An uneven game of hide and seek

Hiding botnet CnC by encrypting IPs in DNS records

Andersen, Martin Fejrskov; Pedersen, Jens Myrup; Böck, Leon; Vasilomanolakis, Emmanouil

Published in:
2021 IEEE Conference on Communications and Network Security, CNS 2021

DOI (link to publication from Publisher):
10.1109/CNS53000.2021.9705029

Publication date:
2022

Document Version
Accepted author manuscript, peer reviewed version

An uneven game of hide and seek: Hiding botnet CnC by encrypting IPs in DNS records

Martin Fejrskov
Technology, IP Network and Core
Telenor A/S
Aalborg, Denmark
mfea@telenor.dk

Jens Myrup Pedersen
Cyber Security Group
Aalborg University
Copenhagen, Denmark
jens@es.aau.dk

Leon Böck
Telecooperation Lab
Technische Universität Darmstadt
Darmstadt, Germany
boeck@tk.tu-darmstadt.de

Emmanouil Vasilomanolakis
Cyber Security Group
Aalborg University
Copenhagen, Denmark
emv@es.aau.dk

***Abstract*—Botnets frequently use DGA and fast-flux techniques to ensure the availability of their command and control (CnC) infrastructure. However, the CnC IP addresses are still exposed in plain-text in publicly available DNS A records, which can be exploited by defenders to disrupt botnet operations. This paper presents the concept of the IP Generation Algorithm (IGA) as a novel method, usable by botmasters, to encrypt the CnC IP address in DNS records to avoid plain-text IP address exposure. This raises the bar for blacklisting malicious IP addresses, and can also be combined with existing techniques to further harden the CnC. For use by defenders, an IGA botnet detection method based on the combination of DNS and NetFlow data is presented and validated using an emulated botnet and an ISP data set.**

***Index Terms*—Botnet, NetFlow, DNS, encryption, detection**

I. Introduction

Many botnets use the DNS protocol and infrastructure to establish command and control (CnC) connections between a bot and the botmaster. Defenders can identify the botnet related domain names and block them in the DNS infrastructure. This is typically made significantly harder by the botmasters by using Domain Generation Algorithms (DGAs) to frequently create and register new domain names [1]. As the DNS responses to the DGA domains still reveal the IP address of the CnC host, defenders can choose to block DNS requests relating to that IP address, or block IP connections towards the CnC host IP address. This is made more difficult by the botnets by using fast-flux (FF) to frequently change the IP address registered with the DGA generated domain name.

The scarce resource in this game of hide-and-seek is the IP address. A usable IP address must represent an infected host, whereas the domain names can be freely chosen. For the botmaster, it would therefore be attractive not to expose the plain-text IP addresses of the CnC host in DNS records.

In this paper, we propose the IP Generation Algorithm (IGA) as a novel technique to encode the CnC host IP address. The botmaster would use the IGA to encode the plain-text CnC host IP address using a time-variable key, and register the domain with the encoded version. The bot would use the IGA to decode the retrieved address to reveal the plain-text IP address of the CnC host. Similar to a DGA, the purpose of the IGA is to generate random, legitimate looking IPs. However, as opposed to a DGA, the IGA is a two-way function.

A botmaster using the DGA and IGA techniques in combination could choose not to flux the IP address at all, effectively bypassing any FF detection techniques. Alternatively, the botmaster could use FF with a much higher frequency with IGA encoded IP addresses, as the amount of IP addresses available is not longer a limiting factor. In both cases, the botmaster does not reveal the plain-text IP addresses in the DNS messages.

The defender, however, faces potentially severe consequences. First, FF detection methods can be irrelevant when identifying malicious domains and IP addresses. Second, it would become impossible to generate IP address blacklists by only studying DNS data, as it does not reveal the plain-text IP address. Third, a defender unwittingly blocking an encoded IP address could result in the blocking of a benign host that matches the encoded address.

The primary contributions of this paper are:

- The IGA concept and a Python based implementation for encoding and decoding the CnC IP address.
- A DNS and NetFlow based IGA detection algorithm validated using Internet Service Provider (ISP) data and an emulated IGA botnet.

Section II of this paper introduces the threat model. Section III describes the IP Generation Algorithm, Section IV describes an IGA detection method that is validated in Section V. Section VI summarizes related work and VII concludes the paper.

II. Threat model

The goal of the botmaster is to use the DNS infrastructure to find one or more IP addresses AND subsequently create a connection towards the discovered IP address for CnC purposes. The botmaster is assumed:

- not to have control of any type of DNS servers.
- not to be able to actively obfuscate, spoof or modify the addresses and ports in the TCP/UDP/IP layers.
- to only use A-type DNS records (although the method presented in this paper could be extended to other types).

Funded by Telenor A/S and Innovation Fund Denmark, 2021.

- to be able to create decoy connections towards any IP addresses present in clear-text in A-type DNS responses.

The defender is assumed to be able to inspect/modify/block DNS traffic at the application layer and CnC connections at the transport and network layer. Also, the defender is assumed not be obstructed by the use of DNS-over-TLS or DNSCrypt, such as the case where the defender controls the resolvers. Thus, a typical ISP can assume the role of defender.

The botmaster could choose to combine IGA with DGA/FF techniques and thereby be subject to existing detection methods for these techniques. This paper will assume that such techniques are not used, and will not depend on them for detection.

III. The IP Generation Algorithm

This section describes the proposed implementation of the IGA. The objective of the IGA is to encode a globally routed, plain-text IP address into an encrypted version, that can also be represented in the form of a globally routed IP address.

The proposed IGA encoding implementation contains three steps: ranking, encryption and mapping. The description in this section will, for brevity, focus on the encoding only. The decoding process is simply the reverse of the encryption process. These three steps are depicted in Figure 1 and elaborated in the following paragraphs. The four coloured dots in Figure 1 each represent a CnC IP address and visualize how the encryption process changes their position in the respective value ranges.

