# Privacy-friendly threat detection using DNS

Rijnders, G.

*Award date:*
2018


Department of Mathematics and Computer Science
Security and Embedded Networked Systems Research Group

# Privacy-Friendly Threat Detection using DNS

*Master Thesis*

Gijs Rijnders

Supervisors:
Luca Allodi, TU/e
Roland van Rijswijk-Deij, SURFnet

Utrecht, August 2018

# Abstract

It is increasingly important for network operators to be able to detect the threats that their users are facing. The DNS (Domain Name System) is a useful tool to do so, as almost every action on the Internet is preceded by one or more DNS queries. Attackers widely leverage DNS for attacks such as malware and phishing. Therefore, many threats can be identified by IoCs (Indicator of Compromise) that reside in DNS queries, such as domain names. However, monitoring DNS queries is also very privacy invasive. The DNS queries that a user performs give a lot of information about the user and their behavior. Naively monitoring all DNS queries on the network is therefore unacceptable from a privacy perspective.

In this thesis, we introduce a threat detection method based on a concept called Bloom filters. DNS queries can be collected and stored in these Bloom filters, which provide strong privacy guarantees. For example, we cannot enumerate the queries that are stored, and we can only look for an exact occurrence in the Bloom filters. Using this threat detection method, we can perform limited detection of IoCs that reside in the DNS.

We specify goals for the privacy-friendly threat detection method, and identify challenges and limitations. Then, we design and develop a prototype of the detection method based on existing software to fulfil the goals. Furthermore, we validate the prototype using three real-world scenarios. We first look at Booter queries, which can be related to DDoS attacks. Furthermore, we look at blacklist hits from a spam filtering service. Finally, we test high-quality IoCs from a security intelligence community in the network. To validate the prototype, we used real-world DNS queries from the operational recursive name servers at SURFnet, the National Research and education network in the Netherlands. The results of the validation shows that we can indeed build a privacy-friendly threat detection method that fulfils the goals we set.

The main contribution of this thesis is a novel, privacy-friendly threat detection method based on Bloom filters. Furthermore, an open-source prototype is developed and validated, that integrates with all major DNS resolver applications. This thesis aims to provide a starting point for network operators to get started with the prototype easily.

# Acknowledgements

This is the master thesis I submitted to complete my master Information Security Technology at Eindhoven University of Technology (TU/e). The project was carried out at SURFnet, providing me the resources to conduct the research.

First of all, I would like to thank my supervisor at TU/e, Luca Allodi, for his support and helpful feedback. Furthermore, I would like to thank Roland van Rijswijk-Deij, my supervisor at SURFnet. Without their help and support, this project would not have been possible. I would also like to thank all my colleagues at SURFnet for the wonderful times, lunches, feedback, and in particular the table football sessions. These were a great distraction during my project.

I also want to thank Matthijs Bomhoff from Tesorion. The initial software came from Tesorion, and I got help and feedback on my work during meetings we organized. Without the help of Matthijs, I would have missed some useful insights.

I got the opportunity to present a poster of my work at TNC18[1], an international networking conference featuring great subjects in the field. The poster I presented received positive attention from the community at TNC18. My colleagues at SURFnet and GÉANT[2], the organization of TNC18 provided me with great help and feedback in this effort. Moreover, GÉANT made my conference trip possible. I want to thank GÉANT and my colleagues at SURFnet for the great support in this effort. The poster can be found in Appendix A, and online at `https://tnc18.geant.org/core/poster/30`.

Finally, I would like to thank my girlfriend for her unconditional support. She was of great help during my studying time.

[1] https://tnc18.geant.org/
[2] https://www.geant.org/

# Contents

# List of Figures

# List of Tables

# Listings

# Chapter 1

# Introduction

**Preamble** This master thesis is the final report of the graduation project for the Information Security Technology master at TU/e (Eindhoven University of Technology). The graduation project was carried out at SURFnet in Utrecht, the Netherlands. SURFnet is an (ISP) Internet Service Provider for academic institutions such as universities and academic hospitals, their constituency.

**Motivation for this research** It is increasingly important for network operators to be able to detect the threats that its users are facing. A way to do this is by using IoCs (Indicators of Compromise), which often reside in the DNS (Domain Name System). DNS is a vital part in Internet communication, as it is responsible for mapping IP-addresses to human-readable domain names. The DNS is widely used as leverage for malware by attackers, and has been argued to be a very effective countermeasure as well. Therefore, it is interesting to use DNS query information for threat detection.

Dodopoulos [16] showed that several methods for threat detection using DNS data are available. However, not all of them respect the privacy of users. Moreover, the IETF (Internet Engineering Task Force) DNS-Privacy working group pointed out that there are severe privacy concerns regarding DNS [14]. The GDPR (General Data Protection Regulation) that recently came into force, asks organizations to review their privacy policies critically. In that trend, SURFnet wants to look for a threat detection method using DNS that is more privacy-friendly. As a solution, we introduce a concept named Bloom filters [9]. Bloom filters can best be described as a statistical way to test for set membership. Using Bloom filters, no query information about users is stored, giving stronger privacy guarantees.

The Dutch company Quarantainenet (now a Tesorion company), that specializes in network security, developed software that uses Bloom filters to detect threats in DNS data using IoCs. SURFnet would like to use this software to identify possible malware infections in its constituency. In collaboration with SURFnet, Quarantainenet has therefore open-sourced its Bloom filter software. SURFnet would like to be able to inform an organization in its constituency upon a possible infection, without pointing fingers at a specific person. The privacy of people in the institutions remains preserved that way. Furthermore, the IETF DNS-Privacy working group is maintaining an Internet draft about service operator recommendations [15]. The Bloom filter based solution will be part of that Internet draft as well.

**Research questions** In this report, we develop and analyze a proof-of-concept of IoC detection using this software. The research question we answer in this report is as follows.

*How can Bloom filters be used to enable DNS-based detection of IoCs in a privacy-friendly way?*

The following objectives are set to answer the research question.

**1** *What DNS-based IoCs and IoC detection methods exist, and can they be performed using Bloom filters?*

We investigate what types of detection methods exist, how they work and whether it would be possible to perform the detection they provide using Bloom filters. Since we are looking at storing DNS log information, the study focuses on detection methods in the passive measurement spectrum.

**2** *What requirements does a privacy-friendly DNS-based IoC detection method have to meet?*

We define what privacy-friendly means, and specify requirements that a privacy-friendly detection method must meet.

**3** *Can we build a Bloom filter-based solution that satisfies those requirements?*

We argue about the properties that Bloom filters provide and the requirements we specified, and whether it is possible to build a Bloom filter based IoC detection solution that meets those requirements.

**4** *How can we measure the performance of the proposed solution?*

We define performance indicators, that specify how we are going to measure the performance of the Bloom filter solution. Performance in this case refers to how well the detection works. To perform the measurements, we define a set of use cases, consisting of input data, a ground truth and hypotheses.

**5** *How does the proposed solution perform?*

We test the Bloom filter solution against a set of validation scenarios. Furthermore, we present the results of those measurements and argue about the performance of the solution.

## 1.1 Contributions

The contributions in this thesis are as follows.

- A theoretical underpinning for the Bloom filter method for privacy-friendly threat detection.
- An extension of the open-source software for Bloom filters to make it suitable for use in an Internet Service Provider (ISP) context, validated using three real-world scenarios and real-world data from the operational recursive name servers at SURFnet.

## 1.2 Outline

The outline of this report is as follows. Chapter 2 contains background information about DNS, IoCs and Bloom filters. Chapter 3 contains a review of related work, describing existing DNS-based IoC detection methods. In Chapter 4, we set the goals of the new IoC detection method. Furthermore, we identify and describe challenges and limitations that are introduced when using Bloom filters. Chapter 5 describes the approach for the prototype development and configuration. Chapter 6 contains the experiment setup, validation scenarios and validation results. Furthermore, the results are discussed and recommendations for future work are presented. Finally, Chapter 7 contains the conclusions.

# Chapter 2

# Background

This chapter contains background on relevant topics, such as the Domain Name System (DNS) and privacy concerns regarding DNS. Furthermore, we discuss Indicators of Compromise (IoC) and sketch an example from the real world. We also go over the threats that users on the Internet are facing. In the context of the GDPR, we then briefly go over the privacy aspects that are required by law, and finally, we explain the concept of Bloom filters.

## 2.1 DNS

Internet communication is based on the Internet Protocol (IP), where hosts communicate with each other using IP-addresses. For example, when a user wants to visit the SURFnet website, it must visit *145.100.190.243*. IP-addresses are not very easy for humans to memorize. Therefore, a mapping exists between IP-addresses and friendly names, hereafter called *domain names*.

The Domain Name System (DNS) is a vital link in Internet communication, as it provides this mapping. For example, it maps *surf.nl* to *145.100.190.243*. When a user connects to a server behind a certain domain name, it connects to a DNS name server first. A DNS name server is a server which provides the user the IP-address that it should connect to. This concept is called *Forward DNS*. DNS also provides a way to map an IP-address to a domain name. This concept is called *Reverse DNS*, and is primarily used for administrative purposes.

### 2.1.1 Domain Names

Domain names are case insensitive ASCII strings, structured in levels. The levels are separated by dots, and represent a hierarchy. An example hierarchy of domain names is shown in Figure 2.1. The root is managed by the the Internet Corporation for Assigned Names and Numbers (ICANN). The names in the first level below the root are called Top Level Domains (TLD). TLDs are classified as gTLD (generic TLD) or ccTLD (country code TLD), where a ccTLD is often used to indicate a particular geographic location. The ICANN delegates responsibility for TLDs to *registries*. The next level in the tree contains second-level domains (SLD). These domain names commonly belong to organizations and individuals. Further levels in the tree are managed by the entity responsible for the second-level domain. Common domain names in these levels are *www* and *mail*, which are often used to point to webservers and mailservers [39, 26]. The full domain name consisting of all levels in the hierarchy is called the FQDN (Fully Qualified Domain Name).

### 2.1.2 Architecture

DNS is a distributed, client-server system. A client application asks a *stub resolver* for a domain name. A stub resolver is a minimalistic resolver application that usually only transforms questions from applications into DNS queries. Stub resolvers are usually installed in the operating system

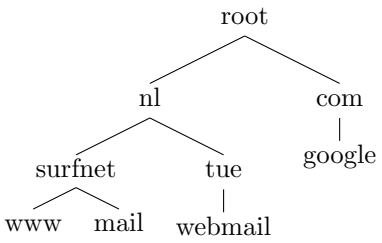



Figure 2.1: An example of a domain name hierarchy in DNS.

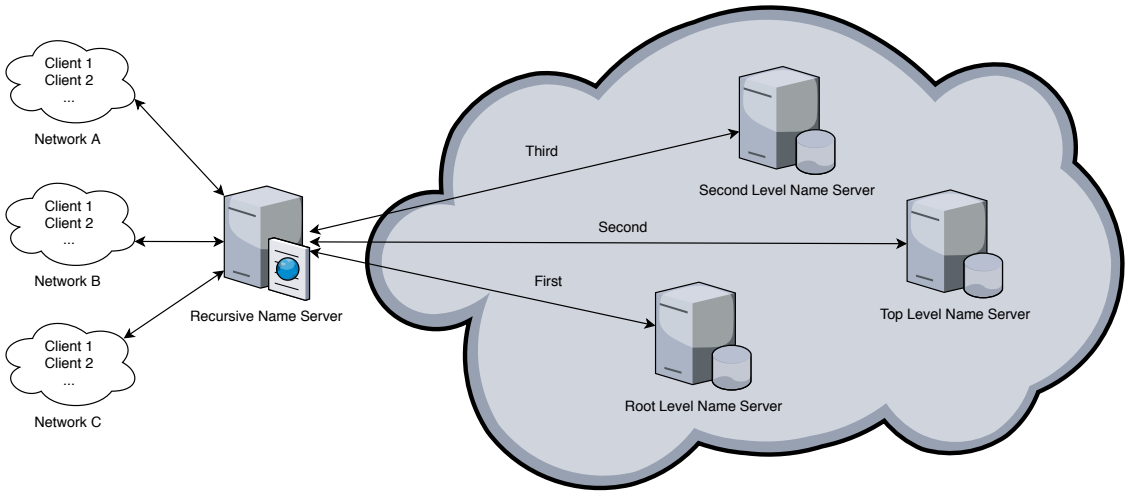



Figure 2.2: An overview of the DNS architecture.

of a client. It processes queries, and replies back with answers. The stub resolver forwards queries to a *recursive name server*.

The recursive name server is responsible for recursing through the domain name hierarchy. It forwards queries to other name servers on the Internet to get the answer. Moreover, it caches answers for queries that were asked before to reduce the number of queries to other name servers.

It is possible that the recursive name server does not know the answer to a query. If so, it asks the *root name server* for the domain first, which is the first level of the domain name hierarchy. This name server will then reply with the IP-address of the *top level name server*, which manages the TLD. The recursive name server then forwards the query to this top-level name server. This name server then replies with the IP-address of the *second level name server*. The recursive name server can then forward the query to this name server, which will be able to answer the query of the client [16]. An overview of the DNS architecture including the order of the discussed steps in a DNS query is depicted in Figure 2.2.

## 2.2 DNS Monitoring

Monitoring DNS activity is useful for troubleshooting network issues, or investigating service (ab)use. ISPs commonly have several recursive name servers for their customers, having thousands of unique users per day. If they are able to monitor that activity, they can better respond to security incidents, or even prevent them from happening.

```
1509929280 192.168.0.101 www.example.net/1/1
```

Figure 2.3: An example of a DNS query log entry.

### 2.2.1 Active and Passive DNS

DNS carries malicious and benign information, and therefore it is useful to monitor DNS traffic. Two types of DNS monitoring exist: *Active DNS* and *Passive DNS* [39, 43]. Active DNS is a technique in which data is collected by generation. DNS queries are generated for possibly malicious domain names, and then executed. The responses that are received for the queries are then analyzed. The OpenINTEL framework by van Rijswijk-Deij et al. [41] performs active DNS measurement. They collect second level domain names from DNS zone information, and perform a set of queries on these domain names. The replies are then stored and analyzed. Active DNS is useful for demographics, measuring the state of DNS globally. Aspects such as domain responsiveness can be measured, and changes on the Internet can be tracked. The OpenINTEL framework [41] creates an overview of the state of DNS globally, measuring the evolution of the Internet.

Passive DNS is an alternative for active DNS. In Passive DNS, queries from users are recorded instead. This technique has advantages, such as that it is easier to implement. Furthermore, the recorded data is representative of the actual use of the monitored infrastructure. Finally, passive DNS does not introduce any additional network load, and attackers cannot detect the use of passive DNS [16].

### 2.2.2 DNS Logs

Passive DNS can be used to log the queries that are performed by users. The DNS resolver application answering the queries can perform this task. A popular and widely used DNS resolver application is `Unbound`[1]. For example, Unbound is used for DNS services at SURFnet. An example log entry in unbound format is depicted in Figure 2.3. It contains a set of fields, separated by spaces and slashes. The first field is the timestamp in seconds since Unix epoch, the second is the source IP-address, and the third the domain name that was queried. Furthermore, the log entry contains the DNS class identifier and DNS record type in numeric form, separated by slashes. The recursive name server can log replies in a similar way. Such log entry contains the reply code and the value of the record, if the reply code did not indicate an error.

### 2.2.3 Vantage Points

For any monitoring system that evaluates user activity, passive DNS is a natural choice. To use passive DNS, a vantage point in is needed in the network. We reconsider the DNS architecture in Figure 2.2, and consider two vantage points for monitoring DNS traffic. The *Name Server Side* vantage point monitors the traffic between the recursive name server and the root, top and second level name servers. This vantage point records queries generated by the resolver that it needs to execute to respond to queries from users. It does not record information about individual users, and hence their privacy is preserved [35]. However, because no user information is recorded, one will not be able to identify infected users.

The *Client Side* vantage point monitors the traffic between the clients and the recursive name server. The largest amount of information is captured on this side. However, this vantage point reveals the identity of users, and hence is not privacy-friendly. Moreover, using this vantage point enables one to identify infected users. The architecture overview including the vantage points is depicted in Figure 2.4.

[1] http://unbound.net/

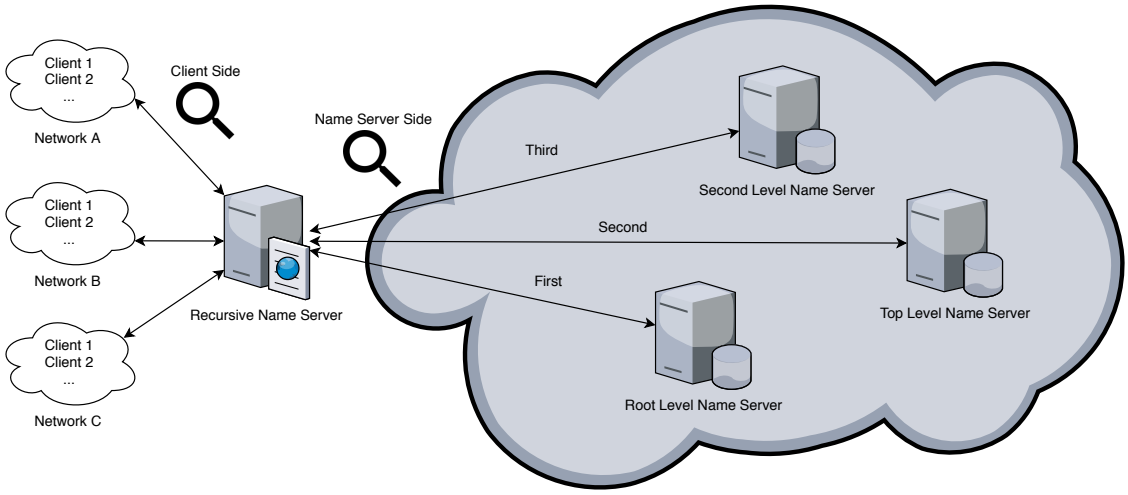



Figure 2.4: An overview of the DNS architecture with two passive DNS vantage points.

## 2.3 Using DNS to Detect Malicious Activity

Dodopoulos [16] comprehensively explained the advantages of DNS monitoring. DNS is only a small fraction of the network traffic, and therefore monitoring DNS traffic is scalable. Capturing all network traffic in large networks often requires additional dedicated infrastructure.

Since the IP-address associated with a DNS query is not known in advance, a connection cannot be initiated before the query is performed. Therefore, malicious activity can be detected before it takes place. However, this is challenging because the query and connection initiation happen in a short time frame.

### 2.3.1 Indicators of Compromise

Indicators of compromise are a vital component in the detection of malware infections. An IoC is an artefact that can be observed on the network. When such an artefact is detected, one can say there is a malware infection with high confidence. Monitoring the network for IoCs is a very common way to look for malware infections. Organizations create, distribute and use IoCs from different sources [30]. SURFcert, the Incident Response team at SURFnet receives IoCs from other Incident Response teams in its community.

For example, let us look at a case of so called *CEO Fraud*. On August 30, 2016 SURFcert reported an incident of CEO Fraud, which is a targeted type of phishing. The fraudsters impersonated the CEO of SURFnet by name and email address, and asked the financial department for immediate assistance in a transfer of money. They registered a domain name that looks similar to *surfnet.nl*, for example, *surfnet-nl.net*. Such domain name could be an artefact to look for on the network, and is therefore an example of an IoC. In this case, it does not refer to a malware infection in particular. However, it would indicate that this specific type of fraud is active within an organization [39].

**Examples IoCs in DNS** Malware, spyware and ransomware infect user machines and commonly connect to remote servers using domain names. The IP-address of the client machine, domain names, DNS record type and timestamp are included in a DNS query. A corresponding DNS reply would contain the TTL (Time To Live) of the queried record, and a response code. The response code is *NXDOMAIN* if the queried domain name does not exist. Malware performs DNS queries for specific values of those attributes, distinguishing the malware from benign activity. Therefore, the attributes can be used as IoC to detect an infection.

| Indicator of Compromise | Description |
|---|---|
| Domain name | The FQDN that was queried by a client, which can be split up into a domain name hierarchy. |
| Record type | The DNS record type that a client queried for. Record type *A* contains an IPv4 address associated to the domain name. A complete list of record types can be found at IANA [24]. |
| TTL | The Time To Live (TTL) of a DNS record is the maximum amount of time that a reply for the DNS record should be cached by the client. |
| Client IP-address | The IP-address that sent a DNS query, which we call the client IP-address. |
| Timestamp | The timestamp at which a DNS query or reply was performed. |
| NXDOMAIN | A DNS reply code that indicates that the domain name that was queried for does not exist. |

Table 2.1: Possible Indicators of Compromise that reside in DNS query and reply log data.

Booter websites that perform DDoS attacks are accessed by domain name. Once a list of domain names related to Booter websites is known, these domain names can be used as IoC to detect accesses of Booter websites. Users who accessed Booter websites could be related to a DDoS attack in the same time frame.

Phishing attacks such as of CEO fraud use domain names that slightly differ from the legitimate domain name. That means that the domain name could be used as an IoC for the detection of phishing attacks. DNS query and reply logs contain more information than just the domain name, as we discussed in Section 2.2. This information could be used to detect malicious activity in the network in different forms. Table 2.1 gives an overview of the kind of data that resides in common DNS activity logs, and could be used in IoCs.

