i-DarkVec: Incremental Embeddings for Darknet Traffic Analysis

i-DarkVec: Incremental Embeddings for Darknet Traffic Analysis

LUCA GIOACCHINI, LUCA VASSIO, and MARCO MELLIA, Politecnico di Torino, Italy
IDILIO DRAGO, Università di Torino, Italy
ZIED BEN HOUIDI and DARIO ROSSI, Huawei Technologies Co. Ltd, France

Darknets are probes listening to traffic reaching IP addresses that host no services. Traffic reaching a darknet results from the actions of internet scanners, botnets, and possibly misconfigured hosts. Such peculiar nature of the darknet traffic makes darknets a valuable instrument to discover malicious online activities, e.g., identifying coordinated actions performed by bots or scanners. However, the massive amount of packets and sources that darknets observe makes it hard to extract meaningful insights, calling for scalable tools to automatically identify and group sources that share similar behaviour.

We here present i-DarkVec, a methodology to learn meaningful representations of Darknet traffic. i-DarkVec leverages Natural Language Processing techniques (e.g., Word2Vec) to capture the co-occurrence patterns that emerge when scanners or bots launch coordinated actions. As in NLP problems, the embeddings learned with i-DarkVec enable several new machine learning tasks on the darknet traffic, such as identifying clusters of senders engaged in similar activities.

We extensively test i-DarkVec and explore its design space in a case study using real darknets. We show that with a proper definition of *services*, the learned embeddings can be used to (i) solve the classification problem to associate unknown sources' IP addresses to the correct classes of coordinated actors and (ii) automatically identify clusters of previously unknown sources performing similar attacks and scans, easing the security analyst's job. i-DarkVec leverages a novel incremental embedding learning approach that is scalable and robust to traffic changes, making it applicable to dynamic and large-scale scenarios.

CCS Concepts: • **Security and privacy → Network security**; • **Networks → Network management**; **Network monitoring;**

Additional Key Words and Phrases: Word2Vec, Network Measurements, darknet

ACM Reference format:
Luca Gioacchini, Luca Vassio, Marco Mellia, Idilio Drago, Zied Ben Houidi, and Dario Rossi. 2023. i-DarkVec: Incremental Embeddings for Darknet Traffic Analysis. *ACM Trans. Internet Technol.* 23, 3, Article 45 (August 2023), 28 pages.
https://doi.org/10.1145/3595378

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. This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.
Authors' addresses: L. Gioacchini, L. Vassio, and M. Mellia, Politecnico di Torino, Italy; emails: luca.gioacchini@polito.it, luca.vassio@polito.it, marco.mellia@polito.it; I. Drago, Università di Torino, Italy; email: idilio.drago@unito.it; Z. B. Houidi and D. Rossi, Huawei Technologies Co. Ltd, France; emails: zien.ben.houidi@huawei.com, dario.rossi@huawei.com.

1533-5399/2023/08-ART45 $15.00
https://doi.org/10.1145/3595378

1 INTRODUCTION

Darknets are sensors that observe traffic received by networks that are announced on the Internet but host neither production services nor client hosts [31]. Such unsolicited packets represent a privileged source of information for network security and debugging activities [30, 37], exposing threats such as scans, brute-force attempts, and misconfigured hosts [24].

Darknets receive traffic from hundreds of thousands of sources targeting all TCP/UDP ports. Each received packet can be mapped into three main dimensions: (i) the target service, coarsely represented by the transport protocol and destination port; (ii) the time of arrival; and (iii) the space, represented by the sender source IP address. All dimensions are highly variable, with new sources arriving over time, others disappearing and reappearing with different ports as targets, with different sending rates, and so on. Some sources are part of coordinated efforts to scan the Internet, with *groups of sources* that take part in the same action. These groups include botnets looking for vulnerable machines, scanners from security companies (and researchers) that build maps of the Internet, and misconfigured hosts that contact the darknet in the search for a non-existent service. In addition, attackers may use spoofed IP addresses—randomly picking addresses in the darknet space—so the victims' responses may reach the darknet. This superposition of patterns makes it difficult to identify relevant events in such a diverse aggregate.

This article aims at automatically extracting actionable information from darknet traffic. We present i-DarkVec,[1] an improved version of DarkVec that we previously introduced in Reference [34]. In a nutshell, i-DarkVec is a scalable and robust methodology to process darknet traffic and automatically extract complex patterns from raw darknet packet traces, as illustrated in Figure 1. It borrows techniques from **Natural Language Processing (NLP)**, making the parallel between words that co-occur in sentences and documents and sources that co-occur in time in the sequence of packets received by a darknet. Specifically, we rely on a word embedding algorithm, which is a recognised 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. This rich representation allows one to exploit machine learning algorithms [41], e.g., to find words with similar semantic, to translate texts, to suggest the next word in a sentence and so on [42].

In a similar way, i-DarkVec *learns a representation* for the darknet traffic, projecting sources reaching the darknet in a latent space. i-DarkVec uses darknet traffic to define *sentences* as temporal *sequences* of sender source IP addresses. The sender source IP addresses are treated as words in sentences. Given a time interval, packets directed to different *services* form separated sequences.

The set of all sequences defined by all services forms our *corpus*, which we process to create our embeddings with the NLP algorithms. As for NLP applications, i-DarkVec opens the way for the execution of several machine learning tasks in the embedded space, such as classifying and clustering sources presenting similar behaviour, as well as identifying new sources that present anomalous behaviour.

We systematically explore the design space of i-DarkVec and show its capabilities with a comprehensive set of experiments run using darknet traces. We find that the embeddings produced by i-DarkVec map senders that perform similar activity in the same latent space regions. In detail, we employ i-DarkVec embeddings in two specific machine-learning tasks. First, we show that i-DarkVec is instrumental to solve a classification problem, i.e., i-DarkVec correctly assigns sources to known groups with 96% accuracy. More newsworthy, we show that i-DarkVec embeddings can be used for successfully identifying previously unknown clusters of sources that we identify

[1] *i* stands for incremental.

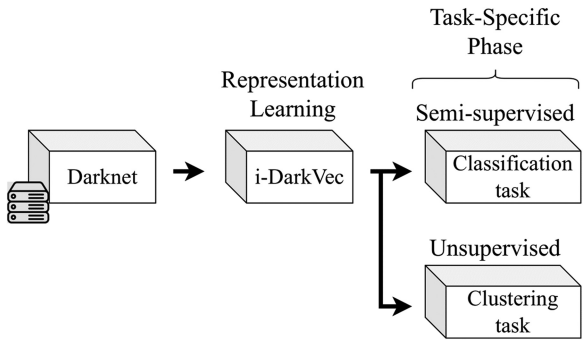


Fig. 1. High-level view of i-DarkVec with two possible follow-up tasks.

as new groups of scanners. In a nutshell, i-DarkVec embeddings feed clustering algorithms that uncover several novel groups of sources that were not reported in popular security databases.

Overall, we show how word embeddings shed light on noisy darknet traces. Beyond the darknet traffic use case, we hope our results and methodological insights can inspire the application of i-DarkVec to the analysis of other network traffic traces, too. For that, we release i-DarkVec source code and an anonymised version of the dataset used in the article.[2]

As previously said, i-DarkVec extends our prior work [34] that explored the use of word embeddings to learn informative representations from noisy Darknet traffic. In this article, we (i) extend the methodology to perform incremental training, with fast learning performance and without loss in accuracy, which eases the application of i-DarkVec in practical scenarios; (ii) compare different corpus definitions, improving the performance when compared to Reference [34]; and (iii) show the usefulness of i-DarkVec in supporting the security analyst's job in the identification of unknown coordinated actions; (iv) provide real use cases of exploration of clusters, shedding lights on new senders' activities in our darknet that were not previously documented in public security feeds.

After discussing related work (Section 2) and the properties of observed darknet traffic (Section 3), we describe i-DarkVec service and embedding definition (Section 4). We then show how representative the embeddings are (Section 5) and how to extract clusters and new patterns (Section 6). We conclude the article by summarising our findings (Section 7).

2 RELATED WORK

The literature is rife with studies that analyse darknet traffic for various purposes. Data coming from darknets have helped profiling attack strategies [30, 37, 44, 53], detecting and characterising Internet scans [28, 29, 48], and studying malware spread [54]. For instance, authors of References [35, 55] rely on genetic algorithms to automatically modify the fingerprints of attacks and automatically identify new variants. Such prior studies rely on ad hoc algorithms to characterise darknet traffic and have repeatedly proved the value of such networks. Here, we extend our previously proposed methodology [34] to automate such an analysis, applying word embedding to simplify the uncovering of groups of senders contacting darknets. As introduced in the previous section, we extend the methodology to increase its performance and make it incremental, allowing its application in practical deployment scenarios.

Darknet traffic has been modelled using complex networks. Authors of References [39, 40] adopt such an approach, modelling the traffic as a graph to detect transport-layer ports co-targeted by

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

scanners. Authors of Reference [52] build a bipartite graph for representing darknet traffic and then apply community detection to it, obtaining clusters of autonomous systems characterised by similar behaviour.

Authors of References [38, 47] rely on deep autoencoders to represent darknet traffic sources using internal darknet features (i.e., amount of generated traffic) and external information (i.e., querying Censys [8] online databases). They apply traditional clustering algorithms on the obtained source representation to spot senders' similarities. These approaches are complementary to i-DarkVec, as they focus on particular traffic features. We propose an approach that puts together multiple dimensions, e.g., finding patterns characterised 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 [27], IP2VEC [50], and DarkVec [34]. All apply Word2Vec [41, 43] to extract features from traffic traces. DANTE and DarkVec focus on darknets as a traffic source, whereas IP2VEC is a more generic flow-level traffic analysis methodology.

DANTE exploits the sequence of ports that each sender targets within an observation window. The authors treat the sequence of the ports reached by senders as a sequence of words in an NLP problem. They then build a separate Word2Vec embedding for each port. Finally, each sender IP address is associated with a vector by averaging the embeddings of the ports that the sender has contacted. The outcome is analysed with standard clustering algorithms. In a similar yet more generic direction, IP2VEC embeds IP addresses (and also ports and protocols) by building Word2Vec models that consider as words the sequence of flow-level variables, such as destination IP addresses, port numbers, and the used transmission protocols.

