Iteratively Detecting Collaborative Scanner Fingerprints

An Iterative Approach to Identifying Fingerprints using Stratified Sampling

Jelt Jongsma[1]
Supervisors: Harm Griffioen[1] **& Georgios Smaragdakis**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Jelt Jongsma
Final project course: CSE3000 Research Project
Thesis committee: Harm Griffioen, Georgios Smaragdakis, Kubilay Atasu

An electronic version of this thesis is available at http://repository.tudelft.nl/.

***Abstract*– The first step of many cyber attacks is the reconnaissance phase. One of many reconnaissance methods employed by adversaries is *internet-wide scanning*, which probes the entire internet to find which hosts have open ports. These scans are practically impossible to detect by a firewall or Intrusion Detection System if an attacker chooses to distribute their scan on multiple hosts. Many of these scans embed a *fingerprint* in their packets, which can easily be detected if they are known. Previous studies have developed an algorithm that is able to identify these fingerprints, but they were not able to identify fingerprints for large portion of their data. This study proposes an iterative approach using stratified sampling, in order to see how this affects accuracy. An experiment showed the algorithm is able to identify fingerprints for sets of packets that make up less than 0.5% of all packets, and less than 0.0001% of sources. Analysis of the fingerprinted groups indicated that these groups are not part of a collaborative scanner, but hold for the same fingerprint by coincidence.**

***Keywords*– Internet-wide scan, fingerprint, genetic algorithm, network telescope, stratified sampling**

I. Introduction

As the internet continues to expand, more services are becoming interconnected. Most of these services are susceptible to discovery by a method called internet-wide scanning. Internet scanning serves various purposes, ranging from security research to identifying exploitable services. As internet scanning is often employed by adversaries to find vulnerabilities on a network, some administrators might want to block this traffic. Against aggressive *non-distributed* scanners, an administrator could implement firewall or Intrusion Detection System (IDS) rules that can drop traffic if it exceeds a certain threshold. Although this is a well-known practice, there is no reliable solution for doing the same with *collaborative* scanner traffic. And with public scanning tools, such as ZMap [1], making it increasingly simple for adversaries to distribute their scans, the detection of internet scanning and our comprehension of its implications are more important than ever.

Packets sent by an internet-wide scanning tool can be detected using network telescopes. A network telescope [2], also know as "Darknet", is composed of IP addresses that host no services or devices, which means any traffic that reaches the telescope is inherently malicious. The packets that reach the telescope are usually either backscatter from DoS attacks, or scanners probing the internet. Therefore, any data accumulated by a telescope can be used to monitor malware propagation and internet scanning activities [3].

Many studies have looked into different approaches to identify collaborative scanners. Tanaka et al. [4] categorized these approaches as follows: port-based, time-series, behaviour-based and fingerprint-based. While the other approaches have been researched quite thoroughly as summarized in [4], fingerprint-based approaches are relatively unexplored, with the works of Griffioen and Doerr [5], and Tanaka et al. [6], [4] being the only research into header field fingerprint generation techniques. Which are only preceded by manual inspection of code from open-source scanners [7], [8], and signatures based on fuzzy hashes [9].

Previous work reports that many internet scanners embed a pattern in IP and TCP header fields to distinguish their scan from backscatter traffic [5]. This pattern is considered a *fingerprint* and consists of specific combinations of header fields, e.g. ZMap sets the IP identity field of their packets to 54321 [6]. Implementing firewall or IDS rules to check incoming packets for known fingerprints is quite trivial; identifying unknown fingerprints is a substantially harder task as there are practically infinite possible patterns.

In 2020 Griffioen and Doerr [5] proposed the first method to identify header field patterns, which was to apply XOR's between packets within the same scanning group to identify common patterns. Following this research, Tanaka et al. [6] proposed a new algorithm, based on the genetic algorithm, that is able to generate flexible header field patterns. This algorithm manages to detect scanners that make up less than 0.5% of the total packets they considered. In 2023 they improved this algorithm to be able to identify low-rate and well-coordinated scans by calculating a fingerprints effectiveness using the appearance rates of hosts instead of the appearance rates of packets [4].

