Covert DNS Storage Channel Detection

Uncovering surreptitious data exchange using the phonebook of the internet

S.R.P. van Hal

TU Delft

Covert DNS Storage Channel Detection

Uncovering surreptitious data exchange using the phonebook of the internet

by

S.R.P. van Hal

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday July 14, 2021 at 16:00.

Student number:	4310055	
Project duration:	November 2019 – July 2021	
Thesis committee:	Drs. B. Vermeulen,	Ministerie van Defensie, daily supervisor
	Prof. dr. ir. R.L. Lagendijk,	TU Delft, chair
	Dr. ir. S.E. Verwer,	TU Delft, supervisor
	Dr. M. Finavaro Aniche,	TU Delft

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

TU Delft

Abstract

The cyber arms race has red and blue teams continuously at their toes to keep ahead. Increasingly capable cyber actors breach secure networks at a worrying scale. While network monitoring and analysis should identify blatant data exfiltration attempts, covert channels bypass these measures and facilitate surreptitious information extraction. The many legitimate uses and widespread availability of DNS, the “phone book” of the internet, make it an attractive protocol for such covert channels. Covert DNS storage channels encode information in the payload of outbound DNS queries.

This thesis aims to assess the effectiveness of using machine learning methods to detect covert DNS storage channels. Our literature survey identified distinct differences in *1)* algorithm type, either unsupervised anomaly detection or supervised classification, and *2)* the information source for features, either isolated DNS queries or query sequences.

We performed experiments with (Extended) Isolation Forest algorithms for anomaly detection and Random Forests for classification, combined with different feature set compositions to evaluate their relative performance. Payload-only features were derived from isolated queries and behavioral features were extracted from time-based or fixed-length sliding windows over per-domain query sequences. We evaluated our models using a large-scale corporate DNS dataset of real-world proportions and a novel dataset of connection tunneling traffic and simulated credit card exfiltration malware.

We found that the majority of experiments were able to achieve high detection rates of 98.6% or more on a variety of storage channel threats, at low false positive rates. Classification models significantly outperform anomaly detection models on threats seen during training. Evaluation on unseen threats, however, revealed that generalization is difficult, provided the limited set of training threats and showed anomaly detection models more capable at detecting a variety of threats than classification models. We furthermore showed that feature sets with a behavioral component consistently outperform payload-only features, although our experiments were inconclusive regarding the relative performance between composite feature sets.

Given the prevalence of benign storage channels misusing DNS for legitimate data transfer, we recommend rigorous filtering of training data beforehand to improve model optimization and evaluation. Furthermore, extending the malicious training set with DNS command-and-control (C2) malware is a promising future research direction to improve generalization of classification models.

Preface

I like to think that the preface is the one part of your thesis that you reread when you're older, bringing back memories and producing a melancholy (or pitiful) smile while musing about bygone times. In hindsight, everything seems easy. Looking back, however, neither takes into account other possible outcomes nor the efforts spent at finding your way towards a moving finish line.

In times where "isolation" was suddenly applicable to more than just query instances for feature extraction, I've discovered that Murphy's law is not a quip but an actual law of nature and that you're ultimately left no choice but to keep pushing and incrementally create something worthwhile.

I would first and foremost like to thank Bas, for the continuing support, symphatic ear and feedback, my close friends, for pulling me through when times were toughest, Leonie and Frank, for enabling me to continue my work as time passed by, colleagues, for much needed distractions, David and Sille, for the invaluable last-mile feedback, and finally my thesis committee, for their efforts in reviewing and assessing this work.

Rarely have I encountered a time so different from expectations than the previous year and a half. I've attempted to make the most out of a confluence of incredible events and an at times excruciating lack of support. It would, however, be foolish to take away anything from these imperfect circumstances but the valuable life lessons learned, now all that remains is to challenge the future with newfound positivity.

S.R.P. van Hal
Delft, July 2021

Mille Periculis Supersum

Contents

1 Introduction

Increasingly advanced and capable actors are active at present-day cyber battlegrounds. Secure networks are compromised at worrying scale, leading to numerous high-level data breaches and the disclosure of sensitive information. This has resulted in a need to continuously monitor and analyze network traffic in security-sensitive networks, in order to prevent exfiltration attempts. Every network packet leaves a trace, but you need to know where to look.

The Domain Name System (DNS) is the phone book of the internet [60]. It allows you to specify host names instead of IP addresses to reach other devices in a network. DNS is inextricably linked with the contemporary internet and is as such widely available. Legitimate use of DNS entails looking up resource records associated with a domain – e.g. an IP address. However, the nature of DNS and its widespread availability have attracted actors with malicious intentions.

Covert channels are a means to communicate information "in a manner that hides the fact that a communication channel is actually established at all" [36]. DNS is a facilitator of a particularly stealthy covert channel that allows for arbitrary data exchange, shrouded by benign interactions. Covert DNS channels are not a novel threat: security researchers discussed their potential as early as in 1998 [33]. However, their prevalence today makes effective detection of covert DNS storage channels as important as ever.

For instance, in 2014, U.S. retailers Sally Beauty and Home Depot announced that their payment systems had been compromised and that credit card details were stolen. In both cases, a variant of Point-of-Sale (PoS) malware *FrameworkPOS* was used to scrape information from the memory of payment terminals. While the Sally Beauty breach was limited to (reportedly) 25,000 payment details due to the malware malfunctioning [48], Home Depot reported over 56,000,000 credit card details stolen during a period of six months [47]. FrameworkPOS uses a covert DNS channel to exfiltrate stolen information [71].

Advanced actors use covert channels to surreptitiously communicate with and control compromised devices. An advanced persistent threat (APT) is "an adversary that possesses sophisticated levels of expertise and significant resources which allow it to create opportunities to achieve its objectives by using multiple attack vectors (e.g., cyber, physical, and deception)" [20]. In 2019, APT *WINNTI GROUP* was observed using an open source DNS connection tunneling tool for covert communication [84]. Covert DNS channels are still used in cyber campaigns as of today [32].

The covert DNS *storage* channels used in these scenarios encode (or *store*) outbound information in the DNS query. Optional downstream information is received via DNS responses. Queries to an appropriately configured domain are resolved through an attacker-controlled DNS server that, as a result, receives the encoded information. DNS provides for an attractive covert channel, because at no point a direct connection is established between victim and attacker devices: queries are resolved

via a (trusted) intermediate server.

Given the ever increasing amounts of network traffic produced in enterprise networks, manual inspection of DNS traffic to uncover possible threats is infeasible. Moreover, detecting known threats is insufficient given the diversity of storage channel implementations. Current research is therefore focused at developing automated and generalizable methods, aided by the rise of machine learning in the field of cyber security.

In recent years, there have been proposed many different methods to describe and classify DNS traffic using machine learning. This work aims to assess and compare the effectiveness of different algorithm types and feature engineering rationales to detect a diverse set of storage channel threats, comprising open-source connection tunneling tools and data exfiltration malware.

1.1. Research Questions

This thesis is aimed at understanding the effectiveness of machine learning-based DNS storage channel detection systems. We investigate the detection capability of different algorithm types as well as (combinations of) features extracted from either isolated queries or query sequences. The main research question is therefore formulated as follows:

How effective are machine learning methods at detecting covert DNS storage channels?

We decompose the main question into the following three individual sub-questions.

SQ1: In current literature, what (traditional) machine learning-based DNS storage channel detection methods exist, what features are effective and how is detection capability measured?

SQ2: What is the difference in detection capability between unsupervised anomaly detection and supervised classification?

SQ3: What are the effects of considering only payload features, only behavioral features or using composite feature sets?

We first identify which current methods are effective at detecting DNS storage channels, which features best describe storage channel characteristics and which properties of models are important to measure detection performance. Based on these insights, we design classification and anomaly detection experiments using combinations of features extracted from single query instances and query sequences, in order to analyze both the effects of different algorithm types and feature engineering rationales.

1.2. Threat Model

Given the diversity of legitimate as well as malicious DNS uses, we impose the following restrictions on the problem setting considered in this thesis.

1. We only consider traffic valid by the DNS specification, as malformed DNS queries may be rejected by legitimate DNS servers. In practical settings, when e.g. corporate networks require DNS traffic to flow through internal DNS servers, these malformed queries would not reach the malicious DNS server.

2. We assume that every storage channel uses a single primary domain. While storage channels could possibly distribute queries across multiple domains, common DNS connection tunneling tools use a single domain (e.g. [27, 29]), as have all four malware strains considered in this work in the past ([22, 38, 49, 72]). This threat model is in line with other storage channel detection research [16, 54, 62]. Note that our methods may still be able to detect multi-domain storage channel threats, as each domain can be viewed a distinct storage channel.

1.3. Contributions

- We create a novel malicious DNS threat dataset, comprising *iodine* and *dns2tcp* connection tunneling tools and *BernhardPOS*, *FrameworkPOS*, *MULTIGRAIN* and *UDPoS* credit card exfiltration malware. We design and capture traffic from a challenging tunneling scenario, varying tunneling parameters, and simulate all data exfiltration aspects of the respective malware at different time intervals.
- We provide for the first time a comprehensive comparison between different DNS storage channel detection methodologies based on either unsupervised anomaly detection and supervised classification algorithms, using (combinations) of payload and behavioral features.

1.4. Outline

This work is structured as follows. Chapter 2 provides background information about the DNS protocol and covert DNS storage channels. Chapter 3 contains a literature survey of current machine learning-based detection methodologies and feature extraction techniques. Chapter 4 describes how the real-world benign and simulated malicious datasets used in this research are collected and processed. Chapter 5 then describes our feature extraction and detection methods and experiment design. Chapter 6 presents and analyzes the outcomes of our experiments. Chapter 7 discusses the context in which the results are to be interpreted and provides recommendations for future work. Finally, the conclusions and limitations of this research are presented in Chapter 8.

2

Background

In this chapter, we introduce background knowledge about key topics used throughout this thesis. First, the purpose, terminology and syntax of the domain name system (DNS) are explained in Section 2.1. Then, covert DNS storage channels and data encoding techniques are introduced in Section 2.2. Finally, Section 2.3 describes different techniques to collect and analyze network traffic.

2.1. Domain Name System

The domain name system is a framework to store and retrieve arbitrary information associated with a named host in a network. A common use of DNS is to retrieve the IP address associated with an internet domain name. For example, the DNS lookup for the IPv4 address (*A-record*) associated with `www.tudelft.nl` returns (at the time of writing) `54.73.174.150`. DNS can be considered the *phone book* of the contemporary internet.

The DNS protocol was first introduced in 1983 to provide a consistent name space for network resources and a more descriptive and ergonomic alternative to using IP addresses directly. Its initial purpose was to provide a mapping of host names to IP addresses, as applications started to span multiple hosts and IP addresses changed dynamically [59]. Over the years, the DNS specification has been extended considerably and now allows for a wide variety of information types.

DNS consists of three core components: the domain name space and associated resource records, name servers and resolvers [60].

Domain Name Space / Resource Records
: The tree-structured *domain name space* defines the hierarchy of domain names within the domain name system. Each node in the name space has one or more associated resource (data) records. The hierarchy of an example domain name is illustrated in Figure 2.1.

Name Servers
: *Name servers* describe the domain name space and resource records for one or more domains. A name server is *authoritative* for a domain name when it provides its definitive resource records. Non-authoritative name servers can delegate (sub)domains to other name servers.

Resolvers
: *Resolvers* facilitate the DNS protocol by searching for and providing the requested resource records for a DNS query. Queries are resolved recursively: the name server for every part of the queried domain name (right-to-left) is queried until the resolver reaches the authoritative name server for a definitive answer.

DNS is a distributed and decentralized system in which has multiple *zones* (sub-spaces in the domain name space) controlled by different entities. Every zone has an authoritative name server that provides

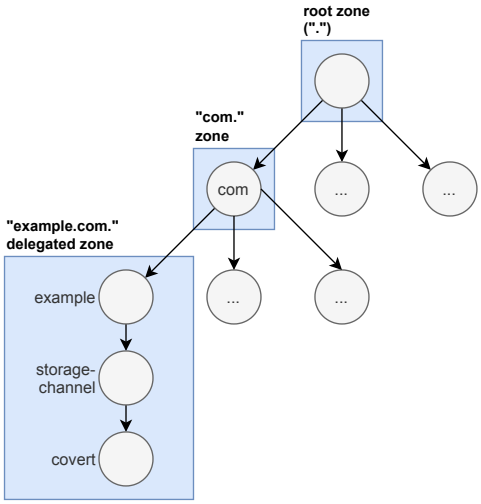


Figure 2.1: Tree structure of the domain name space, illustrated by means of example domain name "`covert.storage-channel.example.com.`".

its definitive resource records. Authoritative name servers can delegate lower-level zones to other name servers. Every zone has a (domain) name, except for the *root* zone, which has the empty label [60].

The DNS hierarchy is illustrated by means of example domain name "`covert.storage-channel.example.com.`" in Figure 2.1. The `example.com` domain and its subdomains are assumed to be in the same zone, which is a common scenario for e.g. websites.

DNS operates at Layer 7 (application layer) of the OSI networking model, in parallel with for example HTTP, and uses port 53 by default. As DNS requests and responses are isolated transmissions and do not require an open connection to a server, the stateless UDP protocol is used. Clients or servers may elect for TCP, e.g. when the expected DNS response is too large for UDP [17]. A side-effect of using a stateless protocol is that DNS clients often repeat the same query, for example after a timeout or even preemptively for performance reasons.

2.1.1. Terminology

This section introduces the DNS terminology used throughout this thesis. As DNS and its naming conventions have evolved organically over the years, not all terms have an exact or non-ambiguous definition. We consider the RFC 8499 document *"DNS Terminology"* [35], published in 2019, to be authoritative in this regard.

The different parts (*labels*) of a domain name, separated by a period, have a distinct purpose and designation, introduced below. Refer to Figure 2.2 for a visual dissection of example domain name "`covert.storage-channel.example.co.uk.`".

Label
: Sequence of zero or more characters that identify a node in the domain name space. The *root label* is the empty label that is the root node in the domain name space.

Domain name
: A list of one or more labels, ordered by decreasing distance from the root. Labels are separated with a period ('.') in domain names.

Host name
: Historically used to identify actual machines ("hosts") in the domain name space, but nowadays considered a domain name that follows the preferred name syntax (see Section 2.1.2).

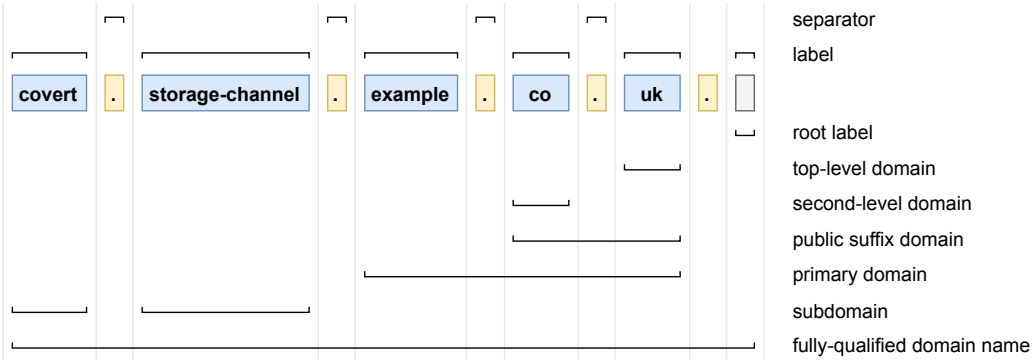


Figure 2.2: DNS terminology illustrated by means of the dissected example domain name "`covert.storage-channel.example.co.uk.`".

Fully-Qualified Domain Name (FQDN)
Domain name that is specified up to the root label.

Top-Level Domain (TLD)
The first label below the root label.

Second-Level Domain (2LD)
The first label below the top-level domain.

Public Suffix Domain
Ordinal-free term to denote domains under which end-users can directly register a domain name [61]. Sometimes referred to as *eTLD* (effective top-level domain).

Primary Domain
Domain name that comprises a public suffix domain and one label below.

Subdomain
Lower-level domain contained within another domain. In this research, we consider all labels below the primary domain *subdomains*.

Every resource record associated with a domain name has a type. The *record type* describes the purpose of the record and the kind of information it contains (e.g. an IP address or arbitrary text). Record types impose restrictions on the size and format of a resource record. A bidirectional communication channel over DNS, for example, requires a different number of queries to transmit the same amount of data depending on the resource record.

When DNS resolvers perform a lookup, the following terminology is used in the remainder of this thesis. A lookup for a domain name and record type consists of a *DNS request* originating from a client and a *DNS response* from a DNS server. A request contains, among other information, a *DNS query* (the domain name) that is considered its payload. In similar fashion, the payload of a *DNS answer* contains the requested resource record(s).

Common record types used in the remainder of this thesis are `A` (IPv4 address), `AAAA` (IPv6 address), `NS` (nameserver) and `TXT` (arbitrary text data).

2.1.2. Syntax

The syntax of DNS domain names is fragmentarily defined in numerous RFC documents. We summarize the aspects relevant to our research below and consider RFC 8499 and RFC 2181 (Section 11)

[17, 35] definitive information sources about DNS syntax.

Domain names consist of one or more labels. The size of a label is defined in octets[1] (or: bytes). Labels can be at most 63 octets long. The root label is *unnamed*; all other labels have to contain at least one character. Labels are concatenated with a period ('.') to form a fully-qualified domain name (FQDN) of at most 253 characters.

Host names only contain characters from the *letters, digits and hyphens (LDH)* subset. Although the DNS standard does not explicitly impose restrictions on the content of labels – any arbitrary octet is allowed, including e.g. unicode characters – the *preferred name syntax* [60] is to use only LDH-characters. Therefore, conforming programs transcode unicode domain names to *punycode*, a unicode representation in ASCII [25]. Newer standards also allow the use of a leading underscore to distinguish non-hostname domain names [17]. Hyphens may furthermore never lead or trail a label or occur adjacently.

Conforming DNS domain names contain one or more labels with one or more LDH-characters and possibly underscores, although arbitrary bytes are not explicitly disallowed. We strengthen this notion of validity by assuming that a valid, publicly accessible domain name has a *primary domain name* that contains only LDH-characters and has a suffix included in the Public Suffix List [61].

However, as DNS lookups may be issued for *any* domain name, "DNS servers must not refuse to serve a zone because it contains labels that might not be acceptable to some DNS client programs" [17]. Whether or not a perceived non-compliant request is processed is at the discretion of the DNS server. Therefore, DNS queries have to be processed without presuming they conform to the standard beforehand.

2.2. Covert DNS-Based Storage Channels

Besides the benign and intented use of DNS described in the previous section, the protocol can also be misused for arbitrary data exchange. Covert channels are a means to communicate information "in a manner that hides the fact that a communication channel is actually established at all" [36]. Covert channels using the DNS protocol are referred to as *covert DNS channels*.

2.2.1. Methodology

DNS is an attractive protocol for covert channels because, even though information can be exfiltrated from a secure network in myriads of ways, there is no direct connection established with the malicious server. Queries are often relayed via trusted (local) resolvers and DNS is widely available.

A covert DNS channel operates as follows. Any individual is free to register an available domain name. Its zone can be delegated to another, authoritative nameserver, which will receive incoming queries for the configured domain. When the delegated server is under our control, we can communicate information using DNS queries to the registered domain by encoding it in the subdomain labels.

Assuming we have registered and delegated the `example.com` domain and wish to exfiltrate the message `secret`, all we have to do is issue a DNS lookup for `secret.example.com`. The query is possibly relayed and resolved by a local DNS server, which recursively looks up the autoritative nameserver for first `.com` and then `example.com`. It then queries the server under our control, which now receives the query and as such the encoded information. This scenario is illustrated in Figure 2.3.

Covert channels that "transfer information (...) by writing to a shared storage location" [2] are referred to as *covert storage channels*. While there exists no exact definition, current research agrees that the *covertness* of a covert channel is proportional to the difference between its theoretical capacity and actual use, i.e. *covertness* $\propto$ (*capacity* - *transmission rate*) [36]. In the context of DNS storage channels, the covertness is therefore dependent on the utilization of the maximum query capacity and time interval between subsequent queries.

Covert DNS storage channels can further be characterized by their data flow. Figure 2.3 describes a scenario with *unidirectional* data flow: the exfiltration is one-way and the optional DNS response

[1] We often measure the length of a label in *characters*, as ASCII-characters are one octet in size.

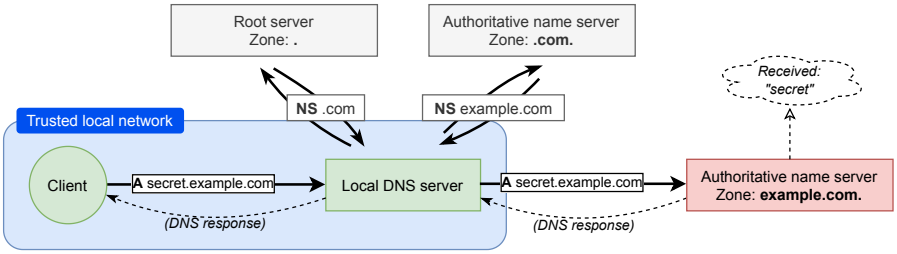


Figure 2.3: Schematic overview of a covert DNS channel, using a delegated DNS server.

is irrelevant. *Bidirectional* storage channels also use the DNS response to return a (malicious) payload. A common threat in either category is info-stealer malware for unidirectional channels and DNS (connection) tunnels using bidirectional data flow (see Section 3.1).

2.2.2. Data Encoding

Textual data can often be embedded as-is in DNS queries. Other data may need to be encoded as LDH-characters for successful transmission. This section contains a brief overview of the data encoding and encryption techniques used by the storage channel threats considered in this thesis.

Base encoding

Base encoding is a method to convert binary data to ASCII characters. A Base-*n* encoding uses a reduced ASCII-alphabet of size n to represent $\log_2 n$ bits of input data per character. Common n values include 64 (6-bit chunks), 32 (5-bit chunks) or 16 (4-bit chunks). Although any printable ASCII-subset may be used for the encoding alphabet, conventions exist for Base64, Base32 and Base16 [78]. The process of encoding `THESIS` in Base64 and Base16 is demonstrated in Figure 2.4.

UTF-8	T	H	E	S	I	S
Binary	01010100	01001000	01000101	01010011	01001001	01010011
6-bit chunks	010101	000100	100001	000101	010100	110100	100101	010011
Decimal (index)	21	4	33	5	20	52	37	19
Base64	V	E	h	F	U	0	l	T

4-bit chunks	0101	0100	0100	1000	0100	0101	0101	0011	0100	1001	0101	0011
Base16 / hex	5	4	4	8	4	5	5	3	4	9	5	3

Figure 2.4: Demonstration of the encoding process of `THESIS` in Base64 and Base16.

Base16 is also referred to as *hex* encoding, as the hexadecimal numeral system represents numbers using radix (base) 16.

Base encoding inherently introduces space overhead. Representing the 8-bit input by 6-bit chunks in Base64 results in a string $^8/_6 * 100\% = 133.33\%$ the original size. This is 160% and 200% for Base32 and Base16, respectively. The input data additionally has be padded to reach a length that is a multiple of the chunk size.

Conveniently, the default Base64 alphabet – excluding the padding character – is equal to the (case-insensitive) LDH character set plus underscore. The default Base32 alphabet includes only lowercase letters and digits and the Base16 alphabet contains only digits and the letters A-F. All encodings can therefore be used to construct valid DNS names.

The aforementioned (standardized) Base-encodings all encode to printable ASCII characters. However, other alphabets can be used for Base-encoding as well, especially since the DNS specification does not explicitly forbid non-printable bytes in DNS query names. DNS tunnel *iodine* [29], for example, uses a custom Base128 encoding.

Encryption
Adversaries may hide the actual information transmitted via a storage channel with encryption. The threats considered in this thesis that apply encryption use either a substitution cipher, an XOR cipher or RC4 encryption. While an in-depth description of each technique is beyond the scope of this thesis, a brief overview is included below.

Substitution cipher	Provided a mapping from plaintext units to ciphertext units, substitute each unit it in the plaintext with its corresponding ciphertext. Units are often single letters, and plaintext units for which no substitution exists are passed through.
XOR cipher	Additive cipher that perfoms a bitwise XOR ($\oplus$) operation between a plaintext and encryption key. The key is repeated when it is shorter than the plaintext.
RC4	Symmetric stream cipher that constructs a ciphertext by performing a bitwise XOR between the RC4 keystream, based on an encyption key, and the plaintext.
RSA	Asymmetric public-key cryptosystem that relies on large prime number factorization for security. Encrypts the plaintext using a RSA key, commonly 1024 to 4096 bits in size.

The output of any encryption algorithm can be (Base) encoded to produce valid LDH-ciphertext for use in DNS queries.

Benign Storage Channels
Covert DNS storage channels, i.e. with the purpose of surreptitiously transferring information, are considered malicious by definition in this work. However, regular DNS traffic may contain lookups that resemble covert storage channels, but do not have malicious intent.

Chen *et al.* [21] examined *disposable domain names*, designated for one-time use, similar to DNS storage channel queries. The authors show a number of prevalent examples: McAfee embeds file hashes in a DNS query to establish file reputation scores and Google has used disposable domains for an IPv6 availability experiment. Two such domain names (from [21]) are for example:

```
0.0.0.0.1.0.0.4e.135jg5e1pd7s4735ftrqweufm5.avqs.mcafee.com
p2.a22a43lt5rwfg.ihg5ki5i6q3cfn3n.191742.i1.ds.ipv6-exp.l.google.com
```

We refer to DNS storage channels that (mis)use DNS for data exchange without malicious intent as *benign storage channels*.

2.3. Network Traffic Analysis

In this section, we introduce three techniques to capture and analyze DNS traffic: packet capture, flow monitoring and traffic analysis. Each method provides a different level of granularity.

Traffic that passes through (part of) a computer network can be intercepted and logged with a packet analyzer. The resulting packet captures (or **pcaps**) contain decoded copies of the network packets in the captured data stream.

Pcaps provide access to the individual fields of packets and are as such suitable for e.g. analyzing low-level network problems. This level of detail, however, comes at a cost as packet capture requires expensive hardware and substantial infrastructure for storage and analysis in high-speed networks [39].

A less computationally expensive alternative to packet capture is **flow monitoring**. Flows describe sets of IP packets that are aggregated by a common properties, e.g. source and destination IP, protocol and port number [39]. Flows describe only the metadata of a connection (inbound and outbound bytes, TCP flags, etc.).

Flow monitoring is too limited for this research, because flows only describe connections between users and DNS servers. While anomalies in these statistics could possibly be used to identify hosts

that use high-throughput tunneling, lower-throughput exfiltrations fall well within the bounds of benign DNS flows.

Traffic analysis is a compromise between packet capture and flow monitoring. Specialized tools analyze packet captures and generate traffic logs that describe every connection – much like flows – but annotated with the salient details of that connection.

Zeek (formerly Bro) is a passive, open-source network traffic analyzer [64]. Zeek can analyze DNS traffic and generates (among others) a log file that describes every observed DNS lookup. This traffic log contains all decoded DNS packet fields, without other lower level (IP) packet information. Zeek provides the required level of detail, i.e. the timestamp, query name and query type of DNS queries, without the storage overhead of pcaps. Zeek DNS logs are used as data source in this thesis.

2.4. Summary

In this chapter, we provided background knowledge about key concepts used throughout this thesis. The purpose, terminology and syntax of the domain name system protocol was explained in Section 2.1. We introduced the concept of DNS storage channels that (mis)use DNS by storing arbitrary information in DNS queries in Section 2.2. Unidirectional storage channels – commonly info-stealer malware – only exfiltrate information; bidirectional storage channels tunnel arbitrary connections by facilitating both sending and receiving.

Furthermore, we provided a brief introduction to Base-encoding, data encryption techniques used to encode and obfuscate information in DNS queries and benign storage channels. Lastly, we described three techniques to capture and analyze DNS traffic in Section 2.3: packet capture, flow monitoring and traffic analysis. Open-source network traffic analyzer Zeek is used to process DNS traffic in this thesis.

3

Literature Review

This chapter provides an overview of research efforts that have been made in the field of DNS storage channel detection. The purpose of this chapter is to answer the first research question: *"In current literature, what (traditional) machine learning-based DNS storage channel detection methods exist, what features are effective and how is detection capability measured?"*.

The scope of this review is defined by the problem setting and research questions. We only include works that attempt to detect DNS-based storage channels with (classic) machine learning techniques, i.e. with manually derived features, in order to be able to combine feature sets. Papers that focus on e.g. DNS timing channels are excluded under our threat model.

Furthermore, works that rely primarily on features extracted from DNS response payloads are only included if they provide valuable additional insights, as these features are not descriptive of unidirectional storage channel threats.

This chapter is structured as follows. Firstly, an overview of common research subjects is provided in Section 3.1. Section 3.2 then surveys features used to describe these threats and Section 3.3 contains an overview of the methods and algorithms used for detection. Section 3.4 summarizes common performance evaluation measures and attempts to define *detection capability*. Lastly, related work is briefly surveyed in Section 3.5.

3.1. Threat Landscape

Different types of DNS storage channels are considered in current literature. It is important to establish the nature of these threats, in order to better understand the detection methods. We identify two threat categories: *connection tunneling* (Section 3.1.1) and *arbitrary data transfer* (Section 3.1.2). Most research focuses only on tunneling: all of the surveyed works use these threats for evaluation. Arbitrary data transfer is only considered by [1, 16, 26, 62, 67, 73].

3.1.1. Connection Tunneling

DNS tunnels relay arbitrary networking data via DNS by encapsulation of its packets, i.e. they provide a means to tunnel another protocol over DNS. Tunnels require both a client-side and server-side component to split and reassemble tunneled networking data [73].

Upstream data is encoded in the DNS query name, downstream data is sent back via the DNS response. As DNS requests have to be initiated at the client, tunneling tools poll for queued downstream data at regular intervals [29]. Because of the stateless nature of DNS, tunneling tools implement a custom TCP-like protocol on top of DNS to maintain the order and integrity of the transferred data [12].