In Reference [34], we already showed that DarkVec outperforms both DANTE and IP2VEC in classification problems while reducing the computational time required to compute the embeddings (with DANTE not being even able to complete the embedding computation). Thanks to the novel incremental approach we introduce here, i-DarkVec further reduces the computation time needed to build the embeddings.

All in all, i-DarkVec makes it practical to apply Word2Vec to large darknet setups. Our thorough exploration of word embeddings design space, including service definitions and the choice of parameters, provides the guidelines for the application of word embeddings to other use cases where the goal is to automatically learn from sequences of categorical variables (e.g., log events, DNS queries, or HTTP requests).

3 DATASET AND BASELINE MODELS

We set up a /24 darknet in the IP range of a university campus network and use it for running experiments. We focus on 30 days of traffic covering the period from 2021-03-02 to 2021-03-31, the same we used in Reference [34] to simplify the comparison. In Table 1, we detail statistics about the dataset. From the 30 days of data, we consider the first 29 for training algorithms and keep the last day as an independent test set. The darknet observes several thousands of distinct sender IP addresses that send several tens of millions of packets in total. We observe packets sent to all ports, with a very skewed distribution in which the top targeted ports get a high percentage of packets, from tens of thousands of sources.

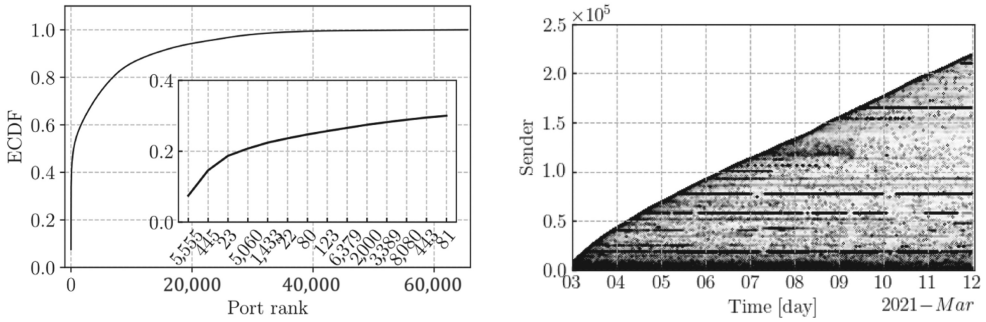
3.1 Darknet Traffic Overview

Figure 2 gives an overview of the traffic the darknet observes along the (i) service, (ii) space, and (iii) time dimensions. In detail, Figure 2(a) reports the **Empirical Cumulative Distribution Function (ECDF)** of the number of packets received by each port in one month.[3] As already observed, all

[3]UDP and TCP ports are added together for simplicity.

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.

Fig. 2. Darknet traffic overview.

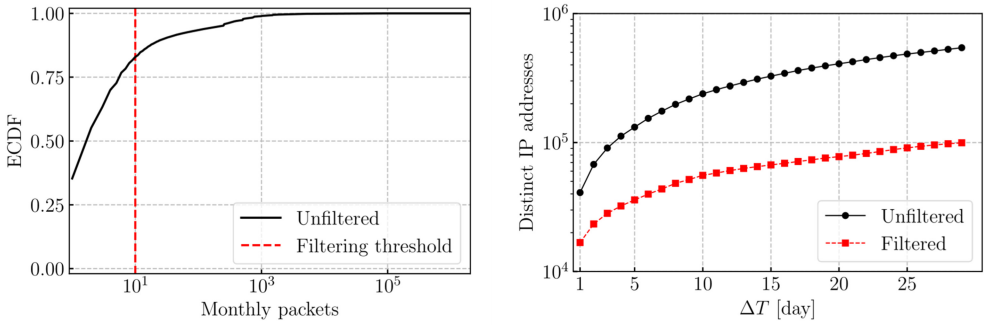
ports get some unsolicited packets, and most traffic is directed to specific ports, as detailed in the inset showing the top-14 ports—each easily linked to well-known services or vulnerabilities.[4]

Figure 2(b) showcases the activity of each sender over time. On the y-axis, each line represents a given sender IP address, sorted by the timestamp of the first appearance in the trace. The x-axis represents time. A dot is a packet received from a sender at a given time. In the first 10 days, we observe more than 220,000 senders sending about 1M packets. As expected [31], we witness a continuous growth of the number of senders over time. Some senders are persistently present (long horizontal lines); some senders appear sporadically, then disappear and reappear (horizontal segments); some senders are seldom visible (sparse dots).

To complete the overview, Figure 3(a) 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., the backscatter phenomenon [37, 49]. Yet, there exist many senders that are quite active. We keep in our analysis only senders that send 10 or more packets to the darknet in the considered period. We call them *active senders*, for which we have enough evidence to perform a reliable analysis. These senders (20% of the total) are responsible for the majority of the darknet traffic.

At last, Figure 3(b) shows the count of distinct IP addresses seen over an increasing period of time. On the first day, we observe about 40,000 distinct senders. This figure quickly grows over time so 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 characterise 100,000 active senders.

[4]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.

Fig. 3. Senders characterisation and filtering criteria definition.

Table 2. Ground Truth Classes—*active senders* Observed in the Last Day of the Collection

Label	Source	Senders	Packets	Ports	Top-3 Ports (% Traffic)
GT1	Mirai-like [32]	7,351	88,192	75	23/TCP (89.6%), 2323/TCP(3.9%), 5555/TCP(1.7%)
GT2	Censys [8]	336	233,004	11,118	5060/TCP(3.4%), 2000/TCP(2.9%), 443/TCP(0.4%)
GT3	Stretchoid [22]	104	57,144	91	22/TCP(3.5%), 443/TCP(3.5%), 21/TCP(2.7%)
GT4	Internet Census [14]	103	9,396	231	5060/TCP(10.4%), 161/UDP(9.8%), 2000/TCP(7.7%)
GT5	BinaryEdge [7]	101	7,646	21	15/TCP(10%), 3000/TCP(9.6%), 4222/TCP(6.7%)
GT6	Sharashka [19]	50	5,436	485	5986/TCP(0.48%), 2103/TCP(0.48%), 2,052/TCP(0.44%)
GT7	Ipip [6]	49	17,342	41	5060/TCP(41.5%), ICMP(10.9%), 8000/TCP(2.3%)
GT8	Shodan [20]	23	13,566	349	443/TCP(0.9%), 80/TCP(0.9%), 2222/TCP(0.9%)
GT9	Engin-umich [15]	10	506	1	53/UDP(100%)
Unknown	–	14,272	2,971,687	10,520	445/TCP(9.4%), 5555/TCP(9.4%), 1433/TCP(1.8%)
Total		22,399	3,403,959	19,882	445/TCP(8.3%), 5555/TCP(8%), 23/TCP(3.5%)

3.2 Ground Truth

One of the main difficulties in automating the analysis of darknet traffic is the lack of ground truth for evaluating the results. i-DarkVec's aim is to learn meaningful representations that allow us to identify senders performing similar activities over time. We validate i-DarkVec using our domain knowledge to label senders that perform coordinated activities that can be observed in the darknet. In particular, we exploit two sources of data: (i) the presence of the widely known Mirai-like malware(s) fingerprint [26, 32] in packets; (ii) our knowledge about security search engines and research projects such as Shodan [20] and Sonar [17] that make publicly available the IP addresses they use.

As said, in our 30-day-long dataset, we observe about 100,000 unique active senders. The manual labelling of all of them is unfeasible. We thus focus on the most active senders that are present on the last day of our collection, which we use as a labelled dataset in our validation. Here, we observe 22,399 active senders in total. Among these senders, we identify nine **ground truth (GT)** classes, summarised in Table 2. We identify senders that are 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 [8] that target more than 11,000 unique destination ports. The smallest groups include 10 senders of the Engin-umich project [15] that perform scans focusing on DNS (port 53/UDP) only. 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.

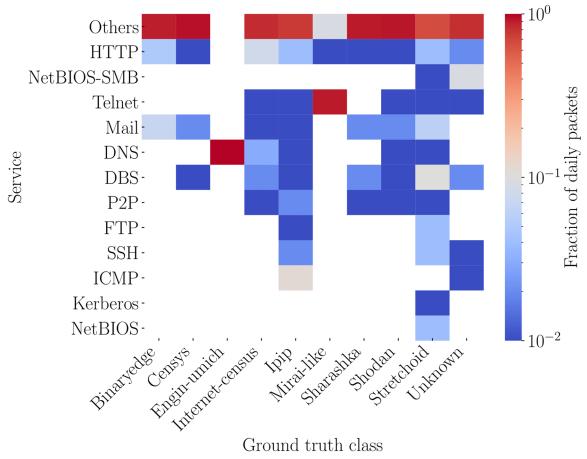


Fig. 4. Fraction of daily packets sent to generic services, normalised by columns.

3.3 Baseline Definition

To motivate the need for advanced representations of the darknet traffic, we set up a simple classification task that directly leverages traffic-related features to classify senders reaching the darknet. We consider simple features, such as top-destination ports, numbers of packets, and others. Intuitively, one could argue that classifying senders using such features could already lead to the identification of coordinated groups without the need of projecting senders in a latent space with our proposed embeddings.

To illustrate and check this intuition, Figure 4 shows the fraction of daily packets sent by senders of each ground truth class to generic services. Here, we identify a service (y-axis) from the ports typically used by the service, e.g., we label traffic with destination ports $\{25, 110, 143, 587, 993, 995\}$ as "Mail," while ports $\{80, 443, 8080\}$ identify HTTP. Fractions in the heatmap are normalised by column. The heatmap clearly shows that a naive port-based approach would not be able to classify all traffic. Some services can be characterised by their focus on a single service, such as the Engin-umich group, which generates only DNS traffic. Yet, other groups produce DNS traffic, too, and these overlaps are hard to solve based only on such features.

Indeed, to check whether it would be possible to group senders with a supervised approach, we design a classifier that uses as features the fraction of traffic each sender generates to top destination ports. We take the last day of traffic and label senders according to the 10 classes in Table 2, i.e., the 9 GT classes and the "Unknown" class. For each class, we extract the top-5 ports in terms of packets and merge the obtained ports as features in a single set. We then project each sender to the feature space by computing the percentage of traffic it sends to each selected port.[5]

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 neighbours. We use the cosine distance to identify the k nearest neighbours. 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 $k = 7$. Table 3 details the results. Overall, the performance is quite unsatisfactory. Only for the most popular class (Mirai-like) the classifier can

[5]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. In total, we obtain a space of 28 features for the 10 classes.