The aforementioned algorithm only considered a small set of binary operations and feature extractions due to computational constraints and does not take different approaches, such as time-series and behaviour-based approaches, into account. Because of this, the algorithm was only able to discover fingerprints for 50.6% of the scanners and 18.8% of the packets from their data [4], despite being able to fingerprint packets that make up less than 0.5% of the total dataset. Which means nearly half of the scanners still go undetected, and over three quarters of the packets are not fingerprinted.

In this paper I aim to answer the following research question:

> *How does an iterative approach to generating fingerprints for collaborative scanners, using stratified sampling, affect accuracy when compared to existing algorithms?*

To answer this question I propose a new fingerprint generation technique that iteratively samples a fraction of all collected packets, computes fingerprints for these samples, removes all packets with the generated fingerprints, and then repeats these steps until no more fingerprints are found or a maximum number of iterations is reached.

In summary, this paper makes the following contributions:

1. *I consider the approach by Tanaka et al. [6], and propose an adaptation to their algorithm by iteratively applying their approach using sampling.*
2. *I show an iterative approach to identifying fingerprints is able to identify fingerprints for sets of packets that make up less than 0.5% of all packets, and less than 0.0001% of sources.*[1]

The remainder of this paper is structured as follows. Section II discusses related work and shows a more in-depth review of previous fingerprint generation techniques. Section III describes the algorithm used during the experiments in this paper. Sections IV and V discuss the experimental settings and results, respectively. Section VI describes ethical decisions made during the research. And finally sections VII and VIII conclude the paper with a discussion, a conclusion and future recommendations.

II. Related work

There are many techniques that can be used to generate fingerprints. This section will first more closely inspect the work by Griffioen and Doerr [5], and will then do a deep dive into the algorithm designed by Tanaka et al. [6], [4] to get a better understanding of existing solutions and how they might be improved.

Griffioen and Doerr [5] proposed a method to identify and fingerprint distributed scanners based on recurring patterns in header fields. They did so by first clustering scanning groups using SLPA [10] and assuming a fixed-form fingerprint. The fingerprints are then identified by applying a sequence of XOR's to different packets within the same scanning group. Their experiments showed a high detection rate for various scenarios, such as distributed scanners and limiting packets per host [6]. As mentioned before, their algorithm is based on fixed-form fingerprints, which means scanning campaigns with varied fingerprints are overlooked.

Based on Griffioen and Doerr's work [5] Tanaka et al. [6], [4] designed a new algorithm that is able to generate flexible fingerprints that can be used to identify scanning groups. They considered a genetic algorithm (GA) and proposed a new algorithm that utilized certain aspects of a GA.

[1]However, it is important to note that the identified groups in this study are not collaborative scanners, but coincidentally share the same patterns.

In this algorithm fingerprints are represented as combinations of signs. These are pairs of *TCP functions* and binary numbers, (f, b), where a *TCP function* is a function that takes as input a packet and returns a binary number. If $f(packet) = b$ then the packet is considered to have the sign. If many of these packets contain the sign, it is considered an *effective sign.* A single effective sign can be considered a fingerprint, but when a set of packets has multiple effective signs they can be combined using ANDs to form a fingerprint. In order to find these fingerprints the algorithm first generates n *TCP functions*, and then proceeds to find effective signs for each function by computing the *appearance ratio* of each sign by number of packets [6] (or hosts [4]) and to what extent each binary influences the variance of the *appearance ratio.*

The step in which the algorithm finds effective signs runs in $O(nc * |P|)$, where n is the number of considered functions, P is the set of all packets, and c is the runtime for computing the *appearance ratio* of each binary. This step makes the runtime of the other functions negligible in comparison, as $|P|$ is very large (37.1 billion in [4]). Because of this we want to minimize either the number of functions to consider for effective signs, or the number of packets we run the functions on, while maximizing the detection rate.

For this we look at the generation scheme used in the experiment. Their generation algorithm randomly decides to perform a *binary operation* on two previously generated functions, or a *feature extraction* where a previously generated function is composed with a function that extracts part of the binary result (e.g. return only the left two bytes of a binary) [6].