A wide variety of open source and readily available tunneling tools exist. Table 3.1 contains an overview of research subjects observed in the surveyed works. We briefly describe the prevalent *iodine*, *dembourHSCOutilsDns2tcp2007* and *dnscat2* tunneling tools below, which differ at the level of encapsulation, data encoding and resource record types used.

dns2tcp TCP-over-DNS port forwarding tool that encapsulates connections at the TCP level [27]. The server-side application specifies available resources (ports) that are forwarded using a DNS tunnel upon client request. Dns2tcp supports `TXT` and `KEY` record types and uses Base64 for both up- and downstream data encoding.

dnscat2 Provides a custom Command-and-Control (C2) channel over DNS and also supports tunneling of any TCP connection (e.g. SSH sessions) [12]. Combinations of `TXT`, `CNAME`, `MX`, A or `AAAA` query types are supported and the up- and downstream data is hex-encoded.

iodine Highly configurable IPv4-over-DNS tunneling tool that encapsulates connections at the IP level [29]. Iodine requires a dedicated (virtual) TUN/TAP adapter, all of whose traffic is tunneled over DNS. Upstream data can be encoded with either Base128, Base64 or Base32 and downstream queries use `NULL`, `PRIVATE`, `TXT`, `SRV`, `MX`, `CNAME` or `A` record types. Downstream data is optionally compressed to increase throughput. Iodine probes the configured DNS server on initialization to find the optimal tunneling parameters.

Connection tunneling is ergonomic: existing applications or protocols can be used as if there was no tunnel. Tunneling, however, provides little control over the emitted query structure and causes significant overhead (up to 1500% in [58] and 2000% in [83]).

3.1.2. Arbitrary Data Transfer

Arbitrary data transfer over DNS, without other protocol constraints, allows for more fine-grained control over the query structure and exfiltration schedule. Adversaries can tune their exfiltration to make detection as difficult as possible.

The main threats in this category are malware: Remote Access Trojans (RAT) that establish a C2 channel over DNS and info-stealer malware that exfiltrate credit card details. Besides malware, other research subjects include a tool for file transfer over DNS and custom exfiltration scripts.

As with DNS tunnels, the main distinction between threats is based on the upstream data encoding and query types. The exfiltration schedule (or timing) is also relevant, as the frequency is often low to prevent detection.

Saeli *et al.* [73] evaluate their detection method using numerous real-world malware samples, from which DNS traffic was captured in a sandbox environment. These threats comprise Remote Access Trojans (RAT) and Point-of-Sale (PoS) malware that use Base32, Base64, hex or custom data encodings and optionally encrypt the transmitted information. The authors also use the *dnsfilexfer* tool to transfer arbitrary files via DNS.

Nadler *et al.* [62] simulate the *FrameworkPOS* and *Backdoor.Win32.Denis* malware variants. The FrameworkPOS simulation exfiltrates three sets of Base64-encoded credit card details per second. Beaconing queries with a custom encoding from Backdoor.Win32.Denis were generated every 1.5

Table 3.1: DNS tunneling tools used in current literature.

Name	Description	Used by
dns2tcp	Open-source tunneling tool that forwards TCP ports over DNS.	[3, 4, 6–8, 14, 16, 54, 62, 73]
dnscat2	Open-source C2 and connection tunneling tool.	[9, 16, 54, 73, 75]
iodine	Open-source IPv4-over-DNS tunneling tool.	[6, 7, 14, 16, 41, 54, 62, 73, 75]
Cobalt Strike	Commercial adversary simulation and red team operations software. The DNS Beacon feature establishes a communication channel over DNS.	[16]
DNScapy	Open-source tunneling tool to encapsulate SSH connections. Uses Python package Scapy for packet manipulation.	[14, 73]
dnscat	Open-source IP-over-DNS tunneling tool. Unrelated to dnscat2.	[26]
OzymanDNS	Open-source tunneling tool that encapsulate SSH sessions.	[54, 75]
TUNS	Simple open-source IP-over-DNS tunnel.	[14]
Your-Freedom	All-in-one firewall and proxy bypassing tool, with DNS tunneling mode.	[73]

seconds. The exfiltration intervals are based on incident reports of real-world campaigns in which the malware was used.

Das *et al.* [26] synthesize four datasets of which one is based on the *BernhardPOS* malware. These queries contain a Base64-encoded credit card number and a fixed primary domain.

Ahmed *et al.* [1] use the DNS module of open-source *Data Exfiltration Toolkit* (DET) to encode and encrypt random credit card details in DNS queries. DET uses AES-256 to encrypt information and hex for data encoding. The authors further evaluate their system with a small sample of 17 queries from PoS-malware (*BernhardPOS*, *FrameworkPOS*) and RAT-malware (*DNSMessenger*, *DNSpionage*).

A custom DNS exfiltration mechanism is used in two other works. Buczak *et al.* train and evaluate their system on data from their *Pick Pocket* tunneling tool, which was "created with the purpose of circumventing IDS defenses" [16]. Preston [67] creates a comprehensive DNS exfiltration suite that varies numerous parameters, including the amount of data exfiltrated, record type, data encoding, use of encryption and query lengths.

3.2. Features

This section provides an overview of the machine learning features used in current literature to detect DNS storage channels. In machine learning, *features* are individual measurable quantities that describe an observed phenomenon [74]. In order to effectively detect storage channels, informative and descriptive features have to be engineered that capitalize on the difference between benign and malicious DNS traffic. Current literature proposes many such features[1].

Most works share common feature engineering rationales: their features are based on storage channel characteristics or benign traffic characteristics and consider the limitations of the DNS specification. Some works also consider the practicality of deployment (e.g. [26]) or take privacy implications into account (e.g. [42]).

A major distinction between observed features, however, is the information source from which they are derived. We first describe *single-instance* features in Section 3.2.1 and summarize *query sequence* features in Section 3.2.2

3.2.1. Single-Instance Features

Single-instance features, or *payload* features, are extracted from a single DNS packet or query. Because the surveyed works use different combinations of features to detect different threats, we provide a thematic overview based on the underlying feature engineering rationale.

Query space utilization

By definition of DNS storage channels, all data is stored in the DNS query name, which is restricted by the standard to at most 253 characters. Storage channels have a tendency to use more of the available query space to increases their efficiency.

The following four features that describe query space utilization in current literature are derived from either "pockets" in DNS packets where data can be stored without affecting the DNS process [16], experimental evidence from the datasets used [54] or prior success in related work [42].

Packet size [7, 16, 42, 54]
: The size of the IP packet or encapsulated DNS packet.

Query length [1, 7, 16, 26, 42, 62, 75]
: The length of the DNS query name. Some works only consider the concatenated subdomains to remove the influence of the primary domain length. Yu *et al.* [85] additionally derive the ratio between processed and original query length.

Open space [16]
: Describes the lack of query space used, calculated as $253 - \mathit{query\ length}$.

[1] N.B.: based on the scope of this review, features derived from the payload of a DNS response are omitted. DNS responses are fully controllable by malicious actors and are not used by unidirectional storage channels.

Fill ratio [73]
The fraction of available query space used, calculated as $\frac{query\ length}{253}$.

A major concern with query space utilization features is the fact that storage channels do not necessarily use long queries. While these features may be effective at separating tunnels from other DNS traffic, low-throughput malware would be indistinguishable from benign traffic.

These features should also take the length of the primary domain into account, or queries to benign but long primary domains would inadvertently appear more suspicious.

Query structure
Using more query space must result in either longer or more labels in a DNS query. Therefore, the amount or labels or label lengths might be predictive of storage channels.

Ahmed *et al.* [1] demonstrate that the vast majority of benign DNS queries have an average label length of at most 10 characters.

Number of subdomains [1, 16, 26, 85]
The total number of subdomains, or all labels in a FQDN.

Average subdomain length [1, 67]
The average length of all subdomains, or the average length of all labels in a FQDN.

Maximum subdomain length [1, 16, 73]
The length of the longest label or subdomain in a FQDN. Saeli *et al.* [73] use the ratio between the longest label and maximum label length of 63 characters instead.

Because information can be freely divided over labels in a query, query structure features are most effective when used together, as a change in one feature also influences other features (given the same amount of characters in a query).

As with query space utilization features, the structure of the primary domain should again be taken into account. Private suffixes with many labels would else reduce the effectiveness of these features.

Information density
As the purpose of DNS storage channels is to transfer information, measuring the information density (or: randomness, uncertainty) in queries has proven to be an effective predictor of storage channels. The reason behind this is twofold:

1. Increasing information density by compressing query names allows for shorter queries that contain the same amount of information, or for more information contained in queries of the same length
2. Well-designed encryption algorithms make it difficult to distinguish ciphertexts from random noise, hence have a high information density

The following three features are used to measure information density in current literature. X denotes a DNS query name.

Entropy [1, 7, 14, 26, 42, 54, 62, 67, 73, 75, 85]
(Shannon) entropy [76] is a widely used measure of the uncertainty of a random process. *Character entropy* in the context of DNS query names describes the average amount of information conveyed per character. Given a sequence of characters X, character entropy is defined as:

$H(X) = -\sum_{c \in X} \Pr(c) \cdot \log_2 \Pr(c)$

Entropy with the base-2 logarithm is measured in bits. Character entropy values are bounded by the number of possible characters, i.e. $\log_2 64 = 6$ for LDH-queries and $\log_2 256 = 8$ for queries with arbitrary byte values.

Besides (single) character entropy, other feature variants include bi- or trigram entropy [54], normalized entropy [26], the entropy of only LDH-characters [62], the maximum entropy between the full query and longest label [73] or the (binary) entropy of the full DNS packet [7].

Gini impurity [85]
The Gini impurity of a character sequence measures the probability that two randomly selected characters from the character distribution are not the same. Gini impurity is calculated as follows.
$G(X) = 1 - \sum_{c \in X} \Pr^2(c)$
This measure is closely related to character entropy, but is bounded by zero (all characters are the same) and one (all characters are different).

Compressed query length [65, 75]
The length of the DNS query name after compression, using e.g. *gzip* or *bzip* compression algorithms. A more rudimentary approach to measuring information content.

Entropy-based features have a non-trivial relation to the query length. The maximum entropy of short queries is bounded by the query length, as fewer possible characters could have been used. The maximum entropy of a query of length 8 is for example only $\log_2(8) = 3$. Normalizing by query length (*metric entropy*, $H/|X|$) to mitigate this effect, however, would generate high values for very short queries (e.g. length 1 or 2), which is not the intended effect of this feature.

Alternatively normalizing by maximum entropy (*efficiency*, H/H_{max}) has drawbacks as well. The amount of possible characters is used to calculate the maximum entropy, but LDH-characters are far more prevalent than other arbitrary bytes. Regardless, this method too would result in high values for short query names.

Lexical properties
Lexical features are related to the words or vocabulary of a language. We adopt a loose interpretation of *language* that also includes features influenced by the DNS (preferred name) syntax (see Section 2.1.2).

The following four features observed in current literature exploit either the difference in character distribution between benign and malicious traffic or draw on similarities with human languages. X denotes a DNS query name.

Character type frequency [1, 16, 26, 67, 73, 85]
The intuition behind character type features is that benign traffic uses the preferred name syntax and encrypted or encoded traffic does not [1].

Different features variants either describe the absolute count or relative frequency of character groups. The features with their corresponding character sets are included below.

Uppercase count [1]	[A-Z]
Numeric count [1]	[0-9]
Lowercase ratio [26]	[a-z]
Non-lowercase ratio [85]	^[a-z]
Uppercase ratio [26, 73]	[A-Z]
Numeric ratio [26, 73]	[0-9]
Alphabetic ratio [26]	[a-zA-Z]
Alphanumeric ratio [67]	[a-zA-Z0-9]
Distinct ratio [16]	$\{c \in X\}$

N-gram distribution [73, 85]
Character *n-grams* are contiguous overlapping character subsequences of length n. The set of bigrams for "NGRAM" is for example {NG, GR, RA, AM}. The frequency distribution of a collection of n-grams is expressive of the underlying dataset. Born and Gustafson [11] empirically show that bi- and trigram models of DNS tunneling traffic are significantly different from those of a corpus of English texts.

Yu *et al.* [85] train bi- and trigram character models of English words and calculate percentile-based scores for each query name. Saeli *et al.* [73] create uni- and bigram models of English and Italian corpora and sum the statistical distances between these distributions and those for a given query name.

Longest meaningful word (LMW) ratio [62, 67]
The longest subsequence of characters in a DNS query label that exists in a predefined vocabulary of "meaningful" words, divided by the total query length.

This feature is presumed to be effective as "subdomains [usually] contain meaningful [and readable] English words" [62].

English content ratio [67]
Similar to the LMW-ratio, defined as the number of characters that represent English words, divided by the total query length. The *English content* is calculated by recursively searching for the longest English word in the sequence of remaining characters.

Features based on human language characteristics are inherently biased towards those languages. The underlying assumptions about DNS traffic are not necessarily valid in every deployment setting. The quality of n-gram features depends further on the comprehensiveness and diversity of the training corpora.

DNS usage
The final category of single-instance features describes uncommon query types and unique query traffic characteristics.

Query type [16, 54, 62, 75]
Query (or: record) types have a characteristic distribution in normal DNS traffic, e.g. `A`, `NS` and `PTR` records are very common, while `NULL` records are not [54, 85]. DNS tunneling tools often use uncommon record types that hold more information than other (common) types [29].

The query type can be used as categorical single-instance feature by e.g. one-hot encoding [75].

Request / response delta [54, 85]
While not strictly a single-instance feature, when both DNS request and corresponding response packets are available for feature extraction, the time delta between those packets is descriptive of DNS storage channels. Liu *et al.* [54] observe that the uncached unique queries of storage channels take longer to resolve than the (often) cached benign queries.

A major concern of query type features is that they are fully controllable by an adversary. While DNS tunneling tools may have to resort to obscure query types for efficiency, arbitrary data exfiltrations can simply use the common `A` record type to appear benign. Also, the response delta feature is not only subject to DNS server behavior, but to any networking irregularities. This feature may therefore be unstable in unreliable networks.

3.2.2. Behavioral Features

Single-instance features may not be descriptive enough of storage channels that resemble benign DNS traffic. The context in which particular queries exist then helps to determine whether or not they are malicious. This context can be provided by analyzing query sequences. Query sequence features describe the behavior of storage channels and are also referred to as *behavioral features*.

This section provides an overview of query grouping techniques for feature extraction, as well as as overview of common behavioral features.

Aggregation Methods
In order to generate query sequences, isolated queries have to be grouped by a common denominator. This denominator determines which entities are ultimately classified as suspicious, e.g. end-users or primary domains[2]. Four different aggregation levels are used in current literature.

Time [3, 4]
Time-based aggregation groups DNS traffic by time intervals, regardless of origin, destination or primary domain.

[2]Note that aggregation options may be restricted by the underlying dataset. In our research, for example, client IP addresses cannot reliably be determined (see Chapter 4), which prohibits per-user query sequence analysis.

Domain [16, 62, 67]
Per-domain aggregations group queries that share a common suffix. This common suffix is e.g. the full primary domain or the second- and top level domains.

User + Domain [16, 26, 85]
User + Domain aggregations group queries by origin IP and domain. Note that *user* in this context may also refer to a (resolving) DNS server, depending on the packet capture position in the network.

User + Server + Domain [54]
This aggregation considers source IP, destination IP as well as domain.

Per-domain aggregations are able to identify isolated storage channels, as, by our definition, storage channels exfiltrate to a single primary domain. Time-based aggregations, in contrast, identify suspicious temporal activity that has to be manually analyzed in order to determine the actual threat [41].

Given a sequence of aggregated queries, features can be extracted by either reducing the query group, or by using a sliding window.

Sequence Reduction [3, 4, 26]
Sequence reduction considers all queries in the group, and is therefore able to incorporate all available information per group at once. This technique produces one feature vector per group.

Aiello *et al.* [3, 4] extract statistical features from queries in contiguous, non-overlapping time bins. Das *et al.* [26] extract features from the concatenated subdomains of all queries with a common primary domain.

Sliding Window [16, 54, 62, 67, 85]
Sliding (or: rolling) window feature extraction operates on a sliding partition of queries in the sequence. The window is either statically sized, i.e. contains a fixed number of queries, or dynamically sized based on a (time) offset. Sliding window analysis produces one feature vector per window, i.e. as many feature vectors as observations when sliding over queries.

Compared to the sequence reduction approach, sliding windows are more memory-efficient as only the queries in the current window have to be retained. Sliding windows can also be used in streaming settings where not all data is available beforehand.

Time Nadler *et al.* [62] slide over contiguous time bins instead of queries. The authors recommend a window size of 24 and 15 minute bins. The use of time bins reduces the required processing power of the system while still retaining its ability to detect exfiltrations that occur within $24 \cdot 15 = 360$ minutes. Features not related to query frequencies are extracted from the concatenated payloads in the window.

Fixed Buczak *et al.* [16] use a sliding window of size 50 for feature extraction and Preston [67] computes its windowed features in a window of size 20. Liu *et al.* [54] experiment with fixed window sizes between 1 and 20.

Fixed sliding windows can identify extremely slow exfiltrations, as the time interval between queries is ignored. However, this can also be achieved by time windows of sufficient length, as demonstrated by Nadler *et al.* [62].

Fixed windows have predictable space complexity, i.e. they bounded by the number of possible aggregations in the dataset (e.g. primary domains) and the size of the window. Dynamically sized sliding windows, however, may grow arbitrarily large for high-frequency domains.

Hybrid Yu *et al.* [85] use a hybrid window with a fixed capacity that optionally expires queries based on a time delta before the capacity is reached. The motivation behind this hybrid approach is to “guarantee the data freshness and preserve storage space” [85].

Aggregation levels and feature extraction techniques influence the resulting dataset. Class imbalance, for example, decreases when using a per-domain aggregation in combination with sequence reduction, as there are fewer distinct primary domains than queries. It is furthermore only possible to combine behavioral features with single-instance features when using sliding window analysis.

Features
This section provides an overview of query sequence features used in current literature. Most query sequence features are grounded in storage channel characteristics: high query frequencies and/or many unique queries. These *behavioral* storage channel traits are invisible when only looking at single instances.

Single-instance feature statistics [16, 54, 62, 67, 85]
Many works derive statistical features from separate single-instance features in the window.

These features include the average query length [62], average subdomain length [67], total entropy of all payloads in the window [85], the mean and variance of the request / response delta, request packet size and entropy features [54] and average LMW-length or LMW-ratio [62, 67].

Aiello *et al.* [3] and Buczak *et al.* [16] consider the average, variance, skewness and kurtosis of all windowed features. Yu *et al.* [85] use the count, sum, min, max and average values per single-instance feature in the window.

Query volume [73]
The number of queries in the window. Saeli *et al.* [73] exclude DNS retransmissions from this count.

Unique query volume [62, 67, 73]
The number of unique queries in the window.

Unique query ratio [62, 67]
The amount of unique queries, divided by the total number of queries in the window.

Unique subdomain volume [26]
The amount of unique subdomains in the window.

Query type ratio [16, 62]
The number of queries with a specific query type, divided by the total number of queries in the window. Nadler *et al.* [62] consider the ratio of `A` and `AAAA` records, Buczak *et al.* [16] create separate features for 12 common and uncommon query types.

Query similarity [16, 67]
Buczak *et al.* [16] determine the similarity between lowest-level domains in the window. Preston [67] calculates the average character overlap between query pairs in the window.

Behavioral features are able to capture context that single-instance features cannot describe. The level of aggregation, in conjunction with the feature extraction method, defines at what granularity threats can be identified: per query, per domain, per user or per time interval. Behavioral features can be paired with single-instance features to paint a more complete picture, as demonstrated by e.g. Buczak *et al.* [16].

3.3. Detection Methods
This section provides an overview of the different machine learning-based detection methodologies proposed in current literature. We observe a clear distinction between works in terms of learning type: either supervised classification or unsupervised anomaly detection.

3.3.1. Supervised Classification
When a labeled set of training data is available, *supervised learning* algorithms exploit this a priori known information about the classes to find a decision boundary [74]. Although labeled datasets are often not available in our domain – labeling large real-world network traffic datasets by hand is infeasible – ground truth can be derived by either presuming normal traffic clean (e.g. [75]) or by using allowlists of reputable domains (e.g. [26]). Supervised learning is further enabled by generating or synthesizing network traces from known storage channel threats [73].

The following works use supervised learning for classification. Given the large number of parameters, i.e. differences in the considered storage channel threats, feature sets, query aggregation levels,

datasets and evaluation metrics, we only briefly summarize their methods and reported conclusions.

We first review approaches that use only single-instance features.

Das *et al.* [26] propose a two-stage approach: suspicious queries are first tagged using heuristics, and tagged queries are subsequently grouped by their primary domain for feature extraction. Their feature set describes information density, query lengths, character frequencies and query structure. A logistic regression model is used to classify queries as either benign or malicious. Their method reportly achieves a recall of 94.5% and a false positive rate of 0.19% on a real-world dataset of 16M queries with query type TXT and artificial exfiltration traffic.

Shafieian *et al.* [75] experiment with ensembles of random forest (RF), multi-layer perceptrons and k-nearest neighbour (k-NN) classifiers. Their feature set describes DNS usage, query length and information density, and similar features are also extracted from DNS responses. Different ensemble configuration and combination rules are evaluated, benign traffic which all show “very high” accuracy, misclassify at most 100 out of 500K test instances and at most 37 false positives. The authors show that a well-tuned ensemble of classifiers outperforms single classifiers, although adding more classifiers does not always increase performance.

Almusawi and Amintoosi [7] use a multiclass support vector machine (SVM) classifier to detect and predict tunneled protocols. Their feature set, comprising request and response packet size, query length and information density features, is discretized using k-means clustering to bound and scale feature values. The authors show that the kernel SVM approach outperforms a Bayesian classifier, but their testbed is limited (only 530 samples in total) and the false positive rate is at best 3.3%.

Bubnov [14] uses multiclass-classification to detect specific tunneling tools with a three-layer feedforward neural network. Features extracted from the payload of DNS requests and responses are based on information density and record type usage. While the model is able to correctly identify large portions of tunneling traffic, it is only 83% accurate on the (limited) set of normal traffic. The test bed is limited as well, comprising only 65,000 samples in total.

The following works use behavioral features as well.

Aiello *et al.* [3, 4] use a per-time aggregation and experiment with various algorithms: the Naive Bayes and Boundary classifiers in [4] and linear discriminant analysis (LDA), k-NN, multi-layer perceptron and SVM algorithms in [3]. Suspicious time-intervals are identified using a small set of statistical behavioral features. Experiments with different ratios of mixed tunneling traffic demonstrate the feasibility of their supervised learning approach, although the performance worsens significantly as the concentration of malicious traffic in the dataset decreases. Repeated trials and majority voting reduce this error rate.

Buczak *et al.* [16] use a comprehensive set of 59 features that describe various characteristics of both queries and corresponding responses. Statistics of a subset of single-instance features are calculated using a sliding window and per-domain aggregation.

Two Random Forest models using data that is either collected inside a network or at the network perimeter. Both models are able to achieve a very high detection rates (> 99.9%) on known threats, whilst reporting almost no false positives. Queries from a custom, unseen threat were also correctly identified, albeit at varying recall (27.6% – 95.89%).

Liu *et al.* [54] compare the efficacy of the decision tree, logistic regression and SVM classifiers using behavioral features based on query space utilization, information density and DNS usage. Features are extracted using a sliding window of fixed size over DNS request and response pairs.

While all three classifiers show high detection rates (> 95.9%), the SVM model outperforms the others (at most 99.3%). The authors acknowledge, however, that their malicious tunneling dataset is insufficient to train models that generalize well over unknown threats and stress the importance of

dataset quality for supervised model training.

Preston [67] experiments with a variety of supervised learners: different bagging (Random Forest and a generic tree bagging classifier) and boosting (AdaBoost and Gradient Tree Boosting) algorithms, as well as SVMs with a linear or radial kernel. The feature set is based on information density, lexical properties and query structure and is computed from a per-domain fixed sliding window.

The system is evaluated with a sizeable dataset of real DNS traffic, from which only queries with types `A`, `AAAA` and `TXT` are retained. This dataset is filtered using an Isolation Forest model to mitigate the concern that the dataset would contain malicious traffic.

All trained models achieve high precision and recall, reportedly over 99.6% on both metrics, with the bagging and boosting algorithms slightly outperforming SVM. A subsequent test with simulated (unseen) malware queries and the Random Forest model confirmed the efficacy of the approach. However, it is unclear whether the final results are obtained used the filtered benign class, which would severely inflate the results. Few information is reported about the custom exfiltration queries as well.

Supervised classifiers are predominantly used for DNS tunneling detection, which is motivated by the relative ease with which malicious training traffic can be acquired. The choice of algorithm is seemingly irrelevant: works that compare different classifiers report minor performance differences. Tree-based, often Random Forests, and SVM-classifiers are prevalent and seem to perform well. However, it is difficult to compare the relative performance of surveyed works, given the large variability in datasets used. The effectiveness of detectors against unknown or unseen threats is, unfortunately, rarely evaluated as well.

3.3.2. Unsupervised Anomaly Detection

The focus of recent research has shifted towards unsupervised anomaly detection, motivated by its improved ability to generalize over unseen threats. To date, few works ([1, 62, 73]) have experimented with an anomaly detection approach in conjunction with features derived from DNS *queries*.

Nadler *et al.* [62] build an anomaly detection system using the Isolation Forest (iForest) algorithm. Their (behavioral) feature set is based on traffic characteristics, information density and lexical properties. Features are extracted from groups to distinct primary domains at predefined time intervals. The anomaly threshold of the system is tuned by a predefined acceptable false positive rate. The authors evaluate their system on a large and diverse dataset of DNS traffic and report a detection accuracy (per domain) of 100% for a variety of storage channel threats, whilst maintaining a low false positive rate of 19 primary domains during six days of operation.

Ahmed *et al.* [1] propose a similar iForest-based system, but advocate for stateless (single-instance) features instead, as they allows for faster intervention after detection and reduce the amount of state to be kept. Their stateless feature set is based on query space utilization, information density and character frequencies. Ground truth for training queries are based on primary domain reputation in a public allowlist.

The amount and height limit of trees and contamination rate (a proxy for the anomaly score threshold) hyperparameters of the iForest model are tuned to improve accuracy. While the system is able to detect up to 98.5% of malicious queries, a significant amount of at least 1.6% (top 10,000 most popular domains) or 21.6% (remaining domains) of benign queries is flagged as anomalous. This would suggest that their feature set is too limited for use in practical situations. A small sample of unseen malware queries, however, were all correctly identified as anomalous.

The authors also train, for comparison, a Random Forest classifier on a balanced dataset containing presumed benign queries and queries from one exfiltration threat. The supervised model outperforms the unsupervised approach on a hold-out set, but performs rather poorly on unseen threats, presumably due to the limited malicious query diversity in the training set.

Saeli *et al.* [73] propose a three-stage detection system based on the One-Class Support Vector Machine (OCSVM). The first step comprises rigorous filtering of queries in the dataset, based on e.g. allowlisting, query response codes and query length.

The second offline phase comprises the training of the OCSVM model on a set of filtered (and

presumed benign) queries, using four features based on character frequency and query structure. In the subsequent online phase, DNS queries that are flagged as anomalous are further analyzed by computing an anomaly score based on behavioral, lexical and information density features. When the final anomaly score exceeds a preconfigured threshold, the query is flagged as malicious.

The system is evaluated with a diverse set of data exfiltration malware and tunneling samples, all of which are successfully detected at various detection rates (between 12% and 100% of queries). However, the false positive rate on a limited set of benign traffic is reportedly 10%, which is significant but in line with [1].

In summary, unsupervised anomaly detection systems use either the Isolation Forest or OCSVM algorithms. The main benefit of anomaly detection over classification is the fact that no class labels are required for training and the increased ability of detecting unknown and unseen threats, at the expense of a higher false positive rate.

3.3.3. Data Collection Remarks

Researchers with access to unlabeled, real-world datasets often distill a ground-truth for the benign class by considering only popular domains, using The Alexa Top Sites, Majestic Million and Cisco Umbrella lists, ranking popular websites. This, however, introduces bias as less popular benign domains are not included in the training set. The model performance may also be inflated when evaluated on the same subset of known domains. Furthermore, research has shown that popularity lists can possibly contain malicious domains [80].

Some works (e.g. [7]) also synthesize the benign class by querying domains in bulk. Real DNS traffic, however, is vastly more diverse, which is why systems cannot be properly evaluated using this method.

For the malicious class, we commonly see artificially generated tunneling traffic in a controlled environment. This tunneling traffic originates in the majority of cases from off-the-shelf tunneling tools, but sometimes from custom tunneling applications developed by the researchers as well.

3.4. Detection Capability

Whether by means of supervised classification or unsupervised anomaly detection, detection systems label DNS traffic as either benign or malicious. Multiple metrics can be used to describe different aspects of detection performance. This section provides an overview of commonly reported performance metrics and how they are calculated. We further attempt to arrive at a definition of *detection capability* to effectively compare different DNS storage channel detection models.

Note that not all works report the following metrics over the same type of observation. Detection methods that use solely single-instance features report, for example, the number of correctly classified *queries*. Works that use only behavioral features would report metrics over their aggregations, e.g. the amount of correctly classified *domains*. It is possible for models to be able to detect few queries but many threats, or many queries but few distinct threats, and report high performance in either situation.

All metrics discussed in this section are based on the amount of correct and incorrect predictions. The four possible categories for binary classification are expressed in the *confusion matrix* in Table 3.2.

Table 3.2: Confusion matrix.

		Predicted	
		Benign	**Malicious**
Actual	**Benign**	True negatives (TN)	False positives (FP)
	Malicious	False negatives (FN)	True positives (TP)

The majority of reviewed papers report one or more of the following common performance metrics.

Accuracy $\frac{TP+TN}{TP+TN+FP+FN}$

(Classification) accuracy describes the ratio of correctly predicted observations, either benign or ma-

licious. Some works report the *classification error rate* or *classification error probability*, defined as $1 - accuracy$, instead.

Precision $\frac{TP}{TP+FP}$

Precision, or *positive predictive value* (PPV), describes the fraction of predicted malicious observations that is actually malicious.

Recall $\frac{TP}{TP+FN}$

Recall, *true positive rate* (TPR), *sensitivity*, or *detection rate*, describes how well a system is able to find all malicious samples.

F_1-score $\frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision}+\text{recall}}$

There is often a trade-off between precision and recall, i.e. higher precision leads to lower recall, and vice versa. The F_1-score is combines both metrics using their harmonic mean.

These metrics predominantly focus on the ability of a system to identify attacks (true positives). However, there is low tolerance for false positives in practice as well [75], given the scarcity of data exchange over DNS [62] and large DNS traffic volumes (e.g. [85] and our own research). The false positive rate is the most often reported metric in this regard.

False Positive Rate $\frac{FP}{FP+TN}$

The false positive rate (FPR), or *probability of false alarm* (PFA), describes the fraction of incorrectly classified samples from the benign class. *Specificity*, or *true negative rate* (TNR), is defined as $1 - FPR$.