Table 3. Baseline 7-NN Classifier Report

	Precision	Recall	F-Score	Support
Mirai-like	0.97	1.00	0.98	7,351
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

Values below 0.50 are highlighted in red.

provide good results, but already for the Censys class—the second most popular one—the recall is very low.

We have also tested other classification models, with similar unsatisfactory results [33]. This negative result strengthens the intuition that an approach based on simplistic features is insufficient. We will show later that i-DarkVec approach, which exploits the temporal information present in the data, can provide a better representation of the darknet traffic and enable the clustering of senders on the obtained latent space.

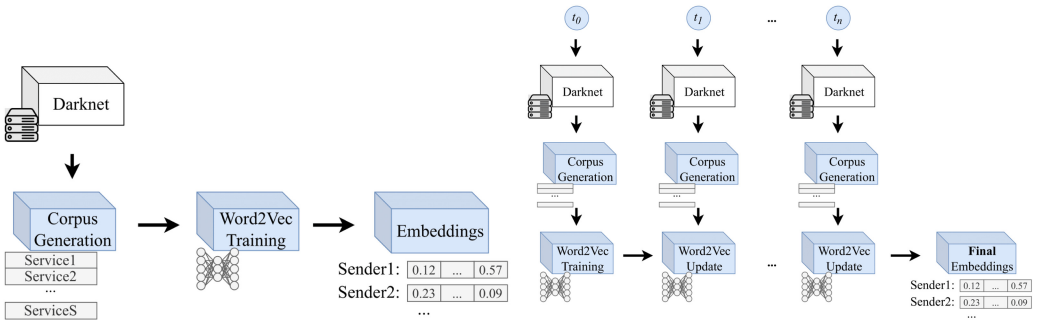
4 I-DARKVEC

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

4.1 i-DarkVec in a Nutshell

To process the aggregate traffic received by a darknet, i-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 latent space, associating a rich feature vector to each word. Although they were built exploiting the co-occurrence of words, these vectors end up encoding the 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 sender's IP address to a vector in the embedding space.

Unlike natural language where a large corpus with all known words is used to pre-build word embeddings in a single pass, i-DarkVec does not see all IP addresses at once, but incrementally discovers them as they hit the darknet (e.g., new senders that start sending traffic or new threats that emerge). In Reference [34], we introduced DarkVec, a single-iteration algorithm that builds a single, static, and immutable embedding space from all the data observed in a darknet. This is what is commonly done in natural language problems. The incremental version of DarkVec presented in this article (i-DarkVec) instead builds and continuously updates the embeddings as new traffic arrives. Figure 5 provides the high-level overview of both methods. In both cases, we first collect packets from a darknet (the top part of the plots). Given packets observed during a time period, we extract separate sequences of senders, considering the services they target. We identify senders by their source IP addresses and define services based on destination ports in the packets. Next, we use such sequences of senders per service to train a Word2Vec model. The original version of DarkVec (Figure 5(a)) uses all data to define a corpus that is then used to build the embeddings



(a) Single training as in the original DarkVec. (b) Novel incremental training in i-DarkVec.

Fig. 5. Comparison of different approaches.

from scratch using the Word2Vec algorithm. The new i-DarkVec conversely receives as input new batches of traffic as time evolves. After building the corpus for each new batch, it updates the previous model to obtain the current embeddings (Figure 5(b)).

In both cases, we map the categorical IP addresses into a multidimensional space using 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 embeddings. The resulting embeddings can be used to solve semi-supervised and unsupervised machine-learning tasks that exploit the correlations in the embeddings.

More in detail, Figure 5(a) illustrates the case of a unique batch used for training DarkVec, which aggregates the history of all the observation period $T = \{t_0, t_1, \ldots, t_n\}$. To generate the embeddings, DarkVec (i) generates a corpus that spans the entire period T and (ii) learns a *single* embedding space with this entire dataset. Accordingly, generating an embedding for a new sample observed in t_{n+1} would require retraining from scratch using the entire dataset. Figure 5(b) illustrates instead the incremental strategy we adopt in i-DarkVec. In this case, at each timestep t_{i+1}, we obtain a new batch of data from which we generate the corpus and incrementally *update* the embeddings starting from the weights computed at the t_i timestep.[6] This latter strategy has two main advantages: (i) it significantly reduces the corpus size and thus speeds up the embeddings learning, and (ii) it lets the system weigh newer information automatically.

We outline the pseudo-code of the incremental algorithm of i-DarkVec in Algorithm 1. Details are explained in Sections 4.2 and 4.3.

4.2 Service and Corpus Definition

To build the embeddings, we need to create ordered sequences of words (i.e., *documents*) as originally done in Word2Vec [41, 43]. Here, we aim at finding similarities among senders' activity considering packets they send to a darknet. We then consider each source IP address associated with an incoming packet to be a *word* w and create documents as the sequence of IP addresses as they appear in time. We define as V the vocabulary containing all sender IP addresses targeting darknets.

We also 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 to consider the different aims of the senders. Given a destination port p, with $p \in P, P = \{0, \ldots, 65,535\}$, we characterise a service s by the set of ports used by services $s = \{p_1, \ldots, p_n\}$. Hence, s results in a partition of P, i.e., $s \subseteq P$, and we call S the set of all the services.

[6] At t_0 i-DarkVec builds the embeddings from scratch starting from random weights.

ALGORITHM 1: i-DarkVec embeddings generation through incremental training.

Function GenerateCorpus
Input: Observation sender sequence W

1: $W \leftarrow$Filter(W) // *Filter trace (see Section 3.1)*
2: $V \leftarrow$GetSenders(W) // *Extract the set of unique IP address senders*
3: $S \leftarrow$ GetServices(W) // *Obtain the definition of services*
4: **for all** $s \in S$ **do**
5: $\quad W^s \leftarrow$ GetSequence(W, s) // *Get the sequence of senders per service*
6: **end for**
7: $C \leftarrow$Merge$(W^{s_1}, \ldots, W^{s_{|S|}})$ // *Get the final corpus as the union of sequences*
8: **return** C, V

Procedure GenerateEmbeddings
Input: Observation sender sequences $W(t_0), \ldots, W(t_n)$

9: $C(t_0), V(t_0) \leftarrow$ GenerateCorpus$(W(t_0))$ // *Process observation sequence* $W(t_0)$
10: $g_0 \leftarrow$ Train$(C(t_0))$ // *Train the embedder* g_0 *on the corpus* $C(t_0)$
11: // *Incremental training on observations*
12: **for** $i \in \{1, \ldots, n\}$ **do**
13: $\quad C(t_i), V(t_i) \leftarrow$ GenerateCorpus $(W(t_i))$
14: $\quad g_i \leftarrow$ Train$(C(t_i), g_{i-1})$ // *Update embedder* g_{i-1} *on the new corpus* $C(t_i)$
15: **end for**
16: $U(t_n) \leftarrow g_n(V(t_n))$ // *Generate embeddings for all senders in* t_n
17: **return** $U(t_n)$

For each observation timing t_i and corresponding sender sequence $W(t_i)$ of senders $V(t_i)$, we split its incoming packets into multiple sequences, one for each service. In detail, taking the sequence of sender IP addresses appearing in a given time interval, the sequence $W^s(t_i)$ of the service s is the sequence of IP addresses sending packets to ports in s in observation time t_i.

Finally, for each observation time t_i, we build a *corpus* $C(t_i)$ as the union of the per-service sequences $W^s(t_i)$:

$$C(t_i) = \cup_{s \in S} W^s(t_i).$$

We use the final corpus $C(t_i)$ to train Word2Vec embeddings. We outline in the pseudo-code of Algorithm 1 the corpus definition in the function GenerateCorpus.

Figure 6 shows an example of the corpus definition. On the left, we have the sequence of packets in a time window. There are only two services, 22/TCP (red) and 445/TCP (blue). 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 that appears both in the port 22 and 445 services.

As we will see later, the definition of services (GetServices in line 3 of Algorithm 1) is fundamental for the construction of good embeddings. Here, we consider three alternatives:

- ***Single service (SS):*** All ports belong to a single service.
- ***Auto-defined services (AS):*** 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 services (DKS):*** We manually assign ports to services based on domain knowledge. Each service groups ports that are commonly used for popular applications. In total, we create 15 services, as detailed in Table A.1 in the appendices.

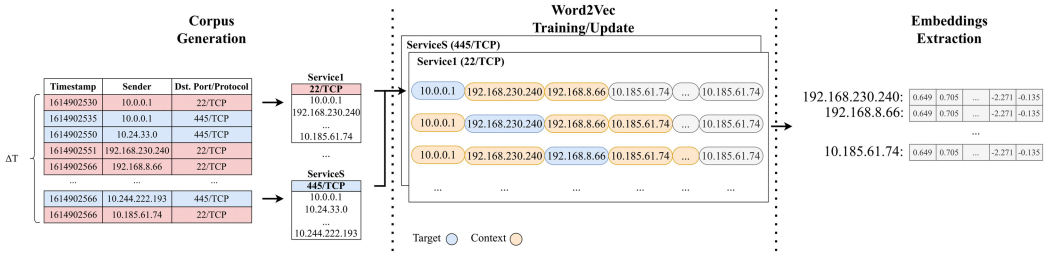


Fig. 6. i-DarkVec training for a single observation time: From the left, the corpus definition by services followed by the skip-gram construction is used to build the embeddings. At the output, IP addresses are mapped to a point in an e-dimensional space.

4.3 Building the Embeddings

Given the final corpus $C(t_i)$ for observation time t_i, we build our embeddings. We employ the skip-gram model (see Section A.1), which provides excellent results in NLP when looking for embeddings that efficiently predict the context of a word in a sentence. Given a specific word in the middle of a sentence (the input word), look at the words nearby and pick one at random. The network is going to tell us the probability for every word in our vocabulary of being the "nearby word" that we chose.

Given a sequence $W = [w_1, w_2, w_3, \ldots, w_n]$ of senders and a context window of size c, we define the context of each w_i as the sub-sequence of the c previous and c following words of w_i, i.e., $[w_{i-c}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+c}]$.