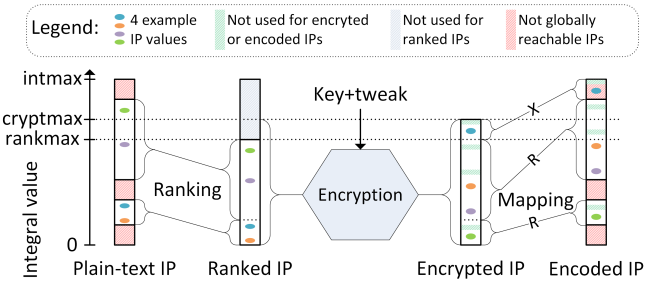

Fig. 1. *Conceptual illustration of the IGA encoding procedure containing three steps: ranking, encryption and mapping.*

A. Ranking

The purpose of ranking is to create a consecutive ordering of all globally reachable IP addresses in order to minimize the probability that the encryption and mapping steps will output an address that is not globally reachable. This excludes for example private, link-local, loopback nets etc. [2] [3]. An IPv4 address is represented by a 32 bit integer in the range $[0..intmax]$ where $intmax = 2^{32} - 1$. A total of $R = 592708608$ IP addresses are not globally reachable, these are represented by red colour in Figure 1. The pool of reachable IPs can be ranked by representing them in the interval $[0..rankmax]$ where $rankmax = intmax - R$. The values never provided as output by the ranking function are represented by blue colour in Figure 1, and the size of the blue area is the sum of sizes of the red areas.

TABLE I
Options for the selection of radix and input/output length

Characters	Radix	cryptmax-rankmax
2	60847	98722
3	1547	35636
4	247	19839394
5	82	5139745

B. Encryption

The input and output for the format-preserving encryption is not an integer, but two or more characters, each character consisting of a symbol from an alphabet. The number of different symbols in the alphabet is called the radix, radix $\in [2..2^{16}]$ [4]. Using this terminology, an IP address can be represented by four characters (bytes) and a radix of 256, as each byte can have a value in the range $[0..255]$. The integer representation of the input and output values is therefore the range $[0..cryptmax]$, where $cryptmax = radix^{characters} - 1$.

The number of characters and the radix should therefore be chosen such that $rankmax = cryptmax$, as this would ensure that a globally reachable IP address could be mapped one-to-one to another reachable IP address, but unfortunately this does not have an integral solution. Choosing $rankmax > cryptmax$ would make it impossible to encode a part of the reachable address space. This is very undesirable to a botmaster, as this would make it impossible to use those addresses for CnC. Choosing $rankmax < cryptmax$ is also undesirable, as the output of the encryption would then need to be mapped to a part of the IP space that is not reachable, indicating that the IP may be encrypted. For this paper, we will assume that botmasters prefer to create DNS A records with unreachable IP addresses rather than being unable to use certain IP addresses for CnC, and therefore opt for the $rankmax < cryptmax$ approach.

Solving $min(cryptmax - rankmax) \ll R$ for integral characters and radix reveal four options listed in Table I as well as the $cryptmax - rankmax$ difference. Two Python-based encryption/decryption libraries have been found. One supports only $radix < 37$ [5], which is not compatible with the options listed in Table I. Another supports only an even length and a $radix \leq 256$ [6], which matches only one possible option in Table I, namely a length of 4 and a radix of 247 yielding $cryptmax = 247^4 - 1$. Note that a botmaster could choose to use a different library with different limitations, and therefore be able to choose a better length and radix combination.

C. Mapping

The purpose of this step is to map the encrypted IP addresses ($[0..cryptmax]$) to a reachable IP addresses ($[0..intmax]$).

The green areas in Figure 1 represent the output values of encrypting the values in the blue area between $rankmax$ and $cryptmax$. This is illustrated by the summed size of the green areas being the same size as the blue area between $rankmax$ and $cryptmax$. As values between $rankmax$ and $cryptmax$ are never output by the ranking step, the values in the green areas are never output by the encryption step (for a given key).

TABLE II
Examples of plain-text IPs and their associated encoded version.

Plain-text	Encoded
193.0.2.255	192.238.197.236
8.8.8.8	154.141.220.55
152.13.43.124	32.72.64.180
212.220.255.3	199.22.251.26

However, as the specific, unused output values of the encryption depend on the key in use, the mapping function is unable to exclude these unused values before performing the mapping. Therefore, the mapping takes as input the range $[0..cryptmax]$ rather than $[0..rankmax]$.

If it was possible to set $rankmax = cryptmax$, the mapping operation would be the reverse of the operation performed by the ranking step. Instead, the mapping function is split into two functions. First, a function operating on encrypted IP values $\leq rankmax$ that performs the reverse operation of the ranking function, marked with an "R" in Figure 1. Second, a function that operates on encrypted IP values $> rankmax$, marked with an "X" in the figure. The latter values are mapped to a segment of the IP address space that is not reachable, and for the implementation provided by this paper, the 224.0.0.0/4 subnet reserved for Multicast is used.

D. General notes

A note on the use of a tweak is important. A tweak can be considered as a non-secret key, that should vary with each instance of the encryption [4]. In an IGA context, an obvious choice for the tweak is the DGA generated domain name: This will ensure that a single (plain-text) IP address would be encoded into several different IP addresses, one for each domain name, even though the key is held constant. As the DGA algorithm already includes time variability, using the domain name as tweak also eliminates the need for time variability in the encoding/decoding step of the IGA.

Without the ranking and mapping steps, the probability that the encryption would output a non-reachable IP address is $\frac{R}{intmax} = 0.138$. By including the ranking and mapping steps, this probability is reduced to $\frac{cryptmax - rankmax}{intmax} = \frac{(radix^{characters}-1)-(intmax-R)}{intmax} = 0.005$. With a crypto implementation supporting an odd number of characters (3) and a higher radix (1547) (as seen in Table I), this probability could be reduced even further to $8.3 \cdot 10^{-6}$.

Python code implementing the IGA algorithm described above is available in [7]. The implementation uses the FF1 encryption and decryption scheme provided by [6], which unfortunately does not support the use of tweaks. Table II contains examples of encoding using the key "someGoodKey".

Although the rank-and-encipher approach is proposed by [2] as a method for semantic-preserving encryption, our paper significantly extends it by considering multiple IP scopes and the associated ranking method, by introducing and solving the problems of differing set sizes for the ranking output and the encryption input, and by providing an implementation.