The false positive rate is a key indicator of the feasibility of use of a model in practice. It is also "portable" and comparable between datasets, as it is invariant to the size of the benign class.

Next, the area under the *receiver operating characteristic* (ROC) curve, or AUC, is sometimes reported as well.

AUC
The ROC curve describes the sensitivity and specificity at all possible decision thresholds of a classifier. The area under this curve therefore describes the overall performance of a model. The AUC is "equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance" [34].

The AUC is a robust metric to describe model performance. It is, in contrast to e.g. precision and recall, insensitive to changes in class distribution [34]. However, higher AUC values do not necessarily result in better models, as "it is possible for a high-AUC classifier to perform worse in a specific region of ROC space than a low-AUC classifier" [34]. This is a relevant shortcoming for DNS storage channel detection, as we arguably only care about (and optimize for) the single best operating point of a model.

Finally, the **amount of identified threats** is a metric that is not often reported due to the fact that many works use few different research subjects. While it is similar to *recall* for per-domain aggregations, it is also relevant in situation where isolated queries are considered (e.g. [73]).

All metrics described in the section express different qualities of a model. There is, in summary, no consensus about a definitive method to evaluate *detection capability* in current literature. Besides the obvious goal of a system to detect as much threats as possible, authors also stress the importance a low false positive rate [62, 75].

Some metrics are ineffective or misleading when used in conjunction with imbalanced datasets and have to be carefully interpreted. Accuracy, for example, overstates model performance. Many of the described metrics are also influenced by either dataset size and/or class distribution. As such, different metrics may be more appropriate in different settings.

In conclusion, *detection capability* depends on three measures: the amount of true positives and false positives and the number of distinct identified threats. The relevant metrics depend on class distribution and dataset size.

3.5. Related Work

Besides the machine learning-based methods introduced in the previous sections, numerous other techniques exist to detect DNS storage channels. This section contains a brief overview of filtering, signature-based detection and other non-ML approaches.

Filtering is a method where successively applied hand-crafted rules are used to filter DNS traffic until a manageable subset of potentially malicious traffic remains. This result set is then either classified as malicious as a whole, or further examined by experts.

Paxson *et al.* [65] filter DNS traffic by excluding cached queries, uninteresting queries (based on timing, volume and payload entropy), estimated compressed information content and allowlisted domains. The system is evaluated on unlabeled DNS traffic of 230B real-world queries and iodine tunneling traffic. Tunnels with a traffic volume of more than 4kB per day could be found with a minimal burden on analysts.

Tatang *et al.* [81] present a similar approach, but only focus on newly-observed hostnames and use simpler features and filters. They first group their data by resource record and apply rules based on known domains, subdomain depth, query volume per fully qualified domain name (FQDN) and known special use cases to each group. Manual analysis of filtering results shows that the approach is effective, as numerous tunnels, both benign and malicious, are uncovered in a large, aggregated dataset of real DNS traffic.

Lastly, Aiello *et al.* [6] perform rule extraction by first clustering DNS traffic with k-means clustering and then extracting rules that capture most of the malicious traffic using a Logic Learning Machine.

Signature-based detection is method to find threats by matching specific characters or byte sequences in network traffic. Numerous signatures and rules are provided in non-scientific publications by Farnham [33], Sheridan and Keane [77] and Jaworski [44], to be used with e.g. the Snort intrusion detection system (IDS) and Splunk security information and event management software. The main drawback of signature-based detection is that only known threats can be detected.

Other non-machine learning methods rely on character frequency analysis [11, 69], statistical tests [5, 18, 28, 30, 42], are grounded in decision theory [15, 56] or use a manually defined decision boundary [19, 79].

3.6. Summary

The variety of features and methods identified in this survey demonstrates the feasibility of using machine learning-based systems to detect DNS connection tunneling and arbitrary data exfiltration. Many proposed methods are able to achieve high to very high detection rates in challenging scenarios.

We identify an emphasis on tunneling detection as opposed to low-throughput exfiltrations. All of the surveyed works are evaluated on tunneling traffic and few ([1, 16, 26, 62, 73]) on more difficult exfiltration traffic as well.

Effective features are derived from either isolated DNS query instances (payloads) or from sequences of queries describing behavioral traits. Behavioral features are extracted from aggregations sequences based on (a combination of) time, primary domain, origin IP or destination IP. Query sequence extraction reduces aggregations to produce one feature vector per aggregation, or uses a time-based or fixed-size sliding window to produce one feature vector per observation.

A multitude of payload-only features proposed in current literature describes one of five aspects: query space utilization, query structure, information density, lexical properties or DNS usage. Behavioral features are based on statistics of payload-only features (e.g. their sum or average), or storage channel characteristics (e.g. unique query ratio or number of unique labels).

We observed a clear distinction between proposed methods in terms of learning type. Unsupervised detectors are easily applied to real-world datasets and not inhibited by potentially problematic artificial ground truth datasets. However, these algorithms focus on anomalies, but benign traffic is diverse by nature and malicious traffic may potentially appear benign. Supervised detectors, on the other hand, attempt to learn a more intricate model of exfiltration training traffic, but need (preferably balanced)

annotated training data and risk overfitting on known threats.

A major concern is the lack of a common baseline (dataset). Studies cannot be compared directly, because *(i)* the problem they solve differs (e.g. detecting one specific type of tunnel or many different exfiltrations), *(ii)* the validation datasets vary in both size and diversity and *(iii)* incompatible or dataset-dependent performance metrics are reported. This makes it impossible to conclusively identify the state-of-the-art in this field.

Furthermore, the evaluation scenarios of some works can be considered unrealistic. It is apparent that artificial datasets that lack diversity or are too *clean* provide for a considerably easier detection problem.

The practical usefulness of presented approaches is another aspect to consider, as requirements besides a high detection rate may be required. A low false positive rate (FPR) is for example more important than a high detection rate in high-volume scenarios, when all reported alerts have to be processed by human analysts.

Finally, as established in Section 2.2.2, the DNS protocol is sometimes misused for legitimate data transfer. There is no consensus in literature about how to handle this traffic category. Many works ignore this aspect completely, other works (e.g. [16]) consider it – perhaps justifiably – as regular tunneling traffic, whereas still others mitigate the problem by ignoring known benign tunneling domains (e.g. [1]). It follows that the way this traffic is to be handled depends on the context in which the system operates.

3.6.1. Research Gaps

Having surveyed relevant works in the field of DNS storage channel detection, we identify the following two research gaps that are addressed in this thesis.

- Payload-only features are cheap and fast to extract, because they do not depend on other queries in the dataset and no state has to be kept. While detectors based on either payload or behavioral features show promising results, it is unclear to what extent behavioral features improve detection capability over payload-only methods.
- In similar fashion, both anomaly detection and supervised classification show promising results. However, their ability to detect a diverse set of storage channel threats at low false positive rates is difficult to establish, as current research cannot be compared due to the lack of a baseline dataset, different preprocessing and incompatible performance metrics. Few research compares both methods ([1, 67]), but arguably in a biased setting.

4

Datasets

To effectively evaluate DNS storage channel detection methods, sizeable datasets comprising both benign and malicious DNS traffic are required. This chapter describes all datasets used in this research. Having been provided a dataset of real-world, enterprise-level DNS traffic, we manually generate dataset for a variety of storage channel threats.

This chapter is structured as follows. The malicious dataset collection is described in Section 4.1. The preprocessing of both benign and malicious datasets is explained in Section 4.2. Section 4.3 then provides an overview and traffic statistics of all datasets used in this research.

4.1. Malicious Dataset Collection

In Section 3.1, we reviewed the landscape of threats considered in current literature. This analysis has shown that DNS storage channels are predominantly from one of two categories: connection tunneling or arbitrary data exfiltration. An effective detection strategy therefore has to be able to generalize over both threat categories. This requires a large and diverse dataset of threats for training as well as evaluation purposes. Unfortunately, few suitable and public datasets are available. As a result, we collect our own dataset of both DNS connection tunneling and arbitrary data exfiltration. The motivation, methods and research subjects are described in this section.

4.1.1. Motivation

Supervised classifiers require both large and diverse datasets and a reasonable balance between classes to be effective. Given a large dataset of benign DNS traffic, many malicious samples are needed to maintain an acceptable class balance. In practice, however, DNS tunneling is scarce.

Also, multiple different storage channel threats have to be considered to reduce bias towards a possibly small amount of threats in the training dataset. Artificially generated datasets may not consider diverse, real-world resembling scenarios and lack characterizing behavior.

Table 4.1 contains an overview of relevant open research datasets from current literature. For this research, only the timestamp and payload of DNS requests is required (see Chapter 5). Having reviewed existing open datasets, we conclude that most datasets do not provide the required level of detail, are too limited in size, are not diverse enough or contain only the author's extracted features.

Table 4.1: Publicly available datasets of DNS storage channel threats.

Source	Origin	Format	Suitable?	Reason
Ahmed *et al.* [1]	Artificial	CSV	✗	No timestamps, payload only.
Bubnov [14]	Artificial	CSV	✗	No timestamps, payload only.
Palau *et al.* [63]	Artificial	CSV	✗	No timestamps, payload only. Few tunneling samples.
Homem and Papapetrou [41]	Artificial	JSON	✗	Processed features only.
Berg and Forsberg [9]	Artificial	PCAP	✓	Packet capture from one DNS tunneling application.

Of these datasets, only the `dnscat2` dataset by Berg and Forsberg [9], comprising three long-lasting packet captures of DNS-tunneled file transfers, has the required level of detail. However, as considering only one DNS tunneling threat is too limited, we use this dataset only for evaluation purposes (see Section 4.3.3).

Besides academic datasets, numerous online sandbox environments exist (see [1]) in which malware can be safely executed and analyzed. For most DNS exfiltration malware, network packet captures are available. However, their behavior in sandbox environments is not representative when, for example, no credit card details are processed on the target systems – hence nothing is exfiltated – or when malware has sandbox evasion techniques. As a result, most packet captures are small in size.

We decide to create a new and diverse dataset of DNS traffic from real data exfiltration and tunneling threats, because *1)* there is a lack of suitable and public DNS storage channel datasets, *2)* packet captures from DNS malware in sandboxes are too limited and *3)* network traffic generated under the same conditions allows for a fair comparison of detection performance between different models.

4.1.2. Data Collection Experiments

As established in Section 3.1, the DNS storage channel threat landscape is diverse and the most prevalent threats are (open-source) connection tunneling tools and info-stealer malware. We design and enact a threat scenario with two tunneling tools and collect the resulting network traffic. For info-stealer malware, we implement and simulate four strains with different characteristics based on publicly available malware reports and simulate credit card exfiltration.

Connection Tunneling

DNS tunnels encapsulate an existing connection in DNS packets. Open-source connection tunneling tools *iodine* and *dns2tcp* have been selected for this research, primarily due to their prevalence in current literature, public availability, configurability and ease-of-use. We now briefly describe these tunneling tools and configuration parameters.

Iodine [29] tunnels IPv4 connections over DNS. The client application processes all network traffic flowing through a dedicated (TUN/TAP) adapter and constructs DNS queries to a configured domain. The server-side application is installed on the authoritative DNS server for the configured domain to process the incoming DNS requests and reply with downstream data.

Iodine has various configuration parameters. We vary the maximum query length, query type and upstream data encoding. Note that because iodine auto-negotiates the best available upstream encoding and does not provide a manual override option, we have compiled four different binaries with hardcoded upstream encoding from its source code.

While iodine by default tries to establish a direct connection between client and server (providing considerable speed advantages), we choose to disable this option and force all traffic to flow through a local DNS server.

Iodine is still in active development, but has no clear versioning system. We use the latest available version at the time of data collection: "`Git version 814a1fd`"[1] (2020), as the most recent stable release dates back to 2014.

Dns2tcp [27] forwards TCP ports via DNS. The server side publishes available services (ports) which can be consumed by clients. In contrast to iodine, dns2tcp requires no dedicated network adapter. According to the original authors, dns2tcp is able to achieve higher throughput, because smaller DNS packets are generated by encapsulating at the TCP level.

Dns2tcp supports fewer configuration options: it always uses Base64 for up- and downstream data encoding and the query lengths are fixed. However, the query type can be configured, as well as whether to use compression or not.

We use the most recent dns2tcp release at the time of data collection, `v0.5.2`.

The tunnel parameter grid used for data collection is described in Table 4.2. Note that query types are varied even though we only focus on the DNS query side for detection, because different resource

[1] `https://github.com/yarrick/iodine/commit/814a1fd7b0a6cf376f38bfc01056084c301d0873`

record types have a different maximum downstream capacity and as such influence DNS traffic characteristics. Also, the maximum query length for iodine has been extended from 100 to 150 when used together with Base32 encoding, as it proved to be impossible to establish a reliable tunnel with shorter queries.

Table 4.2: Connection tunneling parameters.

	iodine	**dns2tcp**
Query type	`MX`, `NULL`, `PRIVATE`, `SRV`, `TXT`	`KEY`, `TXT`
Max. query length	100|150, 255	—
Upstream encoding	Base{32, 64, 64u, 128}	—
Compression	—	No, Yes

The main purpose of our connection tunneling scenario (Scenario 1) is to simulate realistic workloads, with realistic durations and intervals. In order to capture real and diverse DNS tunneling traffic, an exfiltration script was designed that alternates between high-throughput data exchange and periods of no activity. The scenario is executed independently for every combination of experiment parameters for each tunneling subject.

This scenario highlights various aspects of DNS tunneling: prolonged maximum throughput, intermittent bursts and pauses. We perform a speedtest, exfiltrate files of different sizes with or without encryption, establish a SSH session with the victim and exfiltrate sensitive information, and perform regular internet browsing.

The scenario differs slightly between tunneling tools and parameter settings, because not all operations are always supported. For example, `ping` uses the ICMP protocol, which can be forwarded through the iodine IP tunnel, but not with the dns2tcp TCP forward. These optional exceptions are mentioned at their respective step.

Scenario 1 Connection tunneling

1: Generate files of 1K, 10K, 100K, 1M and 10M in size with random data to simulate sensitive data extraction.
2: Request a listing of available tunneling services (ports). *(dns2tcp only)*
3: Pause 20s.
4: Initiate tunneling software, perform handshake and set-up connection.
5: Pause 20s.
6: Ping server 10 times via the tunnel. *(iodine only)*
7: Pause 20s.
8: Perform a regular and reverse speed test. Sleep 30 seconds in between.
9: Pause 20s.
10: Transfer the randomly generated data files *without* encryption, using `netcat` (`nc`). Sleep 30 seconds between each transfer.
11: Pause 20s.
12: Transfer the randomly generated data files *with* encryption, using `scp`. *(iodine: only configurations with query length ≥ 150)*
13: Pause 20s.
14: Establish a SSH session between server and client and execute the following sensitive commands at two-second intervals. *(iodine only and query length ≥ 150)*

1. id
2. uname -a
3. ps -aux
4. env
5. ls -la ~
6. find / -perm /6000 2>/dev/null
7. cat /etc/passwd
8. cat /etc/group
9. cat /etc/shadow
10. ip a
11. ip route

15: Pause 20s.
16: Simulate internet browsing for 10 minutes.

The speed tests were performed with `iperf3`[2]. Regular internet browsing was simulated using a modified version of `web-traffic-generator`[3], which uses a headless web browser instead of direct requests to better resemble actual web usage, as not only the requested page is loaded, but also all dependencies (stylesheets, scripts, etc.). A selection of 15 popular web pages is the starting point for a recursive browsing journey, until a random depth is reached and a new starting page is selected. The customized web browsing code is made publicly available[4].

Arbitrary Data Transfer
While connection tunneling focuses on relaying an existing protocol as well as possible, arbitrary data exchange over DNS has no such restrictions. We have observed a variety of threats that use different query structures, data encodings and possibly encryption (see Section 3.1).

Point-of-Sale (PoS) info-stealer malware is a notorious abuser of DNS for data exfiltration. Installed on PoS-terminals, this malware collects payment details and exfiltrates them at low rate. We select four such malware strains, based on their popularity in current literature (e.g. [1, 62]) and distinct data encapsulation methods: BernhardPOS, FrameworkPOS, MULTIGRAIN and UDPoS.

It is important to exfiltrate "real" information using the actual exfiltration mechanism of the malware samples, because changes in encoded data are visible in the prepared exfiltration queries. For example, using random data instead of credit card details changes query characteristics and propagates to e.g. features based on entropy.

However, most actual malware samples are not publicly available, employ sandbox evasion techniques to prevent analysis or pose a significant risk to the researcher. Furthermore, it is difficult to obtain or resemble PoS terminal software to simulate payments. Therefore, we decide to reverse-engineer and implement only the DNS exfiltration methods of the four malware strains – based on public incident reports – and simulate exfiltrations using imitation credit card details and a given exfiltration schedule.

In order to understand PoS-malware, some background about credit card payments is required. Most PoS-malware silently monitors the memory of infected terminals for bytes resembling data from the magnetic stripe of credit cards. This stripe contains two relevant tracks: *track 1* and *track 2* [43]. Both tracks contain enough information to complete a transaction, i.e. card number, expiration date and security codes, but *track 1* data is more detailed and also contains for example the name of the customer.

The maximum record length of *track 1* data is 79 characters and 40 characters for *track 2* records. *Track 1* data may contain alphanumeric characters, whilst *track 2* can only contain digits. Both records fit comfortably within a DNS query.

Furthermore, the length (between 16 and 19 digits) and format of credit card number differs per vendor. We select three common schemes for our research: VISA (16 digits), MasterCard (16 digits) and MasterCard (19 digits). Naturally, no actual credit card details are used, but generated instead within the constraints of the selected schemes. We use Python *Faker* library[5] to this end. The credit card generating code is made publicly available[6].

Having established the global functioning of PoS-malware, we now provide an overview of the exfiltration methods of the selected malware samples. Note that this information is gathered from multiple public incident reports. When insufficient information is available, however, implementation details of the respective malware are at the discretion of the author.

BernhardPOS [38]
Simple info-stealer malware that exfiltrates only *Track 2* credit card data in a single DNS label. The credit card details are encrypted with an XOR cipher and subsequently Base64-encoded.

FrameworkPOS [31, 49, 57]
Malware family that has many different incarnations. We implement a variant that sends an install

[2] `https://iperf.fr/`
[3] `https://github.com/ReconInfoSec/web-traffic-generator`
[4] `https://github.com/tudelft-cda-lab/dns-storage-channel-detection/thesis-web-traffic-generator`
[5] `https://faker.readthedocs.io/en/master/providers/faker.providers.credit_card.html`
[6] `https://github.com/tudelft-cda-lab/dns-storage-channel-detection/thesis-ccgen`

beacon and possibly notices about running malware analysis software upon activation. Then, it listens for and exfiltrates *track 1* and *track 2* credit card information. Each exfiltration query contains a host identifier, campaign identifier, the name of the executed command and the actual payload. The payload section is hex-encoded after encrypted with a substitution cipher and XOR cipher.

MULTIGRAIN [22, 55]
Info-stealer malware derived from earlier versions of *NewPosThings* PoS-malware. Send an install beacon on successful activation and scrapes and exfiltrates *track 2* credit card details. Includes a host identifier in the payload to distinguish between targets. All information is encrypted using RSA with a 1024-bit key and encoded with Base32.

UDPoS [72, 82]
UDPoS is the most feature-rich malware strain we consider. On install, it collects and exfiltrates a large amount of sensitive information about the infected device. Then, it sends notices if any active process debuggers are detected and finally emits an install beacon. UDPoS exfiltrates both *track 1* and *track 2* credit card details. Exfiltration queries are encrypted with RC4 and hex-encoded. Analyzing real UDPoS queries revealed that each payload is terminated with newline characters (`\r\n`) before encryption.

Table 4.3 provides an overview of the encryption and data encoding techniques applied.

Table 4.3: PoS-malware data encryption and encoding techniques.

Name	Source	Encryption/Obfuscation	Encoding
BernhardPOS	[38]	XOR cipher	Base64
FrameworkPOS	[31, 49, 57]	Substitution + XOR cipher	Hex
MULTIGRAIN	[22, 55]	1024-bit RSA	Base32
UDPoS	[72, 82]	RC4	Hex

As mentioned, most samples exfiltrate more information besides credit card details, i.e. send install beacons, notices about running malware analysis programs, or sensitive information about the infected host. To generate an accurate reproduction of malware traffic, this behavior is simulated as well. Table 4.4 describes the malware behavior at different three stages: reconnaissance just after installation, beaconing after activation and actual data exfiltration. Each distinct query structure is denoted with an identifier (e.g. **> Notice**), for which sample queries are provided in Appendix A.3.

Table 4.4: PoS-malware exfiltration techniques.

	BernhardPOS	FrameworkPOS	MULTIGRAIN	UDPoS
Recon	—	Scan for running debuggers. **> Notice**	—	Collect and exfiltrate machine information. **> Info** Scan for running debuggers. **> Notice**
Install	—	Send install beacon. **> Install**	Send install beacon. **> Install**	Send install beacon. **> Install**
Exfil (repeat)	Collect *track 2* data. **> Exfil T2**	Collect *track 1* data. **> Exfil T1** Collect *track 2* data. **> Exfil T2**	Collect *track 2* data. **> Exfil T2**	Collect *track 1* or *track 2* data. **> Exfil**

Lastly, the exfiltration rate (or: schedule) has to be defined. We aim to simulate a scenario with realistic intervals, while still generating enough data to support supervised classifiers. Unfortunately, little credible and detailed figures about actual exfiltration campaigns are available.

Nadler *et al.* [62] face this problem as well and decide to simulate three credit card exfiltrations per second, based on the ballpark figure of 56M stolen credit card details in six months during the Home Depot campaign. Following this intuition, we select three different malware operating points to emphasize (extremely) low-throughput: once **per second**, once **per minute** and once **per five minutes**. Random jitter of up to 5% is added for a more realistic scenario.

Regarding malware implementation details, we use the `A` query type – resembling the actual malware queries – and simulate a valid and corresponding server response for completeness. All exfiltration queries are sent to the (locally emulated) primary domains used in their respective malware campaigns; none of which exists in the normal DNS dataset. The malware simulation code is made publicly available[7].

4.1.3. Network Set-Up

A local network of hosts is set-up for the collection the described malicious datasets. Three virtualized hosts are configured to resemble a real-world situation: a client (or: victim), server (or: adversary) and a local DNS server to facilitate the storage channels.

Figure 4.1 depicts the network set-up used for the collection of malicious datasets. We use Docker in conjunction with `docker-compose` to configure, deploy and execute the different containers and experiments: a *Local DNS*, *Client* and *Server* container.

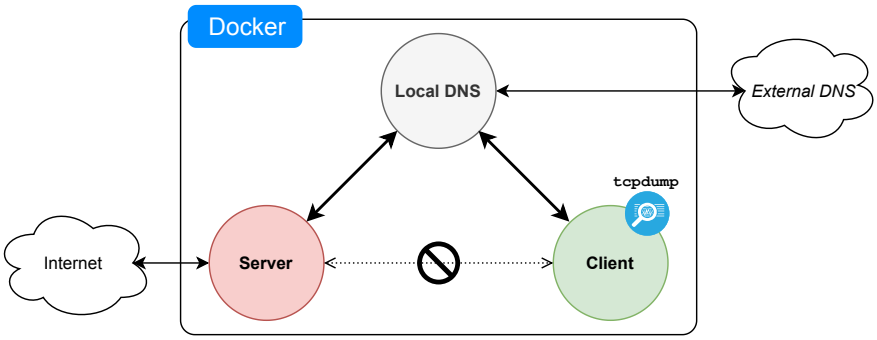


Figure 4.1: Containerized network set-up for the collection of malicious datasets.

Local DNS

The local DNS server (BIND[8] v9.14.12) is configured to delegate zones for the following malicious domain names to the “Server” container:

- tun[.]lan
- 29a[.]de
- ns[.]a23-33-37-54-deploy-akamaitechnologies[.]com
- dojfgj[.]com
- ns[.]service-logmeln[.]network

The local DNS is configured to reject DNS traffic to other domains. Tunneling scenarios that require web browsing resolve other lookups via public DNS server `1.1.1.1` (CloudFlare).

DNS BIND is a widely used DNS server implementation. We therefore consider the constraints imposed on queries and responses in our experiments representative of real-world situations.

Client

The Client container is considered the *victim* in data exfiltration scenarios and the *end-user* in tunneling scenarios. This container either runs the info-stealer exfiltration software or the client-side DNS tunneling software.

Direct traffic between the client and server containers is avoided, and filtered during preprocessing for completeness, such that all communication passes through the local DNS server.

[7] `https://github.com/tudelft-cda-lab/dns-storage-channel-detection/thesis-dns-malware`
[8] See `https://www.isc.org/bind/` for more information.

Server

The Server container is on the receiving end of the DNS storage channels. It hosts custom DNS servers to accept and answer DNS queries for the delegated domains. This is, for tunneling scenarios, the server-side tunneling application and for data exfiltration scenarios a custom DNS server written in Python.

This container is only connected to the internet in tunneling scenarios, to be able to e.g. load actual web pages during DNS-tunneled web browsing.

Network traffic is captured by means of `tcpdump` at the network adapter of the client container. All exfiltration queries are considered relevant to our malicious datasets and not just those that reach the adversary. The collected packet captures are then processed by Zeek (see Section 2.3) to generate traffic logs. We store the for each experiment individually generated `dns.log` files as final output of the data collection process. These log files are used used for further analysis and feature extraction in Chapter 5.

Limitations

The most significant drawback of this data collection method is that the capitalization of characters in DNS queries is lost due to Zeek v3.1.4 storing queries in lowercase. Starting with Zeek v3.2.0, however, the unmodified query is also preserved. This version was unfortunately not available at the time our datasets were collected.

As a consequence, the amount of information that was originally conveyed by a query is not necessarily the same in the one that is logged. This complicates the detection of data encoding in DNS queries, as e.g. Base32 and Base64 encodings now appear similar.

Furthermore, as all DNS data exchange happens on the same device and is directed via one DNS server, the latency is very low and packet drops are rare. In real-world scenarios, resolving a DNS request via possibly many intermediate DNS servers takes more time and has a higher risk of failing. The main consequence is that DNS retransmissions are rare in the collected datasets. Retransmissions may influence features based on the amount or ratio of unique queries.

4.2. Preprocessing and Filtering

The purpose of this processing phase is to prepare the datasets for feature extraction, by filtering redundant or invalid observations and parsing the DNS queries. A comprehensive overview is provided by Figure 4.2 in Section 4.2.5.

4.2.1. Cleaning

Both the collected malicious datasets and the corporate DNS traffic dataset (see Section 4.3.1) are in the common Zeek `dns.log` format. The Zeek logs are stored as plain text files and contain only ASCII characters for portability. Because the use of non-ASCII characters is discouraged by the DNS standard, this should – in theory – not pose a problem. In practice, however, most DNS servers accept non-standard characters in queries. Non-ASCII characters are stored by Zeek as unicode byte escapes. For example, a DNS query for "exämple.com" is logged as "ex**\xc3\xa4**mple.com". Because byte escapes prevent proper feature extraction, we convert escaped sequences back to characters.

Note that UTF-8 graphemes (symbols) may span multiple bytes. The escaped ä glyph from the example, for example, is constructed from two code points, even though it visually appears as one character. These byte escapes are decoded but not combined, as the query length is measured in bytes and not in symbols.

Non-ASCII characters are uncommon in regular DNS traffic. Conforming applications should convert such characters in domain names (i.e. internationalized domain names) to *punycode* [25]. The punycode equivalent of `exämple.com` is for example `xn--exmple-cua.com`. Tunneling tools, however, dispatch DNS queries without this conversion layer and purposefully use non-ASCII characters to achieve a higher information density. Domains in punycode are considered valid DNS usage and are left as-is.

Having processed the log files to obtain the original DNS query, the first filtering step consists of discarding any DNS requests without an actual query or with an empty query. The empty query, used to retrieve the DNS root server locations, can by definition of storage channels not contain any relevant information. Also, in edge cases when Zeek is unable to process a DNS transmission, an empty query may be logged that is filtered as well.

4.2.2. Parsing

A key preprocessing step is the parsing of DNS queries. We split a query into three parts: its subdomains, primary domain and public suffix domain (see Section 2.1.1). The purpose of this processing step is to allow analysis of different label characteristics, independent from the primary domain, as well as query aggregations per domain.

Although DNS queries have a clear structure, determining which parts belong to the suffix, primary domain or subdomains is not straightforward. Suffixes may span any number of labels, whose policy is determined by the respective top-level domain (TLD) registry.

To illustrate this problem, consider two top-level domains: `.nl` and `.co.uk`. Domain names may be registered directly below the `.nl` suffix. The governing body for the `.uk` TLD, on the other hand, has imposed a second-level hierarchy with namespaces for designated entities. Suffix `.co.uk` is for example intended for companies and `.ac.uk` for academic institutions.

These irregularities prevent naive parsing approaches like splitting on the two rightmost separators, as this would result in considering `co` the primary domain and `uk` the suffix for `example.co.uk`, instead of `example` and `co.uk`, respectively.

An additional circumstance is the existence of private suffixes. A private suffix is a lower-level domain that is used in practice as if it were a public suffix. Operators of those domains allow users to register their own domain names directly below it. For example, the Blogger blogging service by Google assigns domain names `[name].blogspot.com`. Considering `blogspot.com` the primary domain in this case would obfuscate the query streams to the underlying domains.

The all-encompassing solution to FQDN parsing is a suffix lookup table. The Mozilla Foundation maintains the canonical Public Suffix List (PSL) [61], which is a list of all known public and private suffixes.

The list is accompanied by a set of rules and algorithm[9] to determine which labels of a FQDN are to be considered the suffix. We parse the DNS queries in our datasets using this PSL, using the version that was current at the time the corporate DNS traffic dataset was collected, and Python package *TLDExtract* (version 2.2.3).

4.2.3. Filtering

A significant part of all DNS queries can be excluded from consideration beforehand, now that more granular information about subdomains and suffixes is available. All excluded entries would be labeled benign in a production setting.

Firstly, following the argumentation from Section 4.2.1, queries without subdomains are removed. Queries with only the `www` subdomain are discarded as well, given its prevalence and inability to convey data for DNS storage channels. This label is the de facto standard label for websites and is, for that reason, often hidden by contemporary web browsers[10].

Next, queries without a top-level domain (TLD) and queries for domains with the following internal or special-use suffixes are filtered.

- .intranet
- .internal
- .private
- .corp
- .home
- .lan[11]
- .arpa
- .example
- .example.com
- .example.net
- .example.org
- .invalid
- .local
- .localhost
- .test

The listed suffixes are reserved for (local) infrastructure [51, Appendix G] and/or special-use purposes [50]. Users cannot register domains below these suffixes by design and it follows that storage channels outside a network perimeter cannot exist using these suffixes. The same reasoning holds for queries without a TLD.