### 2.3.2 Threat Model

Indicators of Compromise enable the detection of threats within the network. SURFnet is a National Research and Education Network, and its constituency primarily consists of students and researchers. SURFnet composed a detailed overview of threats that are relevant for its constituency [10]. Relevant threats include DDoS (Distributed Denial of Service) attacks, malware, spyware and ransomware attacks. Phishing is also a relevant threat. In particular, highly target types of phishing such as identity fraud and CEO fraud. The IoC detection method we propose aims to detect those kind of threats.

**DDoS attacks** DDoS attacks target a machine, user or application within an organizational network. In 2012, SURFnet observed multiple DDoS attacks against educational institutions. Santanna et al. [33] found that these DDoS attacks were purchased by students from so called *Booter* websites, which perform *DDoS-as-a-Service*. In this case, students performed DNS queries for these Booter websites. The attacker has access to the Booter websites and targets a machine or application in an institutional network. The goal of the attacker is to disrupt services such as e-learning, for example, to prevent exams from taking place.

**Malware, spyware and ransomware** Malware, spyware and ransomware attacks target individual machines in an organizational network. They are used to extract information about users or disrupt services. Ransomware disrupts services until users make a payment to the attacker. Once a machine is infected with malware or spyware, information is extracted and transferred to a so called *dropzone*. This is a remote server, which is often located using DNS queries, where the attacker can collect the information. The attacker can also gain remote access to the infected machine. In that case, the attacker has full control over the machine's behavior.

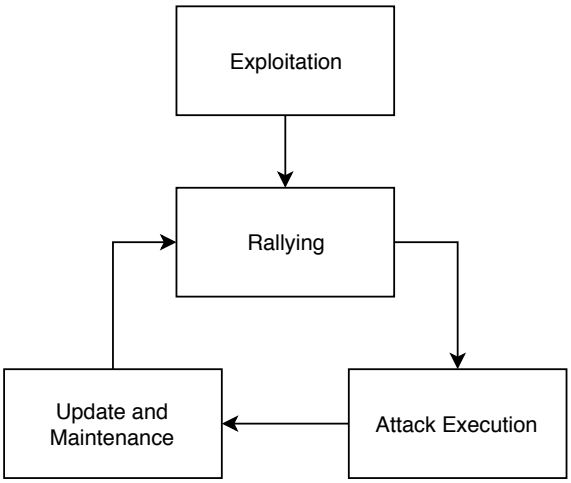



Figure 2.5: A common life cycle for malware with botnet participation.

**Phishing** Phishing is an attack that attempts to acquire sensitive information about users. The attacker impersonates an organization or person in order to gain trust, and asks users to disclose sensitive information at a fake website. This website is referred to using DNS queries. Identity fraud and CEO fraud are examples of highly targeted phishing attacks. The attackers know personal things about their target. Furthermore, they take time to find about the behavior of the person they impersonate. The number of cyber-criminal activities increased in 2017, with ransomware as most significant threat [10].

### 2.3.3 Example Application: Botnets

In this section, we will provide an example of how the DNS plays a role in malicious activity. Malware infections are a threat to users of organizational networks. Malware can infect machines on a large scale, forming a network. Such network is called a *botnet*. Botnets can grow very large in the number of involved machines, and inflict serious damage. In 2016, the Mirai botnet [4], composed of approximately 600.000 infected machines, executed one of the largest DDoS attacks in history.

The life cycle of a malware infection that results in botnet participation is commonly composed of four phases [2]. The first phase is *exploitation*. In this phase, the user machine is exposed to the malware, which exploits a vulnerability to nest itself in the machine. The second phase is *rallying*. A botnet is controlled by remote servers on the Internet, called *Command and Control* servers. In the rallying phase, the malware calls home to these remote servers to join the botnet. The botnet maintainers send commands to the malware in the *attack execution* phase. The Mirai botnet used this phase to perform DDoS attacks. The last phase is *update and maintenance*, in which for example, the malware is reconfigured. After reconfiguration, the malware returns to the rallying phase. An overview of this botnet life cycle is depicted in Figure 2.5.

When malware calls home in the rallying phase, they connect to their command and control servers. To keep botnets maintainable at large scale, the command and control servers are typically exposed using domain names. When command and control servers are seized, the botnet maintainers only have to change the IP-addresses associated to the domain names. Therefore, DNS logs are very useful in the detection of botnet activity.

**Domain Generation Algorithms** If botnet maintainers used static, predefined domain names for their command and control servers, botnet rallying could be easily detected and blocked using static domain blacklists. Hence, some malware uses so called *Domain Generation Algorithms* (DGA) to dynamically generate domain names. A large set of (pseudo)random domain names are

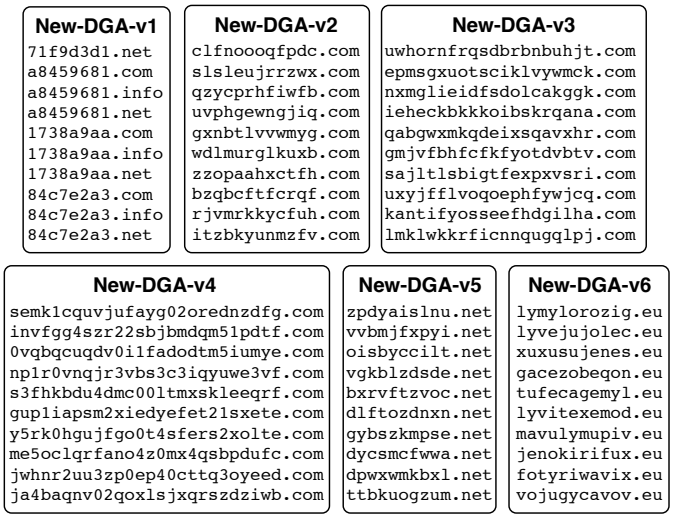



Figure 2.6: A set of example domain names generated by different DGAs (taken from [5]).

periodically generated, and a small subset of those domain names are actually used to connect to the command and control servers. Since the domain names appear random and change periodically, the detection of such is hard [5]. A set of examples of DGA generated domain names is depicted in Figure 2.6.

Users in a network usually do not query random domain names on purpose, and therefore the occurrence of random domain names possibly indicates the presence of botnet activity. The set of domain names that is generated by a DGA also contains domain names that do not exist. The infected machine queries these domain names as well, resulting in numerous NXDOMAIN responses. Therefore, looking at excessive amounts of NXDOMAIN responses is a way to detect possible botnet activity. As botnets leave traces, IoCs are useful tools in detecting botnet activity in organizational networks.

### 2.3.4 Privacy Concerns

Almost every website visit, email sending or mobile phone app operation requires multiple DNS queries. These queries reveal information about users and what websites they visit, but also which services they use at those websites. Therefore, one can learn a lot about the identity of a user by looking at its DNS queries. DNS queries are transmitted over the network in plain text by default. Therefore, anybody that can listen to the network traffic between the client and name server can find out what users are doing.

An entire working group in the IETF, called *dprive*, is dedicated to addressing issues regarding DNS privacy[2]. The working group aims to deprive attackers from the information they can obtain from monitoring DNS user activity. In 2014 they stated that *Pervasive Monitoring Is an Attack*, targeting privacy [32]. The pervasiveness of protocols should be mitigated in their design. The dprive working group proposed DNS-over-(D)TLS as solution to eavesdropping on DNS queries, encrypting the communication between client and name server [45, 31]. Eavesdropping on the plain text DNS traffic would no longer be possible in this case. However, ISPs that log DNS queries that are performed at their name servers, can still see this information.

ISPs may use information they store in ways such as providing parental controls or sharing with third parties [14]. Herrmann et al. [22] showed that given this information it is possible to profile users based on their DNS activity, learning about their identities. Users may also have visited websites they want to keep private. Therefore, performing analytics on DNS query and reply log information significantly infringes the privacy of users.

[2]https://datatracker.ietf.org/wg/dprive

In case of abuse, an ISP would like to be able to inspect the network traffic of a specific user. However, they may only do so once they comply with the privacy requirements that are imposed by law. Therefore, they must obtain permission to do so from their Privacy Officer. Because the privacy guarantees that can be given in data inspection are currently insufficient, it may be hard for an organization to obtain that permission. More details about the privacy requirements that are imposed by law are described in Section 2.4.

## 2.4 Privacy Requirements

The GDPR came into force on May 25, 2018. In an era where privacy is important, we are encouraged to think about privacy more thoroughly. Therefore, we consider *Privacy by Design* in the development of our IoC detection method. This concept calls for considering privacy throughout the complete engineering process of a technology, rather than applying patches afterwards.

Hafiz [21] defined a set of privacy by design patterns that can be used in developing a privacy-friendly system. In the design of our Bloom filter based IoC detection solution, we consider the *Anonymity Set* design pattern. This design pattern states that information about multiple users is mixed in a single set of data. Queries from multiple users can be mixed in a single Bloom filter, providing anonymity.

The *Hidden Metadata* design pattern is used as well. This design pattern states that any metadata that reveals information about the content of sensitive data should be hidden. Bloom filters do not store the content of queries, only their presence. Moreover, the parameters and purpose of the Bloom filter are private to the maintainers. Therefore, the Bloom filter appears random without the metadata. A more detailed study of the privacy requirements is out of the scope of this report.

## 2.5 Bloom filters

When testing for IoCs, large amounts of DNS log data are collected and stored. This is very privacy infringing, and it scales very poorly in terms of required space and execution time of search operations. In this research, we study if Bloom filters can be used as a viable way to solve these problems.

Bloom [9] introduced a concept called the Bloom filter in 1970. A Bloom filter consists of an array of bits, initially set to zero. Elements can be added to the Bloom filter by running them through a number of hash functions. The output of those hash functions is used to determine a set of indices within the array. This conversion from hash values to array indices is depicted in Figure 2.7.

Once array indices are selected, the value of each selected bit is checked. If the bit in the array is 0, it is set to 1. If the bit is already set to 1, no operation is performed. An example of a Bloom filter with domain names added as elements is depicted in Figure 2.8. The domain name *domain1.nl* is added first. The indices[3] 1, 3, and 5 are selected, and set to 1. The domain name *domain2.com* is added next. The selected indices are 1, 2 and 6. The bit at index 1 was already set to 1, so no operation is performed. The bits at indices 2 and 6 are still zero, so they are set to 1. Every domain name that is added points to a set of indices within the Bloom filter. Adding an item therefore never fails. Instead, the false positive rate increases. The more bits are set to 1, the likelier it is that a query will yield a positive result.

One can ask the Bloom filter whether a certain domain name was added to the Bloom filter. The domain name will be run through the series of hash functions again, to determine the indices that represent the domain name in the filter. If all bits at the given indices are set to 1, the answer is yes. If at least one bit is set to 0, the answer is no. If the answer is no, the domain name was not added to the Bloom filter, and this answer is certain. That means that false negatives cannot occur. However, if the answer is yes, the correctness of the answer is probabilistic. The answer is

[3] Indices in the bit array start from 0.

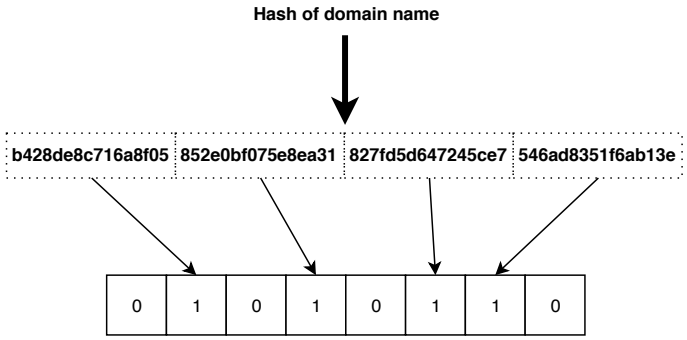



Figure 2.7: A depiction how the hash function output is used to determine array indices.

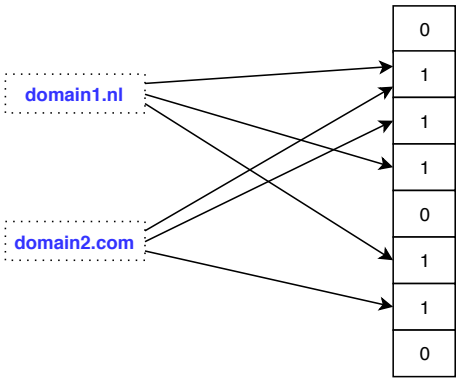



Figure 2.8: A depiction of the Bloom filter bit array with two domain names added to it.

correct with high probability, depending on the Bloom filter parameters. The Bloom filter may incorrectly state that a given domain name was added to the Bloom filter. These errors are called false positives.

### 2.5.1 False Positives

We reconsider the example Bloom filter in Figure 2.8. One asks the Bloom filter whether the domain name *false-positive.net* was added to the Bloom filter. The domain name is run through the hash functions, and the indices 3, 5 and 6 are selected. The answer to the question is yes, because all selected bits are set to 1. However, the domain name *false-positive.net* was never added to the Bloom filter. This case describes when a false positive occurs in a Bloom filter, and is depicted in Figure 2.9.

True negatives are also depicted in Figure 2.9. One asks the Bloom filter whether the domain name *true-negative.name* was added to the Bloom filter. The domain name is run through the hash functions, and the indices 0, 1 and 2 are selected. The bits at indices 1 and 2 are set to 1, but the bit at index 0 is set to 0. Therefore, the answer that the Bloom filter gives is no. The value of a bit in the Bloom filter can change from 0 to 1, but not from 1 to 0. Therefore, a false negative can never occur.

A Bloom filter is a set that does not store the original information as it was added. When asking the Bloom filter whether a certain element is a member of the set, the behavior is probabilistic if the answer is yes. Therefore, Bloom filters are best described as a statistical way to test for set membership.

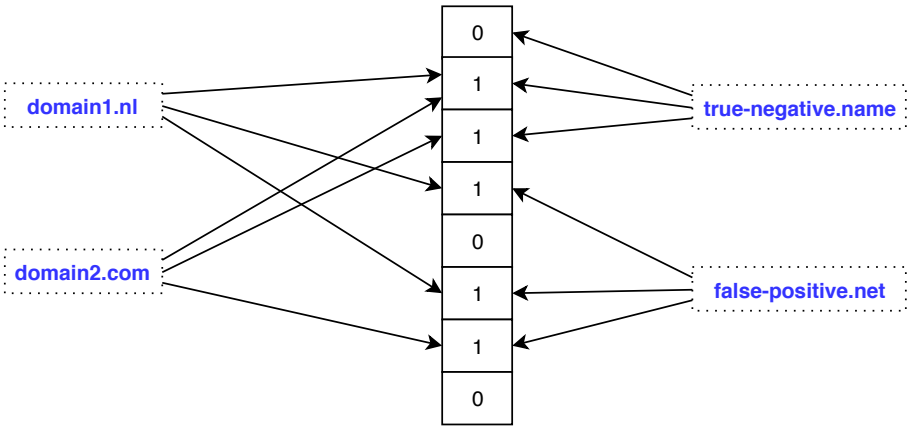



Figure 2.9: A depiction of the Bloom filter bit array, indicating false positives.

### 2.5.2 Parameters

We discussed the meaning of false positives in Bloom filters, and stated that the false positive rate depends on the parameters of the Bloom filter. A Bloom filter has two parameters: the size in bits $m$, and the number of hash functions $k$ that are evaluated for each input element. The size of the Bloom filter allows one to trade space for false positive rate. The smaller a Bloom filter is, the quicker all bits are set to 1, and the false positive rate goes up. Furthermore, increasing the number of hash functions increases the number of bits that are set to 1 when an element is added to the Bloom filter. When a higher number of bits must have the value 1 in order to be evaluated to a positive answer, the false positive rate goes down.

An analysis on the false positive rate of Bloom filters showed that we can calculate the false positive rate $p$, given $m$, $k$ and the number of elements $n$ we want to add to the Bloom filter [12]. We can approximate the false positive rate with equation 2.1. The analysis showed that this formula is incorrect for Bloom filters with small $m$, predicting too small values for the false positive rate. However, we want to store a large number of elements $n$, and therefore we have Bloom filters with large $m$. In this case, equation 2.1 predicts the false positive rate correctly.

$$p \approx \left(1 - e^{\frac{-kn}{m}}\right)^k \tag{2.1}$$

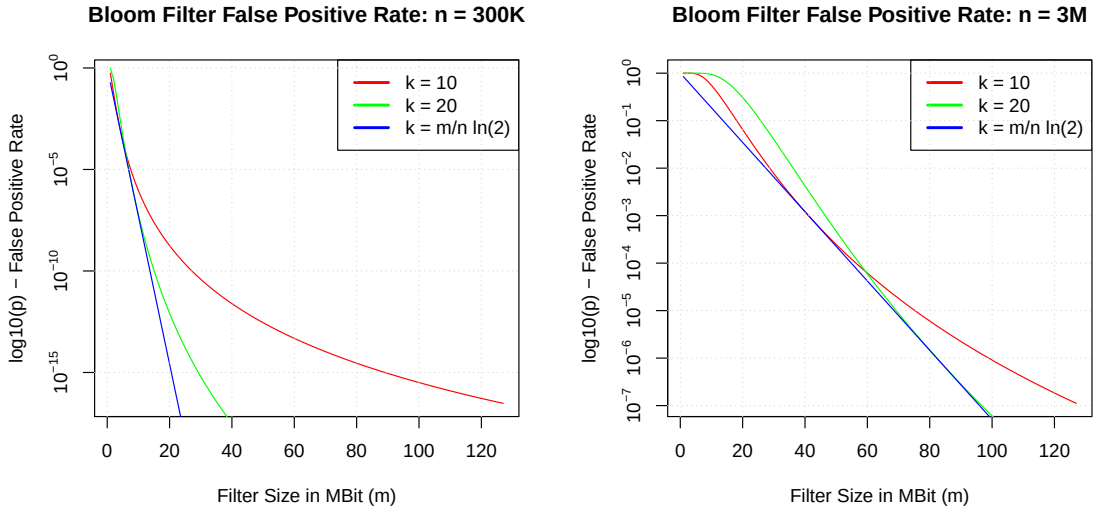
The calculation of $p$ depends on the parameters $n$, $m$ and $k$. The parameter $n$ is the maximum number of unique domain names that are added to a Bloom filter. We will set this parameter to a fixed value in our calculations. The parameters $k$ and $m$ remain as degrees of freedom. We want to set a false positive rate $p$ that we consider acceptable, and tune $k$ and $m$ accordingly. The value of $k$ that minimizes the false positive rate of a Bloom filter is given by equation 2.2 [12].

$$k = \frac{m}{n} \ln 2 \tag{2.2}$$

If we substitute Equation 2.2 in Equation 2.1, we get the false positive rate $p$ with an optimal number of hash functions. Given $k$ and an acceptable false positive rate $p$, we can calculate $m$ using Equation 2.3.

$$m = -\frac{n \ln(p)}{\ln(2)^2} \tag{2.3}$$

We introduce an example to show how the Bloom filter parameters influence the false positive probability. We let $m$ range from 1 to 128 Megabits, and plot the false positive rate $p$ as a function of $m$. The number of elements $n$ is important when calculating the size of a Bloom filter. Figure 2.10a shows the false positive rate of a Bloom filter with the parameters we have set, given that $n = 3 \cdot 10^5$. The blue line shows the false positive rate using optimal $k$. The red and green




(a) False positive rate for $n = 3 \cdot 10^5$.

(b) False positive rate for $n = 3 \cdot 10^6$.

Figure 2.10: A series of plots showing the influence of $k$ and $m$ on the false positive rate, given $n$. The false positive rate is displayed on a logarithmic scale.

lines show how deviating $k$ influences the false positive rate. The graph in Figure 2.10b shows the same calculation with $n = 3 \cdot 10^6$.

Let's consider a false positive rate of $10^{-3}$ acceptable. Given this false positive rate, we can easily see that $m \approx 40$ Megabits is a good size for a Bloom filter that stores at most $n = 3000000$ queries, using $k = 10$ hash functions. 40 Megabits equals to 5 Megabytes of required memory, which is very space efficient for such a large number of queries.

The Bloom filter theory suggests that every bit index is selected by a different hash function, or by using a different salt. However, this is inefficient because only a small number of bits from the hash are used to select the indices. Gerbet et al. [19] showed that any cryptographic hash function may be called once, and the output may be sliced into chunks to select multiple indices. We also took this approach in Figure 2.7. This is the case because cryptographic hash functions have the property that their outputs have a randomly uniform distribution. The minimum number of required bits of entropy to add a query to a Bloom filter can be calculated by $k \cdot \lceil log_2(m) \rceil$ [19].

According to the analysis in [12], the false positive rate of a Bloom filter steadily increases as more queries are added. When designing Bloom filters, an upper bound for $n$ must be determined. However, the number of queries that is actually added may deviate from $n$. When asking a Bloom filter whether a query was previously added, it is important to know what the probability is that the answer is correct. The actual expected false positive probability $p_s$ can be calculated with Equation 2.4, given the number of bits set $s$. The false positive probability is optimal when $\frac{s}{m} = \frac{1}{2}$.

$$p_s = \left(\frac{s}{m}\right)^k \tag{2.4}$$

We use the example Bloom filter with $k = 10$ and $m = 40$ Megabits, to illustrate how the false positive rate increases as $\frac{s}{m}$ increases. Figure 2.11 depicts the false positive rate as a function of $\frac{s}{m}$. As the number of bits set approaches $\frac{1}{2}$, we see that the false positive rate approaches $10^{-3}$ as we discussed earlier. However, the false positive rate steadily increases when more queries are added.