The central part of Figure 6 shows the skip-gram model for the sequences of the Port 22/TCP service. The blue circles represent the target sender's IP address, while the yellow circles represent the context IP addresses to predict.

We build the embeddings using off-the-shelf Word2Vec where we consider also some negative sampling [43]. It consists of giving an additional set of words that are never found in any context window of the given word w_i (thus, "negative" words). As said, the training of Word2Vec consists in training a **Neural Network (NN)** that has the task of predicting the context window words for a given target word. The NN is formed by three fully connected layers, and the weights of the neurons that connect the input to the (hidden) second layer of e neurons build the embedding space that maps each input word into an e-dimensional vector, i.e., $g(w_i) = u_i \in \mathbb{R}^e$. The dimension e of the projection is a parameter that impacts the quality of the embeddings. Notice that the embedder is a function g from the space of observed senders $w \in V(t_i)$ to $\mathbb{R}^e$. We call the set of all embeddings $U(t_i) = \{g(w_i), w_i \in V(t_i)\}$.

We report the pseudo-code outline of the incremental embedding training in Algorithm 1. Refer to Appendix A.1 for more details.

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 embeddings map those IP addresses that frequently appear in the same context window into the same region in the e-dimensional space, i.e., senders that perform similar patterns at a given time are mapped into a compact region.

For our experiments, we use the skip-gram-based Word2Vec Python implementation of *Gensim* [12], which we modify to support the incremental updates of the weights. We make all our source code and anonymised datasets available to the community to allow others to reproduce results.[7]

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

Table 4. Comparison between i-DarkVec, DarkVec, IP2VEC, and DANTE

			5 Day dataset (coverage: 82%)			**30 Day dataset** (coverage: 100%)		
	Corpus	**Epochs**	**Samples**	**ETA**	**Acc.**	**Samples**	**ETA**	**Acc.**
DANTE	Port-based	10	>7B	>10 days	–	–	–	–
IP2VEC	Flow-based	10	38M	~60 min	0.67	–	>10 h	–
DarkVec	DKS ($c = 25, e = 50$)	20	17M	~14 min	0.93	486M	~1.2 h	0.96
i-DarkVec	AS ($c = 5, e = 200$)	1	4M	~18 sec	0.97	21M	~2 min	0.97

5 VALIDATION WITH SUPERVISED TASKS

We now validate the embeddings i-DarkVec creates by defining a supervised learning task in which we use a simple k-NN classifier in the embedding space to label sender IP addresses. We compare i-DarkVec to the original DarkVec proposal and to DANTE and IP2VEC.

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 i-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 speed up training.[8]

5.1 Comparison with DarkVec, DANTE, and IP2VEC.

We consider the traffic observed in our darknet for a period of 5 days (first scenario) and 30 days (second scenario). For each scenario, we select the subset of *active* senders and create the embeddings for each case. We then run the same test done with the baseline classifier to check how senders in the ground truth are projected into the embedding space. Intuitively, good embeddings shall project IP addresses of the same ground truth class to compact regions in the latent space so a k-NN classifier can recover the correct label.

Following a Leave-One-Out approach, we consider each IP address w_i for which we have a label and that results active in the considered dataset. Using the cosine similarity, we measure the distance between w_i and other IP addresses, i.e., $cosine(u_i, u_j)$ measures the similarity between the projection vectors u_i, u_j of w_i and w_j. Then, we extract the k-Nearest Neighbours of w_i in the embedding space and use majority voting to assign the predicted class to w_i. At last, we compare the predicted class against the w_i ground truth class. If the predicted and actual class match, then we have a correct prediction. By repeating the procedure for all labelled IP addresses, we compute the average accuracy, i.e., the probability that an IP address gets associated with 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 corpus size and the training time required to build the embeddings. We consider DarkVec with best parameters as in Reference [34], i.e, with the domain knowledge service definition, context window $c = 25$, and embedding dimension $e = 50$. We consider i-DarkVec with the auto-defined services, context window $c = 5$, and embedding of $e = 200$ dimensions. We will discuss the choice of i-DarkVec parameters in Section 5.2.

We summarise results in Table 4. Consider first the 5-day dataset. With i-DarkVec, we can predict the correct class with a macro accuracy of 0.97, while with DarkVec single training approach, we can reach an accuracy of 0.93. IP2VEC reaches only 0.67. Here, both the incremental training

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

and the flexible definition of services in i-DarkVec play a role, and we will discuss that in detail next. Considering scalability, because of the incremental training approach, i-DarkVec is the most scalable algorithm with a total training time of only 18 seconds—this is the time to process 5 batches and model updates of about 4M samples each. Conversely, DarkVec produces a corpus of over 17M sequences that require more than 14 minutes to complete the Word2Vec training. IP2VEC requires about four times more time to complete the training. The additional complexity is also due to the large usage of negative sampling suggested by authors. This negative sampling, together with a custom definition of context, increases the number of training samples to 38 M, increasing the training time.

DANTE does not scale, and after more than 10 days, we could not complete the training. This happens because DANTE generates around 7B sequences during sequence creation. Recall that DANTE associates each port to a word and generates a different sentence for each IP address. This approach results unfeasible with hundreds of thousands of sources, each generating independent sequences. The original paper presenting DANTE confirms the scalability problem. For the sake of completeness, we perform an additional test, reducing the dataset size by considering only one day of data. DANTE was finally able to complete the embeddings computations in about 2 minutes, resulting in 0.47 average accuracy. i-DarkVec completed the training in just 1 second, reaching 0.64 in accuracy. On the one hand, this result testifies to the scalability issues of DANTE; on the other hand, it shows the need to consider a big dataset to let the embeddings converge to a useful representation.

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 (cf. Figure 3(b)). More data allows us to build embeddings that cover more IP addresses. In fact, the number of active senders found on the last day and covered by the embeddings grows from 17,463 to 22,399. Restricting to those senders for which we have a label, the embeddings built on 5 days of traffic cover 82% of senders and, by construction, the coverage is 100% when using the 30-day dataset.

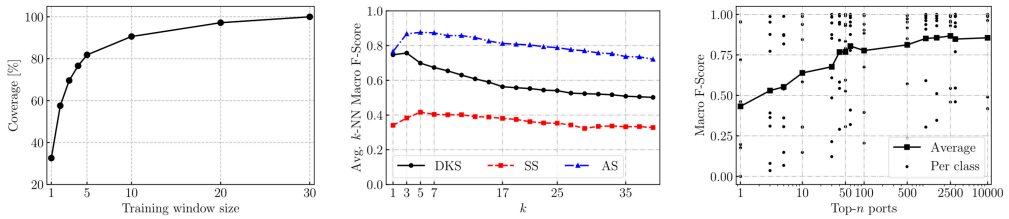
The increase in the data produces a sizeable increase in the number of sequences in the corpus. Thanks to the incremental approach, i-DarkVec processes 29 batches of data, each containing 1 day of traffic, each time updating the ith embeddings to get the $(i + 1)$-th embeddings. i-DarkVec training over the whole 30 days completes in 2 minutes, and we obtain 0.97 accuracy. When trained using a single batch of 30 days, we obtain almost the same accuracy (0.96) using the original DarkVec embeddings, but DarkVec takes 1.2 hours to complete the training. IP2VEC cannot complete the word sequence creation process after more than 10 hours, producing more than 200M training samples.

In sum, the incremental training approach and the simple word sequence creation of i-DarkVec increase scalability. Its flexible service definition and incremental approach result also in embeddings that can be used to train more precise classifiers than the original DarkVec.

5.2 i-DarkVec Parameter Tuning

We perform a sensitivity analysis on i-DarkVec hyper-parameters. We again focus on the classification task, verifying the accuracy following the same Leave-One-Out approach on the IP addresses that are active on the last day of the trace. Here, we use the macro F-score as the main performance indicator. It is a conservative metric that averages the per-class F-score to avoid the class unbalance problem.

We vary the number of days used to train the Word2Vec model and the number of neighbours k, as well as we test the different service definition strategies. We then study strategies to reduce the corpus size, as this is a key parameter for the scalability of i-DarkVec. Finally, we study the impact of the embedding size e and the context window size c. Given the number of parameters to test,



(a) Impact of training window length.

(b) Impact of k on the k-NN classifier. i-DarkVec.

(c) Impact of the top-n ports defined as service. Incremental training.

Fig. 7. Grid search for training window length, k of k-NN classifier, and top-n ports used as services.

we cannot perform a complete grid search. Instead, we follow a greedy optimisation by varying and choosing one parameter at a time. When not otherwise specified, we set $e = 50$ and $c = 25$, as they are the best choices for the parameters in Reference [34]—we later optimise these parameters for i-DarkVec, too. We train i-DarkVec on 30 batches, each corresponding to one day of traffic. At each batch, we update the embeddings by performing a 1-epoch iteration.

5.2.1 Training Data Size. First, we check the impact of the training data size. Considering accuracy, we have already observed how training i-DarkVec with a 5-day- or a 30-day-long dataset let the k-NN reach 0.97 of accuracy. However, the training data size has a significant impact on the coverage, as depicted in Figure 7(a). Given that we restrict the embeddings constructions to those IP addresses seen at least 10 times in the training period, the longer the period, the higher the chance to collect enough observations to build an embedding for a given IP address. For instance, Figure 7(a) shows that using a single day of data would allow us to build the embeddings for about 35% of the senders that could be covered when using 30 days of data.

As such, in our experiments, we fix the training period length to 30 days to maximise coverage.

5.2.2 Impact of k and Service Definition. We next choose the k parameter for the k-NN classifier. Figure 7(b) shows the macro average F-score for increasing values of k. First, observe how the ***Auto-defined Services* (AS)** model performs significantly better than the ***Domain Knowledge Services* (DKS)** and ***Single Service* (SS)** models. Second, $k \in [3, 7]$ offers the best performance. Increasing k improves the average accuracy up to when the neighbourhood starts to include too many samples. When k is too large, the classification becomes uncertain, because popular classes, e.g., the "Unknown" and other classes, start to dominate the neighbourhood.