Then, the remainder of invalid queries are filtered by validating primary domains. Real and diverse DNS traffic may contain noise, broken or improperly structured queries and bogus transmissions. We have observed lookups to, among others, fully qualified URLs (with protocol and port number), email addresses, arbitrary byte sequences and invalid, non-existent domain names.

By our threat model and definition of storage channels, data can only be exfiltrated from a network to a valid primary domain. We use a lenient validation rule based on the domain name syntax in DNS specification [17, see Section 11] to filter all queries to non-conforming primary domains, using the following regular expression:

```
^([_]?[a-zA-Z0-9]+[a-zA-Z0-9\.-]*[a-zA-Z0-9]?)$
```

This rule forces primary domains to contain at least one alphanumeric character, start with an alphanumeric character or underscore and optionally contain hyphens or dots or end with alphanumeric characters. While this rule does not fully conform to the DNS standard – e.g. primary domains with a trailing dot or hyphen are not filtered – it is performant, comprehensible and filters the majority of unwanted noise.

The final filtering step removes fast retransmissions. A retransmission of the same packet occurs when the sender believes that the original packet was not received correctly. DNS predominantly uses the stateless UDP protocol, which does not acknowledge deliveries. Therefore, clients may re-send DNS queries multiple times when an answer is not received in time, or even as a preemptive measure.

Because retransmissions are essentially the same as the original query, we de-duplicate these queries. From a sequence of queries with the same origin IP, response IP, transaction ID, query and query type, observed within 100 milliseconds of each other, only the first query is kept.

Note that this method does not filter similar queries with a different transaction ID, which are considered deliberate duplicates.

[9] See `https://publicsuffix.org/list/` for additional information.

[10] For background on this matter, refer to `https://url.spec.whatwg.org/#url-rendering-simplification` and `https://bugs.chromium.org/p/chromium/issues/detail?id=883038#c114`.

[11] An exception is made for "`tun.lan`", which is used for the malicious tunneling dataset collection (see Section 4.1).

4.2.4. Dataset-Specific Processing

Malicious Datasets

Some DNS service discovery queries, unrelated to the experiment, were captured during the exfiltration and tunneling experiments. As traffic is captured at the client side of the exfiltration, queries that did not originate from the client device, but that were observed in the packet capture, are filtered.

Moreover, the exfiltration clients had internet access to facilitate DNS lookups during the web browsing phase. While any ensuing network traffic was sent via the DNS tunnel, these lookups originated from the client itself and are filtered from the final dataset.

Corporate DNS Dataset

Due to the corporate DNS dataset traffic capture method, the same DNS query could end up multiple times in the Zeek log file, outside the retransmission filtering window of 100 milliseconds. To mitigate this problem, we have identified major relay nodes in the dataset that process the majority of observed unique domain names. Only queries passing through these nodes are retained.

While this approach may discard valuable unique traffic to smaller DNS servers, it greatly reduces the amount of duplicate queries. We believe this filtering methods is valid, as it is highly probable that the vast majority of queries have passed through the selected nodes.

4.2.5. Summary

This section provided an overview of preprocessing and filtering steps applied to every dataset prior to feature extraction. During the first phase, the Zeek-generated DNS logs are cleaned by unescaping non-ASCII byte escapes and filtering empty or erroneously processed entries.

Next, the DNS queries are parsed into its primary domain, public suffix domain and subdomains. The subsequent filtering steps remove queries that cannot be part of a storage channel: queries without subdomains, with the “www” subdomain, with internal or special-use suffixes or invalid primary domains. Then, fast retransmissions of the same query are removed.

Finally, based on the type of dataset, malicious datasets are filtered to only retain relevant (storage channel) queries, and the benign dataset is processed to remove duplicate queries outside the retransmission window, that are artifacts of the data collection method.

Note that the filtering steps presented in this section have a negligible impact on the malicious datasets collected in Section 4.1, as the nature of the data collection methods ensures they are clean. These steps are aimed primarily at the corporate DNS dataset. For completeness and a fair comparison, however, all datasets have been preprocessed the same.

Figure 4.2 contains a complete overview of the data preprocessing pipeline.

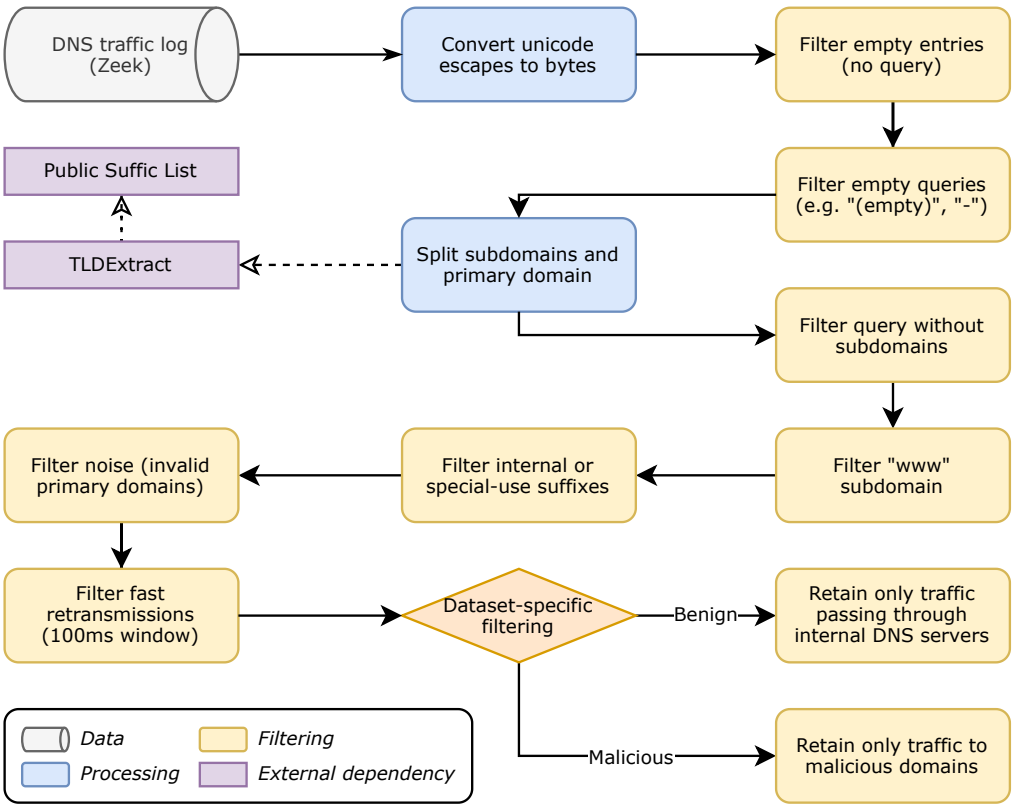


Figure 4.2: Dataset preprocessing pipeline.

4.3. Dataset Overview

This section presents an overview of all filtered and processed datasets used in the remainder of this research. Section 4.3.1 introduces the corporate DNS traffic dataset containing normal DNS traffic, Section 4.3.2 describes the collected malicious datasets and Section 4.3.3 contains an overview of unseen traffic samples used only for evaluation purposes.

4.3.1. Corporate DNS Dataset

We have been provided a large and unfiltered dataset of DNS traffic, originating from multiple corporate networks. The data collection period spanned three consecutive days (3x 24h) in 2020. The dataset is provided in the same format as the Zeek `dns.log` files generated in Section 4.1.

Traffic statistics of the corporate DNS dataset are included in Table 4.5. Besides the query volume per day, the number of (distinct) primary domains per day and queries per domain are provided as well, which are relevant to the feature extraction process described in Chapter 5. Due to privacy concerns, the dataset and actual DNS queries cannot be disclosed.

Table 4.5: Summary of the corporate DNS dataset.

Day	Queries	Unique primary domains (unseen)	Queries per primary domain		
			Min.	Mean ± Std.	Max.
Day 1	5.8×10^7	1.5×10^5 (100%)	1	$400 \pm 9.2 \times 10^3$	1.4×10^6
Day 2	7.0×10^7	2.1×10^5 (47%)	1	$324 \pm 1.0 \times 10^4$	2.5×10^6
Day 3	4.7×10^7	1.9×10^5 (31%)	1	$253 \pm 7.2 \times 10^3$	1.7×10^6
Total	1.8×10^8	3.1×10^5	1	$574 \pm 2.0 \times 10^4$	5.5×10^6

4.3.2. Malicious Datasets

All malicious datasets described in this section have been collected using the methodology described in Section 4.1.2. The experiments have been performed sequentially – no two experiments ran at the sime time – to prevent any interference. Our datasets are available online[12].

Table 4.6 describes the aggregated query statistics of the collected malicious datasets. A detailed dissection per dataset is available in Appendix A, Tables A.1 to A.5.

Table 4.6: Collected dataset statistics (aggregated).

Dataset	# config.	# queries	Mean queries / dataset	Duration (mean ± std.)	Max. burst / sec.
dns2tcp	4	1,711,049	427,762	19m ± 0m	6,814
iodine	40	10,765,229	269,131	22m ± 3m	2,403
BernhardPOS	3	43,807	14,602	11h 57m ± 2m	2
FrameworkPOS	3	43,784	14,595	12h 24m ± 2m	2
MULTIGRAIN	3	43,736	14,579	12h 24m ± 1m	2
UDPoS	3	44,153	14,718	12h 23m ± 2m	2

Data collection remarks

DNS-tunneled connections are not reliable. Query behavior is sometimes erratic, even in our controlled environment. This is especially visible in e.g. the data collection duration and number of queries for different datasets running the same tunneling scenario (see Tables A.1 and A.4).

Iodine, for example, may stall for an arbitrary period of time during operations and only transmit beaconing queries, outside a scheduled pause window. The experiments have not been re-run in an attempt to reduce or prevent this behavior, as we consider these irregularities real-world tunneling behavior and keep the datasets as-is. Figure 4.3, subfigure *iodine (2)*, clearly shows the effects of this behavior. The tunnel stalls for approximately two minutes at the `6m` mark and then again briefly at `10m`.

[12] `https://github.com/tudelft-cda-lab/dns-storage-channel-detection/thesis-malicious-dns-datasets`

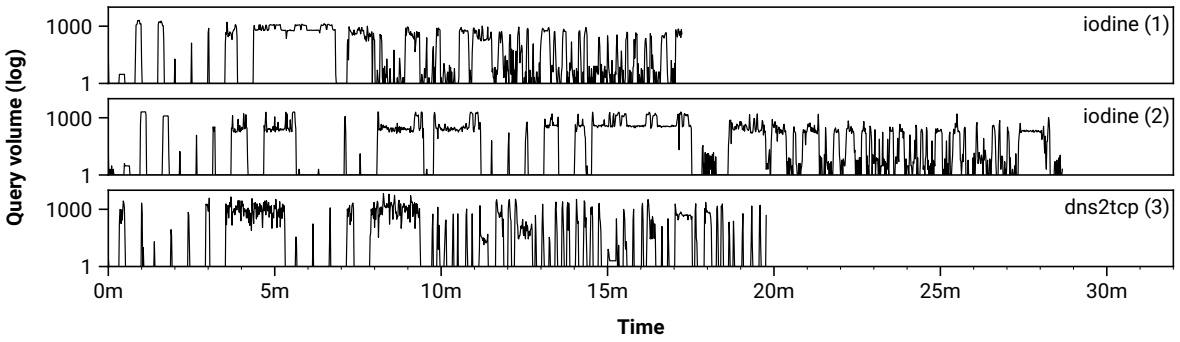


Figure 4.3: Time series plot of query volume (per second). The shortest iodine dataset has configuration PRIVATE / Base128 / 100 and is visualized in subplot (1), the longest iodine dataset has configuration MX / Base32 / 150 and is visualized in subplot (2), and the visualized dns2tcp dataset in subplot (3) has configuration TXT / no compression.

Furthermore, dns2tcp crashed regularly during the web browsing phase. As the cause of this issue could not be determined, the issue was mitigated by restarting the tunneling service as soon it crashed to continue browsing. As the web browsing phases of both iodine and dns2tcp show similar characteristics in Figure 4.3, the impact on the dataset quality is assumed to be minimal.

4.3.3. Unseen Storage Channel Threats

In addition to the generated malicious datasets, we assess the practical detection capability of our models with unseen threats. This set comprises traffic from the *dnscat2* tunneling tool, packet captures from real malware samples and custom exfiltrations based on the generated malware in Section 4.1. All datasets are processed as described in Section 4.2 as well.

dnscat2

Berg and Forsberg [9], as mentioned in Section 4.1.1, published a dataset of packet captures from the *dnscat2* tunneling tool. Although this dataset is deemed suitable for training purposes, we keep it for evaluation only, as all other malicious datasets are generated under the same conditions.

The dataset has been generated in a similar setting, as the authors “set up an authoritative DNS server (...) running the DNS tunneling software dnscat2, a set of virtual computers, hosted on one of our own computers, to control through the DNS tunnel” and record traffic with `tcpdump` [9]. The authors transfer a large file by DNS tunneling either the SFTP, SSH or TELNET protocol.

The traffic statistics of this dataset are provided in Table 4.7.

Table 4.7: Statistics of the dnscat2 dataset by Berg and Forsberg [9].

Tunneled protocol	# queries (unique)	Duration	Mean q/s	Max. q/s
SFTP	994,564 (100%)	43h 46m	6	20
SSH	1,358,727 (100%)	34h 51m	11	24
TELNET	1,148,598 (100%)	38h 44m	8	24

Preliminary analysis shows that, surprisingly, the mean and maximum amount of queries per second are significantly lower than our generated tunneling datasets. The evaluation of our models with this dataset will therefore prove their effectiveness on a tunneling threat with different characteristics.

Real malware samples

Up to this point, all malicious datasets considered have been generated in a controlled environment. Malicious DNS storage channels employed “in the wild” are presumably more diverse. For a complete assessment of our models, we therefore also consider network traffic from real DNS malware. These analyzed samples have generated too little traffic to be used for training purposes, but are suitable for model evaluation.

Saeli *et al.* [73] graciously provided the public sandbox reports of the malware used in their work. In these sandboxes, malware is executed in a controlled and monitored environment and all network activity is recorded. We download the packet captures for each malware strain and process them accordingly. Table 4.8 summarizes these datasets.

Three malware strains fall outside our defined threat model. *Denis* uses multiple domains for exfiltration. *Carbanak* and *Cobalt Strike* are DNS payload droppers, which download malicious data via DNS responses. Nevertheless, we include these samples to assess the effectiveness against related threats as well.

Table 4.8: Dataset statistics – sandbox samples

Malware variant	Encoding	Domain(s)	# queries (unique)	Duration	Mean q/s	Max. q/s
BondUpdater	Hex	withyourface[.]com	23 (100%)	<1m	6	12
Carbanak	–	google4-ssl[.]com	791 (100%)	2m	6	7
Cobalt Strike	–	cisc0[.]net	262 (100%)	3m	1	17
Denis	Base64	teriava[.]com tulationeva[.]com notificeva[.]com vieweva[.]com	23 (100%)	<1m	1	1
DNSpionage	Base32	0ffice36o[.]com	13 (31%)	<1m	1	6
ISMDoor	Base64	basnevs[.]com	2,296 (26%)	4m	8	58
Pisloader (1)	Base32	it-desktop[.]com	13 (100%)	2m	0	1
Pisloader (2)	Base32	it-desktop[.]com	78 (33%)	4m	0	3
UDPoS	Hex	service-logmeln[.]network	121 (100%)	1m	1	2

Plain-text exfiltration
Finally, we implement a custom DNS credit card exfiltration scheme using the same experiment setup as the simulated malware in Section 4.1.2. All four samples described in that section use encryption and data encoding. However, credit card details contain predominantly LDH-characters and can be embedded with minimal processing in DNS queries. The intuition is that “hiding in plain sight” may be an effective strategy to circumvent models that focus on encoded and/or encrypted data.

Table 4.9 summarizes this final evaluation dataset. We exfiltrate both *track 1* and *track 2* credit card data, in the format described by the following two sample queries:

- `4453359077797003.12256437021961772782` .example.com
- `4719942928801031.patricia-jacobson.11205500` .example.com

Table 4.9: Dataset statistics – custom plain text exfiltration

Malware variant	Schedule	Duration	# queries (unique)	Mean q/s	Max. q/s
Plain Text	Per second	11h 59m	42,948 (100%)	1	2
	Per minute	11h 59m	720 (100%)	0	1
	Per 5 minutes	11h 55m	144 (100%)	0	1

4.4. Summary

This chapter introduced the datasets used in this research. Several malicious datasets are generated by means of existing connection tunneling tools and simulating DNS data exfiltration malware. Section 4.1 provides an overview of the data collection methodology and experiments. Section 4.2 describes the preprocessing methods used to clean, parse and filter all datasets. Clean and consistent data is important for the feature engineering process addressed in the following Chapter 5. Finally, a summary of all datasets and their traffic statistics is provided in Section 4.3. Besides the aforementioned collected malicious datasets, a sizable dataset of real-world corporate DNS traffic is introduced, as well as a collection of unseen samples used solely for evaluation.

5

Methodology

In Chapter 3, we surveyed existing detection methods for DNS storage channels. It followed that existing detection mechanisms use different feature engineering rationales and different kinds of machine learning algorithms. The main distinction between these categories is made between supervised classification and unsupervised anomaly detection, and features extracted from single query instances or sequences of queries.

Unfortunately, directly comparing existing approaches is difficult due to the use of proprietary datasets and incompatible performance metrics. As a result, it was necessary to design a novel DNS storage channel detection framework which allows us to analyze both the effects of different feature sets and machine learning methods, in order to answer the following two research questions:

SQ2: What is the difference in detection capability between unsupervised anomaly detection and supervised classification?

SQ3: What are the effects of considering only payload features, only behavioral features or using composite feature sets?

To the best of our knowledge, no other research has compared the effects of both different feature extraction methods as well as detection methods for DNS storage channel detection.

This chapter is structured as follows. First, in Section 5.1, the main assumptions about our datasets and detection methods are stated. Then, the feature engineering process and feature sets used for detection are described in Section 5.2. The modelling methods and algorithms used are described in Section 5.3. Lastly, the experiment design, evaluation metrics and configurations are provided in Section 5.4.

5.1. Assumptions

Besides the constraints imposed by the scope and threat model (see Section 1.2), we make the following additional assumptions about our problem setting.

Firstly, we assume that covert DNS storage channels are detectable in DNS traffic using only their timestamps and DNS query names. The intuition is that these characteristics are significantly different from regular traffic: either due to longer query lengths, a high amount of (unique) queries, an abnormal amount of labels, or the use of encoded or encrypted subdomains. Our feature survey (Section 3.2) identified many such features in current literature. Using this limited amount of information allows for fast extraction of comprehensible features and limits adversarial evasion possibilities (see Section 5.2.1).

Next, we presume the corporate DNS dataset to be attack-free. Based on the low probability of DNS tunneling and exfiltration in the wild and manual inspection of the dataset, it is deemed highly unlikely

that the dataset contains storage channels with malicious intent. We additionally verified that the corporate DNS dataset does not contain any of the malicious domains used by our simulated malware.

However, *benign* DNS storage channels may be present in real DNS traffic. This phenomenon is also observed in several other works [1, 16, 62, 73]. Filtering benign storage channels beforehand is challenging: their prevalence is expected to differ between networks and expert knowledge is required to correctly identify them. Removing benign storage channels beforehand might cloud the performance evaluation, as it is influenced by the quality of the filtering.

While we acknowledge the presence of benign storage channels in the corporate DNS dataset, the total contamination is presumed to be low and insignificant. We choose to ignore their effects during training and evaluation.

Lastly, we assume that it is impossible to obtain usable information about end-users from the corporate DNS traffic dataset. Due to the nature of the data collection method, the DNS traffic cannot reliably be attributed to either a single user or a device that processes traffic of multiple users. The total amount of users in the dataset is, as a result, unknown as well. This is common for traffic captured at different locations in large and complex networks. The main consequence is that the possible aggregation levels for behavioral features are limited.

5.2. Feature Extraction

This section describes the feature engineering methods used to describe covert DNS storage channels. We create three different feature sets based on either single query instances or query sequences, in order to assess their relative performance.

First, we briefly describe adversarial evasion possibilities and our preemptive mitigation strategies in Section 5.2.1. The features used for detection are then designed and analyzed in Section 5.2.2 (payload-only features) and in Section 5.2.3 (behavioral features).

5.2.1. Robustness

Under our threat model, an attacker has complete control over both the DNS queries as well as responses, with the only restriction that they are valid by the DNS standard. Many aspects of a DNS packet can be trivially altered by an attacker – e.g. query type, flags in the header section, response code, response type, response content, etc. – in an attempt to evade detection, without impacting exfiltration ability. Current literature does not always take this into account.

For example, Saeli *et al.* [73] ignored unsuccessful DNS lookups and assume that no exfiltration can take place with those queries. However, as the response code is controlled by the attacker, any response to an exfiltration query can be marked “unsuccessful” to evade detection. At that point, the payload has already reached the malicious DNS server.

Buczak *et al.* [16], Nadler *et al.* [62] considered query type frequencies per domain. Query types are only relevant to the format and capacity of the DNS answer, and have therefore no influence on the exfiltration capability of an attacker, as the format of DNS queries is the same regardless of query type. The malware we use for training, for example, all use the common `A` query type.

Both examples demonstrate the importance of only taking into account properties of DNS transmissions that are difficult to modify without incurring a significant cost in terms of exfiltration capability.

By definition of DNS storage channels, any information has to be exfiltrated via the DNS query. Furthermore, every DNS request is sent at a defined point in time, which is recorded by the observer such that the order of a sequence of requests can reliably be determined, without possible adversarial modification by the sender.

We therefore consider the payload and timestamp of a DNS request the definitive information source for covert DNS storage channel detection. All other aspects of DNS requests and responses can be modified by an attacker without impacting exfiltration capability. Modifying the query or timing between requests, however, directly influences the exfiltration throughput and/or covertness.

An added benefit of this approach is that our method can be used with packet capture, (Zeek) traffic analysis logs or DNS server logs, since each data source contains the required information.

5.2.2. Single-Instance Features

We first consider features extracted from single query instances, or *payload features*. These features have an inherent computational advantage over query sequence features because no state has to be kept.

The payload features presented in this section are based on proven effectiveness and storage channel characteristics as described in current literature (see Section 3.2.1). We argue that the most important payload features are based on query space utilization, information density and lexical properties of queries. These characteristics are valid in any network setting (i.e. are not domain or language-dependent) and capture both *theoretical capacity* and *actual use* aspects of the covertness proposition (Section 2.2).

The influence of primary domains (with different lengths) is removed by only considering subdomains for feature extraction. That is, any lengths, character frequencies, etc. are based on the "`subdomains`" portion of `subdomains.example.com`. Furthermore, the label-connecting dots are ignored for any lexical features. This results in near-identical features for queries with the same payload but different primary domains.

Some works (e.g. [52]) do consider the primary domain a valuable source of information, from e.g. the intuition that malware authors use disposable or algorithmically generated domain names for exfiltration. In our experience, real-world covert DNS channels use readable and inconspicuous domain names to appear benign, using for example typosquatting or combosquatting ([46]). Most importantly, however, the domain name is fully controllable by the attacker and is easily changed to evade detection.

We define a payload feature extraction function fe_p to transform a single DNS query $Q_i = (S_i, D_i)$ to a feature vector, where S_i is the subdomain portion of the query and D_i the primary domain. S_i may be split on the label-separating dot to obtain the tuple of subdomain labels $L_i = (l_{i,1}, l_{i,2}, \ldots, l_{i,n})$. The concatenated payload, equal to S_i without the label-separating dots, is denoted as $P_i = l_{i,1} \,\|\, l_{i,2} \,\|\, \ldots \,\|\, l_{i,n}$.

We extract the following eight features from single query instances, defined in Equations 5.1 to 5.8.

(i) **Number of unique characters**
The amount of distinct characters in the payload.

$$fe_{p,0}(Q_i) = |\{ c : c \in P_i \}| \tag{5.1}$$

(ii) **Unique character ratio**
The ratio between the amount of distinct characters and total characters in the payload.

$$fe_{p,1}(Q_i) = \frac{|\{ c : c \in P_i \}|}{|P_i|} \tag{5.2}$$

(iii) **Number of digits**
The amount of distinct characters in the payload.

$$fe_{p,2}(Q_i) = |\langle c \in P_i : isdigit(c) \rangle| \tag{5.3}$$

(iv) **Number of non-standard characters**
The amount of non-alphanumeric, non-hyphen and non-underscore characters in the payload.

$$fe_{p,3}(Q_i) = |\langle c \in P_i : \neg isalphanum(c) \wedge \neg ishyphen(c) \wedge \neg isunderscore(c) \rangle| \tag{5.4}$$

(v) **Average subdomain length**
The average length of a subdomain in the payload.

$$fe_{p,5}(Q_i) = \sum_{s \in L_i} \frac{|s|}{|L_i|} \tag{5.5}$$

(vi) **Maximum subdomain length**
The maximum length of a subdomain in the payload.

$$fe_{p,6}(Q_i) = max\{\mid s \mid : s \in L_i\} \tag{5.6}$$

(vii) **Entropy**
The (Shannon) character entropy of the data payload. Note that the effectiveness of this feature is impacted by the lowercasing of DNS queries (see Section 4.1.3), as there are 26 less (uppercase) characters to observe.

$$fe_{p,7}(Q_i) = -\sum_{c \in P_i} \Pr(c) \cdot \log \Pr(c) \tag{5.7}$$

(viii) **Fill ratio**
The payload length (including label separators) divided by the theoretical maximum amount of payload space. The maximum payload space is determined by subtracting the length of the primary domain and the payload-connecting dot from the maximum query length of 253 characters (see Section 2.1.2).

$$fe_{p,8}(Q_i) = \frac{\mid S_i \mid}{253 - \mid D_i \mid - 1} \tag{5.8}$$

5.2.3. Behavioral Features

While we consider payload features effective at describing anomalous single queries, they may not describe storage channels using many inconspicuous queries well. Instead of maximizing storage channel throughput by constructing long and anomalous queries, an attacker can exfiltrate a given amount of information just as well with shorter queries, that have ordinary information entropy and no invalid characters, by using more queries and/or shorter intervals.

Moreover, payload features may cause large amounts of false positives or false negatives for benign observations that resemble malicious observations, and vice versa. Ahmed *et al.* [1], for example, report a significant amount of false positives using anomaly detection with payload features.

In order to detect low-and-slow exfiltrations and to better characterize benign observations, we look beyond isolated payloads and extract features derived from local context. By considering sequences of queries, latent exfiltration behavior can be recognized even when isolated queries appear benign.

Defining Context

The global aggregation levels identified in Chapter 3 are either based on time, users or primary domains. A per-time grouping considers queries within a certain time period, a per-user grouping considers queries to (or from) distinct IP addresses and a per-domain grouping considers queries to distinct primary domains. Based on the following considerations, we propose a method that incorporates context by grouping DNS requests per primary domain:

- Features from per-domain sequences are not influenced by temporal characteristics of a network, e.g. differences in traffic volume at different times of day or between different users.
- Storage channels are by definition isolated from other traffic in a per-domain aggregation, as they use a single domain name under our threat model. Detection at this level would enable network administrators to block a domain name as soon as an exfiltration is detected.

Considering all queries in a aggregation at once, however, limits the usability of the detection system, as all data has to be present beforehand, analysis of streaming data is no longer possible and keeping a full query history per domain name is costly. Also, temporal peculiarities or outliers may be smoothed away.

We therefore propose a feature extraction method based on local context, by continuously considering a limited query history per aggregation. Per-domain sequences are analyzed with sliding windows

that contain for a given observation either a fixed amount of queries λ, or all queries from the last δ seconds (including the current observation).

The intuition behind two different sliding window types is that time-based windows are assumed to be effective at detecting query bursts – often observed in tunneling traffic – and that fixed-length windows are effective at detecting low-and-slow exfiltrations with significant pauses between queries – as is the case with info-stealer malware.

Note that the sliding feature extraction method produces one feature vector per original observation. This enables us to create different compositions of feature sets – also with payload features – and compare their effectiveness.

Feature Extraction

For behaviorial feature extraction, both DNS query Q and corresponding timestamp T of a DNS request tuple $R_i = (T_i, Q_i)$ are required. The sequence $G_{D_i} = (R_0, R_1, \ldots, R_n)$ consists of all observations for a primary domain D_i, ordered in time.

We define a behavioral feature extraction function fe_b to transform a per-domain sliding window to a feature vector. The size of the sliding window is defined by either a time delta δ in seconds or by a fixed number of queries λ. The sequence of observations in a sliding window for primary domain D_i at index u is given by Equation 5.9 for time windows and by Equation 5.10 for fixed-length windows.

$$W_{\delta,u}^{D_i} = \{R_j : R_j \in G_{D_i} \wedge (T_u - \delta) < T_j \leq T_u\} \tag{5.9}$$

$$W_{\lambda,u}^{D_i} = \{R_j : R_j \in G_{D_i} \wedge (u - \lambda) < j \leq u\} \tag{5.10}$$

Fixed-length sliding windows are expanding until the maximum amount of observations λ is reached. That is, after the first query, a $\lambda = 2$ window contains one query, after two queries, two, and after three queries, again two, etc.

In cases where the extraction procedure is indifferent to the sliding window type, i.e. $fe_{b,i}(W_\delta) = fe_{b,i}(W_\lambda)$ for a feature i, the window type is omitted for brevity and the sliding window is denoted by W. Also, building on the notation introduced in Section 5.2.2, the following convenience notations are used:

- $S_W = \bigcup_{R_j \in W} S_j$ is used to denote the set of unique subdomain portions (i.e. the query without primary domain) of queries in a sliding window W.
- $L_W = \bigcup_{R_j \in W} L_j$ is used to denote the set of unique subdomain labels in a sliding window W.
- $p_W = ||_{S_j \in S_W} P_j$ is used to denote the concatenation of unique data payloads in a sliding window W, where $||$ denotes the concatenation operation. The order in which the payloads are concatenated is not important as we only consider the total length and entropy.

We extract six features (Equations 5.11 to 5.16) from both sliding time and fixed-length windows and two features (Equations 5.17 and 5.18) from time windows only. The behavioral feature sets comprise features of proven effectiveness in current literature and emphasize the uniqueness, volume and information transfer of storage channel queries.

Because fixed-length windows have variable time duration, any feature based on time is unstable: the duration may either approach zero for high-traffic domains, or become arbitrarily large. The *unique query rate* and *unique transfer rate* are therefore only calculated for time windows.