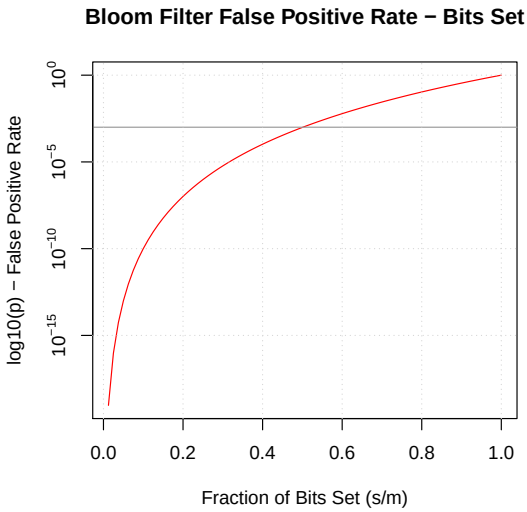



Figure 2.11: A depiction of the false positive rate as a function of the number of bits set. In this plot, $k = 10$, $m = 40$ Megabits, and the false positive probability is displayed on a logarithmic scale. The horizontal line represents the acceptable false positive rate at $10^{-3}$.

### 2.5.3 Properties

We discussed the theory of Bloom filters in Section 2.5. Bloom filters introduce constraints, but also benefits. We refer to a set of constraints and benefits as properties. The following properties follow from the Bloom filter theory [40].

1. The original input data is not stored. Therefore, what queries were performed cannot be trivially recovered from the Bloom filter.
2. Queries can only be looked up using an exact value. Therefore, enumeration of stored information is not possible.
3. Bloom filters only require a small amount of memory to store a large number of queries, and hence scale well.
4. Bloom filters can be exported and stored elsewhere for a long time. Therefore, lookups can be made in history.
5. Queries from different users can be mixed in a single Bloom filter. When mixing queries, it becomes practically impossible to link a query to a specific user.
6. The way Bloom filters are stored in memory allows their states to be easily combined. If two Bloom filters have the same size in bits, and the same set of hash functions, the union of the two can be built by taking the bitwise OR of the bits. This saves storage space, since the contents of two Bloom filters are combined into one.
7. If the hash function(s) of two Bloom filters are different, for example, by an added salt, then their false positive probabilities are independent. Therefore, the false positive probabilities may then be multiplied.

### 2.5.4 Challenges

Bloom filters can be used for IoC detection, but they introduce constraints, as we discussed in Section 2.5.3. We emphasize that overcoming these challenges and working within the constraints is

a significant part of this thesis. For example, the presence of false positives and non-enumerability limit the capabilities of IoC detection using Bloom filters. The challenges and limitations of Bloom filters are extensively discussed in Chapter 4.

### 2.5.5 Attacks

Cryptographic hash functions are collision resistant, and hence we assume that finding collisions is impractically hard. However, interesting attacks remain. Gerbet et al. [19] analyzed the security of Bloom filters under an attacker model where the attacker knows what parameters are used for the Bloom filters, and what data is inserted. In the area of open-source software, this attacker model is reasonable. Moreover, the attacker can insert any chosen element, since he can send any DNS query he likes to a recursive name server.

Under the attacker model discussed above, two attacks were described in [19]. The *pollution* attack aims to increase the number of bits in a Bloom filter that are set to 1. The attacker does this by generating a set of polluting DNS queries that have a high rate of dispersion. That means that the queries all hash to different indices in the Bloom filter. In the worst case, the attacker wants to set all bits in the Bloom filter to 1. This attack is called the *saturation* attack. The goal of these attacks is to increase the false positive rate of the Bloom filter beyond expectation. The attacker wants to confuse network operators with a large number of false positives, to achieve that network operators cannot determine what is correct and what not.

Finding the same vector of bits for a different DNS query has the same complexity as finding a second pre-image. However, it is significantly easier to find polluting items. If one adds $n$ polluting queries to an empty Bloom filter, the probability $p_n$ of finding the $n$-th polluting query can be calculated using Equation 2.5. Let's take the example Bloom filter from Section 2.5.2, with $m = 40$ Megabits, $n = 3000000$ and $k = 10$. Filling in the probability formula, we find that the probability in our example equals $1.6 \cdot 10^{-64}$, which is achievable.

$$p_n = \frac{\binom{m-(n-1)k}{k}}{m^k} \tag{2.5}$$

The pollution and saturation attacks are achievable, but simple to counter. The maintainers of the Bloom filters can use a private salt that is applied to all domain names. When a private salt is used, the attacker does not know what hash will be output, and hence will not be able to determine perfect polluting queries. The attacker will still be able to perform a pollution or saturation attack, but only by rapidly inserting random queries. This will significantly increase the number of queries arriving at the recursive name servers each second, and hence is easily detected.

# Chapter 3

# Related Work

In this chapter we study related work in the context of privacy preservation. We study existing literature about IoC detection techniques and their privacy aspects. Furthermore, we explain what our proposed IoC detection method contributes. We also look at ways for preserving privacy in data analysis, and analyze the privacy guarantees that Bloom filters provide. Finally, we study existing literature about Bloom filters and privacy preservation.

## 3.1 IoC Detection

In 2015, Dodopoulos [16] comprehensively compared existing DNS-based IoC detection methods [5, 6, 8, 11, 27]. These methods were categorized in approaches, and the goal of these approaches is detecting botnet activity and malicious domain names. The approaches are described below.

1. *Group activity* looks for groups of a fixed number of hosts that have the same intermittent activities. For example, these hosts may query the same domain name for a period of time. If that domain name is known for malicious activity, a group of infected hosts can be identified.

2. The *co-occurrence* approach looks for unknown domain names that were queried by a host that also queried known malicious domain names. If those unknown domain names occur, they are automatically considered malicious. Threat information coming from this approach can be used to create new IoCs. For example, detected domain names can be added to a blacklist.

3. *Data Mining and Reputation systems* store DNS information on a large scale and look at features of DNS to determine which domain names are malicious. Features of DNS include properties of domain names, such as its length, its TLD and the TTL value of its associated records. The reputation systems also use *statistical features*. For example, Notos [6] examines the number of IP-addresses that have been assigned to a domain name in the past. Furthermore, it looks at the geographical location of those IP-addresses, including the IP-addresses that have been assigned to any subdomains. The primary goal of IoC detection methods having this approach is detecting unknown malicious domain names.

4. The *sequential correlation* approach looks for correlations between domain names that are queried by a client in sequence. If a particular domain name is queried before or after another, a pattern arises. This approach assumes that infected clients query malicious domain names in sequence. For example, an infected client may query a set of domain names that is known for a specific malware sample.

5. The *NXDomains* approach bases detection upon DNS replies for queries to Non-eXisting Domains. Clients that query a large number of non-existing domains in a specific time window may arouse suspicion.

IoC detection methods using these approaches are not designed to deal with the constraints Bloom filters introduce. The applicability of the discussed approaches will be discussed in Section 4.2.1.

Dodopoulos' comparison also included information about whether the privacy of users was violated. It defined privacy in terms of the vantage points where data was collected. The privacy of users was violated if the client side vantage point as described in Section 2.2.3 was used to collect data.

IoC detection methods that use the name server side vantage point preserve the privacy of users. However, no information about users is collected, as we discussed in Section 2.2.3. Therefore, infected users cannot be identified. These IoC detection methods use information from the upper DNS hierarchy to create new IoCs. DNS traffic is passively monitored, and unknown malicious domain names are identified using statistical tools and machine learning models. The new IoCs can then be used in detection with the client side vantage point, or to extend a blacklist. Client-side detection violates the privacy of users as we discussed, but blacklists are directly used to block certain queries. No client-side monitoring is required, and the privacy of users is preserved. However, using a machine-built blacklist to directly block certain domain names introduces false positives, which impacts the freedom of users on the Internet.

It is important for network operators to be able to identify threats in their networks. However, this currently cannot be done in a way that does not infringe the privacy of users. The Bloom filter based IoC detection solution we propose bridges the gap between security and privacy. In contrast to the existing work, Bloom filters provide stronger privacy guarantees. It enables network operators to identify threats in their networks without violating the privacy of their users.

## 3.2 Preserving Privacy

Gertz et al. [20] and Vaghashia et al. [23] surveyed privacy preservation techniques for data analysis. In this section, we study those techniques and establish their positive and negative properties. Table 3.1 gives a comprehensive overview of the techniques discussed in this section.

### 3.2.1 Anonymization

Anonymization is a process of data sanitization, where personally identifiable information is masked or removed from the data. Certain fields in a dataset can be *suppressed*, meaning that the information is removed or changed to something meaningless. Furthermore, certain fields can also be *generalized*. In this case, the information is changed to a grouped variant that implicitly contains the original information. For example, The IP-address of a host in the network can be generalized to a subnet. The amount of anonymity $k$ that is provided in that case depends on the number of unique IP-addresses that are in use in the subnet. Anonymization makes sure that personally identifiable information is removed from the data. However, one should note that when information is suppressed or generalized, part of the data is lost [23]. Another form of anonymization is pseudonymization, where personally identifiable information in the data is replaced by artificial identifiers called *pseudonyms*.

When data is anonymized, a property that the data may possess is *k-anonymity*. A field in a dataset satisfies k-anonymity if the information of a random entity in the data can be mapped to at least $k$ different entities. In the previously mentioned subnet example, the $k$ different entities are IP-addresses. The k-anonymity property provides a privacy guarantee on the data [38].

However, k-anonymity does not provide for randomization of the data. An attacker can perform re-identification by linking, and still make observations about the data that may result in the identification of an entity. Aggarwal [1] showed that k-anonymity does not work well for datasets with a high number of dimensions, resulting in an unacceptable level of data loss. Furthermore, Angiuli et al. [3] showed that k-anonymity may skew the data in case, influencing the results of analyses.

| Technique | Advantages | Disadvantages |
|---|---|---|
| Anonymization and Pseudonymization | Personally identifiable information can be hidden. | Part of the data can get lost. Furthermore, re-identification by linking is possible. |
| Encryption | The data is protected and remains complete. | Cryptographic keys are required, which scales badly as the number of involved parties grows. Furthermore, re-identification by linking is possible. |
| Hashing | Not trivially possible to recover original information. | Brute-force and Rainbow Tables can still reveal the original information. |
| Bloom filters | Enumeration of stored data is not possible, and recovering the original data from the Bloom filter is not possible. | Stored data can only be queried with an exact value. Storing data with an accurate timestamp is not trivially possible. |

Table 3.1: An overview of existing privacy preservation techniques.

### 3.2.2 Cryptography

Cryptography is a practice of hiding information from unauthorized people. One can hide information in a dataset using a cryptographic algorithm, taking the data as *plaintext* and a cryptographic key as input, and producing a *ciphertext* as output. Only entities that possess this key can unveil the hidden information. This process is called *encryption*. The data can be encrypted before it is stored. There are two types of encryption. If *deterministic* encryption is used, a plaintext will always produce the same ciphertext. In this case, one can still match a plaintext to a ciphertext, and make observations about the data. If *non-deterministic* encryption is used, the data appears random until decrypted. As long as the data remains stored, encryption works well as privacy preservation technique. No data is lost, and the data is well protected. However, every party that wants to access the data must have the cryptographic keys. Therefore, encryption scales poorly when the number of involved parties grows. Moreover, once a key has been given away, the maintainer does not have control over it anymore. The data must also be decrypted before it can be used, and on decryption, all anonymity is removed.

The data can be encrypted as a whole, as previously described. However, the data can also be encrypted by parts. For example, all personally identifiable information can encrypted separately. In that case, researchers could make observations about the data without knowing about which individuals it is. If required, the encrypted data can still be decrypted later. When encrypting data by parts as discussed above, re-identification by linking is also possible.

### 3.2.3 Hashing

Hashing is a process of mapping values from an input space of arbitrary size to an output space of fixed size. Hash functions have multiple applications, one possible use of hash functions is data hiding. A hash function takes input data of arbitrary size and outputs a value of fixed size. Given a hash function and input data, one can take the hash, and store that instead of the original data. We assume that the hash function used is preimage resistant, which is a common property for hash functions. It means that it is not trivially possible to find the input that matches a hash.

However, it is still possible to find the preimage, given the hash and the used hash function. This could be done by brute-forcing all possible inputs and matching the output value to the given hash. Moreover, depending on the application, the character space from which the input data is taken is limited, which decreases the difficulty of brute-forcing. Kumar et al. [25] showed that a more efficient way of finding the preimage exists, using Rainbow Tables. Wander et al. [42] showed that GPUs are also very efficient when computing the preimages of hashes.

## 3.3 Bloom Filter Applications

Bloom filters were invented in 1970. Ever since, they have been used in a countless number of applications. Geravand et al. [18] showed with a survey that Bloom filters have been used in various areas of network security. A Bloom filter based solution to detect spoofed DNS requests was introduced in 2008 [37]. Furthermore, Bloom filter based techniques to counter DDoS attacks exist. The goal of these techniques was posing a feasible solution for a problem that requires a high performance approach at large scale. For example, analyzing network traffic at a high capacity link. Privacy preservation was not an explicit goal for those techniques.

Bloom filters have also been used for privacy preservation. Chen et al. [34] introduced an anonymous routing technique based on Bloom filters. In this technique, Bloom filters were used to hide routing information such as next hops, sources and destinations. Furthermore, Zhu and Mutka [46] introduced a message notification protocol for Instant Messaging (IM) that stores message notifications in Bloom filters. They state that one so called proxy server should proxy the messages from an IM server to the users. However, the proxy server should not know what the messages and notifications contain. Furthermore, the Bloom filters were also used to reduce overhead on the messaging protocol.

It is interesting to see that Bloom filters have also been applied in signature detection. Yan and Cho [44] showed that Bloom filters can be used for e-mail spam filtering. The authors reported that Bloom filters could significantly reduce the required size of a signature database, and improve the time required for lookups.

The privacy preservation concepts that were introduced in earlier work [34, 46] solve privacy issues in an operational context, rather than in monitoring and detection. Moreover, Bloom filters were also used to improve efficiency in this case. The signature detection solution is relevant, because a signature could be considered an IoC. However, the signature detection solution was solely introduced for improving efficiency. Bloom filters were used both for privacy preservation and detection, but not jointly. In this thesis, we discuss a Bloom filter based concept for monitoring and detection that focuses primarily on preserving privacy.

# Chapter 4

# Goals & Challenges

We previously explained how Bloom filters can be used for IoC detection, and why it is useful and desired. A prototype of a Bloom filter based IoC detection system is designed and validated in this thesis. To do so, we specify our research goals in this chapter.

The use of Bloom filters introduces benefits, as we discussed in Section 2.5. However, Bloom filters also introduce constraints which significantly influence their applicability for IoC detection. Therefore, we also look at the applicability of existing IoC detection methods that Dodopoulos analyzed in [16]. Furthermore, we study the challenges and limitations that arise from using Bloom filters for IoC detection, given the research goals.

## 4.1 Goals

**Main Goal** The main goal is as follows. *We want to use IoC detection to gain insight on threats on the network at a strategical and tactical level, while preserving the privacy of individual users.* That means that measures could be taken on an organizational level, based on detection results. For example, if we find that a specific type of malware targets the network, we could decide that this malware must be actively mitigated. Furthermore, we want to make an overview of the state of the network. Which threats are active, and how widespread are they?

**Subgoals** To achieve the main goal, we break up the main goal into the following subgoals. The subgoals are labelled and will be referred to further in this thesis.

**G1** *The privacy of individual users in the network must be preserved.*

DNS information about users is very sensitive, as we discussed in Section 2.3.4. Therefore, we only want to detect the activity of threats in subnets in the network. We do not want to be able to identify individual users.

We must obtain permission from our legal counsel to actively monitor network traffic. Being able to guarantee the privacy of users in monitoring is a firm argument in getting that permission.

**G2** *We want to be able to detect the presence of a threat in the network using existing IoCs, and estimate the time of occurrence.*

IoCs provide information about how to detect the presence of a threat in the network. We obtain IoCs from sources such as communities and blacklists, but the current infrastructure at SURFnet does not currently use that information as a naive implementation may lead to a violation of the privacy of users. Using Bloom filters, we would be able to perform detection without violating user privacy.

We want to able to detect the presence of a threat in the network, and estimate the time of occurrence, accurate to an hour or a day. The frequency of occurrence is not required. Furthermore, gathering security intelligence from the network to create new IoCs is not a research goal.

**G3** *We want to be able to identify the network segment in which a threat is detected.*

Large networks may be divided into network segments, which are defined using subnets. These subnets contain groups of users, and hence they can be associated with an organization or organizational unit. If we can identify the organization in which threats are present, we can notify the local incident responders. The local incident responders can then decide whether the threat is serious enough to warrant temporarily monitoring all DNS traffic to find and mitigate the threat (thus temporarily sacrificing user privacy). Furthermore, knowledge about which threats target organizations helps forming strategies for prevention and mitigation.

**G4** *We want to be able to aggregate information in the detection system over time.*

Part of the main goal is creating an overview of threats in the network. Such an overview is useful, because it allows us to see how specific threats occur over time, what threats are most prominent, and for how long threats remain active. The threat overview becomes increasingly useful as more data is collected. Therefore, this overview should combine threat information from long periods of time.

**G5** *Operationalize data collection and storage in compliance with current regulations.*

The threat detection system must be set against the GDPR, and therefore, we want to minimize the data we collect and store. We only collect the information necessary for the purposes of the main goal. Information that is not stored cannot be retrieved. When detailed information about the activity of the network is requested by a third party, such as law enforcement, we cannot, and do not have to provide anything. Furthermore, data minimization reduces the risk of unauthorized access and security threats in the detection system, because the stored data is not useful for an attacker.

For an overview of the threats within the network to be meaningful, information must be collected and stored for a long period of time, preferably at least a year.

## 4.2 Challenges & Limitations

In this section, we will discuss the challenges and limitations that arise from the constraints that Bloom filters introduce. For convenience, a comprehensive overview of the limitations is given in Table 4.1.

### 4.2.1 Applicability of IoC Detection Methods

The approaches we discussed in Section 3.1 capture their data either from the client side or name server side vantage point. IoC detection methods that use the name server side vantage point produce new IoCs. They use passive DNS data to build statistical or machine learning models, which are used to identify unknown malicious domain names.

Bloom filters are used for data storage. The passively captured DNS queries are stored, and one can ask the Bloom filter whether a certain query exists. It is not possible to enumerate the contents of the Bloom filter. Therefore, building statistical or machine learning models on the stored data is not possible. Existing IoC detection methods that use this approach cannot be performed using Bloom filters.

| Limitation | Description | Mitigation (if possible) |
|---|---|---|
| Applicability of IoC Detection Methods | The approaches of existing IoC detection methods described in Section 3.1 are not suitable for use in Bloom filters. | Existing IoC detection methods focus on finding new IoCs. The Bloom filter based system focuses on testing existing IoCs. |
| DGA Detection | Only known random labels in domain names can be detected. | The Bloom filter based system focuses on detecting known labels. |
| False Positives & Decision Support | False positives apply in detection using Bloom filters. | The probability and impact of a false positive must be considered and dealt with on a case specific basis. |
| Resilience to Adversarial Actions | An attacker is able to pollute the information stored in the Bloom filters. | The number of queries and unique domain names that are stored should be continuously monitored to detect an attack. |
| Configuring Bloom Filters | Deciding how to configure Bloom filters and what to store in them depends on the goals. | An appropriate configuration is chosen based on the goals. |
| Scalability | Choosing suitable parameters for a scalable IoC detection system with an acceptable false positive rate requires trade-offs. | The trade-offs should be made using measurement results, having decided what false positive rate is acceptable. |

Table 4.1: An overview of the limitations in the proposed approach.

IoC detection methods that look for group activity, co-occurrence, sequential correlation or NXDOMAINs require stored queries to be mapped to specific users. This cannot be done in Bloom filters, and therefore, these detection methods cannot be performed using Bloom filters. However, hypotheses from malware analysis can be tested using Bloom filters. For example, if a combination of known domain names is assumed to co-occur in case of an infection, the infection can be detected.

The constraints that Bloom filters introduce significantly limit the capabilities of an IoC detection method. However, these constraints can be worked around. A Bloom filter based IoC detection method then provides limited, but targeted detection capabilities, and preserves the privacy of users. The focus of Bloom filters is testing existing IoCs against a dataset, rather than using a dataset to find new IoCs.

### DGA Detection

Malware uses domain names that are randomly generated by a DGA, as we discussed in Section 2.3.3. These domain names can be stored in a Bloom filter, and the exact domain name can be detected later. However, these randomly generated domain names change rapidly. Moreover, a large set of them is queried by an infected machine, where only a few are actually used by the malware. That is a challenge, which is discussed in this section.