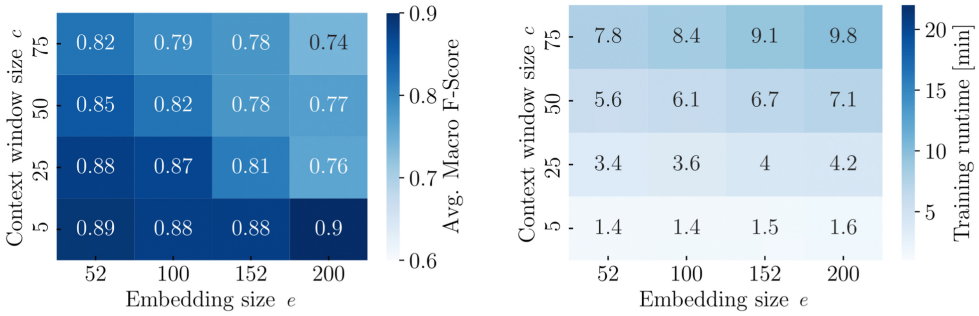
We thus select the AS model and fix $k = 5$.

5.2.3 Top-n Ports for Auto-Defined Services. We now observe the impact of the number of top-n ports to generate sequences. We report results in Figure 7(c). The curve shows the average macro F-Score and the dots depict the per-GT-class F-score. Overall, increasing n makes most GT classes have larger F-Scores, with the best values when $n \in [2{,}000, 2{,}500]$. Here, we choose top-2,500 ports for the Auto-defined Services.

5.2.4 Corpus Reduction and Training Approach. To speed up the training time, we investigate if it is possible to reduce the number of training samples by removing duplicates in the sequences. Recall that we build the sequences as senders reach a particular service in the darknet. We frequently observe multiple packets sent from the same sender to the same port in short time intervals. These repetitions create sub-sequences where a single IP address appears multiple times. We thus check the impact of replacing such sub-sequences with a single occurrence. Table 5 details the corpora

Table 5. Comparison of the k-NN Classifier Performance for the Corpus Reduction

	With duplicates		Without duplicates	
	DKS	**AS**	**DKS**	**AS**
Avg. Doc length	5,276	56	3,003	19
Max. Doc length	290,597	313,756	78,675	35,677
Runtime - 1 day	4.3 s	6.2 s	4.5 s	4.4 s
Runtime - 30 days	88 s	122 s	79 s	80 s
Macro F-Score	0.63	0.87	0.72	0.87



(a) Impact of e and c on F-Score.

(b) Impact of e and c on training run time.

Fig. 8. Impact of embeddings parameter on classification F-Score and training runtime.

size, the embeddings training time, and the average macro F-score. For completeness, we report results for both the DKS and AS definition. Again, AS outperforms the DKS.

As expected, removing duplicates in sequences reduces the average document length. It has little impact on the training time, which is dominated by the back-propagation time during the training of Word2Vec. The removal of duplicates causes marginal differences in performance. Even if the benefits are limited, we create sequences with the duplicate removal strategy, given its marginal benefits in the runtime of the algorithm.

5.2.5 Impact of c and e. We finally study the impact of the context window size c and the number of dimensions of the embeddings e in i-DarkVec. Here, we consider both the model training time—the shorter the training time, the better the performance—and the performance of the k-NN classifier.

Figure 8 details results. Each matrix shows the impact of $c \in [5, 75]$ and $e \in [52, 200]$.[9] F-score tops to [0.89,0.90] when $c = 5$, with marginal impact of e. For large values of the context window c, small values of e perform slightly better. This result suggests that low-dimensional embeddings avoid the curse of dimensionality, simplifying the classification task.

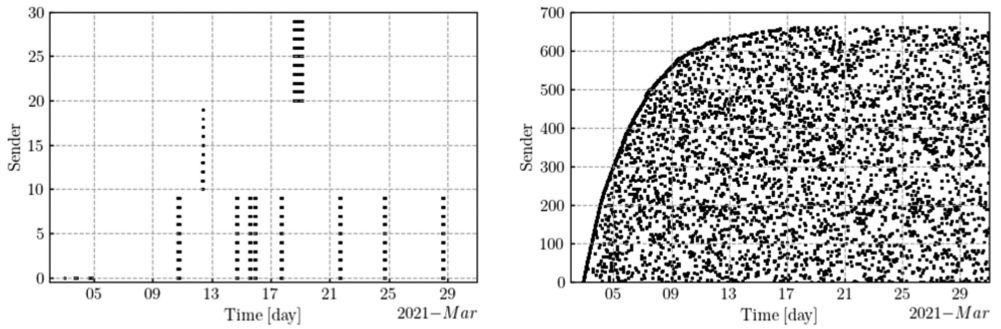
Looking at the time to complete the training, small values of c and e correspond to a simpler NN, which in turn makes the training faster. Based on these results, we set the context windows $c = 5$ and the dimension of the embeddings $e = 200$, which maximises the performance. Not reported here, we repeated the grid search for the SS and DKS cases. Results are consistent with best figures for small values of c and little impact of e.

[9]The Gensim library suggests using embeddings size that is multiple of 4 for efficiency in the calculation.

Table 6. 5-NN Classifier F-Score

	SS ($c = 5, e = 52$)			DKS ($c = 5, e = 152$)			AS ($c = 5, e = 52$)			
	Precision	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**	**Support**
Mirai-like	1.00	0.97	0.98	1.00	0.95	0.98	1.00	0.98	**0.99**	7,351
Censys	0.92	0.93	0.92	0.95	0.92	0.94	0.99	1.00	**0.99**	336
Stretchoid	0.80	**0.04**	**0.07**	0.53	**0.09**	**0.15**	1.00	**0.31**	**0.47**	104
Internet-census	0.97	0.92	0.95	0.97	0.98	**0.98**	0.98	0.99	**0.98**	103
Binaryedge	0.99	0.97	0.98	0.95	0.93	0.94	0.98	1.00	**0.99**	101
Sharashka	0.94	0.88	0.91	0.96	0.90	0.93	1.00	1.00	**1.00**	50
Ipip	0.55	0.76	0.64	0.56	0.76	0.64	0.78	0.57	**0.66**	49
Shodan	0.93	0.61	0.74	0.95	0.87	0.91	1.00	1.00	**1.00**	23
Engin-umich	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**	10
Macro avg.	0.90	0.79	0.80	0.87	0.82	0.83	0.97	0.87	**0.90**	8,127
Weighted avg.	0.99	0.95	0.96	0.99	0.94	0.96	1.00	0.97	**0.98**	8,127

Values below 0.50 are highlighted in red.



(a) Engin-umich activity pattern. (b) Stretchoid activity pattern.

Fig. 9. Activity patterns for some GT classes.

5.3 Detailed Result Per Class

So far, we have reported only the F-score to summarise the results. The macro F-score, however, weights the performance independently on the class support, thus giving equal importance to all 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 appreciate in detail the per-class results, Table 6 summarises the Precision, Recall, and F-score for each GT class. At the bottom, we summarise the average performance using both the macro average (i.e., dividing each metric by the number of classes) and weighted average (i.e., weighing the metric by the support of each class). For the sake of completeness, we report results for all four service definitions, highlighting in red those results that are particularly unsatisfying (<0.5) and in bold the best performance in the F-score.

The Single Service embeddings result is particularly poor for this task. They work well for the Mirai-like botnet(s) but fail in most other classes. Being the largest class, Mirai dominates the weighted metrics. The AS and DKS help Word2Vec to obtain descriptive embeddings even for minority classes. IP addresses of the same class are projected into the same portion of the space so the majority of the $k = 5$ neighbours results of the same class.

Interestingly, the 10 Engin-umich senders (which target port 53 only) are projected into the same portion of the embedding space so the k-NN correctly classifies all of 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 9(a).

The Stretchoid class results in the lowest performance metrics. Looking into its temporal pattern in Figure 9(b), we observe a very irregular pattern, with few packets from each sender arriving at irregular time intervals. Not having a similar context, Word2Vec likely projects these points at random in the embedding space.

Finally i-DarkVec allows us to assign labels to previously unlabeled senders by adopting a semi-supervised approach. Given the set of "Unknown" IP addresses classified as one GT class, we sort them by increasing the 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 identify new senders performing scan patterns very similar to Shodan servers (confirmed by manual investigation), other senders being very likely part of the Censys network, and so on. While qualitative, this analysis lets 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.

6 UNSUPERVISED EMBEDDINGS ANALYSIS

i-DarkVec is able to project senders involved in similar activities into the same region of the latent space. We now investigate how unsupervised approaches let us identify clusters of senders that perform coordinated actions.

6.1 Clustering Methodology

Given the good properties exhibited by k-NN for classification, we design a graph-based clustering for the unsupervised exploration of the embeddings. Given the space of all possible senders V, we define the set of senders for which we have an embedding $V' \subseteq V$. Then, we build a directed graph $G(V', E)$, where we connect each vertex to its k' nearest neighbours:

$$E = \{(w_i, w_j), w_j \in \text{k'-NN}(w_i), \forall w_i \in \mathcal{V}'\}.$$

We assign to each edge (w_i, w_j) a weight equal to the cosine similarity $cosine(u_i, u_j)$ between the embeddings of the vertices w_i and w_j. Note that each edge (w_i, w_j) is directed, since w_j can be among the k' nearest neighbours of w_i, but w_j can have k' different neighbours.

Given the graph G, we use the Louvain algorithm [25] to extract non-overlapping communities or clusters. The algorithm maximises 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 [45] and even for darknet traffic analysis [52].

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.

6.2 Choice of k′

The only parameter for the graph-based clustering is k', the number of neighbours to 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 the auto-defined services. We run i-DarkVec to build the embeddings, build the graph, and extract clusters for different k'. We show the number of clusters (left y-axis, solid red curve) and the

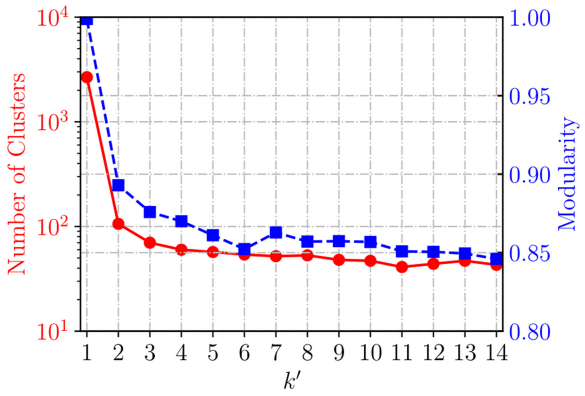


Fig. 10. Impact of k' in cluster detection.

modularity (right y-axis, blue dotted curve) in Figure 10. Intuitively, when $k' = 1$, we connect each IP address to only the nearest point. This creates many disconnected components in the graph, resulting in thousands of tiny clusters. With $k' > 1$, disconnected components start to get connected, resulting in fewer clusters. With $k' = 3$ (suggested by the elbow method [51]), we obtain 70 clusters with high modularity. Larger values of k' have little impact, only slightly decreasing the overall modularity. As such, we use $k' = 3$ in the results that follow.