Even though the experiment only considered a bitwise XOR as the *binary operation* and a small set of *feature extractions*, they had to generate 2.000 functions to be able to identify 9 effective signs [6]. This shows 99.55% of calls to the *find-effective-signs-function* yielded no result. If we are somehow able to generate fingerprints more effectively, then we could generate fingerprints using a larger set of binary functions, without increasing computational cost.

Algorithm 1: Fingerprint identifier

```
Input  : $n_f$ : number of generated TCP functions
         $n_s$ : number of samples
         $n_i$ : number of iterations
         $P$ : set of packets
         $F_0$ : initial set of TCP functions

Output: $R$ : set of fingerprints
1  Function fingerprint_identifier($n_f$, $n_s$, $F_0$):
2      $F \leftarrow$ generate_functions($n_f$, $F_0$)
3      $R \leftarrow \emptyset$
4      for $i \leftarrow 0$ to $n_i$ do
5          $P_s \leftarrow$ sample_packets($n_s$)
6          // Compute fingerprints for sample
7          $S \leftarrow \emptyset$
8          for $f \leftarrow F$ do
9              $S_f \leftarrow$ find_effective_signs($f$, $P_s$)
10             S.append($S_f$)
11         $R_i \leftarrow$ consolidate_signs($S$)
12         // Remove "fingerprinted" packets
13         $P \leftarrow$ filter_packets($P$, $R_i$)
14         R.append($R_i$)
15     return $R$
```

III. Methodology

The previous section discussed the different fingerprinting techniques available, and where these methods fall short. This next section will shortly explain the bulk of the algorithm, and will then go into deeper detail on the individual functions implemented in this method. However, many steps from this algorithm are similar to the steps in the algorithm by Tanaka et al. [6], for this reason the overlapping steps will only be shortly explained.

A. Fingerprint identifier

This method of identifying fingerprints is based on the algorithm proposed by Tanaka et al. [6]. The algorithm is implemented in Golang and its pseudocode is shown in algorithm 1. Below is also a summarized version:

1. *Generate functions.* Based on the same generation scheme as in [6], this method generates n functions by applying *binary operations* and *feature extractions* to an increasing set of initial *TCP functions.*

2. *Identify fingerprints.* Iteratively sample packets from the dataset and search for fingerprints. Once a fingerprint is discovered, all packet matching that print are filtered out of the dataset, so that the algorithm will not "rediscover" fingerprints. This way the algorithm is, in theory, able to identify effective signs for small scanners, without signs like ZMap's (which make up nearly 50% of the dataset), influencing appearance ratios.

   (a) *Sample packets.* The algorithm samples n packets from the dataset using proportional stratified random sampling, where n is determined using the formulas proposed by Cochran, mentioned in section III-C.

   (b) *Find effective signs.* From the sampled packets, identify *effective signs* with hyperparameters adjusted such that the algorithm finds *at most* 5 signs for a given sample.

   (c) *Dynamically adjust sign threshold.* Because the dataset is continually shrinking, the sign threshold might need to be adapted, such that the result does not contain > 15 signs for a single sample. So whenever the algorithm finds too many or too little signs twice in a row, the algorithm either increases or decreases the sign threshold by 50.0, respectively.

   (d) *Consolidate effective signs.* As in [6], the discovered effective signs are merged using ANDs (if possible). Because all signs in a fingerprint are discovered from the same packets, only signs that are found within the same sample have to be merged.

B. Generate functions

As mentioned earlier, the generation scheme used in the algorithm is based on the same technique used in [6]; a set is initialized using a small set of basic *TCP functions*, such as *get-IP-id* and *get-source-port*, and iteratively newly generated functions are added. For initializing the set of *TCP functions* Tanaka et al. [6] identified which header fields could be modified without changing the way a packet behaves, and which are not usually the same for all packets. New functions are generated by either performing a *binary operation* or a *feature extraction* on two or one of the functions from the existing set of functions, with probabilities $r - 1$ and r, respectively. Functions from the existing set are selected with probability according to formula (1), where f_{count} denotes the number of *binary operations* and *feature extractions* within a function.

