day dataset. DarkVec predicts the correct class with accuracy 0.93, while IP2VEC reaches 0.67 accuracy only. Considering scalability, DarkVec takes 14 minutes to complete 20 epochs for training over 17 million skip-grams. IP2VEC requires about four times more time to complete 10 epochs. The additional complexity is also due to negative sampling adopted by IP2VEC [31], which consists on training the model with an additional set of words belonging to different contexts. This technique may improve performance but increases the size of the training data. DANTE does not scale, and after more than ten days, it could not complete the training. This happens because DANTE generates around 7 billion skip-grams during sequence creation. Recall that DANTE associates each port to a word and generates a different sentence for each IP address. This approach turns unfeasible with several thousands of IP addresses each generating independent sequences. The original paper presenting DANTE confirms the scalability problem.

Consider now the 30-day dataset. In this case, the number of active IP addresses grows by a factor of 5, reaching about 100 000 IP addresses (cfr. Figure 2b). More data allows us to build an embedding that covers more IP addresses. In fact, the number of active senders found in the last day and covered by the embedding grows from 17 463 to 22 399 when moving from 5- to 30-day long dataset. Restricting to those senders for which we have a label, the embedding built on 5 days of traffic cover 82% of senders (by construction, the coverage is 100% when using the 30-day dataset). This fact underlines the importance of having a scalable model to extend the dataset and the coverage of the embedding (more details in Section 6.2.1).

The increase in the data produces a sizeable increase in the number of skip-grams in the corpus. Still, the training of DarkVec completes in 1.2 hours, and the accuracy grows to 0.96. IP2VEC cannot complete the word sequence creation process after more than 10 hours, producing more than 200 million skip-grams. In sum, the limited number of services and the simple word sequence creation of DarkVec increase scalability. The resulting embedding outperforms IP2VEC, while DANTE again does not scale.

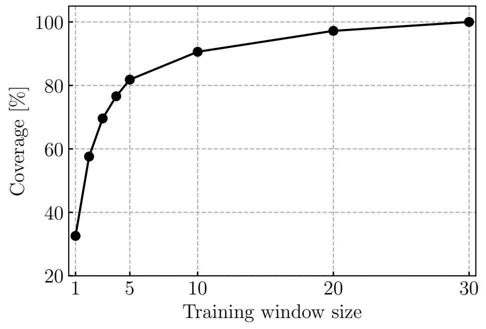


Figure 6: Impact of training window length.

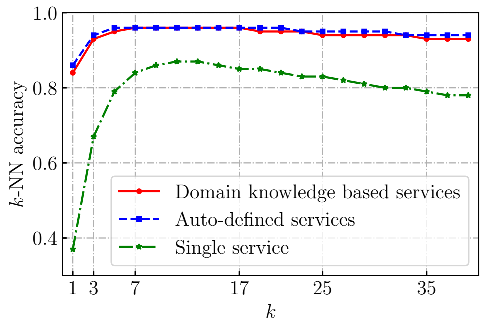
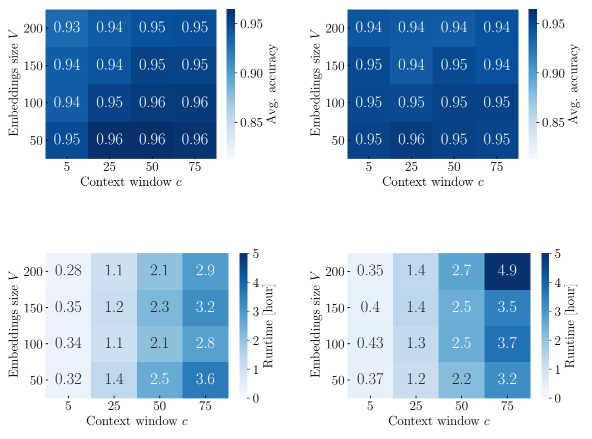


Figure 7: Impact of k on the k-NN classifier.

6.2 DarkVec parameter tuning

We perform a sensitivity analysis on the model hyper-parameters. We focus on the classification accuracy following the same Leave-One-Out approach on the IP addresses that are active in the last day of the trace. We vary the number of days used to train the Word2Vec model, the number of neighbors k, the embeddings size V, and the context window size c. Given the number of parameters to test, we cannot perform a complete grid search. Instead, we follow a greedy optimization by varying and choosing one parameter at a time. When not otherwise specified, we set $V = 50$ and $c = 25$, and train DarkVec on 30 days of traffic for 10 epochs.

6.2.1 Training data size. First, we check the impact of the training data size. Considering accuracy, we already know that a model



(a) Auto-defined (b) Domain knowledge based

Figure 8: Grid search on c and V and impact on accuracy (top) and training time (bottom).

trained on the 5-day dataset suffers a limited impact on the accuracy, with a drop of just 3% compared to using a 30-day long dataset. More important, the training data size has a significant impact on the coverage, as depicted in Figure 6. Indeed, given that we restrict the embedding constructions to those IP addresses seen at least 10 times in the training period, the longer the training, the higher the chance to collect enough observations to build a model for a given IP address.

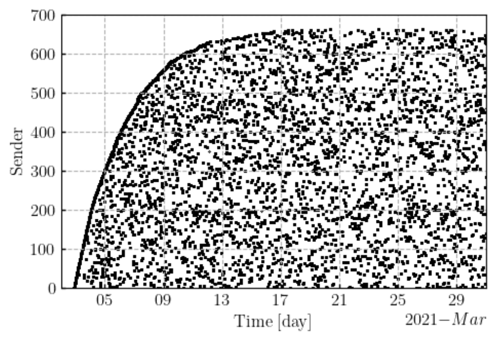
Given these results, we fix the training windows size to 30 days from now on to maximize coverage and accuracy.