E. Summary

This section describes the IP Generation Algorithm and the three steps used to encode a globally routed, plain-text IP address into an encrypted version, that can in most cases also be represented in the form of a globally routed IP address. The encryption step maps a plain-text IP address into an encrypted IP address, and using the DGA generated domain name as tweak ensures time variability in the encrypted IP address. The ranking and mapping steps reduce the probability of the encoded IP being outside the globally reachable address space, thus decreasing the suspiciousness of the encoded IP.

IV. Detecting IGA botnets

The purpose of this section is to outline a method to detect the presence of an IGA-based botnet by using DNS and NetFlow data. The key idea of the method is to identify sets of source IPs that exhibit the group behavior expected by IGA bots: They resolve the same set of domain names and then create unnamed flows towards the same set of destination IP addresses. Each of the steps are described in more detail in the following subsections. The results of applying the method to a real dataset will be the topic of Section V.

Throughout this section, it is assumed that the DGA used by the botnet will generate a domain per day, as this is a common domain validity period for time-dependent DGAs [1]. Adapting the proposed method to a different frequency should be trivial.

A flow is defined as the daily aggregation of all packets sharing the same 5-tuple (protocol and source/destination IP address/ports), timestamped using the first-seen timestamp observed in NetFlow records for that 5-tuple.

As the algorithm is based on identifying source IPs that exhibit the same behaviour over k days, it is a requirement that DNS and NetFlow data is observed $\geq k$ days. From an adversary perspective, it is likely undesirable not to allow the majority of bots to establish CnC communication at least once a day. Therefore, the choice of k can be based on data availability. It is demonstrated in Section V that $k \geq 3$ is a necessity to avoid a high number of false positives in the detection.

A. Dataset reduction

The first step in the detection algorithm is to apply systematic white-listing so that irrelevant data is disregarded (such as flows or DNS request related to popular domains or CDN IPs). The main purpose of this is to reduce the processing requirements for the following steps. Therefore, not all reduction operations may be relevant to apply for all datasets. The remaining parts of this paper will refer to the result of applying all the desired reduction operations as two sets, a set of DNS Resource Records (RRs), $D_{reduced}$, and a set of flows, $F_{reduced}$.

The detection method proposed in this paper is considered complementary to existing DNS-only or NetFlow-only methods, therefore a black-listing approach based on such methods is intentionally not used. The following notation is used:

- D_{all}: All collected A type DNS RRs.
- $D_{all}^{rdataIP}$: The unique rdata IPs in D_{all}
- D_{all}^{FQDN}: The unique qname FQDNs in D_{all}

- $D_{all}^{2ndlevel}$: The unique second level domains found by extracting these from all entries in D_{all}^{FQDN}
- F_{all}: The flows in all collected NetFlow packets.

1) Outside originated flows: All flows originating from outside the network under observation should be white-listed, as the DNS requests related to these flows will not be observable. Notice that when using sampled NetFlow, it is often not possible to identify the flow origin.

2) Frequent second level domains: Two techniques can be used to white-list both DNS RRs and flows based on frequently seen second level domains:

A popular white-listing technique is to disregard any domain names found in lists of popular second level domains (such as the Alexa Top list), which will be denoted $W_{2ndlevel}$. The rationale of this is that is unlikely that a botmaster can retain long-term, unnoticed control of either domains or servers used by such high-volume organisations. The list is used to find the white-listable RRs, $D_{whitelisted} = D_{all} \ltimes W_{2ndlevel}$, which can then be used to find two reduced sets, $D_{reduced} = D_{all} \setminus D_{whitelisted} = D_{all} \triangleright W_{2ndlevel} = D_{all} \triangleright D_{whitelisted}^{2ndlevel}$ and $F_{reduced} = F_{all} \triangleright D_{whitelisted}^{rdataIP}$, that do not include RRs or flows relating to the white-listed second level domain names. The threshold $T_{secondlevel} = \frac{D_{reduced}}{D_{all}}$ is defined to indicate how large a fraction of DNS RRs is retained.

Another technique is to white-list any DNS RRs with qnames or rnames containing FQDNs or second level domains, that are requested by more than $T_{maxbots}$ source IPs, the set of white-listed DNS RRs denoted $W_{2ndlevel}(T_{maxbots})$. The rationale of this is that a specific DGA FQDN will never be requested by a benign client, therefore FQDNs requested by more than $T_{maxbots}$ source IPs will be benign, if the number of bots in the observed network is $\leq T_{maxbots}$. This can be used to construct $D_{reduced}$ and $F_{reduced}$ in a similar way as described above. This technique should only be used to white-list flows if the botnet is not assumed to deploy CnC hosts on servers also serving benign content.

Although these two techniques overlap, practical experiments show that both provide distinct dataset reductions.

3) Frequent rdata IPs: It seems tempting to also white-list DNS RRs and flows based on frequently observed rdata IP addresses. This is, however, not a feasible strategy, as the botmaster has unlimited control over the contents of the rdata IP address, and could therefore choose to implement an IGA, that only uses frequently seen IP addresses (CDN IPs etc.) as output space for the IGA algorithm.

4) Frequent destinations: Two techniques can be used to white-list flows based on frequently seen flow destinations:

For NetFlow data, a popular technique is to disregard Content Delivery Network (CDN) IPs, denoted W_{CDN}, solely serving static/benign content, such as Youtube, Facebook and Akamai CDNs (but not for example Amazon or Azure CDNs, as these can host private virtual servers). The rationale of this is that it is unlikely that a botmaster can retain long-term, unnoticed control of such servers. Notice that as ISPs deploy CDNs locally in their networks, the CDN IPs will likely differ between ISPs. This can be used to construct a reduced set of flows, $F_{reduced} = F_{all} \triangleright W_{CDN}$.

Another technique is to construct a set of white-listed flow destination endpoints (defined by a destination IP, protocol and port combination), denoted $W_{destination}(T_{maxbots})$, such that a flow is in this set if more than $T_{maxbots}$ source IPs contact a specific endpoint. The rationale of this is that a specific botnet CnC endpoint will never be requested by a benign client (assuming that the CnC software does not share the destination port with benign software, such as Apache). This can be used to construct a reduced set of flows, $F_{reduced} = F_{all} \triangleright W_{destination}(T_{maxbots})$.

Although these two techniques also overlap, practical experiments show that both provide distinct dataset reductions.