6.3 Comparison with Other Clustering Algorithms

Next, we compare the results of the Louvain clustering with traditional clustering algorithms. For this analysis, we use 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.

We consider a simple k−Means algorithm and a hierarchical agglomerative algorithm [23]. In both cases, we select parameters (the number of clusters for k−Means and the maximum linkage threshold for the agglomerative clustering) by using the same elbow-based parameter selection.[10]

Figure 11 shows the average per-cluster silhouette for the three algorithms. All clusters tend to have a positive silhouette. The hierarchical agglomerative clustering detects only 3 clusters, while k−Means detects 12 clusters. Given we already know there are at least 10 groups of coordinated sources in this dataset (9 GT classes + 1 unknown), we would expect a higher number of clusters if the clustering algorithm is able to capture such behaviour from the embeddings. Manual inspection shows that these clusters are very large, with samples from the GT classes that are spread in multiple clusters. These results hint at poor clusters, which do not reflect groups of coordinated senders. Instead, the Louvain algorithm on 3-NN Graph offers detects 70 clusters, and 80% of them have positive silhouette values. In the following, we inspect these clusters to show how they represent compact and homogeneous groups of senders.

6.4 Clusters Characterisation

Figure 12 shows the average silhouette per cluster with respect to the cluster size on the x-axis. In general, we observe that the size of the clusters varies a lot (note the logarithmic x-scale). The small

[10]We have also tested DB-Scan, a density-based algorithm. Performance is much worse, facing the well-known curse of dimensionality as well as difficult parameter convergence.

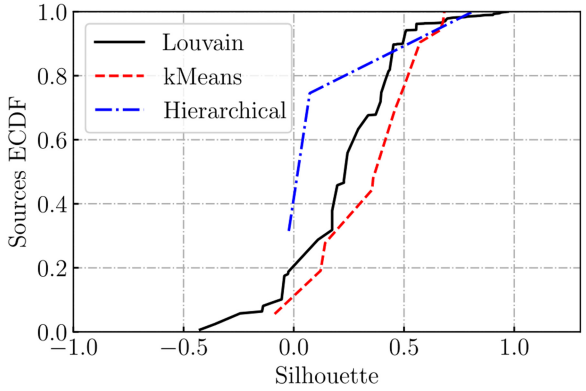


Fig. 11. Clustering algorithms comparison. Louvain detects 70 clusters, $k-$Means 12 clusters, and Hierarchical 3.

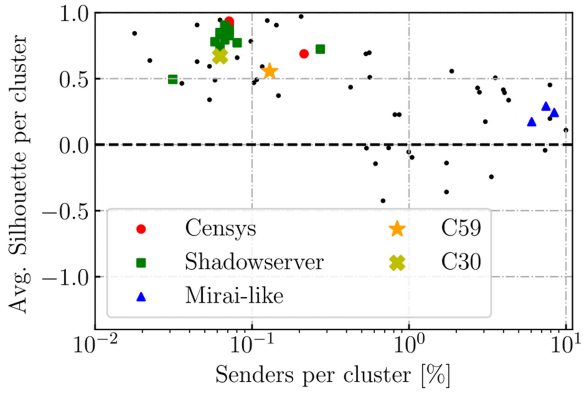


Fig. 12. Average cluster silhouette versus cluster size for the 70 clusters identified over the 3-NN graph.

clusters tend to have a high silhouette, resulting in compact and well-defined groups. For instance, senders in the Shadowserver (green squares) or Censys (red points) classes clearly emerge as well-separated clusters. This confirms that senders belonging to known classes are well-separated from other senders in the embeddings space generated by i-DarkVec.

Interestingly, there are also some large clusters aggregating thousands of senders. For instance, three clusters aggregate a lot of senders in the Mirai class with a positive silhouette (highlighted with blue triangles). As several variants of the Mirai class exist, the identification of several clusters of senders having the same Mirai signature is a sign that i-DarkVec can detect sub-classes of the generic Mirai class. In the following, we analyse clusters with high silhouettes looking for insights and possibly previously unknown classes of scanners.

We summarise the results of our manual analysis in Table 7. Similarly to the steps used to build the GT, 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` database, check public security repositories, and fire HTTP requests to senders' IP addresses to check for "abuse" pages redacted by people running legitimate scanners. Here, the ability to work on groups of senders dramatically simplifies the analysis, showing how i-DarkVec eases the work of the security analyst. We make publicly available the

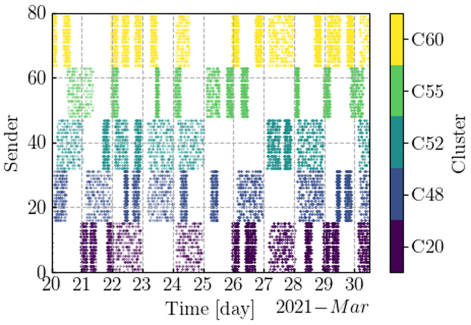


Fig. 13. Patterns of five Censys sub-clusters.

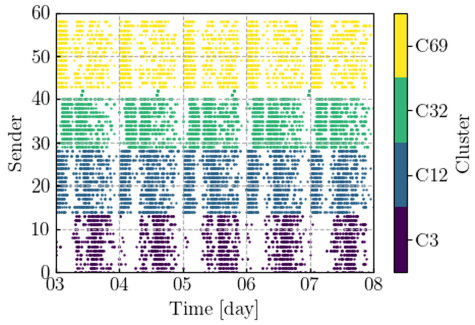


Fig. 14. Patterns of four Shadowserver sub-clusters.

clustering report[11] i-DarkVec automatically generates, providing a brief characterisation of each cluster.

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

6.4.1 Sub-Clusters of Known Scanners. Using a completely unsupervised approach, i-DarkVec coupled with clustering identifies groups of senders already present in our ground truth. Interestingly, it identifies sub-groups inside the set of senders belonging to the same scanner. For instance, consider Censy's service. Recall that Censys targets more than 11,000 ports (Table 2) with 336 IP addresses seen in our data. i-DarkVec divides 144 of those senders into 7 groups.[12]

Figure 13 shows the temporal patterns of these clusters. For visualisation, we report only five clusters. The x-axis represents the time, the y-axis presents senders ordered by the cluster IDs, and points mark the time in which the given sender is active. Colour identifies each cluster. Patterns show that each group is formed by a similar number of IP addresses that are active in different time periods. Not shown, each group targets a different set of ports, too. i-DarkVec highlights the scan strategy employed by Censys, which deploys sets of scanners, each composed of a similar number of hosts and each group searching for particular services on the Internet.

6.4.2 Disclosed Benign Scanners. i-DarkVec allows us to identify addresses belonging to Internet security services like *Shadowserver* [18], which we initially were unaware of and, therefore, did not include in our GT. The service performs scans from its networks and is listed in public security databases [5, 11, 13].

We identify 345 *Shadowserver* senders belonging to the same /16 network that i-DarkVec divides into 13 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 14) than in the Censys case. Yet, i-DarkVec identifies that they target the same group of ports, but with very different intensities (e.g., C3 focuses on port 3389/UDP, while C12 focuses on ports 17/UDP, 32414/UDP). Other notable examples of ground truth extension are the CSN cluster (C28) and the Cortex Xpanse ones (C33, C61). The first contains 46 senders belonging to the **Cloud System Networks (CSN)** company [9]. They perform periodic scans on port 137/UDP, which is related to the NetBios protocol. In the second case, i-DarkVec detected 38 senders belonging to Google CDNs

[11] https://github.com/SmartData-Polito/darkvec/blob/idarkvec/reports/cluster_report.pdf.
[12] The remaining Censys IP addresses have a more sporadic presence and remain in noisy groups with a low silhouette.

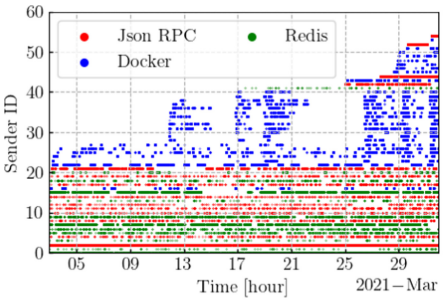
Table 7. Summary of Clusters Exhibiting Strong Signs of Coordination

Type	Cluster	IPs	Silhouette	Ports	Last day pkts.	Description
Confirmed GT	C7	48	0.688	254	58k	Censys sub-clusters
	C16	16	0.935	23	506	
	C20	16	0.916	27	506	
	C48	16	0.935	26	532	
	C52	16	0.904	20	506	
	C55	16	0.909	21	460	
	C60	16	0.902	26	1k	
	C44	119	0.688	245	11k	Binaryedge
	C49	13	0.489	17	88	
	C5	23	0.469	712	20k	Shodan
	C27	12	0.593	355	3k	Internet-census
Benign scanners	C3	14	0.738	39	784	Shadowserver sub-clusters
	C10	16	0.826	40	1k	
	C12	15	0.905	40	1k	
	C32	14	0.836	41	1k	
	C34	16	0.869	42	2k	
	C50	18	0.772	46	2k	
	C63	15	0.794	40	1k	
	C64	61	0.724	40	3k	
	C65	14	0.849	39	1k	
	C66	7	0.494	39	1k	
	C68	13	0.78	36	1k	
	C69	16	0.857	40	1k	
	C67	126	0.051	36	5k	
	C28	48	0.971	2	486	CSN
	C33	32	0.905	16	1k	Cortex Xpanse
	C61	10	0.906	12	402	
Suspicious	C0	965	0.337	109	92k	MSSQL bruteforcer. SQL Slammer Worm-like
	C47	12	0.34	8	2k	Redis miner. RedisWannaMine-like
	C53	14	0.944	1	5k	Json RPC brute-forcers
	C59	29	0.555	24	9k	Docker scanner. All senders target Docker ports. Activity coherent with known botnets
ADB.Mirai	C2	794	0.506	85	262k	Mirai variant targeting Android debug bridge service
Mirai-like	C39	1,885	0.244	85	19k	Mirai-like bots targeting port 23/TCP and 2323/TCP
	C45	1,669	0.292	73	17k	
	C36	1,354	0.174	58	7k	
Emerging campaign	C24	26	0.591	15	2k	4 common ports and unique /24 subnet
	C58	128	0.227	24	117k	SMB scanner
Unknown scanners	C30	14	0.669	35	5k	All senders target the same 35 ports
	C57	24	0.492	53	31k	SIPVicious-like scanner
	C59	29	0.555	24	9k	Kubernetes exploiter
	C43	28	0.94	5	342	Sequential ports scanner
	C37	909	0.393	39	4k	HTTP/Telnet scanner
	C22	95	0.435	416	54k	NSR Mixed scanner
	C14	10	0.692	40	8k	Kamatera Inc. scanner