6.2.2 Impact of k and service definition. Figure 7 shows the impact of k in the k-NN classifier. First, the single service model performs significantly worse than both domain knowledge based and auto-defined service models. Second, increasing k improves the average accuracy up to when the neighborhood starts to include too many samples. Here, classification becomes uncertain because the Unknown senders dominate the neighborhood for large k. In our validation, we consider such cases to be misclassifications, even if some Unknown samples may indeed behave similarly to those in the ground truth (possibly extending it).

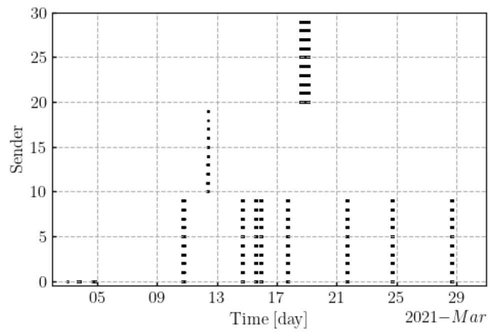
In the following, we drop the single service definition, and we fix $k = 7$, which the figure shows to guarantee accuracy above 0.96 for both auto-defined and domain knowledge based services.

6.2.3 Impact of c and V. Focus now on the context window size c and the number of dimensions of the embedding V. Here we consider both the model training time – the shorter the training time, the better the performance – and the accuracy of the k-NN classifier.

Figure 8 details results by showing on the top the accuracy and on the bottom the running time. From left to right, we compare the auto and domain knowledge-based service definitions. Each matrix shows the impact of $c \in [5, 75]$ and $V \in [50, 200]$. Accuracy tops to 0.96. Neither c nor V significantly impacts average accuracy, even



(a) Stretchoid activity pattern.



(b) Engin-Umich activity pattern.

Figure 9: Activity patterns of some GT classes.

if smaller values of V are preferable because compact embeddings are instrumental to avoid the curse of dimensionality.

Conversely, small values of c and V require less time to complete the training. Based on these results we set the context windows $c = 25$ and the embedding dimensions $V = 50$. Here, the usage of domain knowledge based services brings a slight advantage over the auto-defined ones when considering training time.

6.3 Per class result

The overall accuracy metric is biased toward the majority classes. In our case, we have a considerably unbalanced dataset with few classes with thousands of senders (e.g., Mirai-like) and others with just a handful of senders (e.g., Engin-umich). To check results in details, Table 4 shows the precision, recall, and F-score for all GT classes.[8] For the sake of completeness, we report results for all three service definitions, highlighting in red those results that are particularly unsatisfying (<0.5).

The single service embedding is particularly critical. It works well for the Mirai-like botnet(s), but fails in most other classes. Being the largest class, Mirai dominates the accuracy. Both the auto-defined and the domain knowledge-based service definition

[8]The precision for a class is the fraction of correct instances among those classified as belonging to the given class. Recall measures the fraction of instances that are recovered over the total instances of a class. The F-score is the harmonic mean of precision and recall. The weighted average of recall over the classes is equivalent to the overall accuracy.

Table 4: 7-NN classifier report. Values below 0.50 are highlighted in red.

	Single service (c=75, V=50)			Auto-defined services (c=50, V=50)			Domain knowledge based (c=25, V=50)			
	Precision	Recall	F-Score	Precision	Recall	F-Score	Precision	Recall	F-Score	Support
Mirai-like	0.98	0.86	0.92	1.00	0.98	0.99	1.00	0.97	0.98	7351
Censys	0.63	0.91	0.75	0.96	1.00	0.98	0.91	0.94	0.93	336
Stretchoid	**0.03**	**0.01**	**0.01**	0.94	**0.30**	**0.45**	1.00	**0.35**	0.51	104
Internet-census	**0.41**	0.50	**0.45**	0.79	0.86	0.83	0.94	1.00	0.97	103
Binaryedge	**0.44**	0.74	0.56	0.98	0.87	0.92	0.94	1.00	0.97	101
Sharashka	**0.12**	**0.02**	**0.03**	0.92	0.72	0.81	0.96	1.00	0.98	50
Ipip	**0.42**	0.92	0.58	0.51	0.86	0.64	**0.34**	0.84	**0.49**	49
Shodan	**0.00**	**0.00**	**0.00**	0.94	0.70	0.80	0.93	0.61	0.74	23
Engin-umich	0.67	1.00	0.80	0.62	1.00	0.77	1.00	1.00	1.00	10
Unknown	–	**0.11**	–	–	0.62	–	–	0.67	–	14272

help the Word2Vec to obtain a descriptive embedding even for minority classes. IP addresses of the same class are projected into the same portion of the space, so that the majority of the k neighbors results of the same class. Only the Stretchoid class obtains low recall. Looking into its temporal pattern in Figure 9a, we indeed observe a very irregular pattern; with few packets from each sender at irregular time intervals so that they fall in random sequences and skip-grams. The embedding likely projects these points at random. Interestingly, the 10 Engin-umich senders which target port 53 only are projected into the same portion of the embedding space so that the k-NN correctly classifies them. Notice that there are a lot of other senders that target port 53. Yet, the skip-gram model can perfectly capture the coordinated and very impulsive action of the 10 Engin-umich senders which we depict in Figure 9b.

In a nutshell, DarkVec with proper service definition proves a very effective way to automatically map senders hitting a darknet into a compact and well-organized space where senders performing similar activities over time and services are close to each other.