5) Domains seen yesterday: FQDNs and second level domain names seen in DNS requests the day before the day under analysis form a large white-list, $W_{yesterday}$. The rationale of this is that the purpose of a DGA algorithm is to create a new and unique FQDNs each day, so that a listing of the domain name on a free or commercial domain name blacklists becomes irrelevant. This can reduce the set of DNS RRs to $D_{reduced} = D_{all} \triangleright W_{yesterday}^{2ndlevel} \triangleright W_{yesterday}^{FQDN}$.

B. Unnamed flows

It is an inherent property of using an IGA that the DNS RRs and flows will only be explicitly related by the source IP address. The A record IP in the DNS response is encoded, and therefore not identical to the destination IP in the flow.

Any flows for which a matching DNS record can be found is called a named flow, $F_{named} = F_{reduced} \ltimes D_{all}$. These flows are not relevant for IGA detection and can be white-listed, leaving only the unnamed flows, $F_{unnamed} = F_{reduced} \setminus F_{named} = F_{reduced} \triangleright D_{all}$. The matching criteria are source IPs, timestamps, rdata record and destination IP.

When identifying named flows, the aspect of time is relevant. A flow could be named by the first preceding DNS record, or by all preceding DNS records within a certain time window, for example the window set by the TTL or the window of the current day. In order to make $F_{unnamed}$ as small as possible by making the white-list F_{named} as large as possible, a time window of the current day seems to be a reasonable approach.

Notice that DNS requests for which a related NetFlow record can be found should not be white-listed, as the botmaster could choose to initiate traffic towards the encoded IP address in the DNS request as a decoy to avoid detection.

C. Re-named flows

Given a set of flows, $F_{unnamed}$, that have no related DNS response, and given a set of DNS responses, $D_{reduced}$, the purpose of the remaining steps of the detection algorithm is to identify the specific entries from each set that are actually created by the IGA botnet. To facilitate this, this section introduces the concept of a re-named flow.

A re-named flow is a flow from $F_{unnamed}$ augmented with a relevant DNS RR from $D_{reduced}$. A DNS RR is considered relevant, if the flow and the DNS request originate from the

same source IP address, and if the start time of the flow is within a certain time window, $T_{delaymax}$, of the DNS response. The set of re-named flows, R, is therefore given by the theta-join, or selected cartesian product, $R = F_{unnamed} \bowtie_\theta D_{reduced} = \sigma_\theta(F_{unnamed} \times D_{reduced})$, the θ condition being the source IP and time window constraints.

The choice of $T_{delaymax}$ requires further consideration, as a DNS request should not be expected to be immediately followed within few seconds by an observed NetFlow record. There are multiple reasons for this: A botmaster could deliberately introduce a variable time delay between the DNS lookup and the CnC connection in order to evade detection, including a delay exceeding the TTL of the DNS RR. Also, the NetFlow data used can be sampled, and therefore the observed flow start timestamp is not necessarily equal to the actual flow start time.

Note that DGA domains are typically registered with a low TTL in order to enable fast-flux. However, as the IGA eliminates the need for fast-flux in the CnC phase, such low TTLs cannot be assumed to be used for IGA based botnets.

R contains a number of entries that are irrelevant for the following IGA detection steps. Only the qname, source IPs and the destination IP are relevant, and $R_{reduced}$ is constructed from R by removing any other information and removing duplicates.

Following the assumption that DGA algorithms generate a new domain name on a daily basis, the processes of reducing the dataset, and identifying unnamed and re-named flows can be performed on a daily basis as well. Therefore, given that DNS and NetFlow data is available for k days, k sets of re-named flows, $R^{1..k}_{reduced}$, can be constructed on an individual basis to form $U = \cup_{i=1}^{k} R^{i}_{reduced}$.

D. Vertices and edges

The next step of the IGA detection algorithm is to build a graph based on all of the re-named flows, U. The purpose of the graph is to identify sets of source IPs that exhibit the same behaviour by resolving the same domain names and connecting to the same destination IPs.

1) Vertices: Subsets of U are created, where a subset consists of the entries of U that share a specific combination of qname, destination IP and day. The size of a subset is equal to the number of unique source IP addresses with this combination. Subsets containing $< T_{minbots}$ source IPs are discarded. Subsets containing $\geq T_{minbots}$ entries form the set V, after removing the destination IP information and duplicates. V is therefore a set of source IP, qname and day triplets.

Subsets $v_1 ... n$ of V are created, where a subset v_i consists of all the source IPs that share a specific combination of qname and day. In other words, v_i represents the source IPs that resolve the same domain name on a specific day. Each of the subsets of V are represented by vertices in a graph.

2) Edges: A bidirectional edge connects two vertices if the Jaccard similarity, $J()$, of the two sets of source IPs, v_i and v_j, is larger than a given threshold, $T_{jaccard}$ determined as $T_{jaccard} > J(v_i, v_j) = \frac{v_i \cap v_j}{v_i \cup v_j}$. Calculating this only for vertices representing different days represents the expected behaviour of a set of IGA bots (represented by their source IPs) resolving a different domain each day.

The similarity threshold $T_{jaccard}$, $0 \geq T_{jaccard} \geq 1$ should be chosen sufficiently high to eliminate false positives and sufficiently low to make it unattractive for the botmaster to try to avoid detection by generating many DGA domains each day, or by instructing too high a fraction of bots to not create CnC connections each day.

E. Cliques and communities

Having constructed a graph where two vertices are connected if they share a certain fraction of their associated source IPs enables the final steps of the detection algorithm: Clique and community detection.

A k-clique is a set of vertices that are fully connected to at least $k - 1$ other vertices, representing sets of source IP addresses that resolve the same set of domain names and then create unnamed flows towards the same set of destination IP addresses across all k days.

As several DGA domains could be created each day by a botnet, a botnet may be represented by several k-cliques. Therefore, the graph is used to identify k-communities: A k-community is the union of all cliques of size k that can be reached through adjacent (sharing k-1 nodes) k-cliques [8].

A k-community could represent an IGA botnet, where the source IPs related to the community represents the bots. However, a community could also represent other structures than IGA botnets, such as regular botnets generating traffic to some other, common destination (e.g. for attack purposes) based on information obtained through the CnC channel (thereby creating an unnamed attack flow).