Cybereason reported eight real-world DGA variants in 2016 [36]. The report shows that DGAs take varying approaches, from scrambling English words to randomly generating characters. The DGA generated domain names contain one or more random labels. If the domain name would be stored in a Bloom filter as a whole, successful detection is very unlikely.

An alternative is to store all the labels of the domain name separately. In this case, any of the labels can be subject to detection. The unknown Russian DGA that was reported by Cybereason generated domain names with fixed labels and random labels. A fixed label could be a common word used in legitimate domain name. For example, the report shows that the word *pop* was used as fixed label, which commonly refers to the POP3 mail protocol. The fixed labels would trigger detection, but with a high false positive rate. It is possible to store the position of a label in the

original domain name with it. This allows one to test for a label at a specific position. However, no correlation can be made between two detected labels in a Bloom filter. Therefore, it is practically impossible to find out whether the labels originate from the same domain name.

The existing IoC detection methods that we discussed in Section 3.1 use techniques that detect the behavior of DGAs, rather than a snapshot of the domain names it generates. These detection methods could be used to find out which domain names should be checked for in Bloom filters.

Dealing with DGA is both a challenge and a limitation, because detecting random labels is only possible if they are known. Furthermore, it is not possible to find out whether two detected labels belong to the same domain name. However, DGAs are a significant threat, and as we described in Goal **G2**, we want to detect such. Summarizing, detecting domain names generated by DGA is possible using Bloom filters, but there are limitations compared to existing methods that inspect all DNS traffic.

### 4.2.2 False Positives

Bloom filters have a statistical nature, and therefore introduce false positives. In contrast to deterministic domain name lookups in a list, lookups in Bloom filters could lead to incorrect results. In this section, we study the challenge of how to deal with false positives in IoC detection.

We want to use Bloom filters strategically and tactically. That means that future organizational decisions may depend on detection results. For example, suppose that we see that a certain type of malware targets a specific part of the network. In that case, measures could be taken in that direction.

We also want to use Bloom filters to identify the network segment that a possibly infected machine was found in. The local incident responders will then be notified about what IoC was detected. However, what could be the impact of such decision if it was incorrectly taken?

A decision that lasts for a long time may require an investment of time and money. For example, preventively mitigating a threat in a part of the network requires time, as a mitigation solution must be built. Furthermore, the mitigation may require additional hardware for analyzing traffic. Users in the network must be informed, and users may ask questions about that. Communicating decisions and answering questions also requires time.

### 4.2.3 Decision Support

When we detect a malware infection, we want to notify the local incident responders. We only know that an infection is present, and not which user is infected. Suppose that we detected a threat, and the local incident responders want to find and eliminate the infection. They would have to inspect the network traffic of users in order to find the infected machine, which infringes their privacy. If the incident responders were notified based on a false positive, the privacy of their users would be unnecessarily infringed.

SURFnet is part of a community in which security intelligence is shared. We get IoCs from this community that could be used for threat detection. in return, the detection results are useful for the community. If the detection results are incorrect, wrong information is shared in the community. In such community, one may also want to share IoCs that were produced using other detection methods as discussed in Section 3.1. If an IoC is based on incorrect detection results, other community members may start looking for nonexistent threats.

What actions are performed upon the detection of an IoC depends on the situation, and has to be determined by analyzing the possible impact of a false positive. If the impact is high, the situation should be approached with caution. For example, we would like to make sure that we are not dealing with a false positive, before we take actions. This is a trade-off between impact and effort, where more effort is required on a higher impact. However, if IoC detection is employed in other ways than using Bloom filters, the privacy of users is violated in any case.

### 4.2.4 Resilience to Adversarial Actions

In Section 2.5.5, we discussed the pollution and saturation attacks on Bloom filters. In the attacker model, we assumed that the attacker is capable of choosing and inserting as many DNS queries into the Bloom filters as he likes. In practice, the attacker model is more restrictive. The following two conditions apply.

- The attacker must have access to the DNS resolver in order to be able to insert queries.
- The attacker does most likely not have an active host in every network segment that is monitored. Therefore, he cannot pollute or saturate all Bloom filters under the assumption that queries are aggregated per network segment, and in separate Bloom filters.

Performing a pollution or saturation attack requires a large number of DNS queries to be sent to the recursive name servers. Because we do not know what these queries will be, it is not possible to block them. However, we can monitor the number of queries that arrive at our recursive name server over time. An attack would be visible in these numbers. Furthermore, we could periodically calculate the number of bits set $s$, and test whether it exceeds half the number of bits $m$. It is therefore possible to detect a pollution or saturation attack, but not to prevent it. As a result, the Bloom filters will still be polluted, and we cannot do lookups anymore in a polluted filter. Therefore, it is important to identify and stop the attack as soon as possible.

### 4.2.5 Configuring Bloom Filters

We discussed the properties of Bloom filters in Section 2.5.3. For example, Bloom filters are non-enumerable, and one can only ask a Bloom filter whether an exact DNS query was performed. These properties introduce constraints in IoC detection, and hence challenges and limitations. For example, what information should we store in a Bloom filter, and how to group users together? In this section, we discuss the challenges and limitations.

**Grouping Users** Making correlations between DNS queries in a Bloom filter is not possible. However, if all queries from a single user would be stored in a Bloom filter, it is trivially known that all stored queries were performed by that user. Moreover, storing queries from each user in a different Bloom filter is not very efficient. Therefore, it is important to mix queries from multiple users in a Bloom filter. As described in Goal **G3**, we want to be able to identify the network segment a possibly infected user is in. A natural way to mix users would then be storing the queries of all users in that network segment together.

Mixing queries from users by network segment could be done in two ways. These ways and their advantages and disadvantages are described below.

- Queries from each network segment can be stored together in a separate Bloom filter. A pollution or saturation attack would not pollute all Bloom filters in this case. However, since common domain names such as *google.com* are queried by all subnets, more memory space is required.
- Queries from each network segment can be stored together in the same Bloom filter. In this case, a label indicating the network segment would have to be prepended to the domain name. An advantage is that only a single Bloom filter has to be maintained. However, a pollution or saturation attack would always pollute the entire Bloom filter.

Even though no correlations can be made between DNS queries in Bloom filters, personal websites can be identified. If one knows that a specific domain name belongs to an individual, the presence of that domain name in a Bloom filter may reveal the presence of the individual. However, it is not possible to find out which other domain names the individual visited, as no correlations can be made between queries.

**Time Based Aggregation** As described in Goal **G4**, we want to be able to aggregate DNS queries over time. The timestamp at which a query was performed can be stored in a Bloom filter. However, since no correlations can be made between a timestamp and a query, we cannot know which query was performed at a specific timestamp.

Alternatively, Bloom filters can be bound to a specific timeslot. For example, DNS queries can be aggregated per hour. A new Bloom filter is supplied every hour, and the old one is stored separately. The exact timestamp of a query cannot be determined this way, but it is possible to estimate the time of occurrence. When using a Bloom filter based IoC detection method for strategical and tactical purposes, this restriction is fine. It is sufficient to know that a threat was present during a specific hour and day.

Bloom filters containing DNS queries can be stored for a long time to allow lookups in history. The information in these Bloom filters is complete, and no new information will be added. To save storage space, Bloom filters that are aggregated per hour can be combined, such that a single Bloom filter contains all queries of one day. The combined Bloom filter has the same size as the hourly ones. However, the combined Bloom filter contains queries from all hourly filters, which may increase the false positive rate. Therefore, it is important to always check the actual expected false positive rate of the combined filter.

**Information to Store** As we discussed in Section 2.5.3, Property 2 states it is only possible to ask a Bloom filter whether an exact DNS query was performed. Therefore, it is important to think about what information to store in a Bloom filter. The following aspects should be considered.

- Domain names are case insensitive, as we discussed in Section 2.1.1. However, taking a hash of two domain names with different case results in different hash values. For example, the SHA256 hash of *SuRfNeT.nL* is different from the hash of *surfnet.nl*, while for DNS, the domain names are equal. Therefore, it is important to canonicalize domain names before storing them in a Bloom filter, to avoid mismatches in detection.
- Can the threats we want to detect be identified solely using domain names? If so, it suffices to store domain names only. However, some threats must be identified using combined information. For example, malware may use the TXT record of a specific domain name to store command and control information. The record type would be required in this case.
- As we discussed in Section 4.2.1, detecting DGA generated domain names would require the labels of a domain name to be stored separately. This increases the number of unique domain names, because they are split into multiple parts.

The information that is stored in a Bloom filter directly influences decisions about goals and configuration. Therefore, this should be one of the first things to consider.

**Parameters** Bloom filters have configurable parameters, as we discussed in Section 2.5.2. These parameters depend on the number of unique domain names that will be put in the Bloom filters, and what false positive rate is considered acceptable. That means that measuring the number of unique domain names occurring on the network is very important. What other challenges arise when choosing suitable parameters?

We want to aggregate Bloom filters over time, and store them for a long time. Storage space is saved when hourly Bloom filters are aggregated into a daily one when the day has passed. Bloom filters that are configured with the same parameters can be aggregated by taking the union, as we discussed in Section 2.5.3. However, the parameters of the hourly and daily Bloom filters should then be the same. The daily Bloom filters will contain more unique domain names than the hourly ones. Therefore, both Bloom filters should be configured to store the daily number of unique domain names to avoid higher false positive rates.

We can aggregate DNS queries from individual network segments in separate Bloom filters. In that case, we can identify the network segment in which a query was performed. Not every network segment contains the same number of users, and hence the query diversity between network

segments can be large. For smaller network segments, a smaller Bloom filter suffices to achieve the same false positive rate. Bloom filters containing queries from the same network segment can then still be aggregated. However, Bloom filters from different network segments cannot be aggregated, because the parameters are different. Having different parameters for each network segment therefore requires maintaining as many Bloom filters. Furthermore, measurements should be performed for each network segment, and if the number of queries changes, the parameters should be reviewed.

### 4.2.6 Scalability

The primary task of the recursive name server is answering DNS queries, and the Bloom filter software should not interfere with that. In this section, we discuss the trade-offs that must be made in implementing Bloom filters. The following trade-offs must be considered.

- The Bloom filters should be configured to deal with a specified number of unique domain names, while providing an acceptable false positive rate. Using a larger $m$ and $k$ for a Bloom filter results in a smaller false positive rate, but requires more memory.
- Aggregation of DNS queries over time allows one to determine when a query occurred. Aggregation in a shorter time window allows for more accurate pinpointing. However, more memory space is required to store the larger number of Bloom filters. Moreover, too short time windows may introduce privacy risks, as specific activity may be easier to spot.
- Queries can either be stored in a single Bloom filter, or mixed into multiple. If queries are mixed into multiple Bloom filters, their false positive rates may be multiplied. Moreover, the damage of a pollution or saturation attack is spread over multiple filters. However, more memory space is required.

The trade-offs can be made when measurements have been performed. The number of unique domain names over time is the most important metric to measure.

# Chapter 5

# Approach: IoC Detection with Bloom Filters

In the previous chapter, we analyzed the goals that are set for a Bloom filter based IoC detection system, and identified the challenges and limitations that apply. In this chapter, we discuss the prototype that was designed and developed to fulfil the goals in Chapter 4. Furthermore, we explain how to estimate parameters for the Bloom filters, which is required for the proper configuration of an IoC detection system using the prototype.

## 5.1 Prototype

In this section, we describe the prototype that was developed to implement a Bloom filter based IoC detection system. The Dutch company Quarantainenet[1] developed open-source software written in C, implementing Bloom filters for storing DNS queries and performing lookups. The software is called *Honas (Hostname Searching)*, and we used it as starting point for our prototype.

The goals defined in this thesis are different from the goals that Quarantainenet specified. Therefore, we had to implement the missing functionality. We tailored the Honas software to fulfil the goals set in Chapter 4. Honas is lightweight and designed to run as a separate process on a recursive nameserver. Furthermore, it consists of a set of applications in which the functionality is divided. Below we explain the applications, what functionality they implement, and which goals they fulfil. A summary of the Honas limitations and of the introduced improvements for this project is given in Table 5.1.

**Data Gathering** The data gathering application *honas-gather* is an existing part of Honas, and implements Bloom filters with configurable parameters. DNS queries are read from a file, and the domain names are extracted. Those are then canonicalized and stored in the Bloom filters. The Bloom filters are aggregated per configurable time period, and stored on disk when the time period passes.

DNS queries are very privacy sensitive, and therefore, we need to transport them to the Bloom filters in an untraceable way. Therefore, we implemented a technology called *dnstap*[2] to listen for incoming DNS queries locally. A UNIX domain socket is used to establish a communication channel. The query data is then transported using a *FrameStream*[3], which is a data streaming technology that transports data encoded as *protocol buffer*[4]. This way, query data does not have to be transferred to another host over the network. We chose dnstap, because it is supported by all major DNS resolver applications, making it easy to use for other network operators. Moreover,

[1] https://www.quarantainenet.nl/
[2] http://dnstap.info/
[3] https://github.com/farsightsec/golang-framestream
[4] https://developers.google.com/protocol-buffers/

| Functionality | Limitations | Our Implementation | Goals |
|---|---|---|---|
| Data Gathering | Untraceable input channel for DNS queries is missing, separate labels are not stored, and data aggregation by network segment is missing. | *dnstap* as untraceable input channel, storing all separate labels including some combinations, entity identification by network segment, and dry-run mode for parameter estimation. | **G1**, **G2**, **G3** |
| Data Searching | This functionality did not have any limitations. | The unmodified Honas application was used. | **G1**, **G2**, **G3** |
| Bloom Filter Information | No information about the fill rate or false positive rate was provided. | Information about the fill rate and false positive rate of the Bloom filters. | **G1**, **G2**, **G3** |
| Bloom Filter Combination | No Honas application exists implementing such functionality. | Application that aggregates two Bloom filters. | **G4** |

Table 5.1: An overview of limitations in Honas and the introduced improvements.

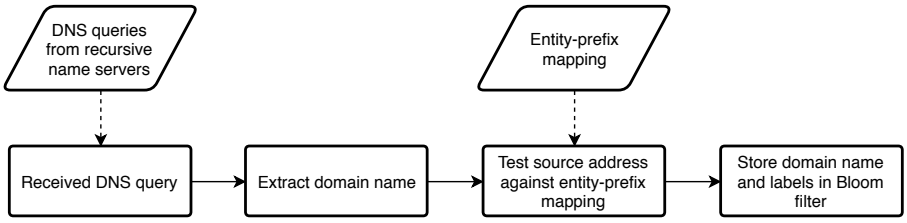



Figure 5.1: The path that DNS queries take from the recursive name servers to Bloom filters in the Honas prototype.

this choice makes the prototype widely applicable in the field. The dnstap functionality was implemented to fulfil Goal **G1**.

The existing data gathering application stores domain names as a whole only. To fulfil Goal **G2**, we extended the application to split the domain name into labels, and store all labels separately. The following information is stored from the labels.

- The domain name as a whole is stored, because it allows one to look up the whole domain name.
- All separate labels in the domain name are stored, because it allows one to look up random labels. The TLD label is not stored, since it is never a random label.
- Domain registrations are performed on SLDs in a specific TLD. Other domains are then a subdomain in such SLD. A malicious domain name may hence be distinguishable by its SLD and TLD. Therefore, the *SLD.TLD* is stored to allow such lookups.

We also implemented entity identification by network segment, to fulfil Goal **G3**. An entity is identified by its name, and it can be mapped to one or more network segments. A JSON file containing mappings between entities and IPv4 and IPv6 prefixes is loaded at startup. The source address of each received DNS query is then tested against these prefixes. If it was successfully matched, the entity name is stored with the domain name. An overview of the steps that the data gathering application takes in processing DNS queries is depicted in Figure 5.1.

To better understand what information is stored in the Bloom filters, we consider the domain name *malicious.site.example.com.* as an example. An overview of which parts of the example domain name are stored, given an example network architecture is depicted in Figure 5.2. If a client in Network *A* performs a DNS query for the example domain name, the label *A* is prepended

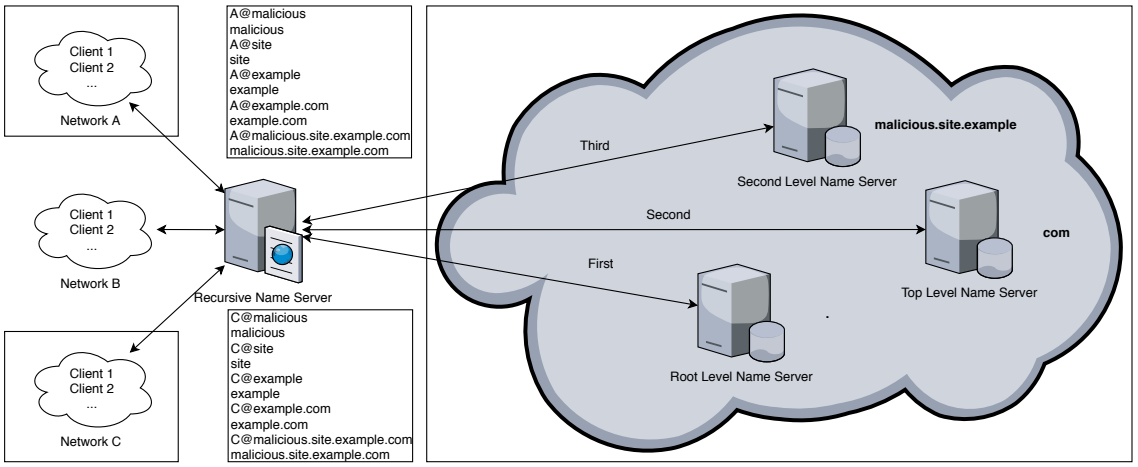



Figure 5.2: An overview of the domain name parts that are stored based on the network and DNS architecture.

to the domain name and its labels. The same goes for clients in network $B$ or $C$. The information that is stored for network $A$ and $C$ is displayed in the top and bottom boxes respectively.

We must estimate the Bloom filter parameters in order to configure the prototype properly, as we discussed in Section 4.2.5. More specifically, we must get an estimate for $n$, the number of distinct elements we expect to insert into the filter, and choose appropriate values for $m$ and $k$ given a desired false positive probability $p$. Therefore, we implemented a dry-run mode that counts the number of unique domain names and labels that are stored in the Bloom filters per hour and per day. An advice about suitable parameters for the Bloom filters is provided daily at midnight, and when the prototype terminates. The advice is based on the highest number of unique domain names observed over time. The theoretical false positive rate from Equation 2.1 is then calculated using the optimal number of hash functions $k$ from Equation 2.2. We want to select an acceptable false positive rate $p$, and calculate suitable parameters. In that sense, the advice contains suitable parameters for $p = 10^{-3}$, $p = 10^{-4}$, and $p = 10^{-5}$. This allows one to choose whatever false positive rate is acceptable. The calculation is rounded up to the nearest hundred thousand for simplicity, and a tolerance of 10% is added on top of that. The tolerance is chosen as a precaution measure to handle spikes during busy days. For convenience, a warning can be generated when the actual expected false positive rate of a Bloom filter exceeds the theoretical one during the gathering of data.

Counting the exact number of unique domain names potentially consumes a lot of memory. Therefore, Honas uses the HyperLogLog [17] algorithm. This algorithm approximates the number of unique domain names within an error bound, rather than calculating an exact value. It requires very little resources and is cheap to execute.

**Data Searching** The Bloom filter searching application *honas-search* is an existing part of Honas, and implements searching functionality for Bloom filters that are stored on disk. It accepts a JSON file containing the domain names or labels to search for, and returns the search results as JSON as well. We use the unmodified application to search the Bloom filters, fulfilling Goal **G2**.

**Bloom Filter Information** The Bloom filter information application *honas-info* is an existing part of Honas, and provides information about Bloom filters that are stored on disk. The existing application gives information about the Bloom filter parameters and the number of bits that are set. We implemented logic that provides the fill rate and the actual expected false positive rate of the Bloom filters. We use the modified application to gain insight into the state of stored Bloom filters, and for performance readings such as false positive rate.

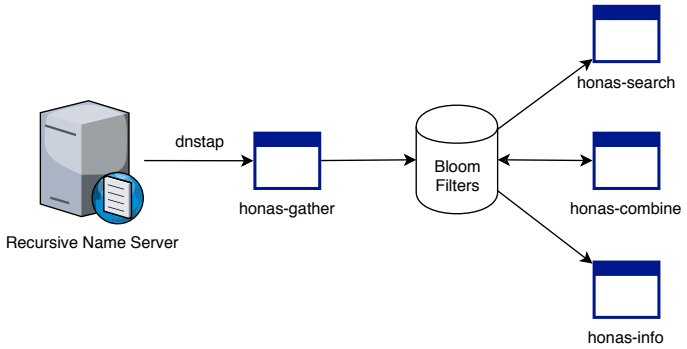



Figure 5.3: An overview of the Honas prototype, including all individual applications.

| Parameter | Description |
|---|---|
| bloomfilter_path | The path in which the Bloom filter state files are stored. Every file corresponds to a *period_length* time frame. |
| subnet_activity_path | The path to the JSON file containing mappings between entities and IPv4 and IPv6 prefixes. |
| period_length | The length of a time frame to store DNS queries for in a Bloom filter. A new Bloom filter is started when the period ends, and the old one is stored on disk. |
| number_of_bits_per_filter | Corresponds to $m$, the size of the Bloom filters in bits. |
| number_of_hashes | Corresponds to $k$, the number of hash functions for the Bloom filters. |

Table 5.2: An overview of the parameters that Honas implements.