6.4 Extending the ground-truth

Adopting a semi-supervised approach, DarkVec allows us to assign labels to previously unlabeled senders. Given the set of Unknown IP addresses classified as one GT class, we sort them by increasing average distance to their k-NN and manually check if the assigned label could be correct. We stop when the average distance becomes higher than the maximum average distance among senders of the given GT class.

With this simple process, we have identified new senders performing scan patterns very similar to Shodan servers, other senders being very likely part of the Censys network, etc. This analysis let us extend our knowledge about already known GT classes. In the next Section we apply this rationale using an unsupervised approach to identify new classes sharing similar activities.

7 UNSUPERVISED EMBEDDING ANALYSIS

We have seen how senders involved in similar activities get projected into the same region of the latent space. This projection paves the road for adopting unsupervised approaches to identify clusters of senders and new patterns of coordinated actions.

7.1 Clustering methodology

We have compared several clustering alternatives, including classic algorithms that work directly in the embedded space such as k-Means, DBSCAN, Hierarchical Agglomerative Clustering [16]. Not reported here for the sake of brevity, these algorithms produce poor results due to the well-known curse of dimensionality as well as their difficult parameter tuning.

Given the good properties exhibited by k-NN for classification, we instead design a graph-based clustering for the unsupervised exploration of the embeddings. In detail, we build a directed graph $G(\mathcal{V}, \mathcal{E})$ where each IP address of the embedding is a vertex $v \in \mathcal{V}$. We then connect each vertex to its k' nearest neighbors so that we have

$$\mathcal{E} = \{e(v, v_i), v_i \in \text{k'-NN}(v), \forall v \in \mathcal{V}\}.$$

We assign to each edge $e(v_i, v_j)$ a weight equal to the cosine similarity $cosine(u_i, u_j)$ in the embedding. Note that each edge $e(v_i, v_j)$ is directed since v_j can be among the k' nearest neighbors of v_i, but v_j can have k' different neighbors.

Given a graph G, we use the Louvain algorithm [18] to extract non-overlapping communities or clusters. The algorithm maximizes the modularity score of clusters, where the modularity – in the range [-0.5,1] – quantifies the quality of an assignment of vertices to clusters. In a nutshell, the algorithm evaluates how much more densely connected the vertices within a cluster are when compared to how connected they would be in a random network with the same degree distribution. The Louvain algorithm has been successfully used for cluster detection in social networks [33] and even for darknet traffic analysis [39]. Among its advantages, the algorithm does not require a pre-defined number of clusters. Moreover, there exist several open source and scalable implementations of the algorithm.

7.2 Choice of k'

The only parameter for the graph based clustering is k', the number of neighbors each vertex is connected to. Since we follow a completely unsupervised approach, the selection of k' cannot be guided by our GT. As such, the k' leading to good clusters may be different from the $k = 7$ used for supervised analysis.

We study the impact of k' considering the 30-day dataset and the corpus obtained through domain knowledge-based services. We run the graph construction and cluster detection for different k' and show the number of clusters (left y-axis, solid red curve) and the modularity (right y-axis, blue dotted curve) in Figure 10. Intuitively, connecting each IP address to only the nearest point creates many disconnected components in the graph, resulting in thousands of tiny clusters. With $k' > 1$, disconnected components start to get connected, resulting in less clusters. With $k' = 3$ (suggested by the

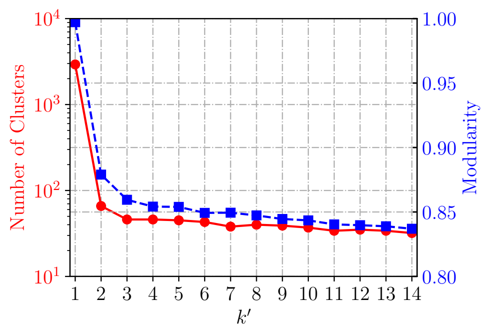


Figure 10: Impact of k' in cluster detection.

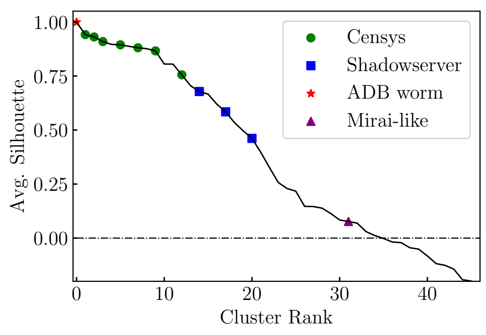


Figure 11: Average silhouette of points within the 46 clusters.

elbow method [38]), we obtain 46 clusters with high modularity. Larger values have little impact, only slightly decreasing the overall modularity. As such, we use $k' = 3$ in the results that follow.

Next, we illustrate the quality of the clusters using the silhouette. The silhouette measures how similar a sample in a cluster is to the other samples in the same cluster (cohesion), compared to how dissimilar the sample is to samples in the other clusters (separation). It takes values in the [-1:1] range. Positive values reflect good cohesion, while negative values suggest possible wrong assignments. Figure 11 shows the silhouette trend of the identified clusters, sorted by decreasing values. We compute the overall average silhouette of all senders belonging in each cluster. More than half of the clusters have silhouettes higher than 0.5, typically considered excellent results. Some clusters present negative silhouette. These cases can represent several situations, such as senders that do not exhibit any temporal patterns in common with other senders, or senders for which the embedding provides little information to characterize their activities (see Figure 9a).

Markers in Figure 11 indicate some notable clusters we detect. We describe and explore them in the following.