$$p(f) = \frac{(1/f_{count})^2}{\sum_{f \in F}(1/f_{count})^2} \tag{1}$$

C. Sampling technique

Cochran [11] describes a general formula for estimating a sample size in his book *Sampling Techniques, Third Edition.* This formula allows for specification of the allowed error margin, the confidence level of the error margin and the expected variance of the data. An expansion of the formula can be used to adjust a first approximation to the actual population size. Following are the aforementioned formulas, where in (2) n_0 is a first approximation, Z is the Z-score based on the determined confidence level, p is the estimated variance, and e is the margin of error. In (3) n represents the adjusted sample size, and N represents the entire population size.

$$n_0 = \frac{Z^2 p(p-1)}{e^2} \tag{2}$$

$$n = \frac{n_0}{1 + \frac{n_0}{N}} \tag{3}$$

Although Cochran proposed this formula first in 1963, it is still commonly used and a widely accepted standard for computing sample sizes.

Algorithm 2: Find effective signs

```
Input  : f : PacketFunction
         P : set of packets
         t : sign threshold
         max : maximum number of signs per function
Output: S : set of effective signs
1  // Modified pseudocode from Tanaka et al. [6]
2  Function find_effective_signs(f, P):
3      $B \leftarrow \{ f(p) \mid p \in P \}$
4      $sorted\_B \leftarrow distinct($sort $B$ according to appearance ratios, $r_a(b) \mid b \in B)$
5      $max\_idx \leftarrow$ -1
6      for $i \leftarrow 0$ to $max$ do
7          if $e_f(sorted\_B[i]) > t$ then
8              $max\_idx \leftarrow i$
9      $S \leftarrow \emptyset$
10     if $max\_idx \neq -1$ then
11         for $i \leftarrow 0$ to $max\_idx$ do
12             $S.append((f, sorted\_B[i]))$
13     return $S$
```

Once the sample size has been determined using Cochran's formula, the dataset is sampled using proportional stratified sampling to accurately represent the complete dataset. Here strata are defined by hour-long intervals within the data, and sampled in proportion to their fraction of the total set size.

D. Find effective signs

The technique used to find effective signs is replicated from the algorithm in [6], and computes effective signs for a function, f, according to algorithm (2). First f is applied to all packets, and each distinct binary outcome is counted. Following this, all binaries are sorted by their appearance ratio and their *effective indicators* are computed. If their effective indicator surpasses the *sign threshold*, then all binaries, b, up until that index are returned as an effective sign, (f, b).

Consider a function $r_a(b)$ = appearance ratio of b, and the multiset $\mathscr{R}_{\leq r_a(b)} = \{r_a(x) \mid r_a(x) \leq r_a(b),\ x \in \mathit{binaries}\}$. The *effective indicator* is computed using formula (4), where $e_f(b)$ is the effective indicator and σ^2 is the population variance.

$$e_f(b) = \sigma^2_{\mathscr{R}_{\leq r_a(b)}} / \sigma^2_{\mathscr{R}_{< r_a(b)}} \tag{4}$$

A higher $e_f(b)$ implies (f, b) is an effective sign, as it means b has high influence on the variance of the multiset of all appearance ratios.

E. Validate results

Because adversaries do not disclose their scans, there is no ground truth when detecting collaborative scanners, which turns validating the results from the algorithm into a difficult task. However, there are some well-known scans that have been fingerprinted in the past, such as ZMap (IPId = 54321) and Massscan (IPId $\oplus$ f_{L2B}(IPDst)[2] $\oplus$ f_{L2B}(TCPSeq) = 0 $\wedge$ TCPAck = 0). These fingerprints will be considered the *"ground truth"*, and if they are present in the set of discovered fingerprints, then the result is considered valid.

IV. Experiments

Next, the algorithm is applied to network telescope data, and its performance is evaluated. Due to time constraints on the project, and many hours spent working on implementation, the algorithm is tested on only a small dataset in order to show what potential the algorithm might have. The implementation used in this experiment can be found on Github[3]. Section IV-A explains how the data used in the experiment is collected, section IV-B explains how the hyperparameters for the algorithm were obtained, and lastly, in section IV-C the identified fingerprints are analyzed.