F. Summary

This section describes the steps of the IGA detection algorithm, which includes a number of possible data reduction techniques. Based on DNS responses and NetFlow records collected over k days, a number of k-cliques are found that may represent an IGA botnet. Several properties are worth noting:

- No assumptions are made about the similarity of two different qnames when identifying cliques, as is the case for example in semantic based DGA detection methods.
- The rdata IP value is only used for data minimization and to identify unnamed flows, but is not used in the re-naming of flows or the clique identification. The rdata IP value is often key in DGA detection methods for example when identifying IP addresses with many associated qnames.
- No assumptions are made about the distribution or similarity of source or destination port numbers (except for when performing white-listing), which is often the case for NetFlow-based detection methods.
- The flow sizes (packets, bytes or time) are not used, which is often the case for NetFlow-based detection methods.

As the botmaster has almost full control of all of the aforementioned features, these properties should be considered important to any detection algorithm.

```
for sourceip in $sourceiplist; do
 dnsflood -n 1 -s $sourceip $botdomain $resolver
 sleep 1s
done
sleep 10s
let sourceport=$RANDOM
for sourceip in $sourceiplist; do
 for ((i=1;i<=512;i++)); do
  sendip -p ipv4 -is $sourceip -p tcp -ts
      $sourceport -td 23 -tfs 0 12.34.56.78
 done
 let sourceport++
 sleep 1s
done
```

Listing 1. *IGA traffic emulation script.*

V. Validation of detection method

In this section, IGA botnet traffic will be injected into DNS and NetFlow data from an ISP to validate the IGA botnet detection method described in the previous section. The following subsections describe in further detail the ISP data available, how emulated IGA traffic is injected into the ISP data, and the specific values chosen for the various thresholds used in the detection algorithm. Finally, the results of running the detection algorithm are presented and discussed.

A. ISP DNS and NetFlow data

DNS and NetFlow data for the 1,5M mobile and 100k broadband subscribers of Telenor Denmark is used for validation. NetFlow data is collected at the Border Gateway Protocol (BGP) Autonomous System (AS) border routers using a sample rate of 1:512. This traffic therefore represents all Internet traffic entering and exiting Telenor Denmark's network. DNS data is collected at DNS resolvers by collecting all DNS response packets. The resolvers are only accessible to Telenor Denmark subscribers, and they are the default choice for all subscribers.

NetFlow and DNS data are anonymized for legal reasons by truncating the internal (subscriber) IP to a /24 prefix for non-NAT'ed subscribers (or truncating the port for NAT'ed subscribers) as well as a number of other measures less relevant to this paper. The anonymization policy applied follows the guidelines of [9] except that varying levels of anonymization is applied to the NetFlow destination IPs and the DNS rdata IPs in order to evaluate the effect of anonymization on the results. Due to anonymization of source IPs, all traffic will originate from only approximately 15k prefixes, each representing somewhere between 0 and 256 customers.

B. Proof-of-concept IGA botnet data

To emulate the behaviour of an IGA botnet, the Bash script available in Listing 1 is used. The script is run once every day to emulate the behaviour of an IGA botnet consisting of 30 bots among the Telenor customers, each of which contact the botnet CnC infrastructure once a day by means of a single DNS lookup and a TCP flow.

The fixed set of 30 IP addresses used as faked source IP addresses are selected from the various prefixes used by Telenor for customers. For each source IP, a single DNS request is created of type A for the domain bot-test.testlab.telenor.dk, giving the response value 192.0.2.3. For each source IP, 512 TCP packets are then created with a random source port, and destination 12.34.56.78:23. This emulates the IGA bot behaviour of decoding the IP address 192.0.2.3 to the plaintext IP address 12.34.56.78, and knowing the destination port number by some other means. The choice 512 packets is made in order to increase the probability that at least one of the packets from each bot is represented in the collected NetFlow records that use a sample rate of 1:512.

As the domain registration process is not scripted, the emulated botnet will always resolve the name bot-test.testlab.telenor.dk. This specific domain is therefore made exempt to the minimization step that removes domains seen the day before. This is implemented in practice by prefixing the domain name with the number of the day of the observation, such as 1.bot-test.testlab.telenor.dk.

TABLE III
Thresholds used for validation.

Metric	Symbol	Value
Number of days observed	k	3-5
Ratio of retained DNS RRs	$T_{secondlevel}$	0.05
Number of whitelisted IPs	$\|W_{cdnip}\|$	$1.3{\cdot}10^6$
Maximum number of bots expected	$T_{maxbots}$	500
Maximum delay from DNS request to flow	$T_{delaymax}$	10 min
Jaccard similarity threshold	$T_{jaccard}$	0.24
Minimum number of bots expected	$T_{minbots}$	4

C. Detection algorithm parameters

All of the recommended methods for reducing the datasets described in Section IV-A are applied. The choices of the detection threshold parameters used for validation are summarized in Table III, and where relevant, the choice of each parameter is elaborated in the following paragraphs.

1) k: Data is collected in two periods, p_1 from 20210318 to 20210321 (4 full days) and p_2 from 20210418 to 20210418 (5 full days). Values of k from 3 to 5 are used, always starting at the first day of the period.

2) $T_{secondlevel}$: To white-list frequent second level domain names, the 1500 most popular second-level qnames and the 200 most popular second-level rnames (where rnames and qnames differ) were white-listed. The qname approach causes approximately 95% of all responses to be white-listed. This is approximately equal to removing second-level domains for which there is more than 100k queries per day. The rname approach causes approximately 95% of all RRs to be white-listed, yielding $T_{secondlevel} = 0.05$. This is approximately equal to removing RRs where more than 170k RRs per day contain a particular second level domain.

3) $T_{jaccard}$: For the Jaccard similarity threshold, a value of $T_{jaccard} = 0.24$ is chosen, meaning that at least a fourth of the source IP addresses in two vertices must be common to the two vertices, for the vertices to be considered connected.

TABLE IV
Example validation data metrics from the first day of the first time period.

Metric	Symbol	Count
Total DNS responses	$\|D_{all}\|$	$3.64 \cdot 10^9$
Total flows	$\|F_{all}\|$	$150 \cdot 10^6$
Minimized DNS responses	$\|D_{reduced}\|$	$631 \cdot 10^3$
Minimized flows	$\|F_{reduced}\|$	$17.9 \cdot 10^6$
Unnamed flows	$\|F_{unnamed}\|$	$13.3 \cdot 10^6$
Relevant re-named flows	$\|R_{reduced}\|$	$33.0 \cdot 10^6$

TABLE V
Validation data detection results.