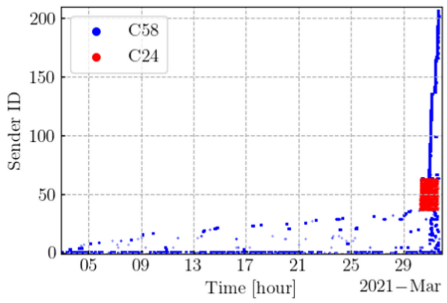
hosting services from the Cortex Xpanse service [10] from Palo Alto Network. The senders are grouped into two clusters according to both their activity and port patterns.

6.4.3 Uncovered Actors. i-DarkVec let us identify groups of senders that are not present in open security databases or related search engines but whose activity patterns suggest the coordination of a large number of IP addresses, sometimes displaying fingerprints that align with botnets. Hence, since we cannot confirm the purpose of their actions, we qualitatively describe them, grouping them into four classes: suspicious activities, emerging campaigns, Mirai variants, and other unknown scanners. We provide more details about these groups in Table 7 and Appendix A.2.

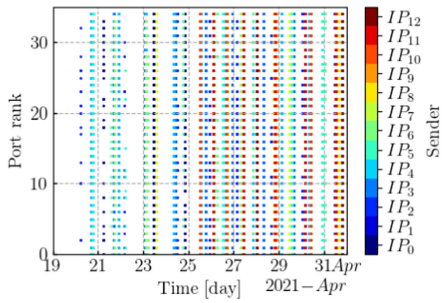
***Suspicious activities*:** We observe several groups of unknown senders whose intentions are suspicious. We depict some notable activity patterns in Figure 15(a) and describe some of the clusters below.



(a) Three clusters exhibiting common patterns associated with suspicious activities.



(b) Emerging campaigns.



(c) Focused scanner.

Fig. 15. Examples of activity patterns of hosts in new clusters.

- *Docker scanners*. We detect a group of 29 unknown senders all targeting the same Docker API ports.[13] This group might be associated to botnets such as Graboid [3], and Kinsing [4].
- *Json RPC brute-forcer*. A cluster of 14 senders sending more than 5,000 packets towards the Json RPC port 8545/TCP. The activity pattern of Figure 15(a) suggests a likely Json RPC brute-force attack.
- *Redis miner*. We observe a group of unknown senders massively scanning for open Redis instances on ports 6378/TCP, 6379/TCP, 6380/TCP, and 6381/TCP. Such suspicious activities could be the first steps of a RedisWannaMine[1] attack.
- *MSSQL brute-forcer*. i-DarkVec detects a group of $>$ 900 senders targeting MSSQL port 1433/TCP (88% of senders) and SMB port 445/TCP (60% of senders). Literature is rife with examples of worms targeting the MSSQL port and probing the SMB one for spreading (e.g., Conficker [36]).

Novel emerging campaigns: Apart from the above suspicious activities, we observe emerging scanning campaigns, such as cluster C58 (Figure 15(b)), where all the 128 senders target SMB port 446/TCP, and most of them become active only 24 hours before our last day of analysis.

Similarly, all 26 senders of cluster C24 target the same four ports: 5060/TCP, 22/TCP, 123/UDP, and 80/TCP, and most of them share the remaining 11 ports. All senders in this cluster also abruptly started their activities on the second to last day of our analysis. They all belong to the same /24

[13]Default Docker API port 10250/TCP, unencrypted Docker API port 2375/TCP, encrypted Docker API ports 8443/TCP and 2376/TCP, and Docker registry API port 4243/TCP.

subnet, with reverse DNS pointing to `bc.googleusercontent.com`. This could be a new coordinated scan activity run by some malicious user using virtual machines hosted in the Google cloud infrastructure.

These examples show that i-DarkVec can ease the identification of emerging scans and attacking campaigns once successfully deployed and possibly assist on the spot of zero-day threats if those result in novel patterns in the darknet traffic.

Mirai variants: Other notable examples of coordinated senders uncovered by i-DarkVec are the Mirai-like clusters (C39, C45, C36). The approximately 5,000 senders that generate the 80% of traffic with the Mirai fingerprint are grouped in these three clusters. They differ by the ports targeted by the senders in each cluster. In detail, 85% of their traffic is directed towards port 23/TCP. The remaining 15% of the traffic is related to a different group of ports, suggesting three different attack patterns or Mirai variants.

Notably, cluster C2 contains more than 704 senders massively scanning the **Android Debug Bridge (ADB)** port 5555/TCP. Since they match the Mirai fingerprint, it is reasonable to assume that i-DarkVec isolates an ADB.Mirai [2] cluster.[14]

Unknown scanners: Finally, manual analysis of detected groups reveals some clusters whose senders' activity patterns suggest coordination. One notable example is C30, containing 14 unknown senders that sent the same portion of approximately 3,000 packets to 40 specific ports. As shown in Figure 15(c), senders follow a temporal pattern in contacting the ports, with each sender being active in a narrow time range and targeting all the shared ports once activated.

C57 is another example of sender coordination detected by i-DarkVec. Indeed, all the 24 unknown senders generate a considerable amount of packets (more than 30,000 in the last day only) towards the same ports 5060/UDP, 5080/UDP, 5062/UDP, 5069/UDP, and 8032/UDP. Those ports are targeted by SIP traffic, and the senders' activity shares similarities with the SIPVicious toolset [21].

While not exhaustive, this analysis shows the benefit of i-DarkVec in supporting the identification of new groups of senders performing coordinated actions. Section A.2 discusses other examples. In a nutshell, i-DarkVec offers the security analyst the opportunity to analyse groups of homogeneous senders, greatly easing the discovery of the actions the coordinated senders perform.

7 DISCUSSION AND CONCLUSIONS

We presented i-DarkVec, a system that relies on Word2Vec to build meaningful representations that can be used to shed light on noisy darknet traces. Similarly to what is commonly done with word embeddings for NLP, the embeddings built with i-DarkVec pave the way for advanced machine learning tasks. In particular, we showed that i-DarkVec embeddings can be used to cluster senders performing coordinated activities that would otherwise be hard to spot in darknet traffic. i-DarkVec thus assists security analysts to extend their knowledge about scans and attacks.

i-DarkVec outperforms state-of-the-art alternatives, in particular, with faster learning times that enable i-DarkVec application to large-scale scenarios. Beyond the darknet traffic use case, our results and methodological insights can inspire the application of i-DarkVec to other sequences of categorical variables often present in networking data, such as in honeypot traffic or network monitoring data.

Compared to NLP algorithms [27, 41, 46] where the resulting embeddings are general, a Word2Vec model trained with network data is hard to generalise. Indeed, i-DarkVec learns the time relationships among co-occurring senders within a certain observation period. Given the rapid changes in Internet traffic, senders' behaviour, and targeted services, the learned embeddings are expected to be very dynamic. On the contrary, embeddings generated from natural languages

[14] ADB.Mirai is a Mirai variant that targets IoT devices running the Android operating system.

are generic, thanks to the intrinsic static nature of the language, where the semantics and usage of words change slowly. i-DarkVec is a powerful analysis tool to shed light on darknet traffic and, thanks to its incremental design, it can cope with the dynamism required in networking use cases.

Many open questions remain, though. We will investigate whether the embeddings learned in one darknet can be useful in other darknets or at different times. First, we will study the impact of the size of the monitoring infrastructure. Here, we relied on a /24 darknet monitoring infrastructure. We expect larger darknet address space could accelerate the collection of data, thus enabling quicker embedding computation (e.g., down to the hourly granularity) and calling for possible fine-tuning of parameters (e.g., the threshold to consider a sender as active). We also envision a federated-learning approach where multiple monitoring infrastructures collaborate to the building of shared embeddings.

Other research directions involve the understanding of the transferability of the embeddings, i.e., if embeddings learned in a network could be used to solve downstream tasks in other networks. While this is common in NLP, we expect the quickly evolving nature of darknet traffic would hardly make it possible to transfer embeddings over multiple networks. The question thus is whether a model trained for one task (e.g., classify sources) on the embeddings of one darknet could successfully achieve the same task on other darknets. We plan to extend our results in these directions in the future.

A APPENDIX

A.1 Word2Vec

Word2Vec [41, 43] is an NLP technique based on artificial neural networks. It allows mapping 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 embeddings.

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 4.3). For example, by considering the sentence *"Chicago is a great city,"* if the word *"a"* is the target one and $c = 2$, then the context for *"a"* is the list of the 2 previous and 2 following words of *"a"*:

$$\text{"a"} \rightarrow (\text{"Chicago," "is," "great," "city"}).$$

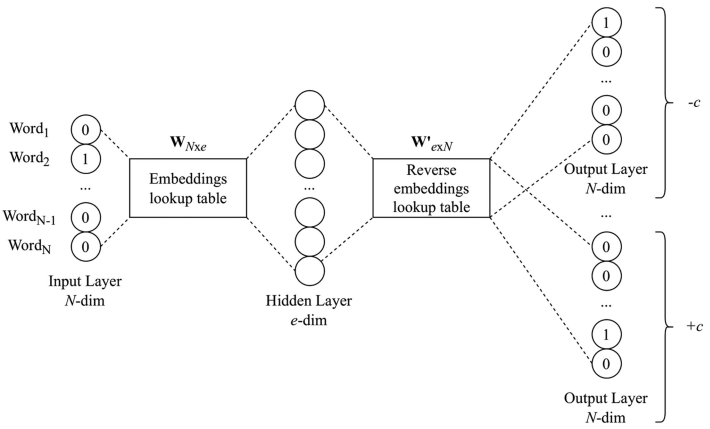


Fig. A.1. Word2Vec skip-gram architecture.