**Bloom Filter Combination** We developed the Bloom filter combination application *honas-combine* to fulfil Goal **G4**. It implements the Bloom filter union operation, which aggregates the data in two Bloom filters by taking the bitwise OR. The result is then stored on disk. We use this application to combine hourly Bloom filters into a daily one.

The functionality in the applications discussed above fulfil all goals except **G5**. Goal **G5** is not explicitly mentioned because we consider it a metagoal. Compliance with the current regulation is the main reason for choosing Bloom filters for IoC detection. The prototype therefore implicitly fulfils Goal **G5**.

The applications we discussed perform different operations on the Bloom filters. The data gathering application writes to the Bloom filters, and the searching and information applications only read. The combination application both reads and writes. An overview of the relationship of the Honas applications to the Bloom filters is depicted in Figure 5.3. Furthermore, Honas implements a set of configurable parameters, including the ones we discussed in Section 2.5.2. The parameters that Honas implements are displayed in Table 5.2.

## 5.2 Parameter Estimation

In order for the prototype to function correctly, we need to configure the parameters for the Bloom filters. In Section 5.1 we explained that these parameters depend on the number of unique domain names that occur on the network. Therefore, this number should be measured. In this section, we discuss why the measurements should be performed. Furthermore, we discuss how the measurements work, and for how long they should be performed.

Bloom filters are sized based on the number of distinct elements inserted into it. The size is fixed, and should be estimated in advance. Moreover, the false positive rate depends on the size

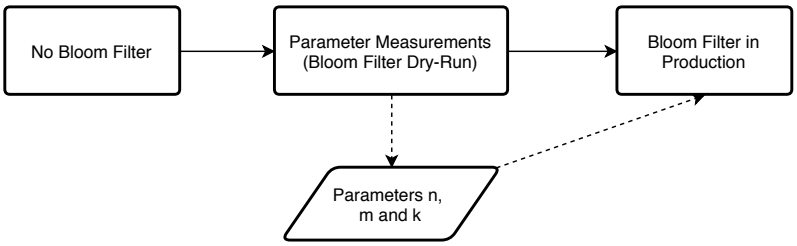



Figure 5.4: The process of parameter estimation and using Bloom filters for IoC detection in production.

and fill rate of the Bloom filter. Setting the size of a Bloom filter by guessing may be incorrect, and as a result, the false positive rate may become higher than acceptable. Moreover, if the Bloom filter is too large, the data is stored inefficiently. By measuring the number of distinct elements, the size of the filter can be accurately determined.

During the parameter estimation, the number of distinct elements that go into the Bloom filters is measured. The size of the Bloom filters and number of hash functions can then be manually configured for production. A visual representation of the parameter estimation process is depicted in Figure 5.4.

As we discussed in Section 5.1, the Honas prototype implements a dry-run feature. This feature can be used to perform the parameter estimation. The dry-run should be performed for at least one week, in which all weekdays are covered at least once. Differences between specific days are best discovered this way. The busiest day should be used as reference for the configuration, as the size parameter is fixed and the Bloom filters must be able to handle data from all different weekdays. The dry-run feature in Honas uses the highest value reported over time for the advice. Therefore, the advice already uses the busiest day as reference automatically. The dry-run is best performed in a week that is representative for the normal use of the network. For example, in a network where the users are students, there is likely less activity during vacations. An example of a real world dry-run can be found in Section 6.2.

# Chapter 6

# Validation

In the previous chapter, we discussed the prototype and Bloom filter parameter estimation process. These are required to properly design and configure an IoC detection system based on Bloom filters. In this chapter, we explain how we validated the prototype, and that we can indeed fulfil the goals that we set in Chapter 4. Furthermore, we discuss the experiment setup, the dry-run and the validation scenarios.

## 6.1 Experiment Setup

In this section, we describe the experiment setup that was used to collect DNS queries, perform the dry-run and validate the prototype. The experiment was conducted in two phases: A *dry-run* for parameter configuration, and a validation set of experiments on three separate case studies.

We validated the prototype at SURFnet, which is the Dutch NREN (National Research and Education Network). We were allowed to collect DNS queries from the operational recursive name servers at SURFnet. These name servers receive approximately 5,000 to 10,000 queries per second on an average day. Furthermore, they serve approximately 150,000 to 200,000 unique IP-addresses every day.

The Bloom filter based prototype gives strong privacy guarantees, and therefore, we were allowed to gather DNS queries from the recursive name servers in the SURFnet network. These queries are performed by users in SURFnet's constituency. To collect these queries, we use *The Extensible Ethernet MOnitor* (EEMO)[1], which is an open-source network traffic analysis tool developed by SURFnet. The client side vantage point is used to monitor queries to the recursive name servers. These queries are then continuously forwarded to a server running EEMO via an encrypted channel.

EEMO features a plugin system, which allows multiple actions to be taken on the same traffic stream. We developed a plugin that outputs the received queries locally to the Honas prototype using dnstap. The data gathering process listens locally on the dnstap interface, and stores the domain names from queries in Bloom filters. The experiment setup as described is depicted in Figure 6.1.

To validate the prototype, we first performed a dry-run for a period of one week to estimate the parameters for the Bloom filters. The dry-run is discussed in Section 6.2. We then validated the prototype using three scenarios, which are discussed in Section 6.3.1. The validation was performed for 23 days, starting on the $1^{st}$ of July, and ending on the $23^{rd}$. During the validation, we collected 8,183,467,204 DNS queries in total.

[1] https://github.com/SURFnet/eemo

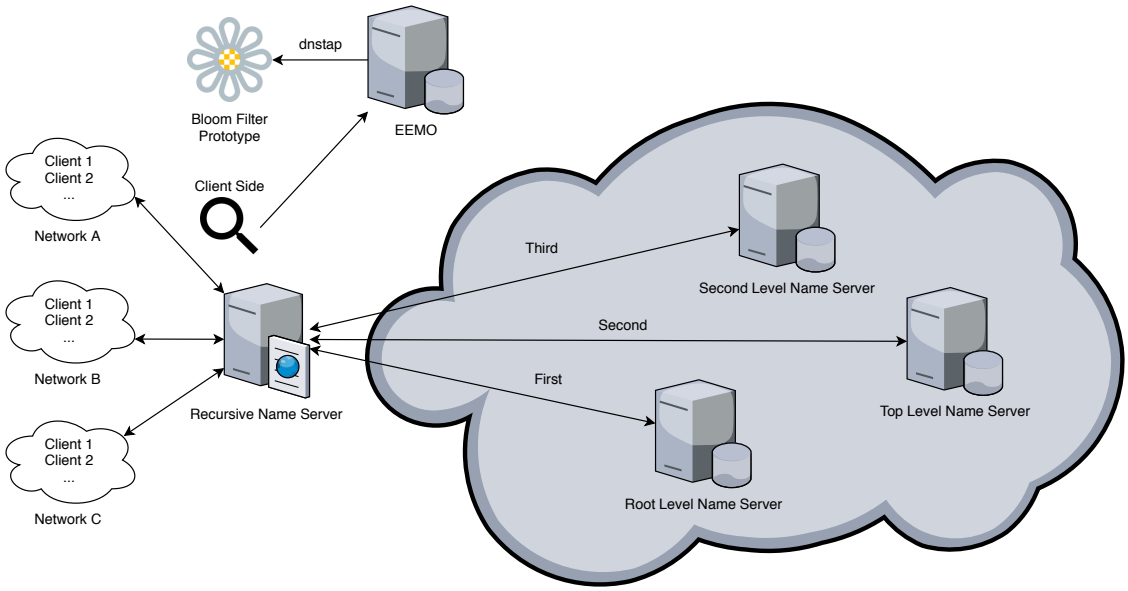



Figure 6.1: The experiment setup for data collection and validation at SURFnet.

## 6.2 Dry-Run

We performed a dry-run using the Honas prototype as first part of the validation. The dry-run was performed for one week, using real world data from the operational recursive name servers at SURFnet. The real world data consists of DNS queries from users in SURFnet's constituency. The dry-run resulted in an advice for Bloom filter parameters. The results of the dry-run are discussed in Section 6.2.1. Furthermore, in Section 6.2.2, we validate the dry-run results and check whether the parameters were correctly chosen.

### 6.2.1 Dry-Run Results

We performed a dry-run to determine proper parameters for the Bloom filters, as discussed in Section 6.2. In this section, we discuss the results of the dry-run, and the parameters that we chose based on them.

The dry-run was performed using the Honas prototype for the period of one week, from Monday until Sunday. The dry-run counted the number of unique domain names per hour and per day, to show the difference between numbers in different aggregation timeslots. Furthermore, we want the Bloom filters that are stored for the long term aggregated per day.

We now discuss the results of the dry-run. A visual representation of the dry-run results is depicted in Figure 6.2. This figure depicts the number of unique domain names over time, which is derived from the query stream. We see that the query stream follows a day-night and work week rhythm. The number of queries is low and constant during nights and weekends. The number of queries gradually increases during the day, until it hits the busiest moment. From that moment on, the number gradually decreases again. Note that the number of queries differs each day. In the week we performed the dry-run, Tuesday was the busiest day. The Bloom filters should be configured to handle the queries during this day. Even though the number of queries is low and constant during nights and weekends, it never goes quiet. There is a baseline of queries that are performed by automated systems, such as servers.

We also see that the number of unique domain names per day is much higher than per hour. This difference is explained by the time at which domain names are queried. Not all unique domain names are queried in the same hour, and not every hour either. For example, the domain name *example.com* could be queried by a user in a first hour. If that user leaves the network

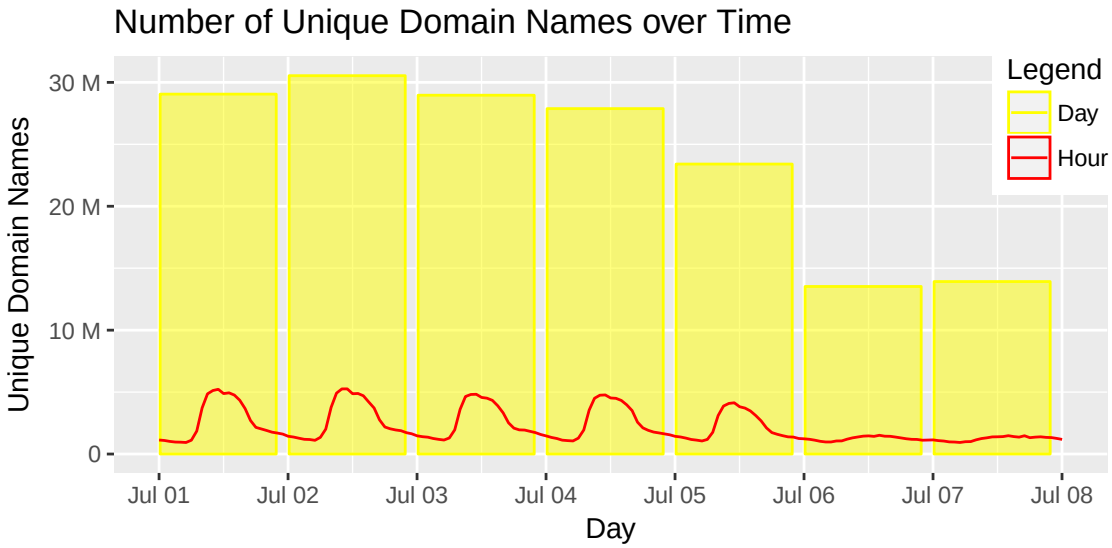



Figure 6.2: A depiction of the dry-run results plotted per hour and per day.

in the second hour, the domain name may no longer be queried. In that case, *example.com* is not recorded in the second hour. However, it is recorded in that day. DNS caching could also cause part of the difference. Larger organizations commonly have their own local recursive name server, which local users query. The local name server then queries the recursive name servers at SURFnet. When a domain name is queried in the first hour, the answer may be cached. As a result, the local name server returns the cached answer in the next hour, and does not perform another query.

The dry-run provided an advice for the Bloom filter parameters based on the number of unique domain names observed. This advice is displayed in Listing 6.1. The advice provides parameter estimations for Bloom filters aggregated per hour and per day. Furthermore, the estimations are tailored to false positive rates of different proportions. This allows one to select which false positive rate is acceptable. Remember that we want to aggregate hourly Bloom filters into a daily one, and that the Bloom filters must have the same parameters in order to do so. Since the numbers per day are significantly higher than per hour, we used the daily advice to configure the Bloom filters for the validation. The false positive rate still has to be chosen. For the validation, we consider a false positive rate of $10^{-3}$ acceptable. Therefore, we chose the size of the Bloom filters to be $m = 491040000$ (491 Megabits), and the number of hash functions $k = 10$.

```
------------------------------ Advice ------------------------------
The numbers are rounded up to the nearest hundred-thousand, and a tolerance of 10
    percent is added.
---------------------------- Hourly Filters ----------------------------
For a false positive rate of 1 / 1000, BF size (m) should be 90750000, based on
    5732164 unique domain names
The number of hash functions (k) should be 10
For a false positive rate of 1 / 10000, BF size (m) should be 120890000, based on
    5732164 unique domain names
The number of hash functions (k) should be 14
For a false positive rate of 1 / 100000, BF size (m) should be 151140000, based on
    5732164 unique domain names
The number of hash functions (k) should be 16
---------------------------- Daily Filters ----------------------------
For a false positive rate of 1 / 1000, BF size (m) should be 491040000, based on
    31043053 unique domain names
The number of hash functions (k) should be 10
For a false positive rate of 1 / 10000, BF size (m) should be 654610000, based on
```

```
    31043053 unique domain names
The number of hash functions (k) should be 14
For a false positive rate of 1 / 100000, BF size (m) should be 818290000, based on
    31043053 unique domain names
The number of hash functions (k) should be 16
──────────────────────────────────────── End ────────────────────────────────────────
```

Listing 6.1: An overview of the results of the dry-run mode of the prototype.

### 6.2.2 Dry-Run Validation

We performed a dry-run and chose the Bloom filter parameters according to the results, as we discussed in Section 6.2.1. We then validated the prototype during a period of 23 days, and evaluated the parameters we chose. In this section, we dicuss whether they were correctly chosen.

The Bloom filter parameters should be chosen such that the actual expected false positive rate $p_s$ does not exceed the theoretical false positive rate $p$, as calculated in [12]. We chose the size of the Bloom filter $m$ to be the daily maximum over the time the dry-run was performed. Remember that we chose $m$ as such because we want to aggregate the hourly Bloom filters into a daily one when the day has passed. Therefore, the actual false positive rate $p_s$ of the aggregated Bloom filters should not exceed the theoretical false positive rate $p$ that we set.

We analyzed the actual false positive rate $p_s$ of the Bloom filters that are aggregated per day, taking into account that $p = 10^{-3}$ is acceptable. The result of that analysis is visualized in Figure 6.3. The bars in the graph show $p_s$ for the Bloom filters aggregated per day. Furthermore, the y-axis shows the corresponding $p_s$ on a logarithmic scale. The horizontal line in the middle represents $p = 10^{-3}$, which represents the acceptable false positive rate. The bars should not reach below the horizontal line, because anything below is larger than the acceptable false positive rate. We clearly see that this does not happen, and therefore, we conclude that we have chosen the Bloom filter parameters correctly.

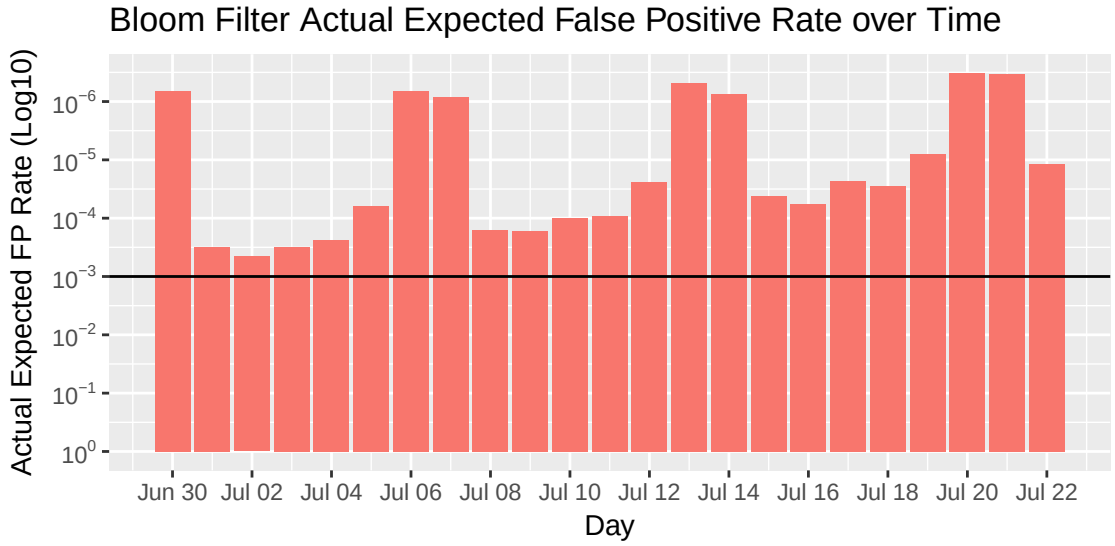



Figure 6.3: A depiction of the actual expected false positive rate $p_s$ per day over time on a logarithmic scale. The horizontal line represents the theoretical false positive rate $p = 10^{-3}$. A higher value indicates a better result.

## 6.3 Validation Scenarios

In this section, we specify three scenarios in which we validated the prototype. We discuss the data we use for each scenario and the corresponding ground truth. Furthermore, we perform the validation and discuss and analyze the results. The validation was performed using the following experiments.

(a) All detected threats that are present in the ground truth are also present in the Bloom filters.

(b) We test the Bloom filter contents against the ground truth. The ratio of false positives occurring in the Bloom filters during this experiment should not exceed the actual expected false positive rate $p_s$. This experiment tests how often false positives occur in practice.

### 6.3.1 Scenario Description

In this section, we specify and describe the three scenarios in which we validated the prototype. For the validation, we collected DNS queries in Bloom filters for a period of three weeks. We then validated whether the prototype performed according to previously discussed hypotheses in three scenarios. These scenarios were tested against the stored Bloom filters. The scenarios are as follows, and are discussed in detail in the upcoming sections.

(1) Booters

(2) Spam Filtering

(3) National Detection Network

#### Scenario 1: Booters

Booters are websites that offer DDoS-as-a-Service, as we discussed in Section 2.3.2. These websites are referred to by domain names. The Booter research by Santanna et al. [33] resulted in a list of known Booter domain names.

The CERT team of SURFnet regularly deals with DDoS attacks against its constituency. Suppose that a specific organization is victim of a DDoS attack. We can test if a known Booter domain name was queried from the network of the same organization. Suppose that a known Booter query was detected inside the network of the targeted organization. That suggests that the DDoS attack may be requested by somebody inside that organization. A real-world example is a student that wants his online exams cancelled. To do so, he mounts a DDoS attack on the school network, so that it becomes unavailable. The anatomy of such inside-job DDoS attack

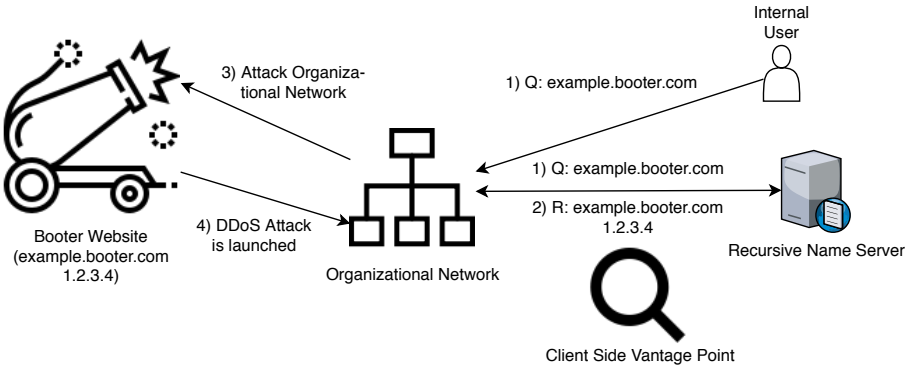



Figure 6.4: The anatomy of an *inside-job* DDoS attack using a Booter website.

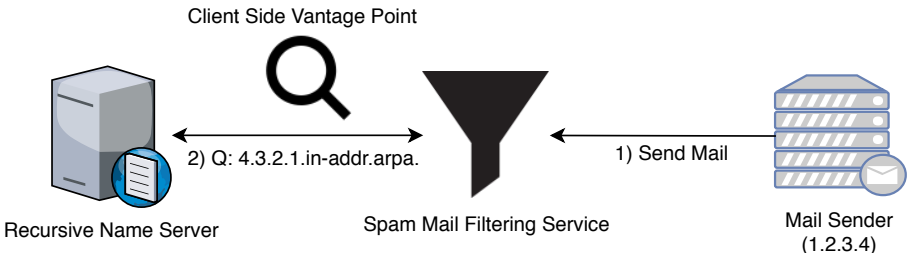



Figure 6.5: The reverse DNS query process in the mail filtering service.

using a Booter website is depicted in Figure 6.4. The location of the client side vantage point is also marked in this depiction.