[2]Extract *Left 2 Bytes* from value

[3]*https://github.com/jeltjongsma/detecting-collaborative-scanners*

A. Dataset

The data used in this experiment is collected by the TU Delft network telescope, consisting of three /16 subnets, during the 1st of February 2024. Some IP addresses in the telescope are routed to actual devices, but when they go offline their traffic is routed to the telescope instead. This means some intervals might contain more addresses than others, but in the case of fingerprint discovery this has no direct influence. On the 1st of February there were ~ 145.3 thousand IP addresses contained within the telescope.

From all collected packets only TCP SYN packets are extracted as these are used to survey active hosts and open ports [5]. Furthermore, preliminary testing showed the algorithm consistently returning fingerprints for ZMap (IPId = 54321, or similar, such as $f_{L1B}(\text{IPId}) = 212$), which make up nearly 50% of all packets and cloud the algorithms ability to detect other fingerprints. For this reason all packets with IPId = 54321 are filtered out prior to running the algorithm. Leaving a dataset containing ~ 246.4 million packets.

B. Hyperparameters

As mentioned earlier, Tanaka et al. [6] identified which header fields could be used for fingerprints. This experiment uses the same initial set, excluding the IP checksum, as it does not return in any previously found fingerprints, and it would lead to a large overhead when preprocessing packets.

$$F_0 = \{\text{IPId, SrcIP, DstIP}\} \cup \{\text{SrcPort, DstPort, Seq, Window}\} \tag{5}$$

Preliminary testing and hyperparameter optimization showed an initial sign threshold of 650 consistently returned at most 5 signs when using a sample size of 1610000 packets. The sample size is determined with a 99% confidence level, an error margin of 0.1%, and an expected variance of 50%. Compared to other sample sizes with a confidence level of 99.5% or higher, 1610000 consistently returned similar signs. Whereas samples with lower confidence levels, and/or higher error margins, returned a significantly higher number of signs that do not reappear when applying them to the entire dataset.

Only the XOR function was used as a *binary operation* on TCP functions. The *feature extractions* used were byte extractions from the left or right side of the binary, where they can both be 1 or 2 bytes in size.

Finally, Go's random package requires two separate uint64s to seed the generator. The seeds used during this experiment were 7992864126721545341 and 2670514226473436009.

C. Analysis of identified fingerprints

The algorithm is applied to the network telescope data and all identified fingerprints are extracted. The algorithm identified just three fingerprints, making up for 1.74% of all considered packets. Gr1 contained four signs, but two of them returned the same thing for every input. So they were manually merged. Even though the algorithm only found three fingerprints, the algorithm did find two for sets of packets that make up less than 0.5% of all packets, one of which makes up less than 0.0001% of sources. The full fingerprints can be seen in table I.

From this table it is clear the algorithm was not able to detect Massscan, which indicates that this approach does not work well as a method for detecting collaborative scanners. The algorithm did find 3 other prints. However, when looking at the number of distinct destination ports targeted by Gr0 and Gr1, they both cover a large set of ports. Which indicates that their fingerprints are coincidences, and they are not distributed scanners.

The final identified group, Gr2, targeted just 11 ports, using 5 source addresses. When looking at all ports targeted by Gr2 however, there appears to be no cohesion, as some are dedicated for e-mail transfer protocols (110, 143, 587), some are for http (80, 8080), and another 6 for different uses (21, 22, 179, 433, 5060, 6667). Looking up the source IP addresses