(i) **Number of unique subdomains**
The amount of distinct subdomains in the sliding window.

$$fe_{b,0}(W) = |L_W| \tag{5.11}$$

(ii) **Entropy**
The character entropy of the concatenated payloads in the sliding window.

$$fe_{b,1}(W) = H(P_W) \tag{5.12}$$

(iii) **Average unique subdomain length**
The average length of unique subdomains in the sliding window.

$$fe_{b,2}(W) = \sum_{s \in L_W} \frac{|\, s \,|}{|\, L_W \,|} \tag{5.13}$$

(iv) **Fill ratio (unique)**
The fraction of the theoretical maximum amount of payload spaced filled in the sliding window. Only unique queries in a window are considered.

$$fe_{b,3}(W^{D_i}) = \frac{|\, p_W \,|}{|\, S_W \,| \cdot (253 - |\, D_i \,| - 1)} \tag{5.14}$$

(v) **Maximum subdomain length**
The maximum length of a subdomain label in the sliding window.

$$fe_{b,4}(W) = max\{|\, s \,| : s \in L_W\} \tag{5.15}$$

(vi) **Unique query ratio**
The fraction of queries that is unique in the sliding window.

$$fe_{b,5}(W) = \frac{|\, S_W \,|}{|\, W \,|} \tag{5.16}$$

(vii) **Unique transfer rate**
The amount of data (characters) transmitted per second.

$$fe_{b,6}(W_\delta) = \frac{|\, p_W \,|}{\delta} \tag{5.17}$$

(viii) **Unique query rate**
The amount of unique queries per second.

$$fe_{b,7}(W_\delta) = \frac{|\, S_W \,|}{\delta} \tag{5.18}$$

Example
Consider the following FrameworkPOS queries from the malicious dataset collected in Section 4.1. The gap between the first beaconing query and subsequent exfiltration queries highlights the significance of two different window types.

Timestamp	Query
0m 0.00s	`4940a08c.grp1.ping.adm.cdd2e(…)dd2cd.cdc4c(…)9defe.f0e197ecfdec.ns` .example.com
25m 57.06s	`4940a08c.grp1.tt2.c8fed0cd(…)dd2c8c8dc.d2c4fefed(…)e9e9e9e9e9.ns` .example.com
25m 58.07s	`4940a08c.grp1.tt1.dcd(…)4fc.99c(…)8c8.cdc(…)9e9.c4c(…)4d0.ns` .example.com
25m 59.09s	`4940a08c.grp1.tt2.dcdcfed2(…)ecddccdc8.d2fcfed0d(…)e9e9e9e9e9.ns` .example.com

The behavioral features extracted from these queries using either a three-query window or three-second window are included in Table 5.1. The first observation is that both windows have aggregated different amounts of queries after different queries. This is visible in the resulting features to varying extents, as for example the number of unique labels is impacted, but the maximum label length is not.

Furthermore, this example illustrates the erratic behavior of the *unique query rate* and *unique transfer rate* features in the fixed-length sliding window. The large gap between the first and second query and the subsequent small intervals make these features unstable and are discarded as a result of this.

Table 5.1: Comparison of sliding window features.

(Queries in window)	Number of unique labels	Unique Query Rate	Entropy	Unique Transfer Rate	Average Unique Label Length	Unique Query Fill Ratio	Maximum Label Length	Unique Query Ratio
Fixed window, $\lambda = 3$								
Query 1 (1)	8	*(10.000)*	3.610	*(790.000)*	9.875	0.410	26	1.000
Query 2 (2)	11	*(0.001)*	3.532	*(0.121)*	15.818	0.476	60	1.000
Query 3 (3)	16	*(0.002)*	3.500	*(0.246)*	22.188	0.638	60	1.000
Query 4 (3)	13	*(1.478)*	3.385	*(206.494)*	29.846	0.692	60	1.000
Time window, $\delta = 3s$								
Query 1 (1)	8	0.333	3.610	26.333	9.875	0.410	26	1.000
Query 2 (1)	6	0.333	3.368	36.333	18.167	0.543	60	1.000
Query 3 (2)	11	0.667	3.407	101.333	26.364	0.752	60	1.000
Query 4 (3)	13	1.000	3.385	139.667	29.846	0.692	60	1.000

5.2.4. Feature Distribution

The following plots summarize the probability densities of the extracted features, per class. Figure 5.1 visualizes payload-only features, Figure 5.2 contains features extracted from a sliding time window of length *2 seconds* and Figure 5.3 contains features from a sliding fixed-length window of size *20*. An overview of the probability density for every feature set used during experiments (see Section 5.4.3), as well as a dissection of the distributions for each group within the malicious class, is provided in Appendix B.

For the benign class, only features extracted from traffic of the second and third days are used, to reduce the effect of novel primary domains and empty sliding windows. For the malicious class, all *dns2tcp* and *malware* datasets are included, and for *iodine* the datasets with query types `TXT` and `NULL`, to improve the balance between queries per group in the malicious class.

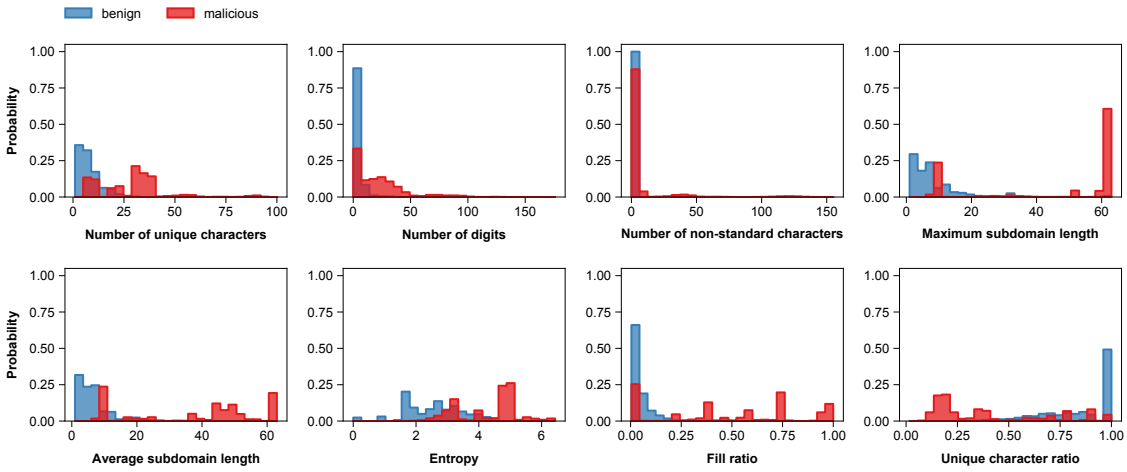


Figure 5.1: Feature distribution, payload-only features

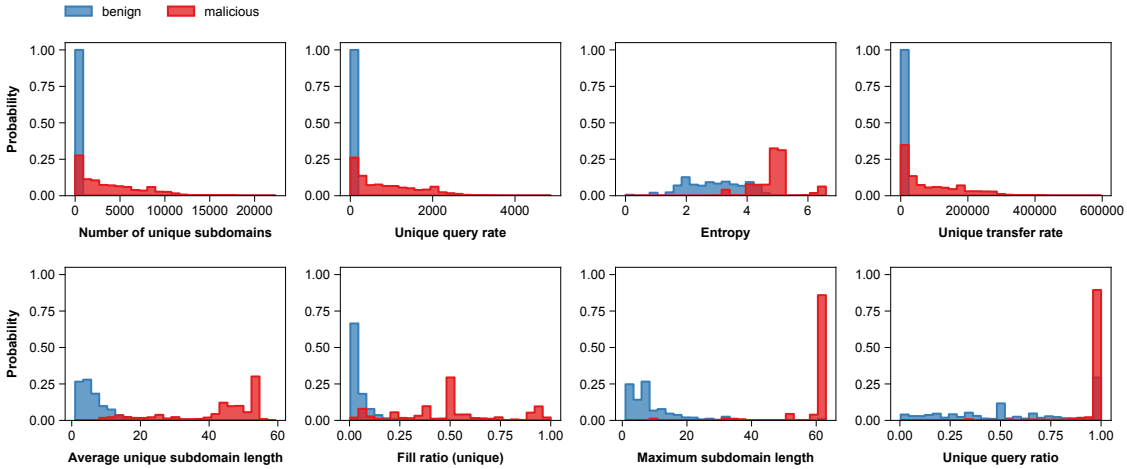


Figure 5.2: Feature distribution, time window (δ = 2 seconds)

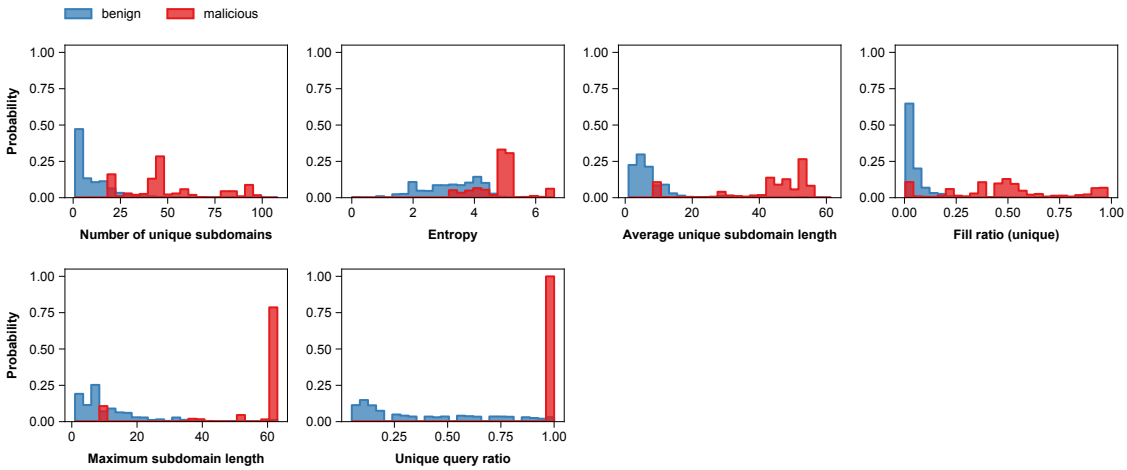


Figure 5.3: Feature distribution, fixed-length window (λ = 20)

Although single features do not have to be discriminative by themselves and algorithms use combinations of features to model the input space, these density plots show that the selected features are clearly descriptive of the benign and malicious classes.

For payload-only features, especially the *maximum subdomain length* and *fill ratio* features – describing query structure and query space utilization – discriminate well between classes. Some sliding window features are highly descriptive as well: for example, the feature values for the *number of unique subdomains* in the sliding time window and *unique query ratio* in the fixed-length window predominantly fall within one histogram bin for the malicious class. The feature distributions between sliding window types differ as well, which make it interesting to evaluate different window combinations.

Clear separations per feature are especially useful for anomaly detection, as it shows that the "anomalies" (malicious class) are sufficiently distant from the benign class to prevent swamping and masking, and that there is little overlap between classes. Overall, the feature sets look promising for use with both classification and feature selection.

5.3. Modeling

Current literature shows promising detection results in both supervised (two-class) classification and unsupervised anomaly detection settings. We use (combinations of) the feature sets proposed in the previous section to model either only the benign class for anomaly detection or both benign and malicious classes for classification.

In this section, we first describe the algorithm selection criteria (Section 5.3.1) and then introduce the Random Forest (Section 5.3.2) and two Isolation Forest algorithm variants (Section 5.3.3).

5.3.1. Algorithm Selection

Every machine learning algorithm has its advantages and disadvantages. The following constraints and requirements apply to our research:

- We perform a considerable amount of experiments, considering multiple algorithms and feature set compositions. Combined with the size of the training set and hyperparameter optimization, algorithms that are slow in terms of training and/or evaluation cannot be used for our experiments.
- Detection models should generalize well over all threats in the class of DNS storage channels, as well as unseen threats.
- Many features are derived from current literature. Algorithms that have been shown to perform well in current research are preferred.
- Given the difference between classification and anomaly detection, algorithms that require no or similar feature processing (e.g. scaling or normalization) are preferred for a fair comparison.

The main algorithm selection criteria are therefore speed, generalization performance and proven effectiveness in current research, and if possible using the same feature processing. Tree-based algorithms satisfy these requirements.

Tree-based Models

Datasets can be modeled with *decision trees*, which recursively split the input space in unique regions that belong to a class. Starting from the root of the tree, each node represents a decision rule to split the data based on one feature, to arrive at the leaves which represent class labels. Learning a decision tree means determining the best split at a node for a given subset of remaining training samples. Different implementations use different metrics for "best", but *information gain* (entropy) or *Gini impurity* are most commonly used.

Fully grown decision trees describe the training data very well, but are highly sensitive to changes: small changes in the data may result in a completely different tree. They are also prone to overfitting, which can be mitigated by e.g. reducing the maximum tree depth or increasing the minimum required samples to remain per leaf.

While single decision trees do not fit our algorithm selection criteria, numerous tree-based algorithms for classification as well as anomaly detection algorithms do suit our needs. These algorithms improve on the shortcomings of decision trees by combining many "weak" trees trained on a sample of training data

and/or a subset of features, to create one strong learner. Advantages of these tree-based ensembles are:

- Fast training and evaluation.
- High generalization power.
- Less prone to overfitting.
- Promising performance in current literature (e.g. [1, 16, 62]).

An added benefit is that decision trees do not require normalized or scaled features, as each split is based on independent features. Also, decision trees require no costly distance calculations between observations, which is beneficial when using large training datasets.

Tree-based algorithms exist for both supervised classification and anomaly detection. We select the *Random Forest* algorithm for the supervised setting and two *Isolation Forest* variants for anomaly detection.

5.3.2. Random Forest

The supervised Random Forest classifier [13] combines multiple decision trees that are trained on different (random) subsets of features and training samples. Each tree independently predicts the output class, and the majority voted wins. The basis of this algorithm lies in bootstrap aggregating (or: *bagging*): aggregating the decisions of many trees trained on random samples, with replacement, from the training data. Random Forests additionally sample the features available for each split.

The main advantages of Random Forests are their ability to handle large and high dimensional datasets (by sampling) and that they are less prone to overfitting. Furthermore, the feature importance can be determined by measuring which features produce the most valuable splits. However, care has to be taken when interpreting importances, as feature correlation influences the importance. Lastly, a practical benefit of using many independent trees is that computations can be trivially parallelized for faster training and evaluation.

Disadvantages of Random Forests are moderate complexity for models with many deep trees (slower training and evaluation speeds) and difficult to interpret models (black boxes). Neither aspect is problematic for our research, as the model complexity is restricted by our choice of hyperparameters, the training speed is deemed sufficiently fast and explainable models are not a requirement.

5.3.3. Isolation Forest

The unsupervised Isolation Forest (iForest) anomaly detection algorithm [53] combines multiple independent *isolation* trees. Similar to Random Forests, the algorithm uses bagging to train trees on sampled training data. The trees, however, resemble decision trees but have no notion of class labels or splitting criteria. Isolation trees are constructed from input data by recursively splitting on a random feature at a random split point.

Isolation Forests are built on the assumption that anomalies are both few and different from normal observations. The intuition is that points that are difficult to "isolate" – i.e. require many splits – share many characteristics with other instances. Conversely, observations that are easy to isolate are apparently different and distanced from other observations and as such anomalous.

The anomaly score for a given sample is based on the (normalized) average path length and the actual path length, i.e. the number of splits until isolation, averaged over all trees in the forest. When the anomaly score crosses a predefined threshold, the sample is marked as anomalous.

Isolation Forests suffer from *swamping* and *masking* [53]. When anomalies are too close to normal instances, more splits are required to isolate points, lowering the average anomaly scores and as a result marking more normal instances as anomaly (swamping). Masking, on the other hand, refers to concentrated groups of anomalies that as a result appear normal. By (rigorously) subsampling the training data per isolation tree, both swamping and masking effects are reduced. We optimize the

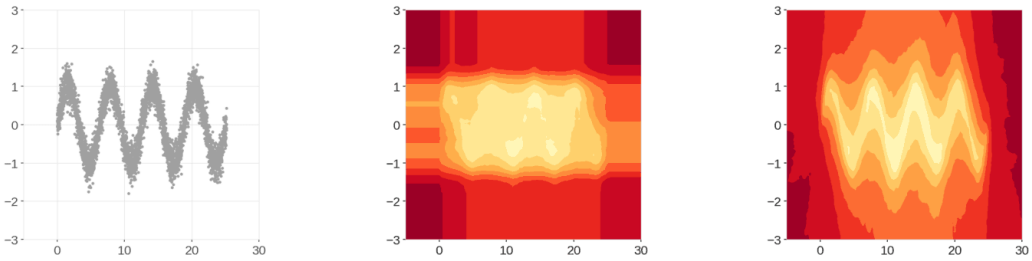
training sample size during experiments.

The main advantages of the Isolation Forest algorithm are state-of-the-art performance combined with fast training and evaluation, as opposed to other anomaly detection methods based on (Euclidian) distance or density. It also scales well with large and high-dimensional datasets. Moreover, Isolation Forests can be fit on training data containing anomalies.

However, the algorithm suffers from bias introduced by one-dimensional splits on a single feature, which may also cause "ghost" anomaly clusters if the input space contains clusters [37]. This bias produces different anomaly scores and ultimately different output labels for observations of similar importance. The *Extended Isolation Forest* algorithm attempts to mitigate this problem.

Extended Isolation Forest

The Extended Isolation Forest (EIF) algorithm [37] modifies the original Isolation Forest algorithm by allowing splits on multiple features. Higher-dimensional hyperplanes produce more effective splits of complex input spaces. Figure 5.4 illustrates the effects of the different splitting methods and clearly shows that the anomaly scores produced by the Extended Isolation Forest better capture the sinusoidal shape of the training samples.



(a) Training observations (b) Isolation Forest (c) Extended Isolation Forest

Figure 5.4: Training data with sinusoidal shape and corresponding anomaly score heatmaps of the regular Isolation Forest and the Extended Isolation forest. Source: Hariri *et al.* [37].

The *Extension Level* of the algorithm denotes the amount of additional features to consider at each split. Extension Level 0 is equivalent to the original Isolation Forest algorithm. We experiment with Extension Levels 1 and 2, considering either two or three features per split, as recommended by Hariri *et al.* [37].

Note that because a combination of features is used to produce a split, features do need to be scaled beforehand, in contrast to the regular Isolation Forest. The EIF package we use for experiments standardizes features at each step (centering features around the mean with unit variance). This approach does not bound features by a range and as such retains outliers in the data.

While we expect no significant issues using regular Isolation Forests, which have shown promising results in previous research ([1, 62]), we also experiment with the Extended Isolation Forest algorithm in an attempt to further improve detection performance.

Anomaly Threshold

Isolation Forest algorithms calculate anomaly scores for each sample. A threshold on this score is used to determine whether or not a sample is malicious. The authors of the iForest algorithm propose a threshold of 0.5 in [53], based on the characteristics of the scoring function.

Another method is to use the *contamination* of the training set to determine the threshold, as used by the Scikit-Learn iForest implementation [66]. The contamination represents the expected fraction of anomalies in the training data. The threshold is then defined after fitting the model by the anomaly scores in the *(1-contamination)*$^{\text{th}}$ percentile.

Given that the actual contamination (by benign storage channels) of our training set is unknown, the original threshold of 0.5 is used instead of determined by a contamination of zero. The latter would effectively define the threshold as the score of the most anomalous point in the benign class, which is

likely very anomalous. This would cause models to be unable to detect malicious but less anomalous samples.

Note that another common technique, selecting the top-*n* samples with the highest anomaly scores, is not a viable solution. Given that there are multiple heterogeneous threats in our dataset, with distinct behavior and query characteristics, this approach would only identify observations belonging to the most anomalous threats.

5.4. Storage Channel Detection

Combining the feature sets and machine learning algorithms proposed in the previous sections, this section introduces the storage channel detection experiments. In order to create effective and generalizable detection models, we require carefully partitioned datasets for training and testing. Section 5.4.1 explains which datasets are used and how they are combined. Then, the evaluation strategy and metrics used to determine *detection capability* of models is described in Section 5.4.2. Finally, the experiment design and parameters are outlined in Section 5.4.3.

5.4.1. Data Partitioning

The DNS traffic datasets used in this research have been introduced in Section 4.3. At our disposal are a sizable dataset of 180M normal DNS queries and malicious datasets of four different storage channel malware and two connection tunneling tools.

Machine learning models are trained on samples from a *training set*, and their performance is estimated using a hold-out *test set*. Using the same data for training and validation is fundamentally incorrect, as it does not provide insight into how well unseen data is classified and overestimates the performance of overtrained models.

Besides single-split hold-out validation, *k-fold cross validation* is a popular technique that estimates model performance by averaging over k different train-test splits. The dataset is split in k folds and on each iteration, a different fold is selected for testing and the remainder for training. Cross validation is generally preferred when limited data is available, as every sample is at some point considered for testing. The resulting performance estimate is therefore less biased. However, given our large and diverse datasets for testing and the computational overhead of cross validation, we opt for hold-out validation.

Naively dividing the data in two partitions, however, may still result in a biased performance estimate. We first identify and address three concerns that apply to both unsupervised and supervised experiments and then two concerns that only apply to supervised learning:

Hyperparameter optimization

We optimize the hyperparameters of the machine learning algorithms used during experiments. However, the hold-out test set cannot be used to determine the optimal set of parameters, as that would imply that the model learns from test samples, violating the independency rule. Instead, we designate a portion of the training data *validation set* which is only used for optimization purposes and not for training.

Data leakage

Data leakage introduces “information about the data mining target, which should not be legitimately available to mine from” and leads to overestimation of the performance of a model [45]. While there are many different data leakage causes, our main concern is training example leakage in terms, and more specifically *time leakage* in general and *group leakage* in the benign class.

Time leakage occurs when training uses observations that would not have been available at testing time. While this issue is not deemed critical for payload-only features, as single queries are presumed independent in that scenario, behavioral features do incorporate temporal information and should be divided over datasets in contiguous segments and not be shuffled.

Furthermore, while the corporate DNS dataset is large and diverse, traffic to the same primary domain is likely to be correlated or perhaps even identical. Analogous to e.g. the medical domain,

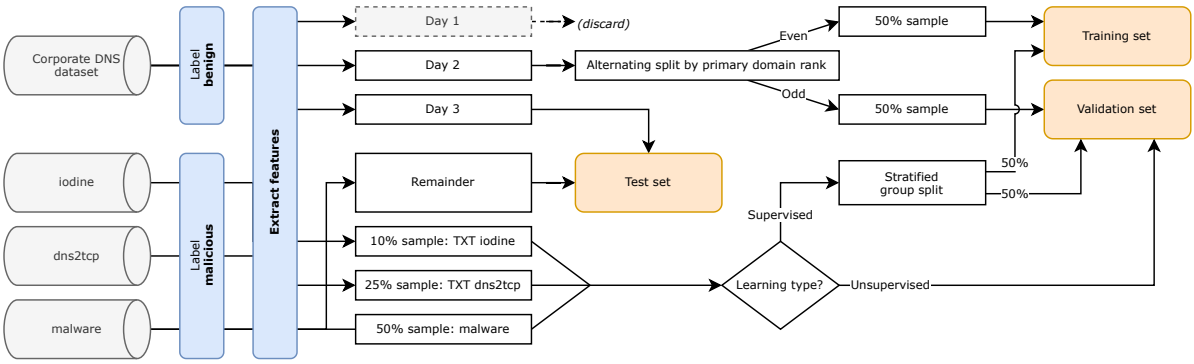


Figure 5.5: Train, validation and test dataset partitioning process.

where observations from the same patient are not to be shared across train-test splits, we make sure that queries to a distinct primary domain end up in either the train or test set, but never both.

Cold-start problem

Query sequence features suffer from *cold-start* problems. When extracting features from a sliding window over queries to a distinct primary domain, newly observed domains have no prior queries in the window. One or few queries cannot describe behavioral aspects well and effectively produce single-instance features. To mitigate this concern, we discard the first day of features from the three-day-spanning benign dataset. Note that still a considerable amount of newly observed domains exists in the remaining data (see Table 4.5).

Unsupervised anomaly detection uses only the benign class for training. The following concerns regard class distribution and balance and therefore only apply to supervised learning.

Class imbalance

The malicious class is significantly smaller than the benign class. Decision trees, and therefore Random Forests, are sensitive to class balance as 1) the minority class has less chance of being in the random sample to construct a tree and 2) splits are affected by class balance. We mitigate this problem by assigning different weights to the benign and malicious classes during training: either globally balanced – the same weight for all trees in the forest, computed beforehand – or balanced per subsample – weights computed independently for each bootstrap sample. This class weight strategy is optimized during hyperparameter optimization.

Threat Groups

The malicious data originates from multiple different storage channel threats and is therefore not independent and identically distributed (i.i.d.). While that is not necessarily a problem, as decision trees (and by extension Random Forests) and Isolation Forests do not make any assumption about the underlying data, uniform random data sampling to build each tree requires a balance between groups. Otherwise, some groups would be considered more important than others.

Furthermore, sufficient data is required to properly estimate the model performance. Therefore, we start with the smallest group (malware) and create a 50% train-test split. As the other groups are significantly larger, we first select only one query type configuration (TXT) and then sample the queries in the resulting subsets at different rates to achieve reasonable balance between groups. The imbalance between groups is resolved by introducing sample weights that weigh each group proportionally.

Based on these concerns and mitigation strategies, the overarching data composition process, visualized in Figure 5.5, is as follows. First, the corporate DNS dataset is labeled *benign* and the malware datasets *malicious*. Then, features are extracted from these datasets as a whole.

The benign class is subsequently split based on the three-day data collection period. The first day of features is discarded to mitigate the cold-start problem of sliding windows. The third day is used for evaluation purposes.

The Day 2 dataset is then split into a train and validation partition by the amount of queries per distinct primary domain. We create a fair split by alternating between assigning primary domains to either train or test dataset by their rank (query count). The domain with most queries is included in the training set, the domain with the second-most queries in the test set, etc. This yields an equal split between primary domains and an approximate even split between (the number of) queries.

Finally, due to memory constraints, the benign class is uniformly sampled at 50% for both train and test datasets. This is not expected to cause undesired side-effects, as the modeling algorithms also rigorously sample the training data. The undersampling also improves class balance.

The malicious datasets are split by their group and configuration. From all queries in the respective dataset, 10% of `iodine TXT`, 25% of `dns2tcp TXT` and 50% of every malware dataset is reserved for training and validation, and the remainder is used for testing.

For anomaly detection experiments, all malicious data is assigned to the validation set as it is not required for training. For Random Forest experiments, however, the malicious training data is additionally partitioned into two equal parts using a non-shuffled *stratified group split*. This divides the data such that the relative frequency of each group is retained across splits. The split is stratified by the dataset parameter configuration, evenly distributing queries from every underlying dataset per threat (see Appendix A.1).

The size and composition of the final datasets is described in Table 5.2.

Table 5.2: Train – validation – test split

Source dataset	Class	Training set	Validation set	Test set
Corporate DNS	Benign	1.78×10^7	1.70×10^7	4.73×10^7
iodine	Malicious	9.64×10^5	9.64×10^5	1.06×10^7
dns2tcp	Malicious	1.01×10^6	1.01×10^6	1.51×10^6
malware	Malicious	4.38×10^5	4.38×10^5	8.78×10^5
	Total:	1.80×10^7	1.73×10^7	5.94×10^7
	Imbalance ratio:	0.014	0.014	0.26

5.4.2. Evaluation

The purpose of this thesis is to assess the performance between algorithms as well as between feature sets. Different evaluation methods used in current literature are described in Section 3.4. However, as no consensus about detection capability exists, we choose to evaluate multiple aspects that we deem important: seen threat detection rates, unseen threat detection rates (generalization performance) and false positive rates. Combined, these describe all relevant aspects of detection capability.

Recall that all metrics are based on the number of correctly and incorrectly predicted observations per class, summarized in a *confusion matrix* (Table 5.3).

Table 5.3: Confusion matrix.

		Predicted	
		Benign	**Malicious**
Actual	**Benign**	True negatives (TN)	False positives (FP)
	Malicious	False negatives (FN)	True positives (TP)

Four metrics have been selected that describe different aspects of detection capability: balanced accuracy, detection rate, false positive rate and the number of false positive domains.

Balanced Accuracy $\frac{\frac{TP}{TP+FN}+\frac{TN}{TN+FP}}{2} \equiv \frac{TPR+TNR}{2}$

Balanced accuracy is defined as the arithmetic mean between the true positive rate (sensitivity) and the

true negative rate (specificity), equivalent to the average recall per class, and is invariant to class balance. While the range of this metrics is [0…1], a model that predicts either all positives or all negatives scores a balanced accuracy of 0.5.

Balanced accuracy is a composite metric of sensitivity and specificity. Both metrics independently provide valuable insight in the relative performance between models as well: correctly classifying observations in either class.

Recall $\frac{TP}{TP+FN}$

Recall (or: detection rate) describes the fraction of correctly classified observations in the malicious class. Recall may be calculated separately per group and averaged to produce a metric (macro-recall) that accounts for query imbalance between groups.

False Positive Rate $\frac{FP}{FP+TN}$

The false positive rate, or $1 - TNR$, describes the fraction of misclassified observations in the benign class.

The aforementioned metrics are based on (the number of) queries. By aggregating results per primary domain, we can measure the amount of detected storage channels, as well as the number of false positive domains. In terms of the practical usability of a model, the number of false positive domains is especially relevant, as few misclassified domains with many queries, perhaps benign storage channels, can be added to an allowlist.

Amount of False Positive Domains
The number of misclassified domains in the benign class, i.e. benign domains for which at least one query was classified as malicious. The number of required detected queries per domain can be increased to reduce the number of alerts (see Section 6.3).

Lastly, all models are evaluated using multiple datasets of unseen threats as well. These datasets comprise traffic from real malware, an unseen DNS tunneling tool and simulated plain-text credit card exfiltration. Refer to Section 4.3.3 for statistics about these datasets. The generalization performance is measured by the detection rate per threat, macro-recall over all threats, as well as the number of identified threats.

5.4.3. Experiments

Our experiments vary different feature sets as well as machine learning algorithms. Table 5.4 provides an overview of the considered machine learning methods (i.e. which algorithm, package and configuration) and feature sets.

Table 5.4: Machine learning algorithms and feature sets used for experiments.