Using the list of known Booter domain names, we can test against the Bloom filters whether Booter queries were performed, and in which organizational network. Prior experience at SURFnet shows that such inside jobs are common. For this reason, SURFnet's privacy officer has given permission to record information about queries for Booter domains. While validating this scenario, we use this permission to record actual queries as ground truth. We continuously log these queries, including source address and timestamp.

### Scenario 2: Spam Filtering

SURFnet provides a spam mail filtering service to its constituency. When the mail filtering service receives mail from an IP address, it performs a reverse DNS lookup. This operation is commonly performed in an attempt to authenticate the sender [13]. This process is described in Figure 6.5. The location of the client side vantage point is also marked in this depiction.

The mail filtering service uses public blacklists containing IP-addresses of hosts that are known for sending spam mail. A list of blacklisted IP addresses from which mail was received including timestamps is generated on a daily basis. We can convert the IP-addresses to the corresponding reverse DNS queries, and test the resulting domain names against the Bloom filters.

Some organizations in SURFnet's constituency host their own mail services. If we find that blacklisted domain names are queried by hosts in the networks of these organizations, they likely receive mail from the blacklisted IP addresses. This way, we can make an overview of which organizations that do not use the mail filtering service offered by SURFnet receive spam email. Furthermore, if we know that a particular spam run is linked to a specific blacklisted IP-address, we can find out what organizations in SURFnet's constituency are targeted.

It would also be interesting to find out if a particular spam run targets a specific type of organizations. Organizations in SURFnet's constituency are categorized by type. For example, an institution for vocational or polytechnical education, or an academic hospital. Suppose that we found that a spam run targets institutions for vocational education in particular. A targeted awareness campaign could be set up to educate users in targeted institutions. Furthermore, the mail filtering service could be improved to block the spam mail.

### Scenario 3: National Detection Network

SURFnet is part of a community called the National Detection Network (NDN), which is operated by the National Cyber Security Centre (NCSC)[2] in the Netherlands. The objective of this community is sharing security intelligence about nationwide threats such as malware. Moreover, the intelligence that is shared is supposed to be of high quality. SURFnet obtains IoCs from this community on a daily basis. Depending on the type of IoC, they contain domain names that can be tested for on the network.

[2]https://www.ncsc.nl/english

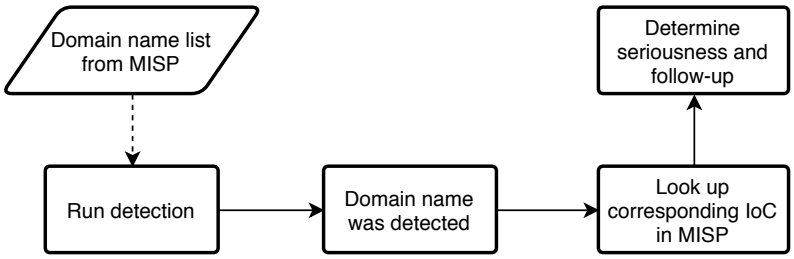



Figure 6.6: The process of detection and impact determination in the NDN scenario.

The IoCs are shared via an application called MISP[3] (Malware Information Sharing Platform). A list of domain names from shared IoCs is exported from the MISP on a daily basis. These domain names are then tested against the Bloom filters. When a domain name from the list is detected, the corresponding IoC is looked up again in the MISP. The MISP contains information about the IoC, allowing one to determine the impact and follow-up actions. A visual representation of this process is depicted in Figure 6.6.

We consider an example for clarity. Suppose that we exported a list of domain names from the MISP, and we tested this list against the Bloom filters. We found the domain name *a.pomf.cat* in the Bloom filters, and we want to know what IoC this domain name represents. To do so, we look up the domain name in the MISP, and find that it is associated with an IoC that indicates the presence of a Microsoft Word document dropping malware. A screenshot of the MISP interface displaying the corresponding IoC is depicted in Figure 6.7.

The NDN scenario is very interesting for SURFnet, because the quality of the IoCs is high. However, plainly logging all DNS queries to test a list of IoCs against does not match the operative privacy requirements internal to the organization. As a result, no information is logged at all, and the IoCs are not used. Using the Bloom filters, we can store all DNS queries in a privacy-friendly way. That means that we can now use the IoCs for threat detection on the network.

Since we cannot plainly test the IoCs against a traffic stream of the recursive name servers, it is harder to obtain ground truth for this scenario. Therefore, we take a different approach here. The privacy policy of the SURFnet network allows targeted investigation of DNS queries for security purposes. Suppose that we detect an IoC containing a domain name that regularly occurs, possibly in the same organizational network. For example, the IoC appears every hour or every day. Under these circumstances, it becomes reasonable and allowed to log queries for this specific domain name, including source address and timestamps. That way, we obtain ground truth for this scenario, representing frequent threats to one or more organizations. However, in a real-world situation we would first look up information about the frequently detected IoC in the MISP to decide whether it poses a real threat.

It would be interesting for SURFnet on a strategical and tactical level to know whether a specific high-profile threat occurred in its constituency. If a certain type of threat targets a specific type of organizations, we could ask those organizations to actively mitigate the threat. Moreover, the knowledge of which threats occur in the network contributes to a global threat overview. SURFnet provides an annual report on cyber threats in its constituency. An example of the 2017 edition of the report is available in [10]. A global threat overview is useful for such report.

### 6.3.2 Validation Results

We configured the prototype according to the dry-run results in Section 6.2.1, and defined three scenarios which we used to validate the prototype. In this section, we analyze and discuss the results of the validation scenarios.

[3] http://www.misp-project.org/

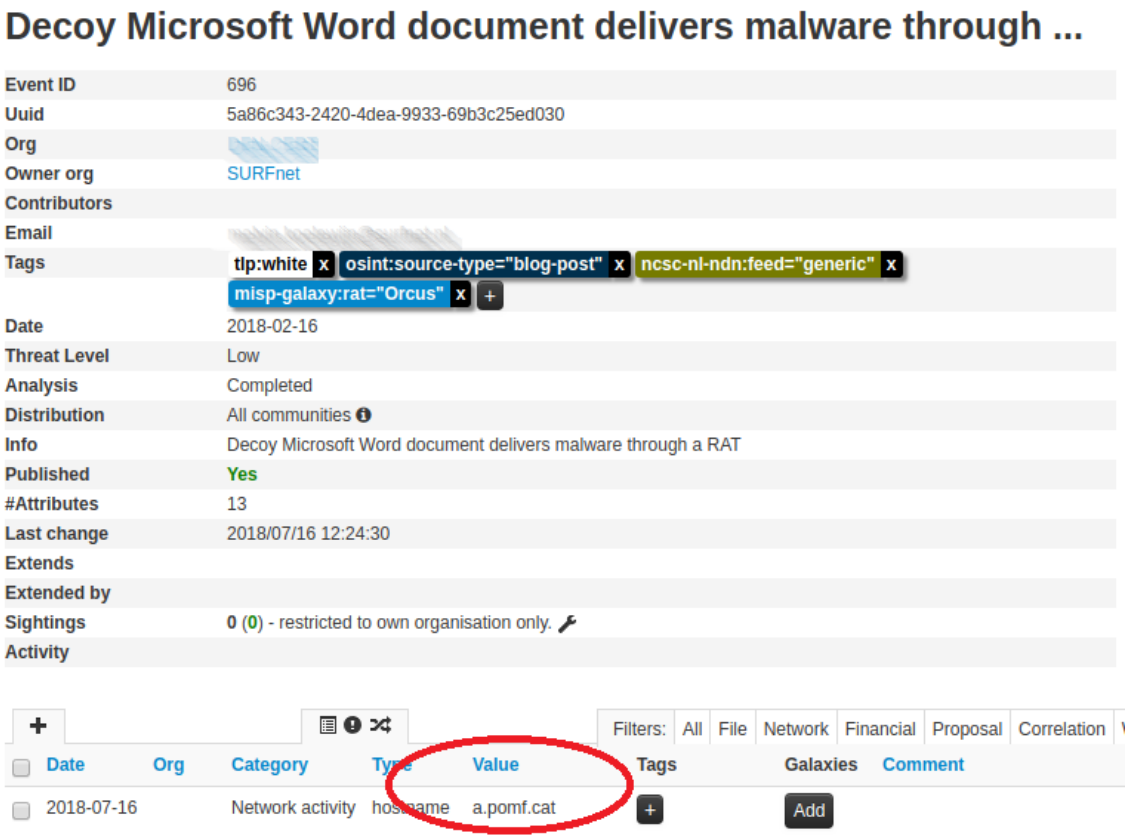



Figure 6.7: An example of an IoC in the MISP platform. The associated domain name is emphasized in red on the bottom. Confidential fields are redacted.

### Scenario 1: Booters

In this section, we analyze and discuss the results of the Booters validation scenario. We first performed Experiment (a) using the detection results, in which we tested the ground truth against the Bloom filter contents, and checked whether all entries from the ground truth are present in the Bloom filters. The results of this experiment are available in Table 6.1a. The first row indicates which daily Bloom filter contents were used. The second row indicates how many detected Booter queries were present in the ground truth. The last row indicates how many Booter queries were present in the Bloom filters. As we can see, all Booter queries from the ground truth appeared in the Bloom filters. Therefore, the prototype performed as expected.

We performed Experiment (b) next, using the detection results. In this experiment, we tested the Bloom filters against the ground truth. We checked every known Booter domain name against the Bloom filters. If a detected domain name is not in the ground truth, and neither in the list of known Booter domain names, we encountered a false positive. The results of this experiment are available in Table 6.1b. The first row again indicates which daily Bloom filter contents were used. The second row indicates how many detection results we encountered in the Bloom filters. The last row indicates how many detection results we encountered in the ground truth. When the number of detection results that occurred in the ground truth is lower than in the Bloom filters, we likely encountered a false positive. We see that all numbers in the second row are equal to the corresponding numbers in the third row. Therefore, we conclude that we did not encounter a false positive. We also did not expect to see any false positives, because the number of detected queries is very small. When the number of detected queries grows, it is more likely to see false positives.

We see a difference in the numbers in Table 6.1a and 6.1b. The number of detected Booter queries is slightly larger in Experiment (b) than it is in Experiment (a). This difference is explained

| Day | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Ground Truth** | 1 | 11 | 5 | 5 | 15 | 7 | 2 | 1 | 6 | 4 | 4 | 6 | 1 | 2 | 1 | 0 | 2 | 0 | 5 | 0 | 2 | 1 | 3 |
| **Bloom Filters** | 1 | 11 | 5 | 5 | 15 | 7 | 2 | 1 | 6 | 4 | 4 | 6 | 1 | 2 | 1 | 0 | 2 | 0 | 5 | 0 | 2 | 1 | 3 |

(a) Results for Experiment (a) in the Booters scenario.

| Day | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bloom Filters** | 2 | 14 | 5 | 6 | 18 | 9 | 2 | 1 | 7 | 4 | 6 | 6 | 2 | 2 | 1 | 1 | 2 | 2 | 5 | 0 | 2 | 2 | 4 |
| **Ground Truth** | 2 | 14 | 5 | 6 | 18 | 9 | 2 | 1 | 7 | 4 | 6 | 6 | 2 | 2 | 1 | 1 | 2 | 2 | 5 | 0 | 2 | 2 | 4 |

(b) Results for Experiment (b) in the Booters scenario.

Table 6.1: Results for Experiments (a) and (b) in the Booters validation scenario.

by the way the Honas prototype stores domain names from DNS queries. It stores the domain name as a whole, but also the separate labels. In the days where the number differ, a DNS query was performed for a subdomain of a Booter query. For example, suppose that we monitor queries for *booter.com*. A DNS query is then performed for *example.booter.com*. The prototype will store *example.booter.com*, but also *booter.com*. However, the ground truth will only contain queries for *booter.com*. In this situation, Experiment (b) matches this query, but Experiment (a) does not.

To gain more insight into how Booter queries appear in SURFnet's constituency, we aggregated the cumulative results per organization type. Remember that the organization type is for example a university or academic hospital. A visual representation of the aggregated data is depicted in Figure 6.8. The top-five organization types are displayed on the horizontal axis, including the number of organizations that are aggregated in the type. The type names are redacted for confidentiality. On the vertical axis, the number of Booter query occurrences for a type is displayed. Looking at the results, we see that one type of organizations, type $A$, performs significantly more Booter queries than others. The number of queries performed by organizations of other types is lower. Furthermore, these numbers lie closer to each other. The presence of a Booter query on itself does not mean anything. However, if organizations of type $A$ are targeted by DDoS attacks in the same time frame, the presence of Booter queries could suggest an inside-job. The higher number of Booter queries in organizations of type $A$ could also be a reason for an awareness campaign targeting this type of organizations. Mounting a DDoS attack against an organization in the Netherlands is considered a crime. For example, a young adult mounted DDoS attacks on banks in the Netherlands using Booters, and will be prosecuted [29] (in Dutch).

We presented the Booter query detection results to the head of SURFcert, the incident response team of SURFnet, and asked whether there have been DDoS attack incidents targeted at organizations for which Booter queries were performed. However, we did not find any matching incidents in the period covered by our validation. This is possibly due to the fact that a large part of the validation period overlaps with the summer vacation. This is discussed in more detail in Section 6.4.

#### Scenario 2: Spam Filtering

In this section, we analyze and discuss the results of the Spam Filtering validation scenario. We first performed Experiment (a) using the detection results, in which we tested the ground truth against the Bloom filter contents, and checked whether all entries from the ground truth are present in the Bloom filters. The results of this experiment are available in Table 6.2a. The first row indicates which daily Bloom filter contents were used. The second row indicates how many detected PTR queries were in the Bloom filters. The third row indicates how many were in the ground truth, and the last row indicates the difference between the second and third row.

As we can see in the last row, there is a slight difference in the numbers every day. The number of detected PTR queries that are present in the ground truth is slightly higher than the number present in the Bloom filters. This difference was unexpected, since we assumed that all queries from the mail filtering services would also end up in our Bloom filters. Therefore, we checked the results with an operator of the mail filtering service at SURFnet, and discovered the cause. As

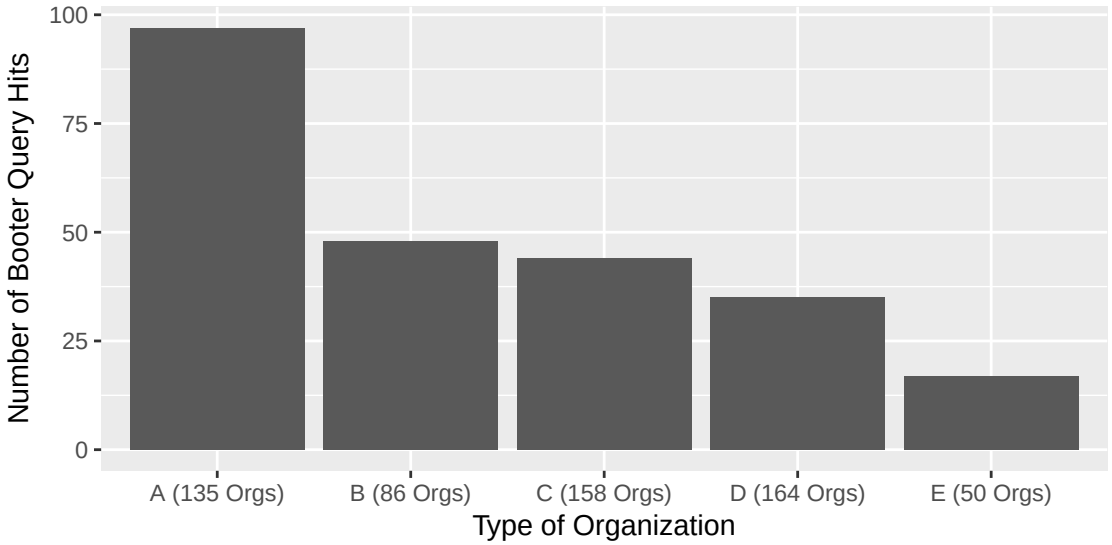
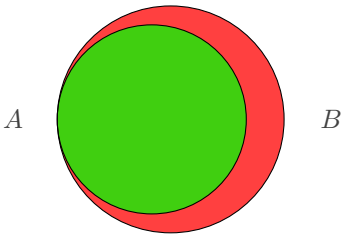



Figure 6.8: The cumulative occurrence of Booter queries in SURFnet's constituency aggregated per organization type. The types are redacted for confidentiality.

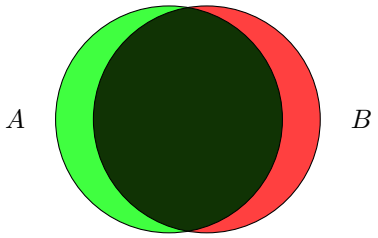
we discussed in Section 6.1, our experiment setup monitored DNS queries to a set of operational recursive name servers. The mail filtering service also uses the monitored name servers. However, they also use an additional name server that our experiment setup did not monitor. The additional name server is not queried often by the mail filtering service, and hence, the number of queries we are missing is small. The difference is hence explained by the missing data. A visual representation of the data we compared and the missing data is depicted in Figure 6.9a. We compared the ground truth $B$ to the Bloom filters $A$. The area of $B \setminus A$ represents the missing DNS queries from the additional recursive name server, as we explained above.

We performed Experiment (b) next, using the detection results. In this experiment, we tested the Bloom filter contents against the ground truth, and checked whether we encountered false positives. The results of this experiment are available in Table 6.2b. The first row again indicates which daily Bloom filter contents were used. The second row indicates how many detection results were present in the ground truth. The third row indicates how many detection results were present in the Bloom filters, and the last row indicates the difference between the second and third row. When the number of detection results that occurred in the ground truth is lower than in the Bloom filters, we likely encountered a false positive.

In these results, we again see a slight difference in the numbers every day. The number of detected PTR queries that are present in the ground truth is slightly higher than the number present in the Bloom filters. We analyzed the difference between the ground truth and the Bloom filter contents, and identified the cause as follows. As we discussed in Section 5.1, the Honas prototype implements entity identification by network segment. When a DNS query is received, the source address is matched to an IP prefix. This prefix is associated to an organization, that is identified by its name. Upon a successful match, the name of the associated organization is stored, rather than the actual source address. The mail filtering service only uses a small subset of IP addresses from the total address space associated to the organization *SURFnet*. However, the prototype only stores the name *SURFnet* for every query it receives from the mail filtering service. The organizational network of *SURFnet* contains more users and services than only the mail filtering service. As a result, we are unable to tell which user or service performed the query, and therefore, we cannot decide whether the missing queries are false positives or not. A visual representation of the data we compared and the missing data is depicted in Figure 6.9b. We compared the Bloom filters $A$ to the ground truth $B$. The area of $B \setminus A$ represents the missing




(a) Venn diagram explaining the missing data in Experiment (a). The area of $B \setminus A$ represents DNS queries to the additional recursive name server that we did not monitor.




(b) Venn diagram explaining the missing data in Experiment (b). The area of $B \setminus A$ represents DNS queries to the additional recursive name server that we did not monitor, and $A \setminus B$ represents the detections missing in the ground truth.

Figure 6.9: A series of Venn diagrams explaining the differences in the results of Experiments (a) and (b). Set $A$ represents the Bloom filters, whereas set $B$ represents the ground truth. Note that the figure is illustrative and the areas are not proportional to the data size.

data from the additional recursive name server in Experiment (a). The area of $A \setminus B$ represents the DNS queries that we could not identify using the ground truth, as we explained above.

The number of detected queries in the Spam Filtering scenario is significantly higher than in the Booters scenario. Moreover, we configured the Bloom filters to have a false positive rate of $p = 10^{-3}$. The number of detected queries is larger than 1000, so it is reasonable that we encountered a few false positives. However, because of the reason discussed above, we are unable to tell whether this is actually the case.

As we discussed in Section 6.3.1, some organizations in SURFnet's constituency host their own mail services. One of the use cases for this scenario was to see whether these organizations also receive spam from blacklisted IP addresses. To do so, we categorized the detection results per organization. We found that several organizations that host their own mail services performed DNS queries for blacklisted IP addresses. A small set of organizations is responsible for a large amount of the performed DNS queries. Most organizations reside in the tail of the data distribution. Therefore, we used the Pareto principle to create an overview of the organizations performing most detected queries. These organizations cover 80% of the detection results, and are listed in Table 6.3. The organization names are redacted for confidentiality. We see that a university queries almost 10% of the blacklisted IP addresses we detected. Other organizations that follow are closer to each other in the number of queries.

#### Scenario 3: National Detection Network

In this section, we analyze and discuss the results of the National Detection Network validation scenario. In Scenario (1) and (2), we had ground truth covering all detection results. However, as we discussed in Section 6.3.1, we must perform detection on all IoCs first to obtain ground truth for Scenario (3). Remember that using the Honas prototype, we are allowed to perform detection on all IoCs, and therefore, we first tested all IoCs from the MISP against the Bloom filters. A visual representation of the number of unique detections we found on each day is depicted in Figure 6.10. We see that the number of detections is between 30 and 50 each day.