7.3 Cluster inspection

We manually analyse the clusters looking for interesting insights and possibly previously unknown classes of scanners. Similarly to the steps used to build the GT in the previous section, we rely on manual inspection, searching for explanations for the senders' activity in each group. To find evidence of the type of activity they perform, we collect reverse DNS hostnames, consult the `whois` and public security repositories, and fire HTTP requests to senders' IP addresses to check for "abuse" pages redacted by people running the scanners. Here, the availability of the groups of IP addresses dramatically simplifies the analysis. Recall that, by construction, IP addresses in the same group target similar destination ports in nearby periods of time.

Table 5 summarizes our findings. All in all, DarkVec identifies (i) well-known Internet scan projects, including those listed in our ground-truth, some not reported for brevity, (ii) Internet scan actions from security services, which were previously unknown to us, (iii) distributed scan events for which the observed patterns suggest coordination from botnets.

7.3.1 Sub-clusters in known scanners. Using a completely unsupervised approach, DarkVec identifies senders already present in our ground truth. However, DarkVec allows us to identify sub-groups inside the set of senders belonging to the same scanner. We provide an example using the Censys service. Recall that Censys targets more than 11 000 ports (Table 2) with 336 IP addresses seen in our data. DarkVec divides 111 of those senders into 7 groups.[9]

Figure 12 shows the temporal patterns of these 7 sub-clusters. The x-axis represents the time, y-axis presents senders ordered by the cluster IDs, and points mark the time in which the given sender is active. Patterns show that each group is formed by a similar number of IP addresses, but that are active in different periods. Not shown, each group targets a different set of ports too. To illustrate this, we compute the Jaccard Index considering the set of ports targeted by IP addresses belonging to pairs of clusters.[10] We obtain an average inter-cluster Jaccard Index of 0.19, i.e., only 19% of the target ports of one cluster is also a target port for another cluster.

In sum, DarkVec highlights a scan strategy employed by Censys, which deploys sets of scanners, each composed by a similar number of hosts, to search for particular services in the Internet.

7.3.2 Scanners from security services. DarkVec allows us to identify addresses belonging to Internet security services like *Shadowserver* [11]. This service was unknown to us, and indeed we have not included it during the manual labeling of our ground-truth. The service perform scans from its subnets and is listed in public security databases [1, 5, 7].

In details, we identify 113 *Shadowserver* senders belonging to the same /16 subnet that DarkVec divides into 3 clusters. All senders belong to the Shadowserver Foundation, which runs the scans for security purposes. Clusters in this case have less evident temporal patterns (Figure 13) than in the Censys case. Yet, DarkVec identifies

[9]The remaining Censys IP addresses have more sporadic presence and remain in noisy groups with low silhouette.
[10]The Jaccard Index is the ratio between the cardinality of the intersection over the cardinality of the union of two sets.

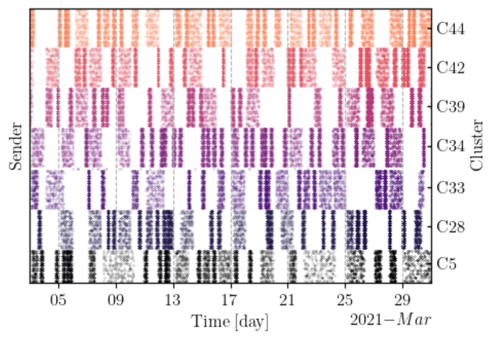


Figure 12: Activity patterns of Censys sub-clusters.

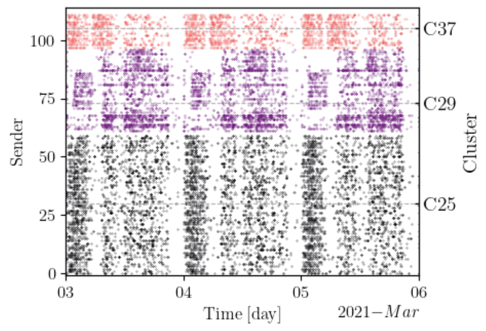


Figure 13: Activity patterns of Shadowserver sub-clusters.

that they target the same group of ports, but with very different intensity as follows:

- C25: 10% of packets to port 623/UDP, 10% to port 123/UDP;
- C29: 25% of packets to ports 5683/UDP or 3389/UDP;
- C37: 63% of packets to ports 111/UDP or 137/UDP.

7.3.3 New scanners unknown to security databases. DarkVec groups senders that are not present in open security databases or search engines. We cannot thus confirm the purpose of their actions. However, their activity patterns suggest the coordination of a large number of IP addresses, in some cases showing fingerprints that are compatible with botnets.

unknown1 *NetBIOS scan*: 85 IP addresses that belong to the same /24 subnet in the Cogent Communications address range. They sent more than 17 500 packet, 60% of which hit the NetBIOS port 137/UDP, with the very regular pattern as shown in Figure 14.

unknown2 *SMTP scan*: 10 IP addresses of the same /24 subnet in the Google cloud address space. These senders send 76% of their traffic to the SMTP port 25/TCP.

unknown3 *SMB scan*: 61 IP addresses scattered into 23 /24 subnets. These addresses target the SMB port 445/TCP with a very regular temporal pattern.

DarkVec finds large groups of senders belonging to multiple subnets in multiple countries and targeting few ports. These signs may indicate botnet activities performing a distributed scan.