Machine learning methods					**Feature sets**	
Type	**Algorithm**	**Package**	**Configuration**		**Type**	**Parameter**
Classification	Random Forest	scikit-learn	–		Payload-only	–
					Fixed window	$\lambda = 10$
				$\times$		$\lambda = 20$
Anomaly detection	Isolation Forest	IsoTree	(Extension Level 0)		Time window	$\delta = 1\text{sec}$
	Extended iForest	IsoTree	Extension Level 1			$\delta = 2\text{sec}$
	Extended iForest	IsoTree	Extension Level 2			$\delta = 5\text{sec}$

Besides experiments with singular feature sets, we also experiment with feature set combinations. For example, payload-only features may be combined with either a single fixed window feature set, a single time window feature set, or with both. However, feature sets of the same type are never combined as they are likely to be highly correlated.

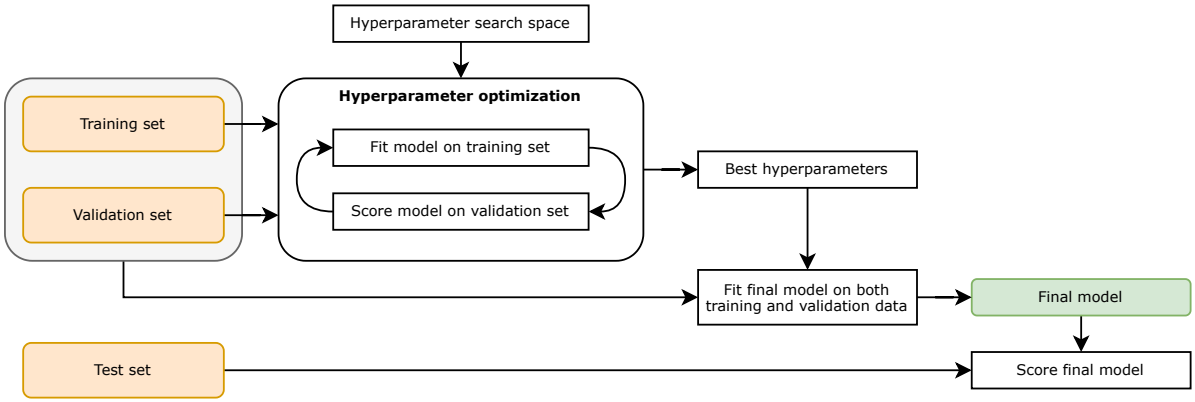


Figure 5.6: Experiment design: hyperparameter optimization, model training and performance evaluation.

There are 23 possible combinations: 6 individual, 11 combinations of two and 6 combinations of three feature set types. As every combination is also used in conjunction with all of the four algorithms, we perform a total of $4 \times 23 = 92$ different experiments.

The *scikit-learn* [66] Python package is used for the Random Forest implementation. While scikit-learn also includes the Isolation Forest algorithm, it does not implement the extended variant. We therefore use the *IsoTree* package [24], which implements both regular and extended Isolation Forest variants. The experiments have been performed with scikit-learn `0.24.2` and IsoTree `0.2.7-post4`, of which the latter includes bugfixes identified by this research[1].

Experiment design

Having established all information necessary to perform the detection experiments, this section describes the experiment design. The full experiment pipeline is visualized in Figure 5.6.

Given training and validation datasets, we iteratively train and score models using different hyperparameters to find the most optimal combination. Hyperparameters control the learning process of machine learning algorithms and are therefore not learned from the training data. Tuning these parameters may result in better models. The best performing hyperparameters are used to fit the final model on both the training and validation data.

The hyperparameters are optimized with respect to a certain metrics. Given the large number of experiments, optimizing with respect to overall performance, using for example the AUC, is not feasible as it takes too long to compute. Furthermore, many common optimization metrics are influenced by class balance and may produce unrepresentative results. Figure 5.7 demonstrates this phenomenon by visualizing the scoring landscape in terms of sensitivity and specificity of the F_1-score (a common optimization metric), accuracy and balanced accuracy, at different rates of class imbalance.

At perfect class balance, accuracy is equally sensitive to changes in the positive and negative class. The F1-score, however, is already biased towards the positive class, as there is little variation in specificity at lower sensitivity values.

Increasing the imbalance to a ratio of 10:1, the existing bias in terms of F1-score is exaggerated, as changes in sensitivity now become increasingly irrelevant. The accuracy is instead barely affect by changes in the true positive rate. At an imbalance ratio of 100:1, the effects on both accuracy and F1-score are even more pronounced. Balanced accuracy, however, weighs both classes equally and is unaffected by class imbalance.

In practice, using accuracy or the F1-score for optimization would produce models that are biased towards either the the negative class or positive class. Instead, the scoring function should be either impartial to class balance, or weigh different errors according to domain requirements (e.g. prefer

[1] See `https://github.com/david-cortes/isotree/issues/17` and `https://github.com/david-cortes/isotree/pull/19`.

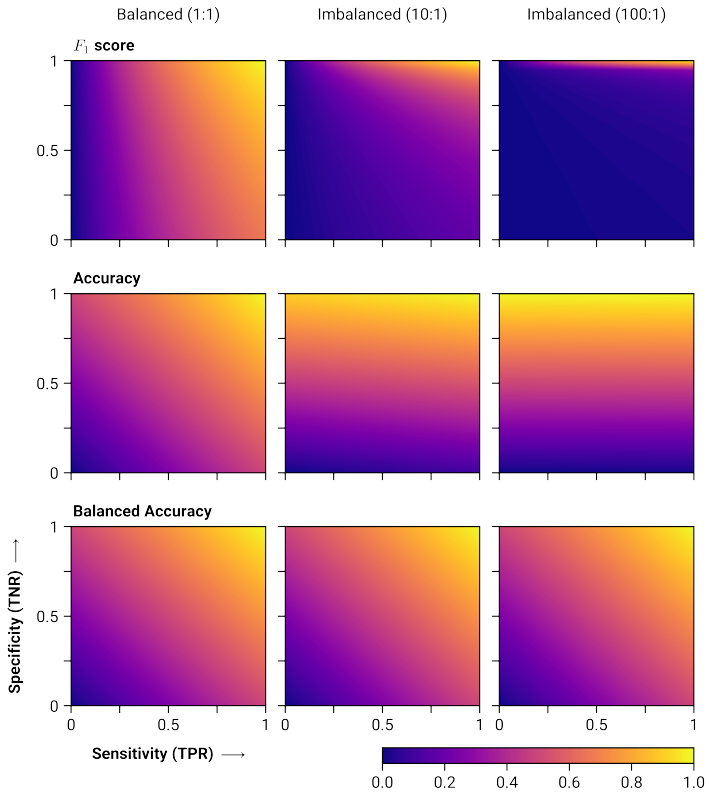


Figure 5.7: Score landscape over sensitivity and specificity of the F1-score, accuracy and balanced accuracy, at imbalance ratios of *1:1*, *10:1* and *100:1*.

false negatives over false positives to reduce the amount of alerts). As determining a suitable cost-sensitive metric is outside the scope of this research, we choose to optimize with respect to the unbiased *balanced accuracy* metric.

The next step is to determine which parameters to optimize. Firstly, the number of trees can be tuned for all (tree-based) algorithms. Tuning this parameter beforehand incurs unnecessary computational cost, because the error rate converges for growing number of trees and the expected gain is low, which is demonstrated by Probst and Boulesteix [68] for the case of Random Forests.

Therefore, to reduce the execution time of each experiment, we initially fix number of trees to 25 – a trade-off between performance and computational time – and optimize the other hyperparameters (Table 5.5). Afterwards, the newly found parameters are fixed and the number of trees is (briefly) optimized. As Probst and Boulesteix [68] explicitly mention that their error rate convergence statement generalizes to “any ensemble method that uses a randomization technique”, we also initially fix the number of trees for our Isolation Forest experiments.

For Isolation Forests, besides the number of trees, only the *sample size* parameter is tuned. This determines the size of the subsample used to construct Isolation Trees. While the original paper recommends a setting as low as 256, we also experiment with values that are orders of magnitude larger (see Table 5.5), given the diversity of traffic in our benign class.

For Random Forests, the main purpose of hyperparameter optimization is to control the tree growth and reduce overfitting. To that extent, the maximum depth of trees in the forest is limited by parameter *max_depth*, the minimum required samples before a node may split by *min_samples_split* and the minimum number of samples to remain at leafs by *min_samples_leaf*. Additionally, the number of features considered at each split is reduced using the *max_samples* parameter, performing implicit feature selection.

Similar to Isolation Forests, we sample the training data from which trees are built. The classes

are then assigned weights based on either global (dataset) balance or the balance in the subsample. Finally, we do not optimize the splitting criterion and use Gini impurity instead of Information Gain, because it is faster to compute and both criteria rarely disagree [70].

The full parameter search space per algorithm is provided in Table 5.5. Common optimization strategies are *grid search* – trying every combination of parameters – and *randomized search* – randomly trying parameter combinations for a fixed number of iterations. We use a different, probability-based search strategy provided by Python package `hyperopt` instead. Hyperopt uses a Tree-structured Parzen Estimator (TPE) to estimate the density of the parameter space [10], in order to make more informed decisions about the search direction. Hyperopt is shown to produce better hyperparameters in less time compared to random search [10], which is beneficial given the large number of experiments to execute. This further allows us to explore a wider parameter range than with grid search.

Table 5.5: Hyperparameter search space, per algorithm and package.

Algorithm	Package	Hyperparameter	Search space (interval or set)
Random Forest	scikit-learn	n_estimators	[5; 125] (step 10)
		max_depth	[4; 32] (step 2)
		max_samples	$[2^8; 2^{19}]$
		min_samples_split	[2; 10]
		min_samples_leaf	[1; 10]
		max_features	[10%; 100%]
		class_weight	{balanced, balanced_subsample}
(Extended) Isolation Forest	IsoTree	ntrees	[5; 125] (step 10)
		sample_size	$[2^8; 2^{19}]$

The hyperparameters are optimized for either 100 (Isolation Forest) or 200 (Random Forest) iterations, given the difference in search space size. Although `hyperopt` supports non-uniform search spaces, we make no assumptions about the shape of the parameter space and define them as either *uniform* (for discrete and continuous variables) or *choice*.

Having determined the best hyperparameter values, the final model is trained on all available training data. This model is subsequently scored using the test set and the metrics described in Section 5.4.2. The optimized hyperparameter configuration for every experiment is provided in Appendix C.

5.5. Summary

This chapter describes the covert DNS storage channel detection methodology.

First, Section 5.1 stated the assumptions about our datasets and feature engineering. We presume our corporate DNS dataset to be attack-free and the contamination of benign storage channels low and insignificant. We further argue that robust features can be derived from only the payload and timestamp of DNS queries.

Section 5.2 presented our feature extraction method for both payload-only features and behavioral features. Behavioral features are extracted from a sliding window over per-domain query sequences and are either time-based or of fixed length. We produce three distinct feature sets that are descriptive of query space utilization, information density and lexical properties of queries for payload features and storage channel behavior for sliding window features.

Next, Section 5.3 introduced the (Extended) Isolation Forest variants and Random Forest classifier used for detection, selected by our requirements of fast training and evaluation, generalization ability and promising performance in current literature.

Finally, Section 5.4 described our experiment design, evaluation metrics and detection methodology based on the proposed features and algorithms. We identify data partitioning difficulties based on data leakage, class imbalance, groups in the malicious class and potential cold-start problems for sliding windows, and adjust our experiments accordingly.

6

Results

This chapter presents the results of our storage channel detection experiments. First, the performance and detection capability on seen threats is analyzed in Section 6.1. Then, the capability of models to detect unseen threats is tested in Section 6.2. Lastly, the nature and cause of misclassifications – false positives and false negatives – is investigated in Section 6.3.

Detailed results per experiment are included in Appendix D. Recall that behavioral features are extracted from sliding windows of either a fixed length, denoted by "λ=length", or time-based, denoted by "δ=duration".

In this chapter, we use the *Mann-Whitney U* and *Wilcoxon signed-rank* statistical tests to determine whether or not distributions of (subsets of) results are significantly different. Both tests are non-parametric, because we cannot assume normality of the experiment results, and test the null hypothesis that there is no significant difference in the medians between groups [40]. The corresponding P-value denotes the probability that the null hypothesis is correct. We reject the null hypothesis when $P < .05$. The Wilcoxon signed-rank test is similar to the Mann-Whitney U test, but is used when samples are paired (e.g. between all results of two algorithms) [40].

6.1. Seen Threat Detection

This section analyzes the detection capability of our models on the test set. The test set contains one full day (24h) of DNS traffic from the corporate dataset and all DNS traffic from *iodine*, *dns2tcp* and all malware traffic that was not part of the training set (see Figure 5.5). We refer to these malicious threats as *seen* threats, as they have been seen by the models during training.

Detection capability is measured by balanced accuracy, false positive rate and recall. To account for imbalance between threats with different number of samples, the recall is calculated per threat and then averaged (*macro-averaged recall* or macro-recall).

Note that experiments may be grouped at different levels of granularity to derive general conclusions and reduce clutter in visualizations. As feature sets are composed of payload and/or behavioral features, they are grouped either by their *composition* – i.e. "Payload", "Fixed", "Time" and combinations – or by the less detailed *composition type*: payload-only, behavioral-only or both.

First, we present our results for anomaly detection (Section 6.1.1) and classification (Section 6.1.2) experiments. Then, we compare the results between algorithms (Section 6.1.3) and feature set compositions (Section 6.1.4). Lastly, we analyze the detection rates between models in Section 6.1.5.

6.1.1. Anomaly Detection

Table 6.1 contains the test set results for the iForest, EIF-1 and EIF-2 anomaly detection experiments. Every algorithm is paired with 23 different feature set combinations. We include the results for the three best performing and three worst performing experiments in terms of balanced accuracy.

Table 6.1: Top-3 and bottom-3 anomaly detection experiment results, by balanced accuracy. Bolded values denote best scores across all experiments for the respective metric.

(a) iForest

#	Payload	Time	Fixed	Balanced Acc.	Macro-Recall	FPR (queries)	FP (domains)
1	✓	1s	20	**0.99575**	0.99920	**0.77%**	2,328
2	✗	1s	10	0.99556	0.99927	0.82%	2,866
3	✗	5s	10	0.99539	0.99962	0.86%	3,479
21	✗	–	20	0.94341	0.92845	8.66%	69,402
22	✗	–	10	0.92678	0.91649	11.15%	74,186
23	✓	–	–	0.88983	0.78924	2.05%	20,925

(b) Ext. iForest (level=1)

#	Payload	Time	Fixed	Balanced Acc.	Macro-Recall	FPR (queries)	FP (domains)
1	✓	5s	10	**0.99748**	0.99936	**0.42%**	456
2	✓	5s	20	0.99733	**0.99956**	0.48%	470
3	✗	1s	10	0.99711	0.99895	0.48%	883
21	✗	–	10	0.97302	0.90795	1.49%	31,139
22	✓	–	20	0.96226	0.90466	0.89%	6,064
23	✓	–	–	0.89423	0.78725	1.03%	7,291

(c) Ext. iForest (level=2)

#	Payload	Time	Fixed	Balanced Acc.	Macro-Recall	FPR (queries)	FP (domains)
1	✗	2s	10	**0.99776**	0.99929	0.38%	723
2	✓	2s	20	0.99753	0.99901	**0.38%**	332
3	✗	2s	20	0.99730	0.99913	0.45%	748
21	✓	–	20	0.95126	0.89627	0.78%	5,188
22	✗	–	20	0.95067	0.92650	7.12%	57,024
23	✓	–	–	0.89496	0.78733	0.90%	6,358

These results demonstrate that anomaly detection is a viable approach to detect DNS storage channels. The best configurations per algorithm type had a detection rate on seen threats of more than 99.5% with a false positive rate of at most 0.77%. Consistent outliers are experiments with payload-only features, with a considerably lower recall and as a result balanced accuracy.

The results also show a major difference in false positives between the best and worst performing models, both in terms of false positive query rates and the number of misclassified domains. Isolation Forest experiments had an FPR of at least 0.77% and at most 11.15%, EIF-1 experiments 0.42% to 1.66% and for EIF-2 experiments 0.38% to 7.12%. The amount of false positive domains ranged between 1,345 to 74,186 for the regular iForest, 231 to 33,976 for EIF-1 and 198 to 57,024 for EIF-2 experiments.

6.1.2. Classification

The classification experiment results are presented in Table 6.2. The 23 experiments in total comprise all different feature sets combinations with the Random Forest classifier.

Table 6.2: Top-3 and bottom-3 Random Forest experiment results, by balanced accuracy. Bolded values denote best scores across all experiments for the respective metric.

#	Payload	Time	Fixed	Balanced Acc.	Macro-Recall	FPR (queries)	FP (domains)
1	✓	1s	20	**0.99991**	**0.99972**	< 0.001%	10
2	✓	2s	10	0.99985	0.99971	< 0.001%	9
3	✗	1s	20	0.99984	0.99939	< 0.001%	8
21	✗	–	10	0.99394	0.99785	0.74%	109
22	✗	2s	–	0.99249	0.99849	1.38%	147
23	✗	–	20	0.99185	0.99943	1.60%	24,785

The results show that classification models were able to near-perfectly separate the benign from the

malicious class, achieving both a recall of 99.99% and false positive rate of less than 0.001%. Remarkably, all experiments were able to detect at least 99.94% of the malicious queries, irrespective of feature set used.

False positive rates vary, however, between < 0.0001 and 1.60% and the worst-ranked experiment misclassified queries for as many as 24,785 domains. Nevertheless, the classification experiment with feature set $\delta = 2s$; $\lambda = 20$ (not in this table) had only two false positive domains while maintaining a recall of 99.92%. Overall, twelve experiments produced at most 27 (0.009%) of 310,000 primary domains in the dataset.

6.1.3. Algorithm Comparison

Comparing the anomaly detection (Section 6.1.1) and classification (Section 6.1.2) results from the previous sections, we find that models using either algorithm type are able to detect seen DNS storage channels with high detection rates and low false positive rates.

Firstly, we compare anomaly detection and classification performance. The median balanced accuracy scores for anomaly detection and classification experiments are 0.9956 and 0.9993, respectively. The distributions in the two groups differ significantly (Mann–Whitney $\mathrm{U} = 223.0$, $n_1 = 23$, $n_2 = 69$, $P < .0001$). Overall, our classification experiments outperformed anomaly detection experiments.

Between anomaly detection algorithms, Extended Isolation Forest experiments (median 0.99653) performed significantly better than Isolation Forest experiments (median 0.99382) in terms of balanced accuracy (Mann–Whitney $\mathrm{U} = 235.0$, $n_1 = 46$, $n_2 = 25$, $P < .0001$). However, between the balanced accuracy distributions of EIF-1 (median 0.99651) and EIF-2 (median 0.99670) experiments, no statistically significant difference is observed (Wilcoxon signed-rank test, $W = 129.0$, $n_1 = n_2 = 23$, $P = .80$).

In summary, classification models have been shown to outperform anomaly detection models on balanced accuracy. Between anomaly detection algorithms, Extended Isolation Forest models achieve a significantly lower balanced accuracy than Isolation Forest models, although there is no such difference observed between extension levels of Extended Isolation Forest experiments.

6.1.4. Feature Set Comparison

Besides algorithms, feature sets were also varied between experiments. We evaluate the relative performance between payload-only, behavioral-only and composite feature sets in this section.

We first determine whether or not combining behavioral features with payload features improves detection capability. Using the Wilcoxon signed-rank test, we find that no significant difference between the balanced accuracy distributions of behavioral-only and composite feature sets exists, neither for anomaly detection experiments ($W = 189.0$, $n_1 = n_2 = 33$, $P = .10$) nor classification experiments ($W = 19.0$, $n_1 = n_2 = 11$, $P = .24$). This means that we cannot reliably attribute improved performance to the addition of payload features to behavioral feature sets.

Figure 6.1 shows, however, that especially the false positive rate distributions are consistently narrower for composite feature sets. We use again the Wilcoxon signed-rank test to determine whether or not a significant difference between false positive distributions exists and find that this is indeed the case for anomaly detection experiments ($W = 137.0$, $n_1 = n_2 = 33$, $P < .05$), but not for classification experiments ($W = 25.0$, $n_1 = n_2 = 11$, $P = .52$). We conclude that the addition of payload features to behavioral-only feature sets does significantly decrease false positive rates for anomaly detection experiments.

Finally, payload-only feature sets are among the worst performing experiments. Figure 6.1 shows payload-only anomaly detection experiments as clear outliers in terms of balanced accuracy, caused predominantly by lower recall. Also, a cluster of outlier anomaly detection experiments appears around a recall of 0.900. The cause of this lower anomaly detection recall is investigated further in Section 6.1.5. Classification experiments showed no such outliers and its payload-only experiment was within the

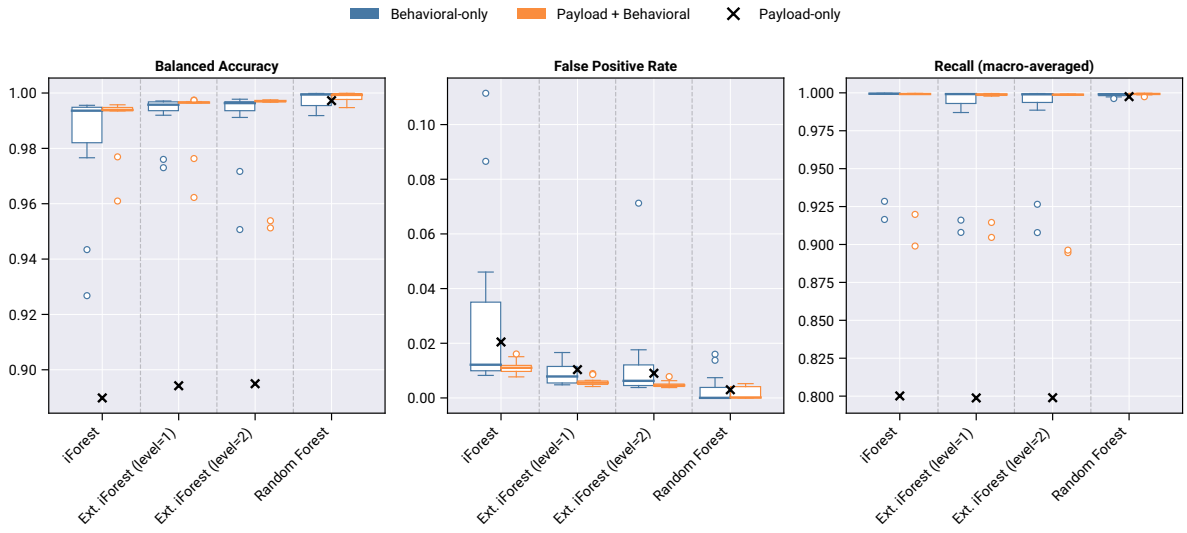


Figure 6.1: Balanced accuracy, false positive rate and macro-recall distributions, visualized per algorithm and split for behavioral-only and composite feature sets. The payload-only experiment scores are added as well.

bandwidth of other experiments, for every metric.

Overall, anomaly detection models had higher false positive rates than classification models. Breaking down the results per feature set composition reveals that this behavior is caused by fixed-only feature sets. Figure 6.2 visualizes the balanced accuracy, false positive rate and macro-recall per feature set composition and clearly shows a fixed-only false positive distribution that is considerably wider and has a higher median value. This figure also indicates that anomaly detection experiments with considerably lower recall use either payload-only, fixed-only or a combination of both feature sets.

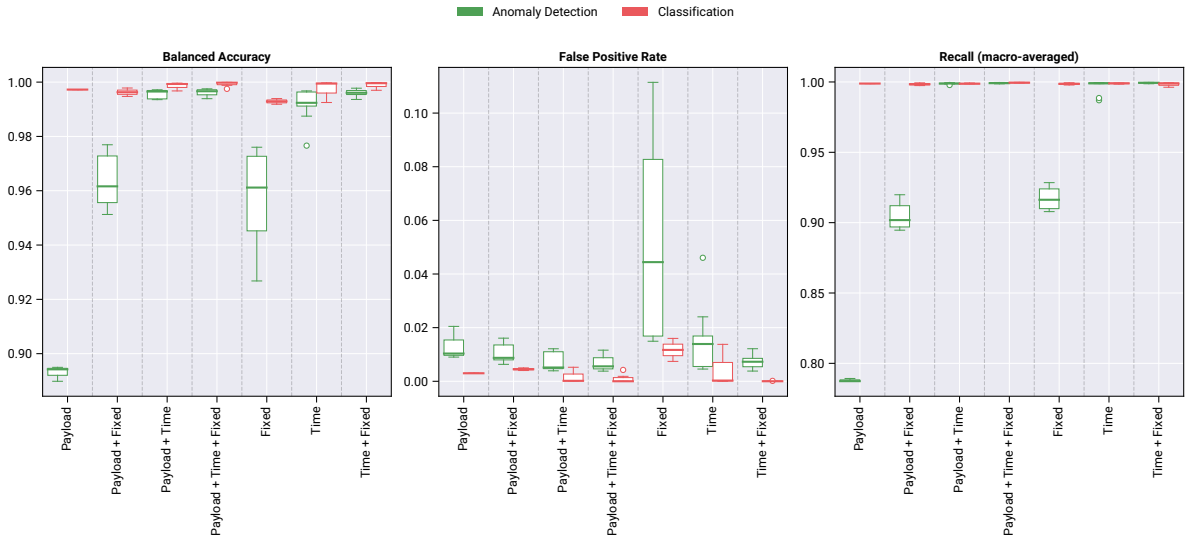


Figure 6.2: Balanced accuracy, false positive rate and macro-recall distributions, visualized per algorithm and split for behavioral-only and composite feature sets. The payload-only experiment scores are added as well.

The experiment results seemed to indicate that feature sets with a time component score better than the same feature sets without it. We use the Mann-Whitney U test to verify whether or not a significant difference exist between the balanced accuracy distribution of experiments with a time component and those without, and do the same for fixed-length sliding window features. We exclude the clear payload-only outliers from this analysis.

For anomaly detection, feature sets with a time component (median 0.99640) have a significantly

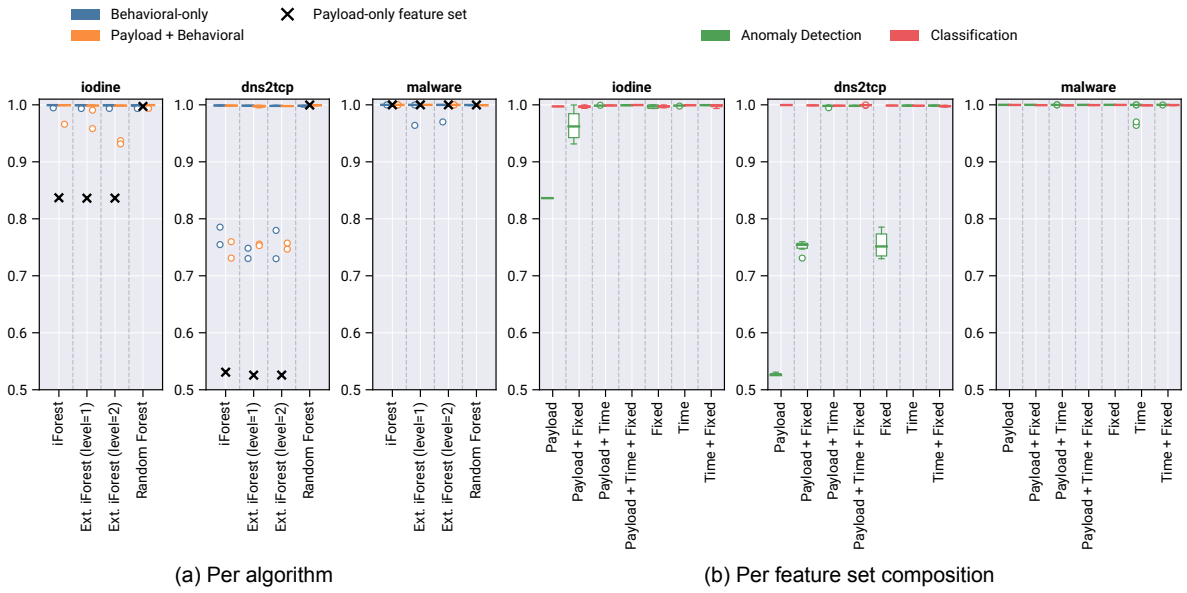
higher balanced accuracy than feature sets without it (median 0.96162, $\mathrm{U} = 1.0$, $n_1 = 12$, $n_2 = 54$, $P < .0001$). For fixed-length sliding window features, the medians are 0.99581 for non-fixed and 0.99573 for fixed, and no significant difference was observed ($\mathrm{U} = 425.0$, $n_1 = 18$, $n_2 = 48$, $P = .46$).

For classification, adding time features (median 0.99959) also provides a significant improvement over feature sets without (median 0.99434, $\mathrm{U} = 6.0$, $n_1 = 4$, $n_2 = 18$, $P < .05$). For fixed-length sliding window features, no significant difference was observed as well ($\mathrm{U} = 40.0$, $n_1 = 6$, $n_2 = 16$, $P = .29$), although the medians of both distributions – 0.99938 and 0.99920 for non-fixed and fixed, respectively – are consistently high.

In summary, our experiments showed that both behavioral-only and combined payload and behavioral feature sets outperform payload-only features. We observed no significant difference in balanced accuracy between behavioral-only and composite feature sets. However, we did find a significant decrease in false positive rates for anomaly detection experiments using combined payload and behavioral feature sets. We also found that fixed-only feature sets cause high false positive rates and that fixed-only, payload-only or combinations thereof are the only feature sets showing considerably lower recall. Lastly, feature sets with a time component showed a significant increase in balanced accuracy compared to feature sets without it.

6.1.5. Recall Analysis

In order to understand the cause of the lower recall outliers observed in the previous section, we visualize of the recall distributions of the underlying malicious threats in Figure 6.3 and juxtapose the dissections (a) per algorithm and (b) per feature set composition. Performance metrics for every experiment, ranked by macro-recall on the test set, are included in Table D.9 for reference.



(a) Per algorithm

(b) Per feature set composition

Figure 6.3: Recall distribution for each threat in the malicious class.

Figure 6.3 shows that, regardless of algorithm or feature set, malware traffic is almost perfectly separable from benign traffic. Two outliers are caused by Extended Isolation Forest models using time-only feature sets ($\delta = 1s$). Therefore, poor (anomaly detection) recall with payload-only features can be attributed to tunneling traffic only.