Remember that we could not plainly log DNS queries for all IoCs we retrieved from the MISP, as we discussed in Section 6.3.1. The privacy policy of the SURFnet network only allows targeted logging for a specific domain name that regularly occurs. Therefore, we had to gather queries in the Bloom filters first to determine the ground truth. We gradually checked which domain names occurred regularly in the Bloom filters. On the $3^{rd}$ day we added the first domain name to the ground truth. On the $16^{th}$ day we added another, and on the $19^{th}$ day we completed the ground truth, consisting of 5 domain names.

| Day | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bloom Filters** | 11564 | 14295 | 14834 | 15351 | 15101 | 14403 | 12467 | 10951 | 14206 | 15462 | 15362 | 15350 |
| **Ground Truth** | 11581 | 14313 | 14972 | 15500 | 15246 | 14537 | 12612 | 11099 | 14801 | 15983 | 15585 | 15458 |
| **Missed** | 17 (0.15%) | 17 (0.12%) | 138 (0.92%) | 148 (0.95%) | 145 (0.95%) | 134 (0.92%) | 144 (1.14%) | 147 (1.32%) | 594 (4.01%) | 521 (3.26%) | 222 (1.42%) | 108 (0.70%) |

| Day | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bloom Filters** | 15021 | 11477 | 10272 | 14077 | 15100 | 15549 | 15275 | 14507 | 11075 | 9048 | 12402 |
| **Ground Truth** | 15171 | 11819 | 10559 | 14123 | 15395 | 16194 | 15864 | 14726 | 11185 | 9294 | 12570 |
| **Missed** | 148 (0.98%) | 342 (2.89%) | 287 (2.72%) | 46 (0.33%) | 295 (1.92%) | 644 (3.98%) | 589 (3.71%) | 219 (1.49%) | 110 (0.98%) | 246 (2.65%) | 168 (1.33%) |

(a) Results for Experiment (a) in the Spam Filtering scenario.

| Day | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Ground Truth** | 18713 | 23312 | 24157 | 24802 | 24774 | 23508 | 20411 | 17709 | 23009 | 24873 | 24987 | 24796 |
| **Bloom Filters** | 18842 | 23431 | 24355 | 24939 | 24984 | 23661 | 20559 | 17844 | 23120 | 24985 | 25103 | 24962 |
| **Missed** | 129 (0.68%) | 119 (0.51%) | 198 (0.81%) | 137 (0.55%) | 210 (0.84%) | 153 (0.65%) | 148 (0.72%) | 135 (0.76%) | 111 (0.48%) | 112 (0.45%) | 116 (0.46%) | 166 (0.67%) |

| Day | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Ground Truth** | 23911 | 17961 | 16355 | 22866 | 24444 | 24568 | 24339 | 23424 | 17710 | 14395 | 19946 |
| **Bloom Filters** | 24046 | 18078 | 16494 | 23013 | 24562 | 24722 | 24498 | 23594 | 17855 | 14513 | 20006 |
| **Missed** | 135 (0.56%) | 117 (0.65%) | 139 (0.84%) | 147 (0.64%) | 118 (0.48%) | 154 (0.62%) | 159 (0.65%) | 170 (0.72%) | 145 (0.81%) | 118 (0.81%) | 60 (0.30%) |

(b) Results for Experiment (b) in the Spam Filtering scenario.

Table 6.2: Results for Experiments (a) and (b) in the Spam Filtering validation scenario.

Using the ground truth and detection results, we first performed Experiment (a), in which we tested the ground truth against the Bloom filter contents, and checked whether all entries from the ground truth are present in the Bloom filters. The results of this experiment are available in Table 6.4a. The first row indicates which daily Bloom filter contents were used. The second row indicates how many detected queries were present in the ground truth. The last row indicates how many detected queries were present in the Bloom filters. As we can see, all queries from the ground truth appeared in the Bloom filters. Therefore, the prototype performed as expected.

We performed Experiment (b) next, using the detection results. In this experiment, we tested the Bloom filters against the ground truth. We checked every IoC from the MISP against the Bloom filters. If a detected domain name is not in the ground truth, and neither in the list of IoCs from the MISP, we encountered a false positive. The results of this experiment are available in Table 6.4b. The first row again indicates which daily Bloom filter contents were used. The second row indicates how many detection results were present in the Bloom filters. The last row indicates how many detection results were present in the ground truth. When the number of detection results that occurred in the ground truth is lower than in the Bloom filters, we likely encountered a false positive. We see that there is no difference in the numbers in the second and last row, which means we did not encounter a false positive.

In the results of Scenario (1), we discussed the difference in numbers between Experiment (a) and (b). The same difference applies to Table 6.4a and 6.4b. On the days where the numbers differ, a DNS query was performed for a subdomain of a domain name that is associated to an IoC. These detections are matched by Experiment (b), but not by Experiment (a).

When we collected queries in the Bloom filters, we looked for regularly occurring domain names, but also for notable ones, such as domain names that look random. These domain names stand out because they look like they are generated by a DGA, and DGAs are often used by malware. One of the most notable domain names that we encountered was *iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com*. When we looked it up in the MISP, we found that it is associated with the WannaCry ransomware. To find out more about where this domain name actually occurs, we added it to the ground truth. We then found out that it was queried on intervals of 5 minutes and 5 seconds, by a single IP-address. A threat report on the WannaCry ransomware

| Organization | Number of Queries |
|---|---|
| A (University) | 1664 |
| B (University of Applied Sciences) | 531 |
| C (University) | 486 |
| D (Vocational Education) | 478 |
| E (Vocational Education) | 475 |
| F (Research Institution) | 464 |
| G (Teaching Hospital) | 461 |
| H (Educational Service Provider) | 425 |
| I (Educational Service Provider) | 422 |
| J (Vocational Education) | 402 |
| K (Educational Service Provider) | 308 |

Table 6.3: The organizations that perform 80% of the DNS queries for blacklisted IP addresses. The organization names are redacted for confidentiality.

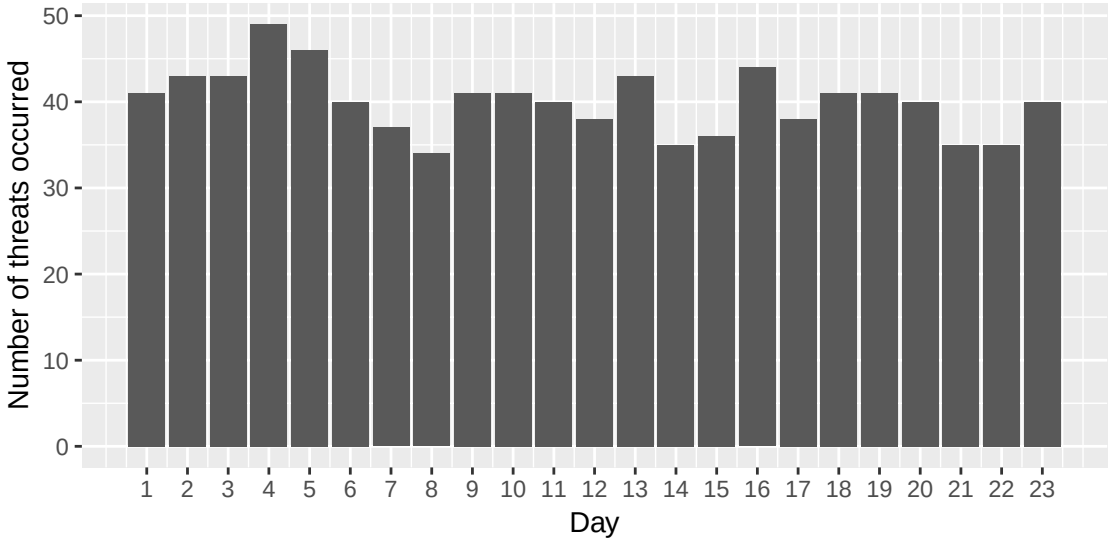



Figure 6.10: A visual representation of the number of unique National Detection Network detections that occurred on each day of the validation.

by FireEye [28] states that this domain name is used as a killswitch. Once the domain name successfully resolves to an IP-address, the ransomware kills itself. This detected threat looks like a single infection. Moreover, since it looks like the killswitch is activated, the user in question may even be oblivious about the infection. We reported the infection to SURFcert, as it is considered a security incident. SURFcert can then decide whether to inform the organization or not.

The WannaCry incident shows the operational aspect of the IoC detection system, rather than the strategical and tactical. For these aspects, we can make an overview containing information about which threats occur most often. We cannot count the number of occurrences. We only know whether a domain name was queried on a specific hour or day. However, if a domain name keeps occurring on a daily basis, we know that the associated IoC (and hence threat), does too. Therefore, we created a threat overview, counting the number of days that a domain name occurred in the Bloom filters during the validation. The largest part of unique threats resides in the tail of the data distribution, and therefore, we again use the Pareto principle to create the threat overview. The overview covers 80% of the threats that occurred, and is depicted in Figure

| Day | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Ground Truth** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 1 | 1 | 4 | 4 | 4 | 4 | 2 |
| **Bloom Filters** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 1 | 1 | 4 | 4 | 4 | 4 | 2 |

(a) Results for Experiment (a) in the National Detection Network scenario.

| Day | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bloom Filters** | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 5 | 5 | 5 | 5 | 5 |
| **Ground Truth** | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 5 | 5 | 5 | 5 | 5 |

(b) Results for Experiment (b) in the National Detection Network scenario.

Table 6.4: Results for Experiments (a) and (b) in the National Detection Network validation scenario.

6.11. We aggregated the information by TLP (Traffic Light Protocol)[4] level, because some IoCs from the National Detection Network are confidential. The TLP is a widely known protocol in the security community that identifies the level of confidentiality in sharing information. The TLP features four levels, that also apply to IoCs shared in the National Detection Network.

- **White:** Disclosure is not limited.
- **Green:** Disclosure is limited to the security community.
- **Amber:** Disclosure is limited to organization.
- **Red:** Disclosure is not allowed.

The fourth level, TLP *Red* is not displayed in Figure 6.11 because no such threats occurred. We see that the threats that occurred are divided in the TLP levels *Amber*, *Green* and *White*. The IoCs shared under TLP White and Green are often public, and shared among worldwide security communities. However, IoCs under TLP Amber or Red are highly confidential, and considered very serious when occurring on the network.

Using the threat overview, we found that the Mirai botnet remains active, composing a large part of the *Green* TLP level. It occurred regularly, and we considered it a serious threat. Therefore, we added it to the ground truth to find out which IP-addresses perform the queries. We found that a single IP-address is performing these queries. The system behind that IP-address happens to be a proxy for the Tor anonymization network, meaning that the Mirai DNS query was performed by an arbitrary user on the Internet. This threat overview covers the data we collected during the validation. However, a threat overview that covers months of collected data would give a much clearer view on the threat landscape. Using this information, security officials in SURFnet's constituency get a better view of which threats to focus on in their organizations. Moreover, it better depicts changes in the threat landscape over time, better preparing security officials in what measures to take.

[4] https://www.first.org/tlp/

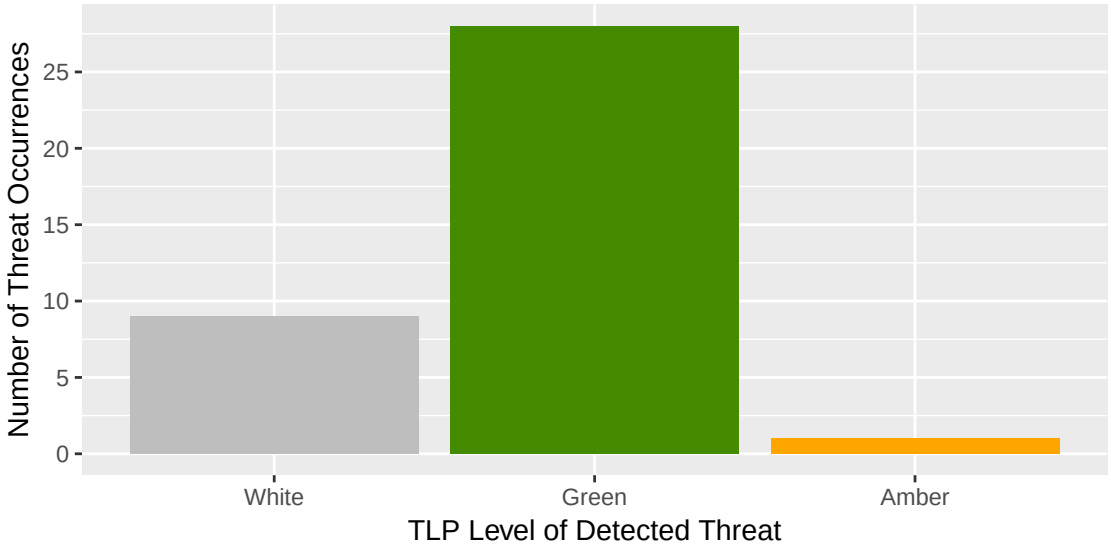



Figure 6.11: A threat overview counting the number of days a domain name occurred in the Bloom filters. The information is aggregated and threat type for confidentiality.

## 6.4 Discussion

In Section 6.3.2, we presented the results of the Honas prototype validation. The results suggest that the Bloom filter based IoC detection method that we developed is useful in practice. In this section, we reflect on the performance of the prototype and discuss its usefulness in practice.

We created a prototype for threat detection using Bloom filters, and validated this prototype using three real-world scenarios. The prototype is based on existing software, which we extended to fit an Internet Service Provider context. The validation was performed using real-world data from the operational recursive name servers at SURFnet. Our implementation allows organizations to perform threat detection easily using operational DNS queries, while preserving the privacy of users. This is very important, because without our implementation, organizations would have little to no insight into the threats their users are facing. An example of immediate impact of this project to business operations is given by SURFcert.

The CERT team primarily focuses on the operational aspect of security, responding to security incidents in SURFnet's constituency. Whenever we detect an IoC using the Honas prototype, we only know in which organization it was detected. If we want to know exactly which IP-address triggered the detection, we must monitor the actual DNS queries. SURFcert usually informs an organization about an incident with information containing an exact IP-address. This information is not known from detection using the Honas prototype, and therefore, SURFcert cannot directly use this information. However, SURFcert currently does not have insight in which IoCs from the National Detection Network occur. Using the Honas prototype, SURFcert can gain this insight. When a threat is detected, and targeted investigation is performed, the actual IP-address can be found out. From that point, the detection results become useful on an operational level. A good example of this is the WannaCry detection we discussed in the validation results of Scenario (3).

The targeted investigation can also be automated. For example, one can perform detection on all IoCs in the MISP on a daily basis, and pick all detected IoCs with TLP level *Amber* and *Red* for targeted investigation. As we discussed in Section 6.3.2, the Mirai detections were categorized in TLP level *Green*. Since Mirai has been around for two years, it is logical that many parties have access to the corresponding security intelligence. However, it is reasonable to assume that the IoCs for Mirai have been categorized as *Amber* or *Red* when Mirai first appeared.

We now zoom in on the individual validation scenarios. Scenario (1) was already shown to

be useful in [33], and therefore, we already knew what to expect from this scenario. We did not manage to link DDoS incidents to Booter queries during the validation period. However, since the validation was performed during the summer vacation, this is not unexpected. Students are on vacation, and not using the organizational network. As a result, DDoS attacks that occur during the prototype validation are unlikely to be related to Booter queries from inside the organizational network. Moreover, students may still have used Booters to order a DDoS attack on a different target. Booters originate from the gaming scene, as *booting* means getting a player kicked out of an online game. Therefore, a Booter may have been used for that purpose as well. In the end, a DNS query also does not automatically imply a website visit.

In contrast to Scenario (1), Scenario (2) and (3) have never been experimented with before. We looked at PTR queries for blacklisted IP-addresses in Scenario (2). Using the validation results of this scenario, we can broaden the threat overview for the mail filtering service. Using the ground truth from the mail filtering service, we could only make an overview of the threat landscape for organizations that use the service. However, organizations that do not use the service would not be covered. Using the Honas prototype and the validation results, we could include the organizations not using the mail filtering service in the threat overview. On an operational level, Scenario (2) is useful too. Suppose that we found that a malicious spam run managed to get past the mail filter, meaning that the mail was delivered to the organizations. We know which organizations use the service, so we could warn them about the spam easily. However, we cannot determine if organizations not using the service also received the spam mail. Using the Honas prototype and the validation results, we could easily check which organizations not using the mail filtering service probably received the same spam mail, and warn them about the spam as well. In the end, Scenario (2) is useful to broaden existing threat overviews.

Scenario (3) was certainly useful, as it helped us detect threats in the network that we previously were not able to. Remember that evidence of detection was required to monitor actual DNS queries for the WannaCry domain name we discussed in the validation results of Scenario (3). If we did not have the evidence from the Bloom filters, we would not have found out about it. Furthermore, we managed to retrieve an overview of regularly occurring threats easily. This information contributes to a global threat overview, providing insight in the threat landscape in SURFnet's constituency. We discussed the validation results with somebody at SURFnet who is involved in writing the annual report on the threat landscape in SURFnet's constituency. Especially the validation results of Scenario (3) were considered useful for the report. The report is aimed at the strategical and tactical level, which was the goal for this thesis.

### Open-Source

Only a fraction of the IoC detection methods that Dodopoulos described in [16] were published as open-source. SURFnet and the IETF dprive working group care about improving privacy in DNS. Everybody should be able to use the solution we propose, and hence, the source code of the Honas prototype is open-source. Furthermore, the software is written such that it integrates with all major DNS resolver packages. It should be easy to get started and use the software. Furthermore, because the software is open-source, the research results in this thesis are reproducible. The source code of the Honas prototype can be found at `https://github.com/SURFnet/honas`.

### Reflecting on the Goals

More generally, this thesis project aimed at fulfilling five goals as discussed in Chapter 4. We proceed to discuss how the goals we set are fulfilled.

**G1: The privacy of individual users in the network must be preserved.** The prototype we built and validated is based on Bloom filters, which provide strong privacy guarantees. As we discussed in Section 2.5.3, we are unable to enumerate the stored DNS queries, and we can only look for exact domain names. Furthermore, it is not possible to make correlations between queries

stored in the Bloom filters. Therefore, the prototype satisfies Goal **G1**, which was the primary goal for this thesis.

**G2: We want to be able to detect the presence of a threat in the network using existing IoCs, and estimate the time of occurrence.** The prototype features a data collection and searching tool, allowing us to look for arbitrary lists of IoCs. These tools were tested and validated using the three scenarios described in Section 6.3.1. The Bloom filters in which incoming DNS queries are stored are aggregated per configurable time period, allowing us to estimate the time of occurrence for a threat up to that level. Therefore, the prototype satisfies Goal **G2**.

**G3: We want to be able to identify the network segment in which a threat is detected.** The data collection tool in the prototype features a subsystem that maps the source address of an incoming DNS query to the entity associated to that network segment. That entity is stored with the query, such that we can look up that combination later. Therefore, the prototype satisfies Goal **G3**.

**G4: We want to be able to aggregate information in the detection system over time.** The prototype features a Bloom filter combination tool, which allows us to aggregate the information in two similarly configured Bloom filters. This saves storage space for long-term storage. The combination of the privacy guarantees provided by the Bloom filters, and the low storage space requirements, result in the fact that we can easily store the data for a long time. Furthermore, the dry-run takes aggregation over time into account by suggesting parameters for hourly and daily Bloom filters. Therefore, the prototype satisfies Goal **G4**.

**G5: Operationalize data collection and storage in compliance with current regulation.** Finally, we wanted to create a threat detection method that operationalizes data collection and storage in compliance with the GDPR. The data is stored in Bloom filters, which are not meaningful without knowing their contents. Moreover, they do not contain personally identifiable information. As a result, we can store the Bloom filters for a long time, and therefore, the prototype satisfies Goal **G5**.

### 6.4.1 Future Venues for Improvement

In Section 6.4, we discussed the results of the validation. We found that we can indeed build a prototype that fulfils the goals we set, and that the prototype is useful in practice. In this section, we suggest future venues for improvement.

**Spam Filtering** An interesting future scenario would be to look at blacklisted URLs, as spam or phishing mails contain such URLs. If the mail filtering service uses URL blacklists too, ground truth for these would be available as well. If we detect a DNS query for a domain name in a blacklisted URL, it may be the case that a user opened the mail and clicked on the link. That information is useful because it indicates how successful a spam or phishing campaign is, which organizations are targeted, and which users click the most. Moreover, it adds knowledge to the mail filtering service, because the service is not able to test whether users clicked the link. However, monitoring all URLs or DNS queries is too privacy invasive. Using the prototype, however, this is possible.