Result set	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Period	1	1	2	2	2	2	2
k	3	4	3	4	5	4	4
Destination IP	/32	/32	/32	/32	/32	/24	/16
Vertices	333	395	530	691	853	1037	598
k-cliques	4k	24k	16k	49k	111k	60k	22k
k-communities	6	2	9	5	18	22	8

D. Results

Seven different result sets are collected and summarized in Table V. The different result sets vary in which of the two time periods are used, how many days of data is used. Also, two result sets are using a truncated version of the destination IP addresses (using the /24 and /16 version of the IP address), in order to evaluate the effect of anonymizing the destination IP adress. Example reference metrics can be found in Table IV.

The full graph for result set 2 is found in Figure 2. The graph for result set 1 (k=3) is structurally similar. The IGA detection algorithm detects two communities. The community in cluster 1 contains the vertexes representing the 4 domains and 26 of the 30 source IPs used in the emulated IGA botnet. The community in cluster 2 contains the vertices representing 124 domains (for example ecy.eu, rfn.de, rae.biz, pms.mx, egln.vg, pbp.ru) and 7 source IPs. Although other clusters exist, they are not 4-communities.

The graphs for result set 3-7, which are all from the same time period, are structurally similar to each other. They all depict the IGA botnet as a separate cluster, less than 10 smaller, non-community clusters, and finally one very large cluster containing the remaining vertices and communities.

E. Discussion

For all result sets, the IGA detection algorithm successfully detects the k-community (containing a single k-clique) with the bot-test.testlab.telenor.dk domains that represent the emulated botnet (cluster 1 in Figure 2). However, additional communities are also detected in all result sets, showing that further data processing is needed. Although this does indicate a high false positive rate for the detection algorithm, we still consider the detection algorithm successful, as it reduces a nationwide traffic data set to a manageable number of 6-22 positives.

As can be deduced from Figure 2 that depicts k=4, using k=2 would provide many false positives. Using k=3 results in six 3-communities and using k=4 results in two 4-communities, which we consider a quite low number given the size of the observed network. As expected, the number of vertices and cliques seem to grow when data from additional days is used. Interestingly, the number of communities detected is lower for $k = 4$ than for $k = 3$ or $k = 5$. This could indicate a sweet spot in the balance of too little or too much data.

Anonymizing the destination IP address by removing the last octet yields result set 6 and removing the two last octets as recommended by [9] yields result set 7. In both cases, the emulated botnet is identified as a k-community and as a distinct cluster. This, combined with the total number of k-communities still being relatively low, could indicate that the IGA algorithm may be feasible to run on anonymized data.

Cluster 2 in Figure 2 could be an IGA botnet, however further investigations in this area were inconclusive. A cluster with similar domain names could not be found in period 2.

Some of the non-community clusters, such as cluster 3, include domain names that look like they could be created by a DGA, and these are probably regular (non-IGA) botnets. Although regular bots do not produce unnamed flows, they produce a lot of DNS requests, and these may by random chance be attributed to non-white-listed, unnamed flows towards common destinations. This indicates that the IGA detection could be an novel method for identifying non-IGA botnets as well.

Cluster 4 contains a lot of mailserver-related names. Although this is not a 4-community, it is surprisingly densely connected. By eliminating day 1 and only looking at at dataset for days 2 to 4, this cluster is reduced to a much smaller cluster of 6 vertices. The cluster is not found in the dataset for period 2. Although this cluster could be non-IGA botnet related as well, it could also belong to SMTP mail servers lookup up the IP address of the sending domain in order to verify the sender, verify SPF/DMARC records or similar. As for the non-IGA botnets, this would create a lot of DNS requests that are by random chance attributed to non-white-listed, unnamed flows.

VI. Related work

The related work falls into current and proposed IGA implementations and IGA detection techniques. Both categories have related work focusing on DNS tunnelling techniques, however, these presuppose that the botmaster has control of the DNS server infrastructure, and are therefore incompatible with the threat model of this paper.Similarly, work focusing on DNS-over-TLS or DNSCrypt techniques is not considered relevant. These techniques describe application-layer encryption between the client and the resolver, whereas the IGA technique describes record-level encryption between the client and the botmaster.

A. IGA implementations and encoding schemes

The Sage 2.0 botnet uses a conceptually different, but similarly named IP Generation Algorithm (using the IPGA acronym) to randomly contact CnC servers among 7702 addresses within four predefined /16 subnets [10]. Most of these addresses are expected to be benign, and the actual CnC IP addresses are not conveyed through the DNS system as suggested in this paper. Therefore, the IPGA and suggested IGA techniques differ significantly.