Payload-only features score consistently worse for both tunneling tools, although there are especially large discrepancies in recall of *dns2tcp* traffic. Payload-only experiments recognize just over 50% of malicious queries. The remaining outliers, ranging between 70% and 80% recall, are experiments with fixed-only or payload combined with fixed-window features. Interestingly, this behavior is similar across all anomaly detection algorithms and can therefore not be attributed to unfortunate sam-

pling during training. We have identified that most of the missed tunneling observations are beaconing instead of data transfer queries, which are further analyzed in Section 6.3.2.

Lastly, Figure 6.3 confirms the excellent class separation abilities of Random Forest models, as each model detected most to all queries for each threat contained in the malicious class, with no outliers.

Table 6.3 contains the top-10 experiments in terms of macro-recall on seen threats. This shows that, while the overall tendency is that classification experiments outscore anomaly detection experiments, the latter are able to achieve high recall as well, at the expense of higher false positive rates.

Table 6.3: Top-10 experiments with highest macro-recall on threats in the test set.

#	Algorithm	Payload	Time	Fixed	Macro-Recall	FPR (queries)	FP (domains)
1	Random Forest	✓	1s	20	**0.99972**	**< 0.001%**	10
2	Random Forest	✓	2s	10	0.99971	**< 0.001%**	**9**
3	iForest	✗	5s	20	0.99970	1.17%	2,116
4	Random Forest	✓	1s	10	0.99964	**< 0.001%**	77
5	iForest	✓	5s	20	0.99964	1.03%	2,375
6	iForest	✗	2s	20	0.99962	0.99%	3,873
7	iForest	✗	5s	10	0.99962	0.86%	3,479
8	iForest	✗	5s	–	0.99958	4.60%	4,619
9	EIF-1	✓	5s	20	0.99956	0.48%	470
10	EIF-2	✗	5s	10	0.99954	0.73%	749

Finally, 77 out of 92 experiments (84%) had a macro-recall of more than 98%. Of the remaining 15 experiments, the macro-recall varies between 93% and 79%, no feature set has a time component and all use an anomaly detection algorithm.

6.1.6. Conclusion

Based on the analysis provided in the previous sections, we conclude the following about the detection capability on seen threats:

- Anomaly detection and classification are both viable DNS storage channel detection techniques. The best performing anomaly detection experiments had a macro-recall of more than 99.92% and a false positive rate of at most 0.77%. Classification models achieved a macro-recall of 99.97% and a low false positive rate of < 0.0001%.

- While the majority of experiments separated benign from malicious queries well, classification models score significantly ($P < .0001$) better than anomaly detection models in terms of balanced accuracy.

- Anomaly detection experiments had higher false positive rates than classification experiments, predominantly caused by tunneling threats. Nevertheless, the best performing anomaly detection experiments outscored the worst classification experiments.

- Extended Isolation Forests had a significantly ($P < .0001$) higher balanced accuracy than the regular Isolation Forest, predominantly due to lower false positive rates. No significant difference in results between extension levels of Extended Isolation Forest experiments was observed.

- We observed no significant improvement in balanced accuracy by combining payload with behavioral features over behavioral-only feature sets. However, false positive rates of anomaly detection experiments do decrease significantly for these composite feature sets ($P < .05$).

- Adding a time component to feature sets improved balanced accuracy significantly over feature sets without it for both classification ($P < .05$) and anomaly detection ($P < .0001$) experiments. No such improvement was observed for the fixed feature set component.

6.2. Unseen Threat Detection

As the majority of experiments achieved high recall on the test set, we now analyze whether or not this holds for unseen but similar storage channel threats as well. Our evaluation dataset includes traffic from nine malware samples also used in [73], three *dnscat2* tunneling samples from [9] and finally a custom plain text version of the simulated data exfiltration malware (see Section 4.3.3).

Note that, as discussed in Section 4.3.3, the unseen *Carbanak* and *Cobalt Strike* malware samples are categorized as DNS payload droppers that embed information in the DNS response instead of the query. These threats are included in visualizations for a better understanding of our detection models, but are excluded from macro-recall analysis for a fair comparison. Furthermore, the *UDPoS* threat is not strictly unseen as it also occurs in the training data. The unseen data, however, originates from actual malware instead of simulated traffic and is included to confirm detection capability and the quality of the simulated dataset.

6.2.1. Anomaly detection

Table 6.4 contains the experiments results on unseen threats, ordered by macro-averaged recall over all threat samples but payload droppers. The false positive query rate and number of false positive domains on the benign class are included for comparison.

Table 6.4: Top-3 and bottom-3 anomaly detection experiment results, by macro-averaged recall over unseen threat samples. Bolded values denote best scores across all experiments for the respective metric.

(a) iForest

#	Payload	Time	Fixed	Macro-Recall	Detected	Incl. droppers	FPR (queries)	FP (domains)
1	✗	–	20	**0.90412**	**11**	**13**	8.66%	69402
2	✗	–	10	0.88616	**11**	**13**	11.15%	74186
3	✓	–	–	0.64418	9	9	2.05%	20925
21	✗	2s	–	0.54350	8	9	2.40%	**1,345**
22	✓	1s	20	0.53893	8	8	**0.77%**	2328
23	✗	1s	–	0.52360	8	8	1.39%	3191

(b) Ext. iForest (level=1)

#	Payload	Time	Fixed	Macro-Recall	Detected	Incl. droppers	FPR (queries)	FP (domains)
1	✗	–	20	**0.67237**	**10**	**12**	1.66%	33976
2	✗	–	10	0.64121	**10**	**12**	1.49%	31139
3	✓	2s	20	0.59651	8	8	0.64%	1320
21	✗	2s	20	0.49213	8	9	0.73%	368
22	✗	1s	10	0.48302	8	8	0.48%	883
23	✓	2s	10	0.46212	8	8	0.56%	388

(c) Ext. iForest (level=2)

#	Payload	Time	Fixed	Macro-Recall	Detected	Incl. droppers	FPR (queries)	FP (domains)
1	✗	–	20	**0.75911**	**11**	**13**	7.12%	57024
2	✓	–	–	0.59697	9	9	0.90%	6358
3	✓	–	20	0.59217	9	9	0.78%	5188
21	✓	1s	10	0.46447	8	8	0.45%	268
22	✓	2s	20	0.46306	8	8	**0.38%**	332
23	✗	1s	10	0.45312	8	8	0.44%	635

The results show large differences in unseen threat detection capability between experiments. Across all experiments, disregarding DNS payload droppers, iForest models detected at least 8 and at most 11 samples (median=9), EIF-1 models 7 to 10 samples (median=8) and EIF-2 models 7 to 11 unseen samples (median=8). The experiments with highest unseen threat macro-recall are among the experiments with the highest number of false positive domains as well.

Interestingly, the experiments with the lowest balanced accuracy on the test set (feature sets $\lambda = 10$, $\lambda = 20$ and payload-only, see Table 6.4) achieve the highest macro-recall on the unseen threats.

6.2.2. Classification

The unseen threat detection results for classification experiments are included in Table 6.5. Again, ordered by macro-averaged recall over all threat samples but payload droppers and with false positive statistics on the benign class.

Table 6.5: Top-3 and bottom-3 Random Forest experiment results, by macro-averaged recall over unseen threat samples. Bolded values denote best scores across all experiments for the respective metric.

#	Payload	Time	Fixed	Macro-Recall	Detected	Incl. droppers	FPR (queries)	FP (domains)
1	✗	–	20	**0.59589**	**11**	**11**	1.60%	24785
2	✗	2s	–	0.41966	7	7	1.38%	147
3	✗	–	10	0.35440	6	6	0.74%	109
21	✓	2s	20	0.13485	6	6	0.00%	21
22	✗	1s	–	0.09496	4	4	0.03%	452
23	✓	1s	–	0.09452	6	6	0.01%	246

Only a single classification model (fixed-only, $\lambda = 20$) was able to recognize all non-dropper threats. This experiment is also the only one that was able to approach the macro-recall of anomaly detection experiments. However, this is again the experiment with the most false positive domains. All other classification experiments had lower macro-recall than every anomaly detection experiment.

The negative outlier (barely) detected four threats and uses only time-based ($\delta = 2s$) features. Overall, our Random Forest models were able to detect 4 to 11 unseen threats (median=6).

6.2.3. Detection Capability Analysis

It is clear from the results presented in Section 6.2.1 and Section 6.2.2 that anomaly detection experiments consistently detected more queries from more unseen threats. The Mann-Whitney U test confirms the significance of these differences in (unseen) macro-recall distribution ($U = 52$, $n_1 = 69$, $n_2 = 23$, $P < .0001$). However, detection rates varied both between unseen threats and experiments, as shown by the visualized detection rates per malicious sample (left) and number of distinct threats detected per feature set (right) in Figure 6.4.

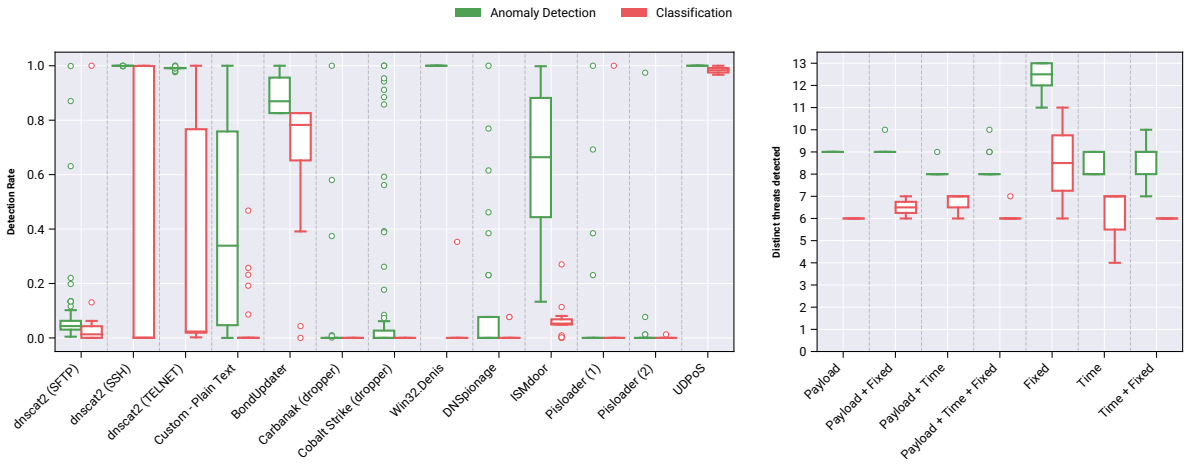


Figure 6.4: Detection rate distributions for each unseen threat (left) and the distinct threats detected distributions for each feature set composition (right), split per algorithm type.

Overall, this visualization shows that anomaly detection experiments recognized more unseen threats for every feature set type: the distributions for anomaly detection experiments have a higher median and their minima are higher than or similar to the maxima of classification distributions.

Table 6.6 provides an overview of the amount of experiments that were able to recognize at least one query per threat. The results show that only four out of 92 experiments were able to recognize every

unseen threat (excluding droppers). The combined variety in detection rates and many experiments that missed samples altogether demonstrates the importance of using unseen threats for model evaluation.

Table 6.6: Number of experiments that detected at least one malicious query for the respective threat.

Threat	Overall	Payload-only	Behavioral-only	Payload + Behavioral	Anom. Det.	Classification
UDPoS	92 (100%)	4 (100%)	44 (100%)	44 (100%)	69 (100%)	23 (100%)
dnscat2 (SFTP)	92 (100%)	4 (100%)	44 (100%)	44 (100%)	69 (100%)	23 (100%)
dnscat2 (SSH)	92 (100%)	4 (100%)	44 (100%)	44 (100%)	69 (100%)	23 (100%)
dnscat2 (TELNET)	92 (100%)	4 (100%)	44 (100%)	44 (100%)	69 (100%)	23 (100%)
BondUpdater	91 (99%)	4 (100%)	43 (98%)	44 (100%)	69 (100%)	22 (96%)
ISMdoor	90 (98%)	3 (75%)	43 (98%)	44 (100%)	69 (100%)	21 (91%)
Custom – Plain Text	74 (80%)	3 (75%)	34 (77%)	37 (84%)	67 (97%)	7 (30%)
Win32.Denis	70 (76%)	3 (75%)	34 (77%)	33 (75%)	69 (100%)	1 (4%)
DNSpionage	27 (29%)	4 (100%)	13 (30%)	10 (23%)	25 (36%)	2 (9%)
Cobalt Strike	21 (23%)	0 (0%)	18 (41%)	3 (7%)	21 (30%)	0 (0%)
Carbanak	6 (7%)	0 (0%)	6 (14%)	0 (0%)	6 (9%)	0 (0%)
Pisloader (1)	5 (5%)	0 (0%)	5 (11%)	0 (0%)	4 (6%)	1 (4%)
Pisloader (2)	5 (5%)	0 (0%)	5 (11%)	0 (0%)	4 (6%)	1 (4%)

Overall, more than 97% of anomaly detection experiments were able to detect at least five unseen malware samples (including *UDPoS*) and all three tunneling samples. Classification experiments were also able to detect the tunneling samples, although only consistently (> 90% of experiments) identifying three malware samples.

Dissecting the performance per unseen threat, an important observation is that all experiments are able to detect UDPoS traffic at high detection rates. This validates both our experimental setup and the quality of simulated malware traffic.

Of the most difficult to detect threats, *Pisloader* stands out. Only five experiments were able to detect any of the two samples. The difference between samples is a larger number of queries as well as more duplicated payloads in sample *Pisloader (2)*. Figure 6.4 shows that experiments using either algorithm type lack detection capability for this threat. *Pisloader* uses DNS for command-and-control, for which no related storage channel threats are included in the training data.

Next, although recognized by every experiment, the difference in detection rates between tunneled *dnscat2* protocols shows that tunneling traffic can be diverse and that the behavior of the tunneled connection influences the characteristics of the resulting DNS traffic. The main difference between samples are shorter queries and a longer duration for the *SFTP* dataset, compared to the other protocols.

There is a significant difference between the detection rates of *Win32.Denis* as well. All anomaly detection models correctly identify every query, in contrast to classification models of which only one is able to identify this threat.

Lastly, the varying detection rates for the *Custom – Plain Text* show that omitting encryption and encoding complicates detection significantly. Recall that these queries follow the same exfiltration schedule as the simulated malware in the training set. Table 6.6 shows that almost all anomaly detection models flag this threat, but far fewer classification models.

False negatives are further analyzed in Section 6.3.2.

Table 6.6 also confirms our initial expectation that some models would be able to detect DNS payload droppers (*Carbanak*, *Cobalt Strike*) as well, although detection is limited to behavioral features. Isolated dropper queries are short and not anomalous. No classification models are unable to detect these threats.

High detection rates on unseen threats come at a cost. Experiments that generalize well suffer from high false positive rates, both in terms of queries and domains. The results of experiment with the highest macro-recall over all unseen threats, all using an anomaly detection algorithm, are included in Table 6.7. Two Random Forest experiments with the highest macro-recall are added for comparison as well, as is the (Random Forest) experiment with the lowest false positive rate.
These results show that anomaly detection experiments that were able to recognize every threat also suffered from high false positive rates. The one classification model in line with anomaly detection

Table 6.7: Top-10 experiments with highest macro-recall on unseen threats, including the two highest scoring Random Forest experiments and the experiment with the lowest false positive rate. The macro-recall excludes DNS payload droppers.

#	Algorithm	Payload	Time	Fixed	Macro-Recall	Num. threats	FPR (queries)	FP (domains)
1	iForest	✗	–	20	**0.9041**	**13**	8.66%	69,402
2	iForest	✗	–	10	0.8862	**13**	11.15%	74,186
3	EIF-2	✗	–	20	0.7591	**13**	7.12%	57,024
4	EIF-1	✗	–	20	0.6724	12	1.66%	33,976
5	iForest	✓	–	–	0.6442	9	2.05%	20,925
6	EIF-1	✗	–	10	0.6412	12	1.49%	31,139
7	iForest	✓	–	10	0.6343	10	1.51%	13,878
8	iForest	✓	–	20	0.6342	9	1.61%	13,102
9	iForest	✗	5s	–	0.6279	9	4.60%	4,619
10	iForest	✓	5s	10	0.6249	10	1.16%	3,160
18	Random Forest	✗	–	20	0.5959	11	1.60%	24,785
71	Random Forest	✗	2s	–	0.4197	7	1.38%	147
77	Random Forest	✗	2s	20	0.3025	6	**0.00%**	**2**

performance also suffered from many false positive domains. Other classification experiments, not included in this table, still recognized six or seven distinct threats (except one outlier at four) at lower macro-recall than anomaly detection experiments, but at sub-1% false positive rates.

Note that a high false positive rate does not automatically result in many false positive domains, and vice versa. The interaction between these metrics, as well as the cause of high false positive rates, is further analyzed in Section 6.3.

In summary, our experiments showed that anomaly detection models are able to recognize more distinct samples and achieve significantly ($P < .0001$) higher macro-recall on unseen threats than classification models. However, only four models were able to recognize every unseen threat. Overall, detection rates vary between samples and experiments, although every experiment achieved high recall on real traffic from our simulated *UDPoS*, validating our simulation and experiment setup.

6.2.4. Feature Set Comparison

Having analyzed the detection capability of both seen and unseen threats, we now compare the overall detection capability of different feature sets. A combined overview of our results dissected per feature set is presented in Figure 6.5. We visualize the macro-recall on seen and unseen threats separately and also indicate the corresponding amount of distinct unseen threats recognized. Moreover, the false positive query rates and the amount of domains for the respective feature sets are plotted for comparison. The unseen macro-recall scores of the same feature set with and without payload features are connected with a line (in orange) to emphasize the relative performance difference.

Anomaly detection algorithms, paired with fixed-only feature sets, achieved the highest macro-recall on unseen threats. These experiments also had the lowest macro-recall for seen threats, confirming our observations in Section 6.2.1. Unseen macro-recall of the other feature sets is within a similar range and shows no distinctive patterns, although the amount of threats detected fluctuates. Combining payload features with fixed-only feature sets, however, decreases unseen macro-recall considerably and reduces performance to the level of the other experiments.

Classification experiments show a more pronounced pattern when combining payload features with a behavioral feature set, which consistently decreases unseen macro-recall. While the number of distinct threats detected is generally low, it does not fluctuate as much as combined to anomaly detection experiments. The observed decreases in unseen macro-recall generally do not lead to fewer distinct threats recognized, but of course lowers the confidence in detection. One outlier experiment, using fixed-only features, was able to recognize all non-dropper threats, again at the highest false positive rate of all classification experiments.

Overall, fixed-only feature sets outperformed all other feature sets in both the number of distinct threats detected and achieved unseen macro-recall. However, their false positive rates were considerably higher than for other experiments. This possibly indicates that *1)* other classification models overfit on

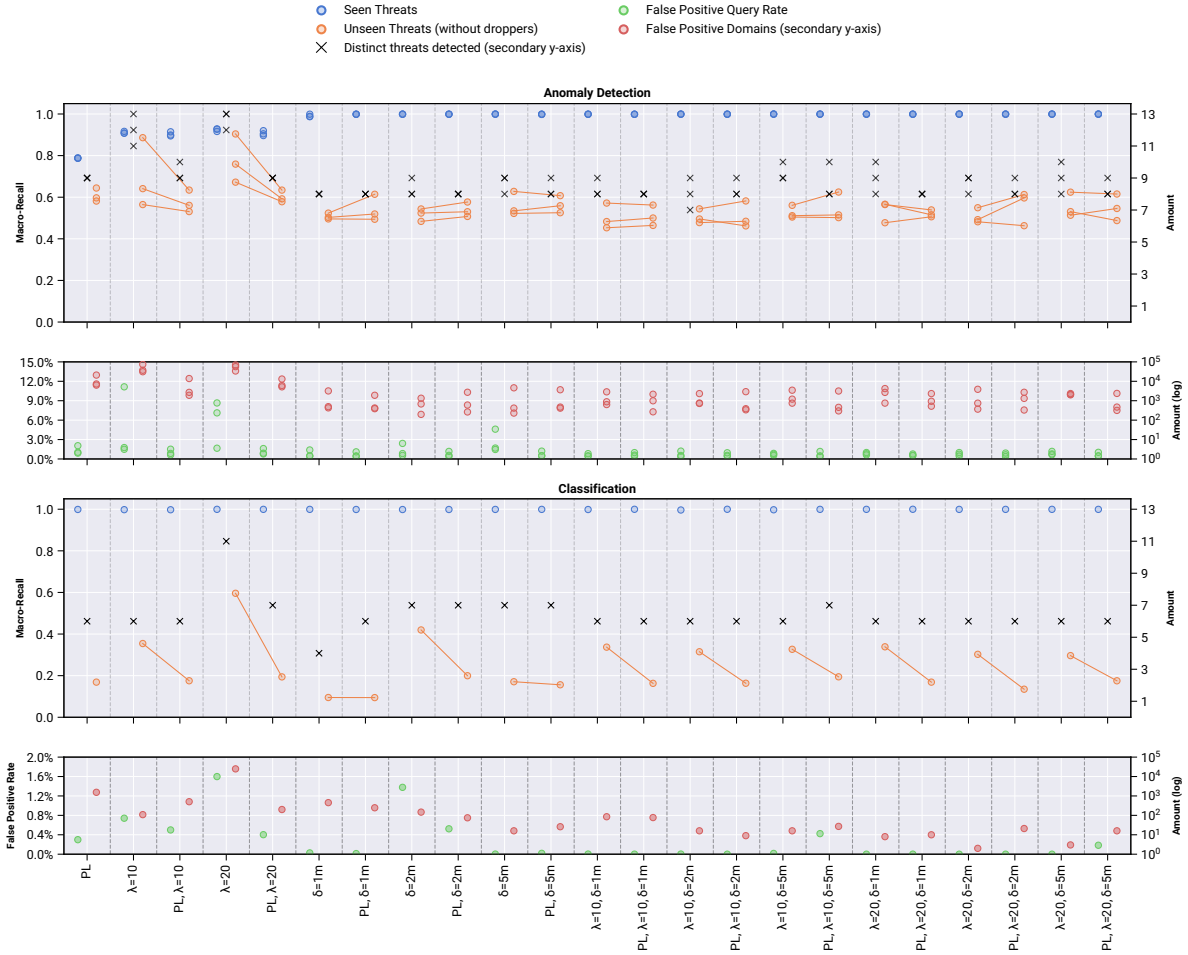


Figure 6.5: Detection capability comparison between feature sets. Visualized is the macro-recall for seen and unseen threats, the false positive (query) rate and amount of false positive domains. Feature set pairs with and without payload features are linked with a line (in orange) to emphasize the relative change in recall. Note the different y-axis ranges for false positive query rates and the log scale on the false positive domains axis.

seen threats, to which fixed-only features are less receptive, *2)* that other features are not descriptive enough of traffic from the unseen threat set and/or *3)* that the flagged false positives are actually mostly benign storage channels that pollute the scoring metrics. Misclassifications are further analyzed in Section 6.3.

Because the experiments that performed well suffered from high false positive rates and that little variation in detection capability is observed for the remaining experiments, we conclude that our experiments provide insufficient evidence to reliably rank specific feature sets or combinations and recommend instead to incorporate the selection of window types, combinations and sizes in the model optimization process.

In summary, behavioral fixed-only feature sets were able to detect most unseen threats regardless of algorithm type, although at high false positive rates. Combining behavioral with payload features consistently lowered unseen threat detection for classification experiments, although this effect was only present in anomaly detection experiments with fixed-only features. The experiments provided inconclusive evidence to rank or recommend window types or sizes.

6.2.5. Conclusion

Unseen threat classification has proven to be an important aspect of evaluation, since our analysis showed that high test set performance does not translate to models that generalize well over similar

but unseen threats. The most important observations are:

- Anomaly detection models were able to recognize more distinct samples and achieved significantly ($P < .0001$) higher macro-recall on unseen threats than classification models. However, only four out of 92 experiments were able to detect every DNS storage channel threat (except DNS payload droppers).
- All three *dnscat2* tunneling samples were recognized by all experiments. Anomaly detection experiments furthermore consistently detected at least five unseen malware samples, opposed to three for classification experiments.
- Every experiment achieved high recall on real traffic from our simulated *UDPoS* malware, validating our simulation and experiment setup.
- In terms of feature sets, behavioral (fixed-only) feature sets detected most unseen threats regardless of algorithm type, although at high false positive rates. The variability in unseen threat detection capability between the remaining feature sets causes our experiments to be inconclusive regarding ranking or recommending window types or sizes.

6.3. Misclassification Analysis

6.3.1. False Positives

To investigate the characteristics of alerts per domain, we recorded for every experiment the average, standard deviation and maximum amount of false positive queries for every domain. The distributions of these three metrics are plotted in Figure 6.6, per algorithm (left) and per feature set composition type (right). Note the log-scale on the y-axis.

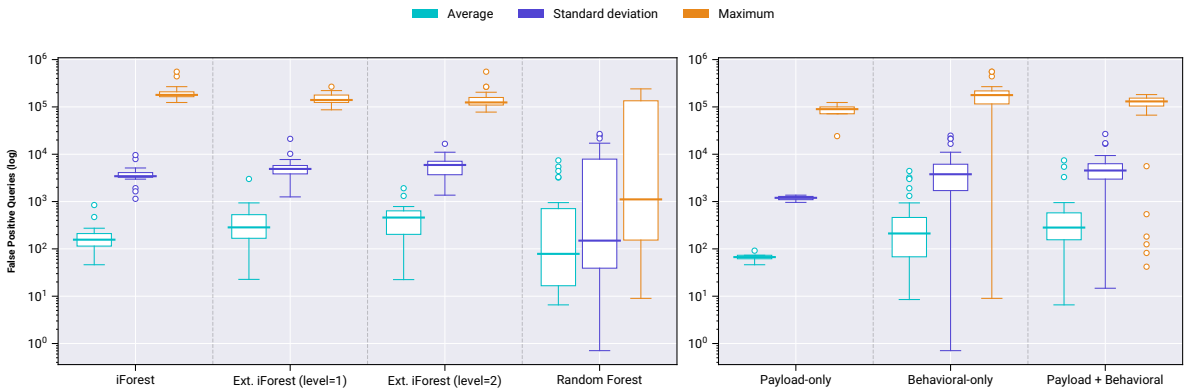


Figure 6.6: Distributions of the average, standard deviation and maximum number of alerts per domain.

Overall, the maximum amount of alerts for a domain was multiple orders of magnitude larger than the average number of alerts. The standard deviation was approximately one order of magnitude larger than the average, indicating a large spread in the number of detections per domain.

Every anomaly detection experiment flagged more than 100,000 queries for at least one domain. In contrast, there are classification models which flagged at most 10 queries per domain. Between anomaly detection experiments, all metrics are within the same order of magnitude. Increasing the extension level increases the average and standard deviation of the number of alerts, although our results have shown that EIF experiments have fewer false positive domains in general.

Classification experiments show less consistent distributions for each metric, although the median standard deviation and maximum is orders of magnitude lower than for anomaly detection experiments.

Within feature set composition types, distributions overlap and are therefore difficult to compare. Again, however, the standard deviation is often larger than the average alerts per domain. Only feature sets

with a behavioral component are able to achieve few alerts per domain. Note that the payload-only distributions are narrower, because only four experiments have been performed.

Lastly, given the wide variety in misclassified domain names and queries between experiments, we are unable to reliably identify common causes for misclassification unique to a particular algorithm or feature set.

Benign Storage Channels
A key assumption made about *benign* storage channels, i.e. legitimate services that use DNS for data transfer, is that the "total contamination is presumed to be low and insignificant" (see Section 5.1). As benign storage channels are not clearly defined, we provide a brief, qualitative analysis of false positive domains and whether or not they are actual mistakes or not.

By aggregating all misclassifications and analyzing the most often occurring domains and domains with either the most or the least alerts, we find that a considerable portion of false positive domains can attributed to benign storage channel services in the following categories:

- Security products, using unique DNS queries for lookups.
- DNS / DDoS protection services, using unique DNS query names to reroute traffic.
- Reverse (IP) lookups, outside the .arpa infrastructure that is filtered from our dataset.
- DNS experiments, by e.g. Google testing IPv6 [21].
- Email providers, using DNS for e.g. spam protection.

These findings are in line with reports of benign storage channels in current literature [1, 16, 21, 62, 73].

To illustrate the impact of benign storage channels, the three most often occurring security product domains account for, on average, 56.3% of false positive queries across all experiments. It is our belief that including benign storage channels in training data is harmful to both anomaly detection models and classification models. For both algorithm types, optimizing with respect to a polluted metric produces sub-optimal models. Furthermore, benign storage channels confuse classification models, because similar feature vectors are present in both the benign and malicious class. Benign storage channels should therefore be identified and filtered before training.

Reduction Techniques
We experiment with three techniques to reduce the amount of false positive domains by filtering the generated alerts. First, we filter the top 10,000 domains from the Cisco Umbrella list of popular domains [23]. As this list is regularly updated, we have used the version actual at the time of benign data collection. Filtering popular domains is a common technique to reduce false positives [1, 26, 73, 85]. Next, we ignore domains with three or less alerts, as domains with more than a few alerts are arguably more interesting. Finally, we assess the effectiveness of combining both techniques. These techniques do not significantly impact detection capability for storage channel threats generating at least three alerts, using primary domains not contained in the Umbrella list.

Figure 6.7 shows the distributions of the amount of false positive domains for algorithms and feature set composition types. Overall, the visualization shows that anomaly detection suffers from more false positive domains, before as well as after the application of reduction techniques. The reduction techniques show similar behavior across algorithms and feature set composition types.

Using the Umbrella allowlist indeed reduces both the median and minima. However, as at most 10,000 domains are removed, the maxima arises mostly unaffected. Thresholding at three alerts per domain, however, reduces the amount of domains considerably for every algorithm and feature set. In contrast to using an allowlist, this technique is especially useful for experiments with flagged many positive domains to begin with. Furthermore, no classification experiment now falsely flags more than 1,000 domains, which is still the median amount of domains for Isolation Forest experiments. As many of the observed benign storage channel domains are not included in the Umbrella allowlist, the practical impact of filtering is presumed to be even more significant.