Name	Packets (%)	#sources (%)	#dest. ports	Fingerprint
Gr0	883 K (0.35%)	22880 (15.75%)	2819	$f_{L2B}(\text{Seq}) \oplus \text{Seq} \oplus \text{DstIP} = 33716$ $\wedge f_{L1B}(f_{L2B}(\text{SrcPort} \oplus \text{Seq})) \oplus \text{DstIP} \oplus \text{Seq} = 131$ $\wedge \text{DstIp} \oplus \text{Seq} \oplus f_{L2B}(\text{DstIP}) = 33716$
Gr1	2915 K (1.18%)	27552 (18.96%)	8731	$f_{L2B}(\text{Seq}) \oplus \text{Seq} \oplus \text{DstIP} = 33441$ $\wedge \text{DstIp} \oplus \text{Seq} \oplus f_{L2B}(\text{DstIP}) = 33441$ $\wedge f_{L1B}(\text{Seq}) \oplus \text{DstIP} \oplus \text{Seq} = 130$
Gr2	492 K (0.20%)	5 (3.44e−5%)	11	$\text{IPID} \oplus f_{R1B}(\text{Seq}) \oplus \text{Seq} = 61016$ $\wedge f_{L1B}(\text{IPId}) \oplus f_{R1B}(\text{Seq}) \oplus \text{Seq} = 238$

Table I: Fingerprints identified by algorithm 1. K denotes 10^3

on AbuseIPDB[4], there is, again, no cohesion. One of them is from the USA (0% confidence of abuse), another from Romania (61%), and three others from China (14%, 100%, 100%). Interesting to note is that the source from Romania sent $\sim$ 492 thousand packets, whereas the other 4 only sent 1. The IP addresses from Romania and China with high percentages might be part of other groups that the algorithm was not able to detect.

V. Responsible research

As a result of the algorithm designed in this paper, fingerprints for internet-wide scanners may be identified with much higher efficiency, leading to a larger set of fingerprinted internet-wide scanners. In the introduction was mentioned how many of these internet scanners have malicious intentions, which means the proposed method will contribute to a safer internet. Inherently meaning it has positive ethical implications on a worldwide scale.

As the algorithm relies on random sampling, I have decided to mention the seeds used for the generators. Also, the code behind the experiment in this paper will be released on Github, and the methodology and the experimental settings are thoroughly explained. This means that another researcher could perform this exact same experiment if they have access to the data. They can also perform a similar experiment on a different dataset.

[4] *https://www.abuseipdb.com/*

VI. Discussion

As mentioned in section IV-C, the algorithm was not able to identify the Massscan fingerprint, which it should be able to using this technique. When compared to previous results, such as Tanaka et al. [6], [4], the algorithm also did not manage to detect any other fingerprints. The algorithm was however able to generate a fingerprint for sets of packets that make up less than 0.5% or all packets, and less than 0.0001% of sources. This means that an iterative approach could work well to find small groups, but it does not seem to detect actual collaboration. This section covers why this might be the case, and what else I have tried to improve the algorithm.

The biggest problem with this approach is that the function generation scheme generates functions that have no actual meaning, such as DstIP $\oplus$ DstIP, or that by coincidence hold for a lot of packets, such as Fgpt0 and Fgpt1 from the experiment. And because at every iteration fingerprinted packets are filtered out, this (1) removes a lot of packets from actual groups, and (2) makes it so that the sign threshold needs to be much higher in order to not find over 50 signs at every iteration. In order to combat this I implemented a check that filters out signs if the packets they hold for target more than 20 distinct ports. This approach did seem to find more meaningful fingerprints, but unfortunately I did not have enough time to properly test this approach.

The first version of this algorithm first sampled a set of packets from the dataset, computed effective signs with a certain threshold, extracted the functions from those signs, and then computed effective signs over the entire dataset with just those functions. Because this is no longer the case in the iterative approach, the algorithm might lose some information when computing effective signs. Which means it might not find all effective signs. The algorithm could be adapted to do this within each iteration, but I did not have enough time to test this method.

Due to time constraints I was also not able to run the algorithm on the complete dataset from the network telescope, which contained data from the entirety of February.

Another, less likely, explanation for not finding any fingerprints is that they are no longer there. The most recent study on fingerprint identifiers was performed on data from September 2021, while the data in this experiment was collected in February 2024. This means it is possible that since then collaborative scanners have stopped implementing header field patterns in their packets. The algorithm was however able to detect ZMap's fingerprint during preliminary testing, so it is very unlikely there are no other internet-wide scanners embedding patterns in their packets anymore.