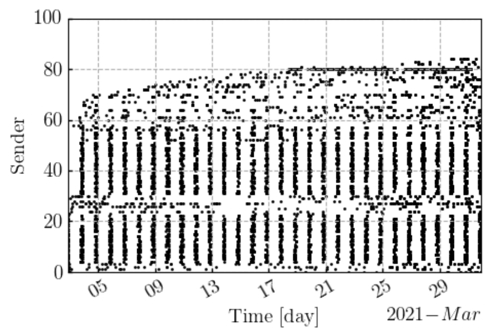


Figure 14: Activity pattern of a unknown1 NetBIOS scan from host in the same /24 subnet.

unknown4: 525 senders sending more than 350 000 packets. The 75% of them are directed toward port 5555/TCP, used by the Android Debug Bridge (ADB) service. The Internet Storm Center reports a spreading of an ADB worm in late March 2021[14]. DarkVec was able to spot some coordinated activity since the beginning of our trace, as Figure 15 shows. The increasing activity of the cluster is coherent with the reported malicious behavior, when DarkVec is able to increase the cluster size and clearly highlight an attack pattern.

unknown5 *Mirai-like scanners* has 1 412 IP addresses in 1 381 /24 subnets targeting 216 ports. Port 23/TCP accounts for 87.7% of traffic, followed by 2323/TCP (2%), 2000/UDP (1%). 71% of the senders send packets with the Mirai fingerprint, with the others that have a very similar pattern over time, without showing the Mirai fingerprint. This group illustrates the usefulness of DarkVec in extending the knowledge about botnets using darknet traffic.

unknown6 *SSH bots* 623 senders show patterns similar to brute-force attempts against SSH servers. Senders target mostly port 22/TCP. Manual verification using honeypot data we run in our premises confirms the brute-force activity performed by these senders.

unknown7: 158 senders that scan 148 ports with an almost equal share, with a very regular daily pattern, hinting to a botnet performing horizontal scans.

unknown8: second group of 22 senders scanning 69 ports with an almost equal share (target ports Jaccard index of 0.82), with a very regular hourly pattern.

While not exhaustive, this analysis shows the benefit of DarkVec in supporting the identification of new groups of senders performing coordinated actions.

8 DISCUSSION AND CONCLUSIONS

In this paper, we showed how word embedding can be used to shed light on noisy darknet traces. Properly parameterized to create services and sequences of senders, DarkVec can assist security analysts to extend their knowledge about ongoing scans and attacks. DarkVec automatically clusters IP addresses performing known activity (semi-supervised learning), and let previously unknown coordinated activity to emerge (unsupervised learning). Beyond the darknet traffic use case, our results and methodological insights can

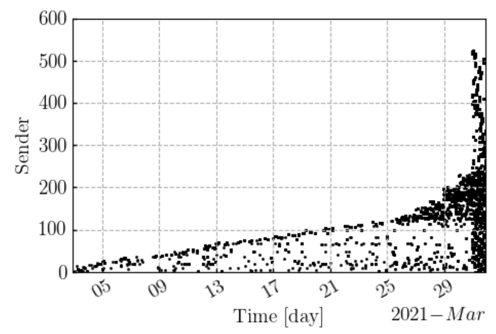


Figure 15: Activity pattern of an ADB mass scan like the spreading of an ADB worm.

Table 5: Summary of extracted coordinated senders.

Name/Type	Cluster	IP	Ports	*Sh*	Description
Censys known scanner sub-clusters	C5	14	19	0.91	Senders of the Censys ground truth class fall into different groups according to the set of ports they target.
	C28	16	21	0.94	
	C33	17	31	0.76	
	C34	16	25	0.87	
	C39	16	13	0.93	
	C42	16	27	0.88	
	C44	16	26	0.89	
ShadowServer known scanner sub-clusters	C25	61	47	0.68	Senders belonging to the ShadowServer /16 subnet and targeting the same set of ports.
	C29	36	42	0.46	
	C37	16	51	0.58	
unknown1 NetBios scanner	C40	85	18	0.62	Same /24 subnet in Congent Communications AS.
unknown2 SMTP scanner	C30	10	12	0.89	Same /24 subnet in the Google cloud. >1 600 packets, 76% to SMTP port 25/TCP.
unknown3 SMB scanner	C13	61	5	0.33	>10 900 packets (99.5% of group traffic) is directed to port 445/TCP.
unknown4 ADB massive scanner	C41	525	141	1.00	75% of traffic to 5555/TCP. The IPs activity pattern is coherent with the spreading of a known ADB worm. (Fig.15)
unknown5 Mirai-like massive scanner	C18	1412	212	0.08	71% of senders has Mirai fingerprint. The most of traffic is towards typical botnet ports 23/TCP and 2323/TCP (85%)
unknown6 SSH brute-force	C26	623	116	0.40	>400 000 packets. 88% of group traffic is directed to SSH port 22/TCP.
unknown7 Massive scanner	C31	158	148	0.03	Mostly 'Unknown'. Daily regular activity pattern. Equal share on 148 ports
unknown8 Massive scanner	C45	22	69	0.80	Mostly 'Unknown'. Regular pattern. Almost equal share on 69 ports

inspire the application of DarkVec to other sequences of categorical variables often present in networking data such as in honeypot traffic or brute-force attacks.

Compared to NLP algorithms [20, 29, 34] where the resulting embedding is general, a Word2Vec model trained through DarkVec is hardly generalizable. We expect that different datasets would generate different embeddings. Indeed, a trained model learns the time relationships among co-occurring senders within a certain observation period. Given the rapid changes in traffic, senders' behavior, and targeted services, we expect each dataset to be a different use case. On the contrary, embeddings generated from natural languages are generic thanks to the intrinsic static nature of the language, where the semantic and usage of words change very slowly. DarkVec is a powerful analysis tool to shed light on darknet traffic, but we believe it is not able to derive a generic model of senders behaviour, as such behaviour keeps changing.