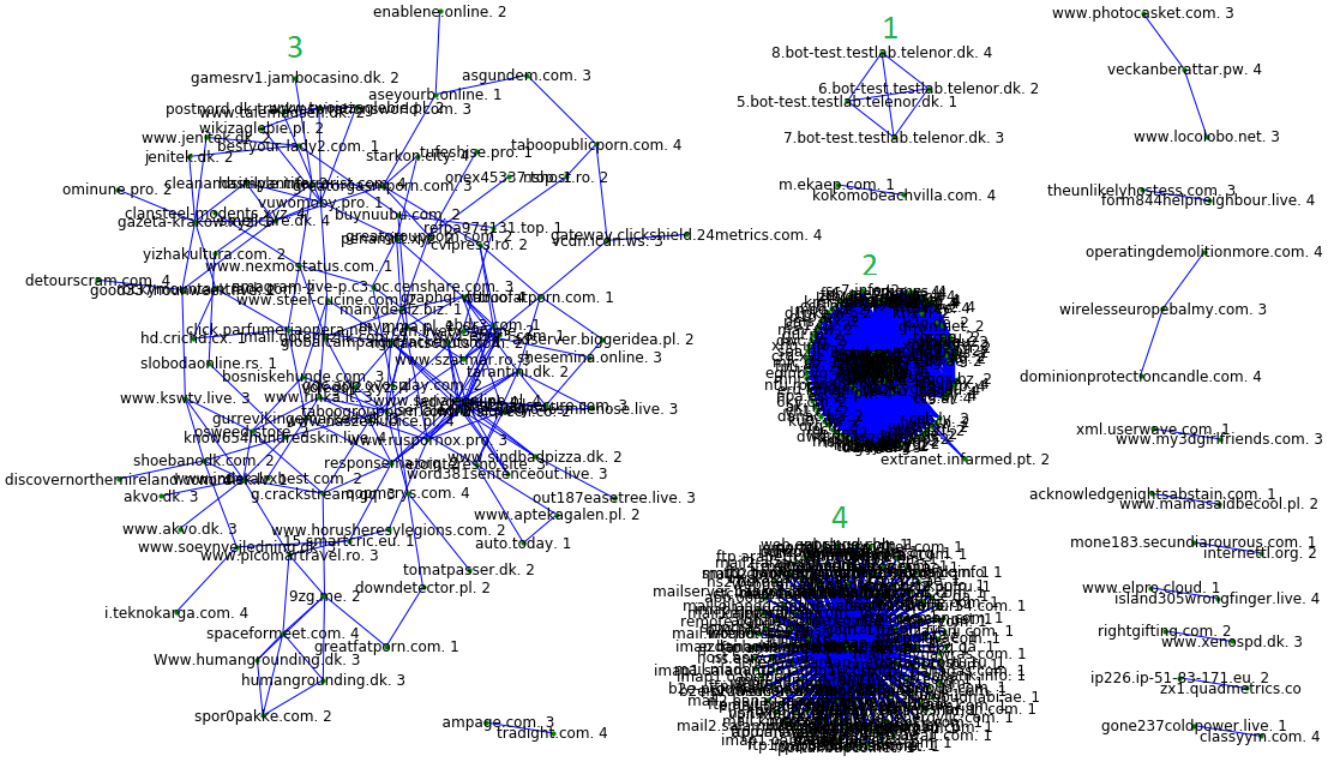

Fig. 2. *A graph depicting the results of the IGA detection algorithm for result set 2, using p_1, $k = 4$ and no destination IP address truncation. Green numbers identify clusters for reference.*

The purpose of Cryptographically Generated Addresses (CGAs) is to generate an IPv6 address with an embedded public key [11]. This scheme transforms the IPv6 address, but does not hide the real IP address. This technique is therefore also of less relevance to our paper.

Much work is available various techniques to anonymize IP addresses [12] [13]. The focus in these papers is typically privacy preservation using one-way mapping functions, such as truncation or hashing. However, the IGA technique requires a two-way mapping as well as an encryption/decryption key, as bots need to be able to obtain the plain-text version of the IP address. This can be achieved using semantic- and format-preserving encryption as described in [2] using cryptographic algorithms as described in [14] and [4]. Preserving semantics is important, as not all possible IP addresses should appear in DNS records. Preserving the format is important, as an encoded IP address must consist of exactly 4 bytes, the length used for IPv4 addresses. If an encoded IP address does not adhere to normal semantics and formats, it will be easy to detect. A different encryption scheme focusing on prefix-preservation is introduced in Crypto-PAN [15], however, the prefix-preservation is an undesired property in our use case, and furthermore the scheme is not semantics-preserving.

A technique for using DNS resolvers as bridge in a two-way communication between two hosts without control of the DNS infrastructure is presented in [16]. However, only clients sharing the same DNS resolver can communicate.

B. IGA detection

To show that a DNS record is implicitly, but not explicitly related to a CnC flow, it is necessary to analyse both DNS and flow data. For this paper, NetFlow/IPFIX data will be used to represent flow data, as this is the simplest, standardized method.

Although commercially available products are difficult to survey due to lack of detailed information, it is appropriate to include for completeness. For NetFlow based detection, Cisco Stealthwatch provides by far the most in-depth documentation of detection capabilities and methods [17]. For DNS based detection, the deepest documentation seems to be provided by Cisco Umbrella (formerly OpenDNS) [18], Infoblox Advanced DNS protection [19] and HP Arcsight DNS Malware Analytics (formerly Damballa) [20]. None of the surveyed products document the ability to perform behavioural analysis on the combination of DNS and NetFlow features, which suggests that commerciall products will not be able to detect IGAs.

Based on academic survey papers, it seems that combining the DNS and NetFlow feature sets may not be a widespread approach. Some papers use NetFlow analysis on DNS packets only, by simply detecting an abnormal amount of DNS traffic within a specific time period for specific hosts [21] [22]. This is still considered a NetFlow-only based approach not applicable to IGA detection, as no layer 7 information is used from the DNS packets, only general knowledge about the DNS protocol.

IP source address entropy derived from NetFlow logs and the

ratio of nxdomain responses derived from DNS logs are used by [23] as detection and validation methods. It is, however, not clear if the two methods are used in combination or independently, thus effectively being a NetFlow based method combined with an DNS based method, rather than a method that combines DNS and NetFlow data before applying the method.

Fuzzy pattern recognition is applied by [24] to both DNS and network flows in a two-stage approach. The DNS related features used are based on the inter-arrival time of requests, and total and failed number of responses. The NetFlow related features used for each destination address are based on the request-response time interval, the number of requests and the payload size. The pattern recognition is then applied in each of these phases, thus analyzing DNS and flow data separately.

IP flows that can not be related to a previous DNS lookup (denoted unnamed flows or non-DNS connections) is one of the topics of [25] and [26]. Such flows account for 5-10% of all internally originated flows in one of the available datasets [25]. As IGA flows will appear unnamed, this property is clearly relevant to exploit in IGA detection.

The topic of [27] is to use traffic analysis on encrypted DNS traffic (DNS-over-TLS etc.) to identify nxdomain response patterns from DGA based botnets. The paper shows that time series analysis and packet size diversity can be used to create IoCs for several specific botnet families. The presented techniques could make it possible to extend the threat model of our paper to allow encrypted DNS traffic as well.

VII. Conclusion

This paper presents the novel concept of the IP Generation Algorithm (IGA) as a method usable by botmasters to avoid exposing the CnC IP address in plain text in DNS A records. An implementation of the concept is provided, and a detection method is presented and validated using an emulated botnet and data from Telenor Denmark's network. Although the results do not indicate that any botnets currently use the IGA method, the method could in the future potentially supplement or replace existing DGA and fast-flux methods.