VII. Conclusion and future work

This study considered an approach by Tanaka et al. [6] to identify header field patterns, also known as "fingerprints", from network telescope data. Their method was adapted to be able to iteratively extract fingerprints from a large dataset using samples. The goal was to find out how this affects the accuracy of the approach.

Preliminary testing showed the algorithm can consistently identify ZMap's fingerprint (IPId = 54321). An experiment on network telescope data from a single day, showed the algorithm is able to identify fingerprints for sets of packets that make up less than 0.5% of all packets, and less than 0.0001% of sources. Analysis of groups that hold for the resulting fingerprints showed that they were not collaborative scanners, though. Which means they shared a pattern by coincidence.

Future work could focus on more effective function generation, or a post-processing step, where "junk" functions are dropped early. They can also look into a validation step for signs to make sure it does not coincidentally hold for unrelated packets. Furthermore, the experiment in this study only considered the XOR as a binary operation and byte extractions as feature extractions, and it was run on data from a single day. It would be interesting to see what happens when the sets of binary operations and feature extractions are larger, and the algorithm is tested on a larger dataset.

References

[1] Z. Durumeric, E. Wustrow, and J. A. Halderman, "Zmap: Fast internet-wide scanning and its security applications," in *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013*, S. T. King, Ed., USENIX Association, 2013, pp. 605–620. [Online]. Available: `https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/durumeric`.

[2] D. Moore, C. Shannon, G. M. Voelker, and S. Savage, "Network telescopes: Technical report," Cooperative Association for Internet Data Analysis (CAIDA), Tech. Rep., Jul. 2004.

[3] M. Kallitsis, R. Prajapati, V. Honavar, D. Wu, and J. Yen, "Detecting and interpreting changes in scanning behavior in large network telescopes," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3611–3625, 2022. DOI: `10.1109/TIFS.2022.3211644`.

[4] A. Tanaka, C. Han, and T. Takahashi, "Detecting coordinated internet-wide scanning by TCP/IP header fingerprint," *IEEE Access*, vol. 11, pp. 23 227–23 244, 2023. DOI: `10.1109/ACCESS.2023.3249474`. [Online]. Available: `https://doi.org/10.1109/ACCESS.2023.3249474`.

[5] H. Griffioen and C. Doerr, "Discovering collaboration: Unveiling slow, distributed scanners based on common header field patterns," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, Apr. 2020. DOI: `10.1109/NOMS47738.2020.9110444`.

[6] A. Tanaka, C. Han, T. Takahashi, and K. Fujisawa, "Internet-wide scanner fingerprint identifier based on TCP/IP header," in *Sixth International Conference on Fog and Mobile Edge Computing, FMEC 2021, Gandia, Spain, December 6-9, 2021*, IEEE, 2021, pp. 1–6. DOI: `10.1109/FMEC54266.2021.9732414`. [Online]. Available: `https://doi.org/10.1109/FMEC54266.2021.9732414`.

[7] Z. Durumeric, M. D. Bailey, and J. A. Halderman, "An internet-wide view of internet-wide scanning," in *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, K. Fu and J. Jung, Eds., USENIX Association, 2014, pp. 65–78. [Online]. Available: `https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/durumeric`.

[8] M. Antonakakis, T. April, M. D. Bailey, *et al.*, "Understanding the mirai botnet," in *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, E. Kirda and T. Ristenpart, Eds., USENIX Association, 2017, pp. 1093–1110. [Online]. Available: `https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis`.

[9] F. Shaikh, E. Bou-Harb, N. Neshenko, A. P. Wright, and N. Ghani, "Internet of malicious things: Correlating active and passive measurements for inferring and characterizing internet-scale unsolicited iot devices," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 170–177, 2018. DOI: `10.1109/MCOM.2018.1700685`. [Online]. Available: `https://doi.org/10.1109/MCOM.2018.1700685`.

[10] J. Xie, B. K. Szymanski, and X. Liu, "Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process," in *2011 IEEE 11th International Conference on Data Mining Workshops*, 2011, pp. 344–349. DOI: `10.1109/ICDMW.2011.154`.

[11] W. G. Cochran, *Sampling Techniques, 3rd Edition.* John Wiley, 1977, ISBN: 0-471-16240-X.