This fact opens challenging questions, i.e., to what extent the embedding learned in one darknet can be useful in other darknets or at different time. There are at least two aspects to be addressed: the transfer of the embedding, and the transfer of learned tasks. The evolving nature of darknet traffic would hardly make the transfer possible over time, but it could eventually help to compare darknet traffic collected from different vantage points during the same time period. Yet, the darknets could have little overlap in terms of sources. The question thus is whether a model trained for one task (e.g., classify sources) on the embedding of one darknet can successfully achieve the same task on the other darknet. We plan to extend our results in these directions in the future.

ACKNOWLEDGEMENTS

The research leading to these results has been funded by the Huawei R&D Center (France) and the SmartData@PoliTO center for Big Data technologies. Luca Gioacchini has been supported by Consortium GARR.

REFERENCES

[1] 2021. AbuseIPDB - IP address abuse reports - Making the Internet safer, one IP at a time. https://www.abuseipdb.com/.
[2] 2021. The Best IP Geolocation Database: IPIP.NET. https://en.ipip.net/.
[3] 2021. BinaryEdge. https://www.binaryedge.io/.
[4] 2021. Censys. https://censys.io/.
[5] 2021. FireHOL IP Lists - IP Blacklists - IP Blocklists - IP Reputation. http://iplists.firehol.org/.
[6] 2021. Gensim, Topic modelling for humans. https://radimrehurek.com/gensim/.
[7] 2021. GreyNoise. https://greynoise.io/.
[8] 2021. Internet Census Group. https://www.internet-census.org/home.html.
[9] 2021. Michigan Engineering - University of Michigan College of Engineering. https://www.engin.umich.edu/.
[10] 2021. Project Sonar. https://www.rapid7.com/research/project-sonar/.
[11] 2021. The Shadowserver Foundation. https://www.shadowserver.org/.
[12] 2021. Sharashka Data Feeds - Security Data That Works. https://sharashka.io/data-feeds.
[13] 2021. Shodan, the search engine. https://www.shodan.io/.
[14] 2021. SPort 5555 (tcp/udp) Attack Activity. https://isc.sans.edu/port.html?port=5555.
[15] 2021. Stretchoid Opt-Out. http://www.stretchoid.com/.
[16] Charu C Aggarwal. 2015. *Data mining: the textbook.* Springer.
[17] K. Benson, A. Dainotti, K. Claffy, A. Snoeren, and M. Kallitsis. 2015. Leveraging Internet Background Radiation for Opportunistic Network Analysis. In *Proceedings of the ACM Internet Measurement Conference (IMC'15).* 423–436. http://dl.acm.org/citation.cfm?doid=2815675.2815702
[18] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.
[19] Joao Ceron, Klaus Steding-Jessen, Cristine Hoepers, Lisandro Granville, and Cintia Margi. 2019. Improving IoT Botnet Investigation Using an Adaptive Network Layer. *Sensors* 19 (02 2019), 727. https://doi.org/10.3390/s19030727
[20] Dvir Cohen, Yisroel Mirsky, Yuval Elovici, Rami Puzis, Manuel Kamp, Tobias Martin, and Asaf Shabtai. [n.d.]. DANTE: A Framework for Mining and Monitoring Darknet Traffic. 88–109. http://arxiv.org/abs/2003.02575
[21] A. Dainotti, A. King, K. Claffy, F. Papale, and A. Pescape. 2015. Analysis of a "/0" Stealth Scan From a Botnet. *IEEE/ACM Trans. Netw.* 23, 2 (2015), 341–354.
[22] Z. Durumeric, M. Bailey, and J. Halderman. 2014. An Internet-Wide View of Internet-Wide Scanning. In *Proceedings of the 23rd USENIX Conference on Security Symposiu (SEC'14).* 65–78. http://dl.acm.org/citation.cfm?id=2671225.2671230
[23] C. Fachkha, E. Bou-Harb, and M. Debbabi. 2015. Inferring Distributed Reflection Denial of Service Attacks from Darknet. *Comput. Commun.* 62, C (2015), 59–71.
[24] C. Fachkha and M. Debbabi. 2016. Darknet as a Source of Cyber Intelligence: Survey, Taxonomy, and Characterization. *Commun. Surveys Tuts.* 18, 2 (2016), 1197–1227.
[25] J. Fruhlinger. 2018. The Mirai botnet explained: How teen scammers and CCTVcameras almost brought down the internet. https:

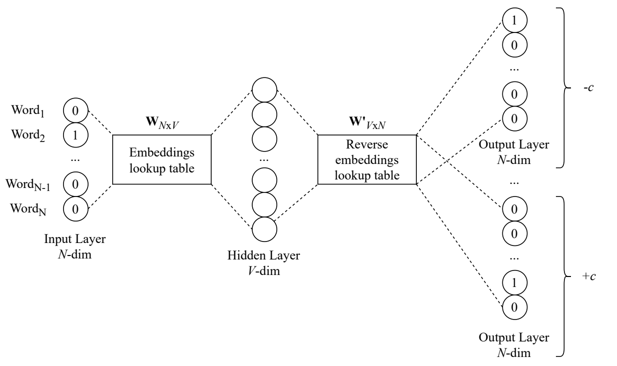


Figure 16: Word2Vec skip-gram architecture.

//www.csoonline.com/article/3258748/\the-mirai-botnet-explained-\how-teen-scammers-and-cctv-cameras-almost-brought-down-the\-internet.html.