Modifications to the detection algorithm, or entirely different detection algorithms, suitable for real-time threat prevention by firewalls should be developed, as the method outlined in this paper is very reactive, as it requires several days of retained data for detection.

Looking further into the detecting and eliminating potential decoy flows or applying existing DGA detection methods on top of the detection method described in this paper could potentially reduce the number of false positives, and could therefore also be interesting topics to address in future work.

References

[1] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, and E. Gerhards-Padilla, "A Comprehensive Measurement Study of Domain Generating Malware," USENIX Security Symposium, 2016. [Online]. Available: https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_plohmann.pdf

[2] G. Ládi, "Semantics-Preserving Encryption for Computer Networking Related Data Types," AIS: International Symposium on Applied Informatics and Related Areas, 2017. [Online]. Available: https://www.crysys.hu/publications/files/Ladi2017ais.pdf

[3] Internet Assigned Numbers Authority, "IANA IPv4 Special-Purpose Address Registry," 2020. [Online]. Available: https://www.iana.org/assignments/iana-ipv4-special-registry/iana-ipv4-special-registry.xhtml

[4] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption," National Institute of Standards and Technology. [Online]. Available: http://dx.doi.org/10.6028/NIST.SP.800-38G

[5] K. P. Dyer, "FFX," 2018. [Online]. Available: https://github.com/kpdyer/libffx

[6] W. J. Buchanan, "FFX schemes," 2020. [Online]. Available: https://asecuritysite.com/encryption/ffx

[7] M. Fejrskov, "IGA: A python module for format- and semantics preserving encryption and decryption of IP addresses," 2021. [Online]. Available: https://github.com/Fejrskov/IGA

[8] NetworkX, "NetworkX algorithms, k clique communities," 2002. [Online]. Available: https://networkx.org/documentation/stable//reference/algorithms

[9] M. Fejrskov, J. M. Pedersen, and E. Vasilomanolakis, "Cybersecurity research by ISPs: A NetFlow and DNS Anonymization Policy," International Conference on Cyber Security And Protection Of Digital Services, 2020. [Online]. Available: https://doi.org/10.1109/CyberSecurity49315.2020.9138869

[10] Swiss Government Computer Emergency Response Team, "Sage 2.0 comes with IP Generation Algorithm (IPGA)," 2017. [Online]. Available: https://www.govcert.admin.ch/blog/27/sage-2.0-comes-with-ip-generation-algorithm-ipga

[11] T. Aura, "Cryptographically Generated Addresses (CGA)," 2005. [Online]. Available: https://tools.ietf.org/html/rfc3972

[12] N. Dijkhuizen and J. Ham, "A survey of network traffic anonymisation techniques and implementations," ACM Computing Surveys, 2018. [Online]. Available: https://doi.org/10.1145/3182660

[13] E. Boschi and B. Trammel, "IP Flow Anonymization Support, RFC 6235," 2011. [Online]. Available: https://doi.org/10.17487/RFC6235

[14] M. Bellare, T. Ristenpart, P. Rogaway, and T. Stegers, "Format-Preserving Encryption," SAC: Selected Areas in Cryptography, 2009. [Online]. Available: https://doi.org/10.1007/978-3-642-05445-7_19

[15] J. Xu, J. Fan, M. Ammar, and S. Moon, "Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography-based scheme," IEEE International Conference on Network Protocols, 2002. [Online]. Available: https://doi.org/10.1109/ICNP.2002.1181415

[16] D. Bernát, "Domain Name System as a Memory and Communication Medium," SOFSEM 2008: Theory and Practice of Computer Science, 2008. [Online]. Available: https://doi.org/10.1007/978-3-540-77566-9_49

[17] Cisco, "Threat detection, Cisco Stealthwatch at work," 2019. [Online]. Available: https://cisco.bravais.com/s/SXhrFcFSKfsqJnOUyF2J

[18] ——, "Cisco Umbrella Investigate," 2019. [Online]. Available: https://docs.umbrella.com/investigate-ui/docs/

[19] Infoblox, "Infoblox advanced DNS protection," 2019. [Online]. Available: https://www.infoblox.com/wp-content/uploads/infoblox-datasheet-infoblox-advanced-dns-protection.pdf

[20] HP, "HP ArcSight DNS malware analytics datasheet," 2015. [Online]. Available: http://www.hp.com/sbso/hpinfo/newsroom/DNSMalwareAnalyticsDataSheet.pdf

[21] M. Grill, I. Nikolaev, V. Valeros, and M. Rehak, "Detecting DGA malware using NetFlow," IFIP/IEEE International Symposium on Integrated Network Management, 2015. [Online]. Available: https://doi.org/10.1109/INM.2015.7140486

[22] D. Huistra, "Detecting reflection attacks in DNS flows," 2013. [Online]. Available: https://pdfs.semanticscholar.org/4ad8/24537f212f70e25e4cbab55498f5a8e43942.pdf

[23] R. Hananto, C. Lim, and H. P. Ipung, "Detecting network security threats using domain name system and NetFlow traffic," ICCSP: International Conference on Cryptography, Security and Privacy, 2018. [Online]. Available: https://doi.org/10.1145/3199478.3199505

[24] K. Wang, C.-Y. Huang, S.-J. Lin, and Y.-D. Lina, "A fuzzy pattern-based filtering algorithm for botnet detection," Computer Networks, 2011. [Online]. Available: https://doi.org/10.1016/j.comnet.2011.05.026

[25] M. Janbeglou, "Understanding and Controlling Unnamed Internet Traffic," 2017. [Online]. Available: https://researchspace.auckland.ac.nz/handle/2292/36323

[26] B. Rahbarinia, R. Perdisci1, A. Lanzi, and K. Li, "PeerRush: Mining for Unwanted P2P Traffic," Journal of Information Security and Applications, 2014. [Online]. Available: https://doi.org/10.1016/j.jisa.2014.03.002

[27] C. Patsakis, F. Casino, and V. Katos, “Encrypted and covert DNS queries for botnets: Challenges and countermeasures,” Computers & Security, 2019. [Online]. Available: https://doi.org/10.1016/j.cose.2019.101614