We consider an example for clarity. The mail filtering service looks at the contents of incoming mail, and tests the contents against URL blacklists. Based on whether the contents triggered a blacklist hit, a score value is added to the mail header. This score is used by the mail application of users to indicate whether a mail is spam. Part of the contents of real-world example phishing mail is available in Listing 6.2. On the first two lines, we see that the mail is marked as spam, and a score is added to the mail headers. How the score is handled depends on the mail server and client configuration. Further down, we see an URL containing the domain name

*www.fashionatingworld.nl*. If the mail filtering service finds this URL in a blacklist, and stores ground truth for it, we can test the URL against the Bloom filters to find whether somebody performed a DNS query for the domain name. Such query suggests that a user may have clicked on the link.

```
X-Spam-Flag: YES
X-Spam-Score: 10.51
.....
<font size="2"><a href="http://www.fashionatingworld.nl/link.php?M=873509&amp;N
    =607&amp;L=6514&amp;F=H" target="_blank" data-saferedirecturl="https://www.
    google.com/url?hl=en-GB&amp;q=https://adsu.link/o/ZewIxk&amp;source=gmail&amp;
    ust=1532004406138000&amp;usg=AFQjCNH-ybDdMq3xpk1chTYbyDWf_6Mh0A">peering@as1101
    .net</a>?</font></span></div>
<span style="font-size: x-small; font-family: courier new, courier;"> <font size="2
    "> </font>
......
```

Listing 6.2: The contents of a real-world example of a phishing mail.

**Parameter Estimation** During the parameter estimation process, we used the highest number of DNS queries occurred over time as input. This worked fine for the validation of the prototype. However, the parameters are estimated using data from a short period of time, usually a week. When the number of queries changes over time, the parameters may be insufficient. Using data from a short period of time, it is hard to predict whether the estimated parameters will be sufficient later. An option is to calculate a confidence interval in the parameter estimation, rather than using the maximum over time. That way, one can choose the parameters such that they will suffice later with high confidence. Furthermore, the prototype could continuously estimate and adjust the parameters automatically in production. This reduces management requirements, but one will have to accept that the Bloom filters cannot be aggregated anymore once the parameters change.

**Privacy Quantification** Bloom filters are a new technique for privacy preservation in the context of IoC detection. We derived that Bloom filters provide strong privacy guarantees based on their properties. However, we did not formally quantify the privacy guarantees of Bloom filters. Bianchi et al. [7] introduced a method to quantify the privacy of Bloom filters. However, their application is different from ours, and privacy quantification is not trivial. Therefore, quantification of the privacy guarantees of Bloom filters is out of the scope of this project. However, for legislation and more formal applications of Bloom filters for IoC detection, privacy quantification may be required.

**Threat Detection Applications** This thesis focuses on IoC detection using DNS. However, many more data sources exist. With slightly different goals and a different data source, the IoC detection method we introduced may be useful as well. For example, in Section 6.4, we discussed using URL blacklists in Scenario (2). Furthermore, in the survey by Geravand et al. [18], Bloom filter applications for DDoS and anomaly detection were discussed.

### 6.4.2 Limitations

The validation of the prototype was performed during a period in which the summer vacation for schools and universities started. As a result, the number of DNS queries that the prototype received every second gradually decreased as the validation was in progress. A visual representation of the decreasing query volume is depicted in Figure 6.12. In the graph, we see that the number of queries at the end of June was a little less than 8000 per second. When we look at the end of July, the number of queries has dropped to under 4000 per second. The baseline however, stays at approximately 2000 queries per second the whole time. Overall, the volume of queries gradually decreased as the summer vacation started, and students stopped using the university network. This is a limitation for the validation, because the number and type of threats occurring

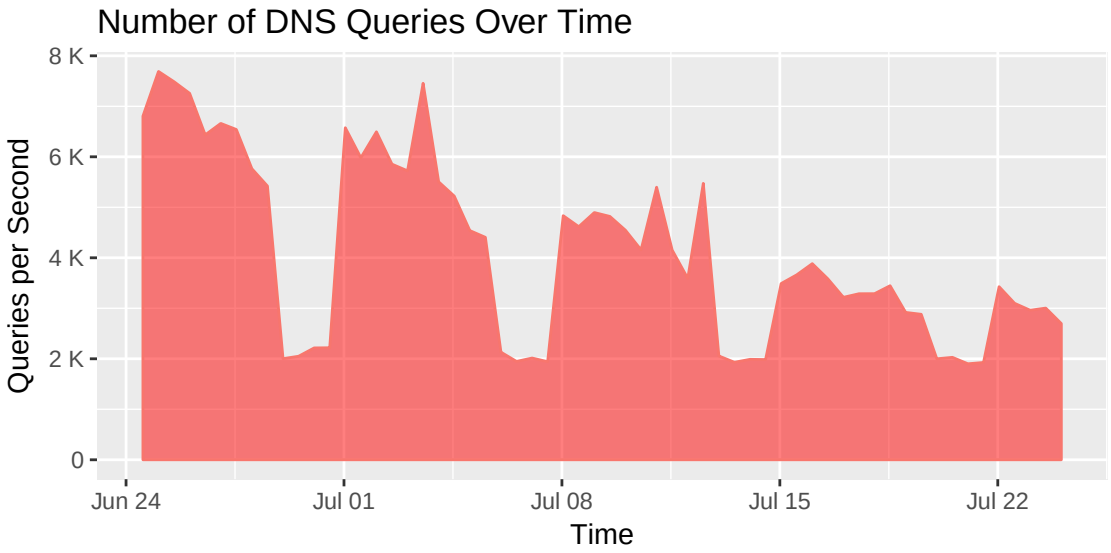



Figure 6.12: The number of queries per second over time during the validation phase.

during the vacation are likely to be different from non-vacation periods. While a more extensive validation period did not fit in the scope of this research project, we strongly suggest that DNS queries should be collected after the academic year starts.

The prototype validation used queries from operational recursive name servers at SURFnet. This way, only users that use these name servers are monitored. Users that use different name servers, such as Google's 8.8.8.8, are not monitored. However, if a few users in every organization use a different name server, threats that target the entire organization would still be visible. Therefore, this is a limitation, but it does not have a large impact on the validation. Furthermore, this limitation is specific to the validation we performed at SURFnet. Users at organizations in SURFnet's constituency are free to use a different recursive name server. On the contrary, the usage policy in an enterprise network could restrict users to the use of internal name servers. Therefore, this limitation does not always apply.

As we discussed in Section 6.2.2, we chose the Bloom filter parameters correctly for the validation. However, the dry-run results showed that on some days, the actual expected false positive rate came close to the theoretical false positive rate. Since the validation was performed during a vacation period, we do not know whether the Bloom filter parameters are also sufficient for non-vacation periods. To be sure, a new dry-run should be performed when deploying the prototype during non-vacation periods. On the other hand, the parameters should also not be chosen too large. When Bloom filters are stored for a long time, the amount of data that is unnecessarily stored grows.

# Chapter 7

# Conclusions

Monitoring all DNS queries for threat detection is very privacy invasive. Therefore, we introduced a solution in this thesis that enables us to perform threat detection providing privacy guarantees. We found that Bloom filters are well suited for this, as its properties offer strong privacy guarantees. However, Bloom filters also introduce challenges and limitations in threat detection. In Chapter 3, we discussed the existing DNS based IoC detection methods analyzed by Dodopoulos [16], and explained whether their goals could be achieved using Bloom filters. In this analysis, we found that the goals of existing work could not be achieved using Bloom filters. The scope in which Bloom filters can be used is limited to testing existing IoCs against collected data.

We set the goals for our Bloom filter based IoC detection method in Chapter 4. To fulfil those goals, we designed and developed a prototype based on existing software. We first performed a *dry-run* to configure the prototype, as we discussed in Section 6.2. Furthermore, we validated the prototype using three real-world scenarios, which were discussed in Section 6.3.1. We first looked at Booter queries, which could be related to DDoS attacks. We also looked at blacklist hits from the spam filtering service at SURFnet, and finally, we tested high quality IoCs from the National Detection Network against the DNS queries on the network. In the validation of the prototype, we used real-world data from the operational recursive name servers at SURFnet. The scenarios tested how the prototype performs. The results showed that the prototype performs as expected.

We also discussed whether the prototype and validation results are useful in practice. Our aim was to introduce a threat detection method that is useful on a strategical and tactical level. The validation results showed that the threat detection method we introduced is certainly useful on a strategical and tactical level. The results of Scenario (1) help to identify inside-jobs in DDoS attacks, and contribute to a global threat overview. Scenario (2) and (3) have never been experimented with before. The prototype enabled us to experiment with them. In Scenario (2), we broadened the threat overview of the mail filtering service, to include threats in organizations that do not use the service. The detection results of Scenario (3) showed that we would miss tens of detections every day without the prototype we developed. Furthermore, with the WannaCry incident we discussed in the validation results, we showed that the prototype features an easy starting point in identifying infected users. Using these results, we also argued that the validation results are also useful for CERTs (Computer Emergency Response Teams), on an operational level. However, on an operational level, more effort is required to perform detection, as one must first conduct targeted investigation on a detected threat.

# Bibliography


[1] Charu C Aggarwal. On K-anonymity and the Curse of Dimensionality. *Proceedings of the 31st International Conference on Very Large Data Bases*, pages 901–909, 2005. 18

[2] Kamal Alieyan, Ammar Almomani, Ahmad Manasrah, and Mohammed M. Kadhum. A survey of botnet detection based on DNS, 2017. 8

[3] Olivia Angiuli, Joe Blitzstein, and Jim Waldo. How to de-identify your data. *Queue*, 13(8):20–39, 2015. 18

[4] Manos Antonakakis, Tim April, Michael Bailey, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, Yi Zhou, Manos Antonakakis Tim April Michael Bailey Matthew Bernhard Elie Bursztein, Bullet J Jaime Cochran Zakir Durumeric Alex Halderman Luca Invernizzi, Bullet Michalis Kallitsis Deepak Kumar Chaz Lever Zane Ma, Joshua Mason Damian Menscher, Bullet Chad Seaman Nick Sullivan Kurt Thomas, and Bullet Yi Zhou. Understanding the Mirai Botnet. In *Proceedings of the 26th USENIX Security Symposium*, pages 1093–1110, 2017. 8

[5] Manos Antonakakis and Roberto Perdisci. From throw-away traffic to bots: detecting the rise of DGA-based malware. In *Proceedings of the 21st USENIX Security Symposium*, page 16, 2012. ix, 9, 17

[6] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. Building a Dynamic Reputation System for DNS. *USENIX Security'10: Proceedings of the 19th USENIX conference on Security*, pages 1–17, 2010. 17

[7] Giuseppe Bianchi, Lorenzo Bracciale, and Pierpaolo Loreti. Better than nothing privacy with bloom filters: To what extent? In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7556 LNCS, pages 348–363, 2012. 52

[8] Leyla Bilge, Sevil Sen, Davide Balzarotti, Engin Kirda, and Christopher Kruegel. EXPOSURE: a passive DNS analysis service to detect and report malicious domains. *ACM Transactions on Information and System Security (TISSEC)*, 16(4):14, 2014. 17

[9] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970. 1, 10

[10] Bart Bosma. Cyberdreigingsbeeld 2017. Technical report, SURFnet, Utrecht, 2017. 7, 8, 41

[11] Hyunsang Choi, Heejo Lee, and Hyogon Kim. BotGAD: detecting botnets by capturing group activities in network traffic. *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*, pages 1–8, 2009. 17

[12] Ken Christensen, Allen Roginsky, and Miguel Jimeno. A new analysis of the false positive rate of a Bloom filter. *Information Processing Letters*, 110(21):944–949, 2010. 12, 13, 38

[13] W. B. de Vries, R. van Rijswijk-Deij, P. T. de Boer, and A. Pras. Passive Observations of a Large DNS Service: 2.5 Years in the Life of Google. *Network Traffic Measurement and Analysis Conference (TMA 2018), Vienna, Austria, 26-29 June 2018*, page 8, 2018. 40

[14] Sara Dickinson. DNS Privacy - The Problem, 2018. 1, 9

[15] Sara Dickinson, Roland Van Rijswijk-Deij, and A Mankin. Recommendations for DNS Privacy Service Operators. 2018. 1

[16] Romanos Dodopoulos. *DNS-based Detection of Malicious Activity*. PhD thesis, Eindhoven University of Technology, 2015. 1, 4, 5, 6, 17, 21, 50, 55

[17] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. In *Conference on Analysis of Algorithms AH*, pages 127–146, 2007. 31

[18] Shahabeddin Geravand and Mahmood Ahmadi. Bloom filter applications in network security: A state-of-the-art survey. *Computer Networks*, 57(18):4047–4064, 2013. 20, 52

[19] Thomas Gerbet, Amrit Kumar, and Cedric Lauradoux. The Power of Evil Choices in Bloom Filters. In *Proceedings of the International Conference on Dependable Systems and Networks*, volume 2015-Septe, pages 101–112, 2015. 13, 15

[20] Michael Gertz and Madhavi Gandhi. *Handbook of Database Security*. 2008. 18

[21] Munawar Hafiz. A collection of privacy design patterns. In *Proceedings of the 2006 conference on Pattern languages of programs - PLoP '06*, page 1, 2006. 10

[22] Dominik Herrmann, Christian Banse, and Hannes Federrath. Behavior-based tracking: Exploiting characteristic patterns in DNS traffic. *Computers and Security*, 39(PARTA):17–33, 2013. 9

[23] PhD Hina Vaghashia, Amit Ganatra. A Survey: Privacy Preservation Techniques in Data Mining. *International Journal of Computer Applications (0975 8887)*, 119:7, 2015. 18

[24] IANA. Domain Name System (DNS) Parameters, 2018. 7

[25] Himanshu Kumar, Sudhanshu Kumar, Remya Joseph, Dhananjay Kumar, Sunil Kumar Shrinarayan Singh, Praveen Kumar, and Himanshu Kumarr. Rainbow table to crack password using MD5 hashing algorithm. In *2013 IEEE Conference on Information and Communication Technologies, ICT 2013*, pages 433–439, 2013. 19

[26] Le Phuoc Tho. *DNSSEC Policies in The Wild*. PhD thesis, Eindhoven University of Technology, 2017. 3

[27] Samuel Marchal, Jerome Francois, Cynthia Wagner, Radu State, Alexandre Dulaunoy, Thomas Engel, and Olivier Festor. DNSSM: A large scale passive DNS security monitoring framework. In *Proceedings of the 2012 IEEE Network Operations and Management Symposium, NOMS 2012*, pages 988–993, 2012. 17

[28] John Miller and David Mainor. WannaCry Ransomware Campaign: Threat Details and Risk Management, 2017. 47

[29] Huib Modderkolk. In gesprek met Jelle S. (18), die met ddos-aanvallen de Belastingdienst en banken zou hebben platgelegd, feb 2018. 43

[30] National Cyber Security Centre. Indicators of Compromise, 2017. 6

[31] T Reddy, D Wing, and P Patil. DNS over Datagram Transport Layer Security (DTLS). 2017. 9

[32] S. Farrell; H. Tschofenig. Pervasive Monitoring Is an Attack. 2014. 9

[33] Jose Jair Santanna, Roland Van Rijswijk-Deij, Rick Hofstede, Anna Sperotto, Mark Wierbosch, Lisandro Zambenedetti Granville, and Aiko Pras. Booters - An analysis of DDoS-as-a-service attacks. In *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM 2015*, pages 243–251, 2015. 7, 39, 50

[34] Chen Sisheng, Xu Li, and Chen Zhide. Secure anonymous routing in trust and clustered wireless ad hoc networks. In *Proceedings of the Second International Conference on Communications and Networking in China, ChinaCom 2007*, pages 994–998, 2008. 20

[35] Jonathan M Spring and Carly L Huth. The Impact of Passive DNS Collection on End-user Privacy. *Securing and Trusting Internet Names: SATIN*, 2012. 5

[36] Uri Sternfeld. Dissecting Domain Generation Algorithms Eight Real World DGA Variants. Technical report, Cybereason, 2016. 23

[37] Changhua Sun, Bin Liu, and Lei Shi. Efficient and low-cost hardware defense against DNS amplification attacks. In *GLOBECOM - IEEE Global Telecommunications Conference*, pages 2062–2066, 2008. 20

[38] LATANYA SWEENEY. k-ANONYMITY: A MODEL FOR PROTECTING PRIVACY. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002. 18

[39] Roland van Rijswijk. *Improving DNS Security A Measurement-Based Approach.* 2017. 3, 5, 6

[40] Roland van Rijswijk-Deij, Matthijs Bomhoff, and Ralph Koning. Let a Thousand Filters Bloom. 2017. 14

[41] Roland Van Rijswijk-Deij, Mattijs Jonker, Anna Sperotto, and Aiko Pras. A High-Performance, Scalable Infrastructure for Large-Scale Active DNS Measurements. *IEEE Journal on Selected Areas in Communications*, 34(6):1877–1888, 2016. 5

[42] Matthaus Wander, Lorenz Schwittmann, Christopher Boelmann, and Torben Weis. GPU-based NSEC3 hash breaking. In *Proceedings - 2014 IEEE 13th International Symposium on Network Computing and Applications, NCA 2014*, pages 137–144, 2014. 19

[43] Florian Weimer. Passive DNS Replication. *Analysis*, 2005. 5

[44] Jeff Yan and Pook Leong Cho. Enhancing collaborative spam detection with bloom filters. In *Proceedings - Annual Computer Security Applications Conference, ACSAC*, pages 414–425, 2006. 20

[45] P. Hoffman Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels. Specification for DNS over Transport Layer Security (TLS). 2016. 9

[46] Danyu Zhu and M Mutka. Sharing presence information and message notification in an ad hoc network*, 2003. 20

# Appendix A

# TNC18 Conference Poster

A poster of the subject researched in this thesis was presented at TNC18, an international networking conference featuring interesting networking subjects in the field. The abstract and poster that was presented is depicted below. It can also be found online at https://tnc18.geant.org/core/poster/30.

**Abstract**

Almost every activity on the Internet starts with a DNS query, sometimes even multiple. Those queries reveal a lot of information about user activity. An entire working group in the IETF, called dprive, is dedicated to improving privacy in DNS. However, DNS is also a very useful tool in security monitoring. Like many network operators, it is increasingly important for SURFnet to gain insight into the threats its users are facing. However, security and privacy are often considered to be mutually exclusive. Previous work introduced a novel, privacy-friendly solution of detecting threats using DNS, based on Bloom filters. Bloom filters are sets with a statistical nature. They are non- enumerable, and it is only possible to ask whether an exact DNS query was performed. While this previous work showed that applying Bloom filters for threat detection is theoretically possible, practical aspects were not covered. In contrast, in this work we focus on the practical aspects of applying Bloom filters. Because of their statistical nature, Bloom filters introduce constraints in threat detection. How should the threat detection system be designed to work within these constraints? The poster will show how we apply Bloom filters in practice. This includes, for example, how to size Bloom filters, what the impact of false positives can be and what information to store in them. Furthermore, we validate the applied threat detection solution against three real- world scenarios. This work aims to be a starting point for other organizations to start using Bloom filters in similar scenarios.

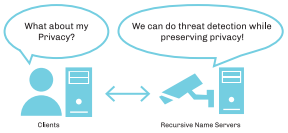
# PRIVACY FRIENDLY

## Threat Detection Using DNS

Gijs Rijnders / gijs.rijnders@surfnet.nl

### MOTIVATION

DNS is a useful tool in threat detection. However, monitoring DNS activity is very privacy infringing. We can perform threat detection using DNS while preserving the privacy of users.




General Data Protection Regulation

### Solution: Bloom Filters

DNS queries are stored in a Bloom filter. We can ask the Bloom filter whether a DNS query is stored.


### BENEFITS:

• Original information is not stored • No enumeration of stored information • Only exact information can be searched for • No correlation between users and queries • Historical lookups are possible • Space–efficient storage solution

### EVALUATION OF SOLUTION

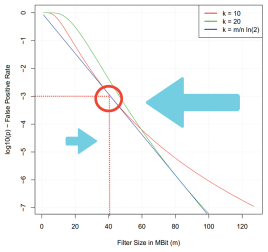
#### Bloom filters False Positives

Bloom filters introduce False Positives in detection. The FP–rate is low, and configurable using two parameters.


Image source: https://continuousimprover.com/2015/06/false-positives-and-semantic-versioning.html

#### False Positive Rate

Bloom Filter False Positive Rate: n = 3M

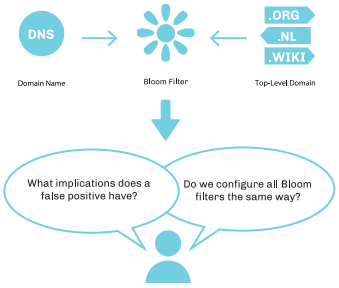



3M unique domain names per filter; False positive rate of 1/1000; Bloom filter size ~ 5 MB

Number of unique domain names is important for filter dimensions.

#### DNS storage in Bloom filters

What DNS Information to store in Bloom filters?




### NETWORK THREAT DETECTION

We obtain Intel from our community, and want to use that for detection in our network. So far we did not have a way to do so, but with the Bloom filters, we can! That brings us a step closer to proper threat detection in our network.

### BOOTERS

Students have launched DDoS attacks on their institution to have online exams cancelled. Such DDos attacks are often inside jobs. Students purchase them from so called Booter websites, offering DDoS–as–a–Service. The Bloom filters allow us to verify whether we are dealing with an inside job or not.

### SPAM FILTERING

We offer our constituency mail filtering services, which use blacklists among other things. Using Bloom filters, we can find out whether blacklisted domain names are also queried by an institution.

Acknowledgements:
My Colleagues @SURFnet | My Supervisor @TU/e: Luca Allodi | Design help: @MOME: Bence Kiss

SURFNET TU/e