[26] M. Jonker, A. King, J. Krupp, C. Rossow, A. Sperotto, and A. Dainotti. 2017. Millions of Targets Under Attack: A Macroscopic Characterization of the DoS Ecosystem. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference (IMC'17)*. 100–113. http://dl.acm.org/citation.cfm?doid=3131365.3131383

[27] Sofiane Lagraa, Yutian Chen, and Jérôme François. 2019. Deep Mining Port Scans from Darknet. *International Journal of Network Management* 29, 3 (2019), e2065. https://doi.org/10.1002/nem.2065

[28] Sofiane Lagraa and Jérome François. 2017. Knowledge discovery of port scans from darknet. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 935–940.

[29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781 [cs.CL]

[30] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. 2013. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168* (2013).

[31] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. arXiv:1310.4546 [cs.CL]

[32] D. Moore, C. Shannon, D. Brown, G. Voelker, and S. Savage. 2006. Inferring Internet Denial-of-Service Activity. *ACM Trans. Comput. Syst.* 24, 2 (2006), 115–139.

[33] Symeon Papadopoulos, Yiannis Kompatsiaris, Athena Vakali, and Ploutarchos Spyridonos. 2012. Community detection in social media. *Data Mining and Knowledge Discovery* 24, 3 (2012), 515–554.

[34] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543. http://www.aclweb.org/anthology/D14-1162

[35] E. Raftopoulos, E. Glatz, X. Dimitropoulos, and A. Dainotti. 2015. How Dangerous Is Internet Scanning? A Measurement Study of the Aftermath of an Internet-Wide Scan. In *Proceedings of the 7th Workshop on Traffic Monitoring and Analysis (TMA'15)*. 158–172. http://link.springer.com/10.1007/978-3-319-17172-2_11

[36] P. Richter and A. Berger. 2019. Scanning the Scanners: Sensing the Internet from a Massively Distributed Network Telescope. In *Proceedings of the Internet Measurement Conference (IMC'19)*. 144–157. http://dl.acm.org/doi/10.1145/3355369.3355595

[37] Markus Ring, Alexander Dallmann, Dieter Landes, and Andreas Hotho. 2017. IP2Vec: Learning Similarities Between IP Addresses. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. 657–666. https://doi.org/10.1109/ICDMW.2017.93

[38] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. 2017. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Trans. Database Syst.* 42, 3, Article 19 (July 2017), 21 pages. https://doi.org/10.1145/3068335

[39] F. Soro, M. Allegretta, M. Mellia, I. Drago, and L. Bertholdo. 2020. Sensing the Noise: Uncovering Communities in Darknet Traffic. In *Proceedings of the Mediterranean Communication and Computer Networking Conference (MedComNet)*. 1–8. https://ieeexplore.ieee.org/document/9191555/

[40] F. Soro, I. Drago, M. Trevisan, M. Mellia, J. Ceron, and J. J. Santanna. 2019. Are Darknets All The Same? On Darknet Visibility for Security Monitoring. In *Proceedings of the IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. 1–6. https://ieeexplore.ieee.org/document/8847113/

[41] S. Staniford, D. Moore, V. Paxson, and N. Weaver. 2004. The Top Speed of Flash Worms. In *Proceedings of the ACM Workshop on Rapid Malcode (WORM'04)*. http://portal.acm.org/citation.cfm?doid=1029618.1029624

A APPENDIX

A.1 Word2Vec

Word2Vec [29, 31] is a NLP technique based on artificial neural networks. It allows to map words (*tokens*) of text sentences (*corpora*) into a latent space as a real-valued array (the *embedding*), such that words belonging to similar contexts have similar embedding.

The core element of the Word2Vec model is the *context*. It is defined as the sequence of words surrounding the one for which the embedding must be generated. The number of words to consider in the context is specified by the context window size c (see Section 5.3). For example, by considering the sentence *'Chicago is a great city'*, if the word *'a'* is the target one and $c = 2$, the context for *'a'* is the list of the 2 previous and 2 following words of *'a'*:

$$\textit{'a'} \rightarrow (\textit{'Chicago', 'is', 'great', 'city'})$$

To generate the embedding, Word2Vec relies on two possible architectures: skip-grams and Continuous Bag Of Words (CBOW). Since we work with the skip-gram architecture, for the sake of simplicity we omit the description of CBOW. By considering a corpus with N distinct words, the model aim at predicting the probability of finding each one of the N words within the context window of a given target word. In Figure 16 we report an overview of the skip-gram architecture. Each word of the sentences is fed as input to the model through a one-hot-encoded input layer. The V-dimensional hidden layer links all the $2c$ context words to the target one. After the model training, the embedding is obtained from the weights matrix $\mathbf{W} \in \mathbb{R}^{N\times V}$. Each of the $i \in \{1, \ldots, N\}$ entries of $\mathbf{W}$ is the embedding in $\mathbb{R}^{1\times V}$ associated to the i-th word.

A.2 DANTE and IP2VEC

Here we provide additional details about DANTE and IP2VEC methodologies.

A.2.1 DANTE. Authors of DANTE [20] provide a framework for analysing darknet traffic through Word2Vec. Rather than generating embeddings directly from senders IP address, they adopt a port-based approach.

Starting from raw darknet traces, DANTE aggregates the sequence of destination ports by (sender, receiver) pairs. Thus DANTE treats each pair as an independent language and the destination ports represents words in the sentences. After training the model, each sender is associated to the sequence of ports it targets. In this way, the final IP embedding is determined as the average embedding among them.