To generate the embeddings, 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 aims at predicting the probability of finding each one of the N words within the context window of a given target word. In Figure A.1, 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 e-dimensional hidden layer links all the $2c$ context words to the target one. After the model training, the embeddings are obtained from the weights matrix $\mathbf{W} \in \mathbb{R}^{N \times e}$. Each of the $i \in \{1, \ldots, N\}$ entries of $\mathbf{W}$ is the embedding in $\mathbb{R}^{1 \times e}$ associated to the ith word.

Table A.1. 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

A.2 Clusters

In this section, we provide further details about the data in Table 7 by reporting more examples of the most notable clusters learned with i-DarkVec.

Here, we complete the description of the clusters uncovered by i-DarkVec that are formed by scanners that we could not explain based on online security databases (cf. Section 6.4.3).

Kubernetes exploiter: All the 29 unknown senders target port 10250/TCP with >24k packets (32.5% of the cluster traffic). The port is usually targeted to exploit Kubernetes vulnerabilities.

Sequential ports scanner: 28 unknown senders. All of them are active in the same period and target the same sequential ports 18081/TCP, 18082/TCP, 18083/TCP.

HTTP/Telnet scanner: More than 900 senders. The 80% of cluster traffic is directed to ports 81/TCP (99.6% of senders) and 23/TCP (94.6% of senders).

NSR Mixed scanner: Half of the 95 scanners belong to the **Net System Research (NSR)** project [16] extending the ground truth, half of them are unknown. All the senders target the same 416 ports with evident activity patterns.

Massive scanner: Ten unknown senders belonging to two contiguous /24 networks and generating 8k packets. They are active in the same period every 4/6 days targeting the same set of 40 ports.

REFERENCES

[1] 2018. RedisWannaMine Unveiled: New Cryptojacking Attack Powered by Redis and NSA Exploits. Retrieved from https://www.imperva.com/blog/rediswannamine-new-redis-nsa-powered-cryptojacking-attack/.

[2] 2019. ADB.Mirai: New Mirai Botnet Variant Spreading via the ADB Debug Port. Retrieved from https://nsfocusglobal.com/adb-mirai-new-mirai-botnet-variant-spreading-via-the-adb-debug-port/.

[3] 2019. Graboid: First-Ever Cryptojacking Worm Found in Images on Docker Hub. Retrieved from https://unit42.paloaltonetworks.com/graboid-first-ever-cryptojacking-worm-found-in-images-on-docker-hub/.

[4] 2020. Threat Alert: Kinsing Malware Attacks Targeting Container Environments. Retrieved from https://blog.aquasec.com/threat-alert-kinsing-malware-container-vulnerability.

[5] 2021. AbuseIPDB - IP address abuse reports - Making the Internet safer, one IP at a time. Retrieved from https://www.abuseipdb.com/.

[6] 2021. The Best IP Geolocation Database: IPIP.NET. Retrieved from https://en.ipip.net/.

[7] 2021. BinaryEdge. Retrieved from https://www.binaryedge.io/.

[8] 2021. Censys. Retrieved from https://censys.io/.

[9] 2021. Cloud System Networks. Retrieved from http://cloudsystemnetworks.com/.

[10] 2021. Cortex Xpanse. Retrieved from https://docs-cortex.paloaltonetworks.com/r/Cortex-XPANSE/1.0/Cortex-Xpanse-Assess-User-Guide/Data.

[11] 2021. FireHOL IP Lists - IP Blacklists - IP Blocklists - IP Reputation. Retrieved from http://iplists.firehol.org/.

[12] 2021. Gensim, Topic modelling for humans. Retrieved from https://radimrehurek.com/gensim/.

[13] 2021. GreyNoise. Retrieved from https://greynoise.io/.

[14] 2021. Internet Census Group. Retrieved from https://www.internet-census.org/home.html.

[15] 2021. Michigan Engineering - University of Michigan College of Engineering. Retrieved from https://www.engin.umich.edu/.

[16] 2021. Net System Research. Retrieved from https://www.netsystemsresearch.com/.

[17] 2021. Project Sonar. Retrieved from https://www.rapid7.com/research/project-sonar/.

[18] 2021. The Shadowserver Foundation. Retrieved from https://www.shadowserver.org/.

[19] 2021. Sharashka Data Feeds - Security Data That Works. Retrieved from https://sharashka.io/data-feeds.

[20] 2021. Shodan, the search engine. Retrieved from https://www.shodan.io/.

[21] 2021. SIPVicious OSS toolset. Retrieved from https://github.com/EnableSecurity/sipvicious.

[22] 2021. Stretchoid Opt-Out. Retrieved from http://www.stretchoid.com/.

[23] Charu C. Aggarwal. 2015. *Data Mining: The Textbook*. Springer.

[24] 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. Retrieved from http://dl.acm.org/citation.cfm?doid=2815675.2815702.

[25] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *J. Statist. Mechan.: Theor. Exper.* 2008, 10 (2008), P10008.

[26] 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. DOI: https://doi.org/10.3390/s19030727

[27] Dvir Cohen, Yisroel Mirsky, Yuval Elovici, Rami Puzis, Manuel Kamp, Tobias Martin, and Asaf Shabtai. DANTE: A framework for mining and monitoring darknet traffic. 88–109. Retrieved from http://arxiv.org/abs/2003.02575.

[28] 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.

[29] 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 Symposium (SEC'14)*. 65–78. Retrieved from http://dl.acm.org/citation.cfm?id=2671225.2671230.

[30] 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.
[31] C. Fachkha and M. Debbabi. 2016. Darknet as a source of cyber intelligence: Survey, taxonomy, and characterization. *Commun. Surv. Tutor.* 18, 2 (2016), 1197–1227.
[32] J. Fruhlinger. 2018. The Mirai botnet explained: How teen scammers and CCTV cameras almost brought down the internet. (03 2018). Retrieved from https://www.csoonline.com/article/3258748/the-mirai-botnet-explained-how-teen-scammers-and-cctv-cameras-almost-brought-down-the-internet.html.
[33] Luca Gioacchini. 2021. *Automatic Detection of Coordinated Events in Darknet Traffic*. Master's thesis. Politecnico di Torino, Torino, Italy.
[34] Luca Gioacchini, Luca Vassio, Marco Mellia, Idilio Drago, Zied Ben Houidi, and Dario Rossi. 2021. DarkVec: Automatic analysis of darknet traffic with word embeddings. In *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*. 76–89.
[35] Chansu Han, Jun'ichi Takeuchi, Takeshi Takahashi, and Daisuke Inoue. 2022. Dark-TRACER: Early detection framework for malware activity based on anomalous spatiotemporal patterns. *IEEE Access* 10 (2022), 13038–13058. DOI: https://doi.org/10.1109/ACCESS.2022.3145966
[36] Mikko Hypponen. 2009. The Conficker mystery. *BlackHat* (2009). https://www.blackhat.com/presentations/bh-usa-09/HYPPONEN/BHUSA09-Hypponen-ConfickerMystery-PAPER.pdf.
[37] 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. Retrieved from http://dl.acm.org/citation.cfm?doid=3131365.3131383.
[38] Michalis Kallitsis, Vasant G. Honavar, Rupesh Prajapati, Dinghao Wu, and John Yen. 2021. Zooming into the darknet: Characterizing internet background radiation and its structural changes. *ArXiv* abs/2108.00079 (2021).
[39] Sofiane Lagraa, Yutian Chen, and Jérôme François. 2019. Deep mining port scans from darknet. *Int. J. Netw. Manag.* 29, 3 (2019), e2065. DOI: https://doi.org/10.1002/nem.2065
[40] Sofiane Lagraa and Jérome François. 2017. Knowledge discovery of port scans from darknet. In *Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 935–940.
[41] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. arXiv:cs.CL/1301.3781.
[42] Tomas Mikolov, Quoc V. Le, and Ilya Sutskever. 2013. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168* (2013).
[43] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. arXiv:cs.CL/1310.4546.
[44] 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.
[45] Symeon Papadopoulos, Yiannis Kompatsiaris, Athena Vakali, and Ploutarchos Spyridonos. 2012. Community detection in social media. *Data Mining Knowl. Discov.* 24, 3 (2012), 515–554.
[46] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the Empirical Methods in Natural Language Processing Conference (EMNLP)*. 1532–1543. Retrieved from http://www.aclweb.org/anthology/D14-1162.
[47] Rupesh Prajapati, Vasant Honavar, Dinghao Wu, John Yen, and Michalis Kallitsis. 2021. Shedding light into the darknet: Scanning characterization and detection of temporal changes. In *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT'21)*. Association for Computing Machinery, New York, NY, 469–470. DOI: https://doi.org/10.1145/3485983.3493347
[48] 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. Retrieved from http://link.springer.com/10.1007/978-3-319-17172-2_11.
[49] 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. Retrieved from http://dl.acm.org/doi/10.1145/3355369.3355595
[50] Markus Ring, Alexander Dallmann, Dieter Landes, and Andreas Hotho. 2017. IP2Vec: Learning similarities between IP addresses. In *Proceedings of the IEEE International Conference on Data Mining Workshops (ICDMW)*. 657–666. DOI: https://doi.org/10.1109/ICDMW.2017.93
[51] 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. Datab. Syst.* 42, 3 (July 2017). DOI: https://doi.org/10.1145/3068335
[52] 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. Retrieved from https://ieeexplore.ieee.org/document/9191555/.

[53] 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. Retrieved from https://ieeexplore.ieee.org/document/8847113/.

[54] 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)*. Retrieved from http://portal.acm.org/citation.cfm?doid=1029618.1029624.

[55] Akira Tanaka, Chansu Han, Takeshi Takahashi, and Katsuki Fujisawa. 2021. Internet-wide scanner fingerprint identifier based on TCP/IP header. In *Proceedings of the 6th International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE, 1–6.

Received 21 July 2022; revised 14 March 2023; accepted 18 April 2023