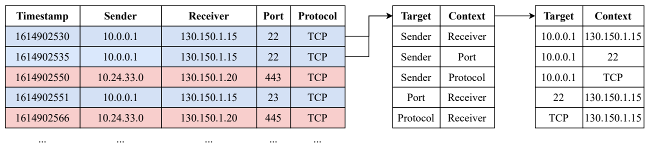
DANTE needs a different language for each pairs (sender, receiver) and the embedding for each language must be trained. This approach does not scale well. Indeed, even by filtering out senders generating a few packets within an observation window, the amount of senders treated as independent word sequences is of the order tens of thousands (Figure 1b) and it largely grows over time. This implies long computational times to process large datasets. Thus, authors suggest to train the model only on framework setup and then use it in different time windows. However, we believe that such a solution strongly decreases the model performance: Given the highly dynamic nature of darknet traffic (see Section 3), the embedding associated to each port should be frequently updated.

Table 7: Domain-knowledge-based service definition used for generating the Word2Vec corpus.

Service	Internet Port/Protocol
Telnet	23/tcp, 992/tcp
SSH	22/tcp
Kerberos	88/tcp, 88/udp, 543/tcp, 544/tcp, 749/tcp, 7004/tcp, 750/udp, 750/tcp, 751/tcp, 752/udp, 754/tcp, 464/udp, 464/tcp
HTTP	80/tcp, 443/tcp, 8080/tcp
Proxy	1080/tcp, 6446/tcp, 2121/tcp, 8081/tcp, 57000/tcp
Mail	25/tcp, 143/tcp, 174/tcp, 209/tcp, 465/tcp, 587/tcp, 110/tcp, 995/tcp, 993/tcp
Database	210/tcp, 5432/tcp, 775/tcp, 1433/tcp, 1433/udp, 1434/tcp, 1434/udp, 3306/tcp, 27017/tcp, 27018/tcp, 27019/tcp, 3050/tcp, 3351/tcp, 1583/tcp
DNS	853/tcp, 853/udp , 5353/udp , 53/tcp, 53/udp
Netbios	137/tcp, 137/udp, 138/tcp, 138/udp, 139/tcp, 139/udp
Netbios-SMB	445/tcp
P2P	119/tcp, 375/tcp, 425/tcp, 1214/tcp, 412/tcp, 1412/tcp, 2412/tcp, 4662/tcp, 12155/udp, 6771/udp, 6881/udp, 6882/udp, 6883/udp, 6884/udp, 6885/udp, 6886/udp, 6887/udp, 6881/tcp, 6882/tcp, 6883/tcp, 6884/tcp, 6885/tcp, 6886/tcp, 6887/tcp, 6969/tcp, 7000/tcp, 9000/tcp, 9091/tcp, 6346/tcp, 6346/udp, 6347/tcp, 6347/udp
FTP	20/tcp, 21/tcp, 69/udp, 989/tcp, 990/tcp, 2431/udp, 2433/udp, 2811/tcp, 8021/tcp
Unknown System	All ports in the [0, 1023] range not classified as before
Unknown User	All ports in the [1024, 49151] range not classified as before
Unknown Ephemeral	All ports in the [49152, 65535] range not classified as before

Table 6: Baseline 7-NN classifier report. Values below 0.50 are highlighted in red.

	Precision	Recall	F-Score	Support
Mirai-like	0.97	1.00	0.98	7351
Censys	0.83	**0.42**	0.56	336
Stretchoid	**0.43**	**0.03**	**0.05**	104
Internet-census	0.50	0.67	0.57	103
Binaryedge	0.97	0.67	0.80	101
Ipip	**0.00**	**0.00**	**0.00**	49
Sharashka	0.94	**0.32**	**0.48**	50
Shodan	0.50	**0.13**	**0.21**	23
Engin-umich	0.71	1.00	0.83	10
Unknown	–	**0.36**	–	14272



Timestamp	Sender	Receiver	Port	Protocol
1614902530	10.0.0.1	130.150.1.15	22	TCP
1614902535	10.0.0.1	130.150.1.15	22	TCP
1614902550	10.24.33.0	130.150.1.20	443	TCP
1614902551	10.0.0.1	130.150.1.15	23	TCP
1614902566	10.24.33.0	130.150.1.20	445	TCP
...	...	...	...	...

Target	Context
Sender	Receiver
Sender	Port
Sender	Protocol
Port	Receiver
Protocol	Receiver

Target	Context
10.0.0.1	130.150.1.15
10.0.0.1	22
10.0.0.1	TCP
22	130.150.1.15
TCP	130.150.1.15

Figure 17: Overview of IP2VEC custom context.

A.2.2 IP2VEC. Authors of IP2VEC [37] propose an approach based on Word2Vec to highlight similarities among IP addresses in general-purpose network traces. Rather than adopting the canonical Word2Vec approach with sequence of tokens, authors define a custom context relying on information aggregated at flow level. Namely, for each sender observed in a traffic trace, they focus on (i) the receiver IP address, (ii) the destination port and (iii) the used transmission protocol. In this way, the target words are not only the senders IP addresses or the destination ports (like DANTE), but all fields in the flow definition. Afterwards authors generate 5 distinct pairs of tokens in the form *(target word, context word)*. In Figure 17 we report IP2VEC definition of the custom context for a flow starting from raw data.

For managing such data type, IP2VEC relies on *negative sampling* [31]. Rather than predicting the probability of all the words being in the same context of the target one, with negative sampling, the skip-gram model predicts if a pair of words belongs to the same context (or they are $2c$-neighbors in a sequence), where c is the context window size.

From a practical point of view, this reduces IP2VEC scalability, as the authors themselves confirm. Indeed, even though the amount of data is reduced through the flow-level aggregation, because of negative sampling, the model must be trained on words belonging to the same context and an additional percentage of words belonging to different ones. Thus, in large datasets, IP2VEC may be too expensive in terms of computational times.

A.3 Domain knowledge based service definition

For the sake of completeness, in Table 7 we report all the services we define in the *domain knowledge-based service* experiment.