Combining reduction techniques improves the minimum, median and maximum amount of false positive domains for every algorithm and every feature set. For classification, the median is now less

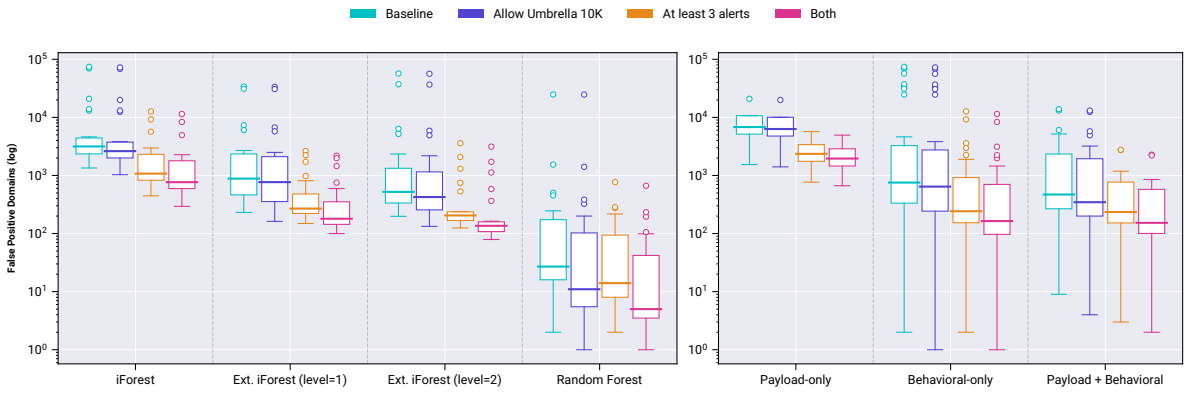


Figure 6.7: Reduction in the number of flagged domains after either allowing the top 10,000 domains from the Umbrella list, after ignoring the first three alerts per domain, or both, visualized either per algorithm or feature set composition type.

than 10 false positive domains. However, anomaly detection experiments still flag at least 200 domains. Between anomaly detection algorithms, increasing the extension level reduces the mean and minima, but has no clear influence on the maxima.

In conclusion, filtering the first n alerts per domain clearly reduces the number of false positive domains. We recommend experimenting with different thresholds to find the optimal trade-off given the setting in which a detection system is deployed. Furthermore, following the observations from the previous section, using fine-tuned allowlists by including the most common benign storage channel domains should further reduce the amount of false positives.

6.3.2. False Negatives

Seen Threats

The most interesting false negatives are caused by *iodine* and *dns2tcp* and (combinations of) payload-only and behavioral features from a fixed-length sliding window.

Firstly, payload-only anomaly detection experiments have the highest false positive rates on tunneling queries. The average query length of unrecognized queries never exceeds 10, for both *iodine* and *dns2tcp* traffic. This indicates that predominantly beaconing queries (see Appendix A.2) are the cause of diminished detection rates, instead of the (longer) data transfer queries. We conclude that our payload features are not descriptive enough of short beaconing queries.

Secondly, fixed-only behavioral feature sets ($\lambda = 10, \lambda = 20$) also show a decrease in recall on tunneling threats. Using the same analysis, we find that again predominantly beaconing queries are missed, although at considerably higher rates for *dns2tcp* than for *iodine*. We again conclude that the fixed-length sliding window queries are not descriptive enough of short queries.

Lastly, there are two outlier experiments (Extended Isolation Forest) using time-based sliding window features ($\delta = 1s$) and malware detection. The misses are caused by *BernhardPOS* exfiltration at a time interval of 1 second.

The majority of misclassified windows (> 87%) contained two queries instead of one, which influences the *unique transfer rate* and *unique query rate*. However, more queries in a window produce more anomalous feature vectors, other windows with two queries were identified correctly and malware using the same exfiltration schedule has no such false negatives. The main difference between *BernhardPOS* and other malware are fewer labels per query.

Missed queries from the remaining anomaly detection experiments are spread across malicious threats, without significant outliers. Most false negatives originate from tunneling traffic, as none of the remaining anomaly detection experiments misclassify more than 10 malware queries.

Random Forest experiments exhibit high overall detection rates, which can be attributed a higher

detection rate on beaconing queries. Compared to anomaly detection experiments, malware detection is slightly worse, missing at most 129 queries.

Unseen Threats
As detection rates vary between experiments for unseen threats, we provide a qualitative analysis of samples that stand out.

The *Pisloader* and *DNSpionage* samples are short, have few queries and relatively short payloads. This renders sliding windows ineffective. We presume that too few payload features are anomalous for anomaly detection models to flag these queries and that no similar threats in the training set cause diminished classification performance. Given the many unique queries in the sample, fixed-length sliding windows eventually collect enough queries for detection.

The *Win32.Denis* sample comprises four storage channels to different domains, and has few queries in total. All anomaly detection models detect all queries from this threat, in contrast to only a single classification models that is able to recognize this sample. *Win32.Denis* queries contain a long prefix of a single repeated character and additionally one subdomain of one character, which may produce features just different from seen threats and on the wrong side of the learned classification boundary.

Regarding the *Custom – Plain Text* threat that we simulated solely for this research, the difference in detection between anomaly detection and classification is notable, given that the queries are generated by the same process as the other malware. By *not* applying encoding and encryption, only few classification models were able to detect a subset of malicious queries. Although more anomaly detection models detect queries from this type, detection rates vary and no clear pattern is distinguishable.

The same is true for *ISMDoor*. As this sample has almost 2,300 queries, it is long enough to provide reliable insights, a worrying observation is that most classification models either fail to detect this threat or flag only few queries. Anomaly detection experiments perform relatively better, but again show variability in detection rates. *ISMDoor* is a quintessential storage channel malware and should have been detected.

Between the three samples of the *dnscat2* tunneling threat, a sharp contrast arises between tunneling SFTP data as opposed to the other SSH and Telnet samples. We attribute this contrast to a significant difference in (average) query length: 22 for SFTP, opposed to 224 and 218 for SSH and Telnet, respectively. The difficulty with shorter queries highlights a shortcoming of our feature selection, even though every experiment was able to detect at least one query per tunneled protocol.

In summary, we observe that using unseen threats for evaluation is crucial to uncover otherwise unknown shortcomings of the detection models. Short queries that are anomalous in only a few aspects are most difficult to detect, which we partially attribute to a lack of similar threats in the training data.

7

Discussion

7.1. Comparison with Current Literature

Although it is difficult to compare results obtained on different datasets, we attempt to place our findings in the context of other research in this section.

Nadler *et al.* [62] propose an anomaly detection-based system using behavioral-only features extracted from contiguous time windows containing queries to distinct primary domains. Their feature extraction method is different to our approach, as we generate feature vectors from sliding windows per observation instead of per time bin. Our detection latency is therefore effectively zero, instead of the window size (recommended 60 minutes) in their approach.

Our features share the same rationale. Their selection, however, includes features based on response type ratios in the window and the longest meaningful (English) word in the query name, which we presume to be either non-portable to other networks and/or susceptible to adversarial manipulation. We further expect their entropy feature to be more effective due to the availability of unprocessed query names, including capitalization.

The authors place a strong limit on the anomaly score threshold to reduce the amount of false positive domains. During six days of evaluation, only 18 domains were flagged in a dataset larger than ours. Our experiments, with the least amount of false positive domains, detect 2 (classification) or 198 (anomaly detection) in 24 hours of traffic. Their method is still able to detect simulated *FrameworkPOS* and *Win32.Denis* traffic and tunneling traffic from *iodine* and *dns2tcp*.

The filtering of low-volume domains as well as the optimization of the anomaly score threshold is expected to contribute considerably to the reduced amount of domains. Both techniques can be applied to our methods as well and are worth exploring to further improve the practical usability. It would, however, be interesting to see to what extent this prevents detection of related but more challenging storage channel threats, e.g. *Pisloader*.

The proposed two-step anomaly detection approach by Saeli *et al.* [73] first identifies anomalous queries based on their payload and subsequently calculates an anomaly score based on additional behavioral characteristics per domain or user. They rigorously filter benign traffic by removing popular domains, known benign storage channel domains, queries with unsuccessful responses, low-volume domains and queries containing IP addresses.

The authors have graciously shared the malware traffic samples used to validate their approach, which we use for unseen threat detection capability evaluation. Saeli *et al.* [73] are able to detect every threat at high detection rates. Only *Pisloader* (96%) and *ISMDoor* (91%) have a recall of less than 100%. In contrast, our detection rates vary between experiments and only few are able to detect at least one query for every threat.

Other test subjects used for validation are *dns2tcp* and *iodine*, also among our training subjects, and *dnscat2*, which is in our unseen evaluation set. However, the total amount of malicious queries (all subjects included) is less than 20,000. Their approach detects 100% of *dns2tcp* and *dnscat2* queries and *iodine* detection varies between 85% and 88%, depending on the configuration. The majority of

our experiments achieve a detection rate of over 99% on an arguably more diverse tunneling dataset of more than 12.5M *dns2tcp* and *iodine* queries. Our *dnscat2* detection varies between tunneled protocols, although most configurations are detected at similar rates.

Their benign class (1.2×10^5 queries) used for testing is several orders of magnitude smaller than ours (4.7×10^7 queries) and the reported false positive rate is 1.7%. The authors share three such queries, which resemble benign storage channels. In comparison, our best performing anomaly detection models in terms of FPR achieve 0.38%, although models that are capable of detecting every unseen threat as well have a FPR of at least 7.1%. However, we identified that most of the false positives belong to benign storage channels, which are more prevalent in our more diverse benign class.

Ahmed *et al.* [1] propose an anomaly detection-based method using payload-only features similar to our payload-only feature selection. Their approach is evaluated using a DNS traffic dataset of similar size to ours.

We agree with their findings that anomaly detection methods outperform classification in terms of detecting new malicious DNS queries. Our payload-only anomaly detection experiments outscore classification experiments on unseen threat macro-recall as well. However, our analysis has shown that payload-only features achieved consistently lower recall on seen threats compared to other feature sets and that behavioral features do improve detection capability.

Their approach has a false positive rate of at least 1.6% on the top 10,000 most popular domains and at least 21.6% for the remaining domains. In contrast, our payload-only anomaly detection experiments have a FPR between 0.9% and 2.1% over all queries in our dataset.

We speculate that the difference in performance stems from 1) the fact that their model is only trained on the top 10,000 most popular domains in the dataset, whose traffic may not be diverse enough to capture characteristics of other domains and 2) their final Isolation Forest model comprises only two isolation trees, which may be insufficient to produce stable anomaly scores.

Other works, using classification methods, predominantly focus on connection tunneling detection.

Buczak *et al.* [16] report a recall of over 99.9% on e.g. *dnscat2* and *iodine* traffic and varying performance (> 27%) on unseen tunneling traffic, with a reported false positive rate of 0%. Shafieian *et al.* [75] combine payload and behavioral features and report 14 false positives for their single Random Forest classifier, further reducing that number to one false positive by stacking classifiers. Misclassifications account for less than 0.02% of traffic. Preston [67] reports a recall and precision of over 99.6%, but his benign class is stripped of anomalies beforehand and only one tunneling subject is considered. Das *et al.* [26] report an (average) recall of 94.5% and a false positive of 0.2%.

Note, however, that the benign class corresponding to each of the aforementioned results is either (very) limited in size or affected by filtering, inflating the reported metrics. Furthermore, none of these works use DNS malware for evaluation.

We conclude that our best classification models achieve similar or better performance compared to current literature, considering that:

- not only tunneling traffic is detected with high recall but DNS malware traffic as well
- our classification models are able to detect at least one other tunneling threat – and often more unseen threats – at low false positive rates
- our experiments had both low false positive query rates and few false positive domains, considering the 47M queries in the benign class of the test set and the presence of benign storage channels

Furthermore, our payload-only anomaly detection experiments improves over a similar method proposed in [1]. The results of other anomaly detection experiments, combined with the reported results in current literature, are insufficient to determine whether or not they provide a tangible advantage over other approaches.

7.2. Recommendations and Future Work

Improving ground truth for training, by incorporating domain knowledge, is a promising future research direction to increase model performance.

Firstly, the observation that many false positive queries actually belong to benign storage channels calls for better and more rigorous cleaning of benign training data. Benign storage channels pollute evaluation and optimization metrics and cause overfitting on seen threats. Identifying and removing benign storage channel domains from the training data therefore improves the quality of ground truth.

Secondly, when acceptable within the risk profile of the deployment setting, increasing the minimum exfiltration threshold for detection by ignoring low-volume domains and/or short queries reduces the amount of uninteresting traffic in the benign class. In addition to our current approach of filtering queries without subdomains or with the `www` subdomain, other common but non-tunneling subdomains are for example `mail`, `ns1`, `ns2`, `dns0`, `dns1`, etc.

Thirdly, sliding window feature extraction allows for fine-grained filtering rules, based on for example the amount of (unique) characters or the information density in a window. Thresholding windows on these metrics improves ground truth by filtering uninteresting traffic, while at the same time improving class balance and reducing training time.

Lastly, extending the Public Suffix List used to parse domain names with known local domain extensions enables either filtering of internal domains or better detection of storage channels to these domains.

Regarding features, we recommend future research to incorporate feature selection for both anomaly detection and classification experiments. While the number of features used for Random Forest experiments is already optimized during training, it is a rudimentary approach and our anomaly detection experiments still use all features. It is expected that, especially when combining fixed-length and time-based windows for low-volume domains, some features will be correlated and redundant. While all algorithms considered in this research should handle correlated features well, it is worth exploring whether or not feature selection improves performance.

Moreover, we expect that tailoring features to a specific window type improves the performance of composite feature sets. Fixed-length window features should focus on uniqueness and information content, while time-based windows best capture anomalous transfer rates and query frequencies.

For classification models, including payload features rarely improves performance and even decreases the overall detection rate of unseen threats, even though the number of distinct unseen threats detected is relatively stable. As we optimize the number of features during hyperparameter optimization, it is unlikely that ineffective features are cause of the problem, but instead overfitting on the limited selection of threats in the test set.

Considering a wider variety of storage channel threats is expected to improve performance. It remains to be seen, however, to what extent this increases false positives as well. We recommend including in particular C2-over-DNS threats for training, besides connection tunneling and credit card exfiltration malware, as this threat category is among the most difficult to detect by our current models.

Without increasing the number of threats used for training, using cross-validation and leaving (part of) the malicious groups out of each fold could reduce overfitting on seen threats as well. Currently, our datasets used for training, validation and testing contain every configuration of each tunneling and malware threat. Training on for instance only tunneling threats and then attempting to classify malware traffic, and vice versa, may well produce models that are able to detect more unseen threats.

Note that although not every unseen threat was consistently detected, all models are able to correctly identify at least one unseen tunneling threat (*dnscat2*) and real traffic from the otherwise simulated *UDPoS* malware, validating our current experiment setup.

While impossible to confirm using our benign dataset, we expect that analyzing traffic more granularly than per-domain could reduce false positive rates, e.g. by also incorporating origin IP addresses. Malicious storage channels are scarce and often originate from a single device, while benign storage channels are often in use by many endpoints. Combining globally behavioral traits per domains, as

confirmed effective in this work, with local behavior would be an interesting extension to this research.

We further expect models to be circumventable by hiding exfiltration queries between many duplicate regular queries, nullifying the advantages of sliding windows. This effect is already noticeable in the lack of recall for the *Pisloader (2)* sample, which contains many duplicate queries. Larger sliding window sizes would be increasingly affected by this evasion technique.

Possible mitigation strategies are to only focus on unique queries in a window or to discard features that are easily polluted by these evasion methods. However, this would require a reassessment of effective features and could be part of a broader feature selection approach. It is our recommendation to confirm detection capability of models when using extreme evasion strategies, e.g. as many duplicate queries as the window size, regardless.

8

Conclusion

In this work, we investigated the effectiveness of machine learning-based methods at detecting covert DNS storage channels. We first answer the sub-questions of this research to arrive at an overarching conclusion and answer to the main research question.

SQ1: In current literature, what (traditional) machine learning-based DNS storage channel detection methods exist, what features are effective and how is detection capability measured?

Current research considers both unidirectional and bidirectional storage channel threats, where the former only exfiltrates information using DNS queries and the latter additionally receives information via DNS responses. Arbitrary data transfer and connection tunneling are the main threats in these categories, respectively. The majority of reviewed works only considers tunneling and only few include, arguably more challenging to detect, low-throughput data transfer threats.

We identified the algorithm type used for detection as a major distinction between works: either unsupervised anomaly detection or supervised classification. Supervised classification is often used to detect tunneling, for which the malicious class can be readily generated. Unsupervised anomaly detection is used when insufficient malicious traffic is available for training, which is often the case for malware.

Besides algorithm type, we identified two different feature engineering rationales: feature extraction from isolated query instances and from query sequences. Query sequences are formed by grouping queries by (a combination of) time, IP address or primary domain. Features are then derived from either all queries in the sequence at once, resulting in a single feature vector, or by using a sliding window over queries to produce as many vectors as observations.

Payload-only features proposed in current literature describe query space utilization, query structure, information density, lexical properties and DNS usage. Common query sequence features are statistics of aggregated payload features (e.g. average query space utilization) or describe behavioral characteristics (e.g. ratio of unique queries).

We were unable to derive a common definition of storage channel detection capability from current research. Works report incompatible performance metrics or metrics that are tied to the dataset used for research, exacerbated by benign class filtering to the extent that it no longer resembles real-world scenarios.

Nevertheless, common evaluation aspects are the detection rate of malicious threats and corresponding false positive rate. We argue that, besides threats seen during training, unseen threats should be incorporated for evaluation as well, as an important reason to use machine learning over rule-based detection is generalization. We therefore arrive at a notion of detection capability that consists of three

aspects: known threat detection, unseen threat detection and the amount of false alerts.

SQ2: What is the difference in detection capability between unsupervised anomaly detection and supervised classification?

Experiments with unsupervised (Extended) Isolation Forest anomaly detection algorithms and the supervised Random Forest classifier showed that models based on either learning type were able to achieve high recall on seen threats. The majority of experiments attained a macro-averaged recall of 98.6% or more over both connection tunneling and malware threats.

Classification experiments had significantly higher balanced accuracy scores on seen threats than than anomaly detection experiments. Classification models were able to almost perfectly separate the benign and malicious classes and the best configuration misclassified queries for just two primary domains.

Anomaly detection models suffered more from false positives query and domains than classification models. Between anomaly detection experiments, Extended Isolation Forest models (at least 0.38% FPR or 198 domains) improved over the regular Isolation Forest (at least 0.77% FPR or 1,345 domains) in terms of false positives, at similar detection rates.

Evaluation with traffic from 11 unseen storage channel threats showed that anomaly detection models generalize better than classification models. Although few experiments were able to detect every unseen threat, anomaly detection models consistently (> 90% of experiments) identified five DNS malware samples and three tunneling samples. While classification models also identified the tunneling samples, only three malware threats were consistently identified, at lower recall.

SQ3: What are the effects of considering only payload features, only behavioral features or using composite feature sets?

We extracted payload features from single DNS query instances and behavioral features from a time-based or fixed length sliding window over per-domain query sequences. Based on effective detection strategies identified in the literature survey, we designed distinct feature sets describing either payload or behavioral characteristics. Besides considering singular feature sets, we also combined features from different types to assess composite feature set performance. Our features use only the timestamp and payload of DNS queries, demonstrating the feasibility of this limited source of information.

Firstly, our experiments showed that classification models are mostly unaffected by feature selection and were able to detect at least 99.6% of queries from seen threats, irrespective of the feature selection. Anomaly detection experiments, on the other hand, achieved considerably lower macro-recall on the test set using payload-only features (78.7%) than behavioral-only features (at least 90.8%) or composite feature sets (at least 89.5%).

Besides high recall, our classification models achieved low false positive rates for all feature sets: the best performing models for each feature set composition type had 0.3% (payload-only) or <0.0002% (behavioral-only or composite).

Anomaly detection experiments had false positive rates significantly lower for composite feature sets than for others. The lowest observed rates were 0.9% (payload-only) or 0.38% (behavioral-only or composite), all by Extended Isolation Forest models.

In general, behavioral feature sets with a time component had a significantly higher balanced accuracy on seen threats than feature sets without it. This effect was not observed between behavioral features from a fixed-length sliding window.

Evaluation on unseen threats showed, however, that high recall on the test set did not translate to generalizable models. Out of our 92 experiments, only four were able to identify at least one query for every unseen sample. Overall, anomaly detection models were able to detect more unseen threats at higher detection rates than classification models. Behavioral features from a fixed-length window detected most unseen threats at the highest detection rates. However, well-generalizing models often

also generate many false positives.

Classification models demonstrated a distinct decrease in unseen threat macro-recall when adding payload features to a behavioral feature set. We therefore conclude that classification models overfit on threats in the training set when using payload features and that our training set did not contain diverse enough threats.

Given the variability in unseen threat detection rates and discrepancies between different sizes for the same window type, our experiments provide inconclusive evidence to rank or recommend window types, sizes and combinations. We recommend instead to *1)* optimize the window size(s) during model optimization, *2)* perform feature selection to remove redundant or uninformative features and *3)* tailor features to the window type, e.g. focus only uniqueness and information content in fixed-length windows and transfer rate in time windows.

How effective are machine learning methods at detecting covert DNS storage channels?

Combining the insights gained by answering the sub-questions of our research, we find that both unsupervised anomaly detection and supervised classification models are viable solutions to detect covert DNS storage channels. We showed that state-of-the-art performance is attainable with either algorithm type, achieving near-perfect recall on malicious storage channel traffic from two connection tunneling threats and four Point-of-Sale malware strains at low false positive rates.

We evaluated our models using a large-scale corporate DNS dataset of real-world proportions and a novel dataset of captured *iodine* and *dns2tcp* tunneling traffic and simulated credit card exfiltrations from *BernhardPOS*, *FrameworkPOS*, *MULTIGRAIN* and *UDPoS*. The malicious dataset is made public to support future research.

The majority of experiments achieved, irrespective of feature selection, a macro-recall on seen threats of more than 98.6%. Moreover, half of the classification experiments produced remarkably few false positive domains: at most 27 (0.009%) of 310,000 primary domains in the dataset.

Furthermore, real *UDPoS* traffic was detected accurately by every model, validating our malware simulation technique and demonstrating that carefully simulated malware traffic can be used to learn to detect real threats.

The main differences between algorithm types are increased false positive rates for anomaly detection and in turn decreased generalization performance of classification models. While behavioral-only feature sets consistently outperform payload-only features, our experiments are inconclusive regarding the relative performance of (combinations of) fixed-length and time-based sliding window feature sets.

Although an all-encompassing comparison with current literature is difficult due to differences in datasets, reported metrics and benign class filtering, we found that our payload-only method has a considerably lower false positive rate than a similar method proposed by Ahmed *et al.* [1]. Furthermore, evaluation on unseen threats is rarely substantially covered by current research, complicating accurate comparison.

Lastly, an important observation is that benign storage channels – DNS queries used for data transfer for legitimate purposes – have a considerable presence in our benign class. Benign storage channels influence both training and evaluation and pollute false positive metrics. We invalidate our initial assumption that their contamination is low and insignificant and recommend to direct efforts towards identifying and filtering benign storage channels before training.

8.1. Limitations

- We considered only one (core) algorithm for unsupervised anomaly detection and supervised classification. Other algorithms, e.g. logistic regression, (One-Class) Support Vector Machines or boosting, instead of bagging, classifiers could produce different results. However, given the magnitude of the difference between the tested algorithms, we suspect that using different algorithms will not close this gap.
- Our data processing and query aggregation pipeline assumes that queries arrive in order. We enforce this constraint by sorting queries per domain prior to feature extraction. When deploying this method on live (streaming) network traffic, however, order is not guaranteed and sliding window features may not be as effective.
- Due to limitations of our data collection method, queries are stored in lowercase, which reduced the effectiveness of features describing e.g. information density and lexical properties. Applying our methods to unaltered queries is expected to improve detection capability.
- Our results are based on a sizable sample of DNS traffic spanning three consecutive days. Although all models have been training and validated on traffic from separate days, the effects of concept drift, e.g. seasonal effects, have not been investigated and could influence detection performance on future traffic.
- Each experiment configuration has been executed only once, using a preset randomization seed. We have not investigated the stability of our results across multiple different seeds.

Bibliography

[1] Jawad Ahmed, Hassan Habibi Gharakheili, Qasim Raza, Craig Russell, and Vijay Sivaraman. Monitoring Enterprise DNS Queries for Detecting Data Exfiltration from Internal Hosts. *IEEE Transactions on Network and Service Management*, 17(1):265–279, September 2019. ISSN 1932-4537, 2373-7379. doi: 10.1109/TNSM.2019.2940735.

[2] Kamran Ahsan and Deepa Kundur. Practical Data Hiding in TCP/IP. *Proc. Workshop on Multimedia Security at ACM Multimedia '02*, 2(7):8, December 2002.

[3] M. Aiello, M. Mongelli, and G. Papaleo. DNS tunneling detection through statistical fingerprints of protocol messages and machine learning. *International Journal of Communication Systems*, 28 (14):1987–2002, 2015. ISSN 1099-1131. doi: 10.1002/dac.2836.

[4] Maurizio Aiello, Maurizio Mongelli, and Gianluca Papaleo. Basic classifiers for DNS tunneling detection. In *2013 IEEE Symposium on Computers and Communications (ISCC)*, pages 000880–000885. IEEE, July 2013. doi: 10.1109/ISCC.2013.6755060.

[5] Maurizio Aiello, Maurizio Mongelli, Enrico Cambiaso, and Gianluca Papaleo. Profiling DNS tunneling attacks with PCA and mutual information. *Logic Journal of IGPL*, 24(6):957–970, December 2016. ISSN 1367-0751, 1368-9894. doi: 10.1093/jigpal/jzw056.

[6] Maurizio Aiello, Maurizio Mongelli, Marco Muselli, and Damiano Verda. Unsupervised learning and rule extraction for Domain Name Server tunneling detection. *Internet Technology Letters*, 2 (2):e85, 2019. ISSN 2476-1508. doi: 10.1002/itl2.85.

[7] Ahmed Almusawi and Haleh Amintoosi. DNS tunneling detection method based on multilabel support vector machine. *Security and Communication Networks*, 2018, January 2018. ISSN 1939-0114. doi: 10.1155/2018/6137098.

[8] Md. Ahsan Ayub, Steven Smith, and Ambareen Siraj. A Protocol Independent Approach in Network Covert Channel Detection. In *2019 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, pages 165–170, August 2019. doi: 10.1109/CSE/EUC.2019.00040.

[9] Andreas Berg and Daniel Forsberg. Identifying DNS-tunneled traffic with predictive models. *CoRR*, abs/1906.11246, June 2019.

[10] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages I–115–I–123, Atlanta, GA, USA, June 2013. JMLR.org.

[11] Kenton Born and David Gustafson. Detecting DNS Tunnels Using Character Frequency Analysis. In *Proceedings of the 9th Annual Security Conference*, Las Vegas, NV, April 2010.

[12] Ron Bowes. Dnscat2, GitHub Repository. https://github.com/iagox86/dnscat2, 2020.

[13] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, October 2001. ISSN 1573-0565. doi: 10.1023/A:1010933404324.

[14] Yakov Bubnov. DNS Tunneling Detection Using Feedforward Neural Network. *European Journal of Engineering Research and Science*, 3(11):16–19, November 2018. ISSN 2506-8016. doi: 10.24018/ejers.2018.3.11.963.

[15] Yakov Bubnov. DNS Data Exfiltration Detection Using Online Planning for POMDP. *European Journal of Engineering Research and Science*, 4(9):22–25, September 2019. ISSN 2506-8016. doi: 10.24018/ejers.2019.4.9.1500.

[16] Anna L. Buczak, Paul A. Hanke, George J. Cancro, Michael K. Toma, Lanier A. Watkins, and Jeffrey S. Chavis. Detection of Tunnels in PCAP Data by Random Forests. In *Proceedings of the 11th Annual Cyber and Information Security Research Conference*, CISRC '16, pages 1–4, Oak Ridge, TN, USA, April 2016. Association for Computing Machinery. ISBN 978-1-4503-3752-6. doi: 10.1145/2897795.2897804.

[17] Randy Bush and Robert Elz. Clarifications to the DNS Specification. RFC 2181, RFC Editor, July 1997.

[18] E. Cambiaso, M. Aiello, M. Mongelli, and G. Papaleo. Feature transformation and Mutual Information for DNS tunneling analysis. In *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 957–959, July 2016. doi: 10.1109/ICUFN.2016.7536939.

[19] Tomas Cejka, Zdenek Rosa, and Hana Kubatova. Stream-wise detection of surreptitious traffic over DNS. In *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 300–304, December 2014. doi: 10.1109/CAMAD.2014.7033254.

[20] Ping Chen, Lieven Desmet, and Christophe Huygens. A Study on Advanced Persistent Threats. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Camille Salinesi, Moira C. Norrie, and Óscar Pastor, editors, *Advanced Information Systems Engineering*, volume 7908, pages 63–72. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. ISBN 978-3-642-38708-1 978-3-642-38709-8. doi: 10.1007/978-3-662-44885-4_5.

[21] Yizheng Chen, Manos Antonakakis, Roberto Perdisci, Yacin Nadji, David Dagon, and Wenke Lee. DNS Noise: Measuring the Pervasiveness of Disposable Domains in Modern DNS Traffic. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 598–609, Atlanta, GA, USA, June 2014. IEEE. ISBN 978-1-4799-2233-8. doi: 10.1109/DSN.2014.61.

[22] Cian Lynch, Dimiter Andonov, and Claudiu Teodorescu. MULTIGRAIN – Point of Sale Attackers Make an Unhealthy Addition to the Pantry. https://www.fireeye.com/blog/threat-research/2016/04/multigrain_pointo.html, April 2016.

[23] Cisco. Umbrella Popularity List. https://s3-us-west-1.amazonaws.com/umbrella-static/index.html, 2020.

[24] David Cortes. IsoTree, GitHub Repository. https://github.com/david-cortes/isotree, May 2021.

[25] Adam M. Costello. Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA). RFC 3492, RFC Editor, March 2003.

[26] Anirban Das, Min-Yi Shen, Madhu Shashanka, and Jisheng Wang. Detection of Exfiltration and Tunneling over DNS. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 737–742, December 2017. doi: 10.1109/ICMLA.2017.00-71.

[27] Olivier Dembour and Nicolas Collignon. Dns2tcp - Hervé Schauer Consultants (HSC). https://web.archive.org/web/20070521023610/http://www.hsc.fr/ressources/outils/dns2tcp/, February 2007.

[28] M. Dusi, M. Crotti, F. Gringoli, and L. Salgarelli. Tunnel Hunter: Detecting application-layer tunnels with statistical fingerprinting. *Computer Networks*, 53(1):81–97, January 2009. ISSN 1389-1286. doi: 10.1016/j.comnet.2008.09.010.

[29] Erik Ekman and Bjorn Andersson. Iodine, GitHub Repository. https://github.com/yarrick/iodine, April 2021.

[30] Wendy Ellens, Piotr Żuraniewski, Anna Sperotto, Harm Schotanus, Michel Mandjes, and Erik Meeuwissen. Flow-Based Detection of DNS Tunnels. In Guillaume Doyen, Martin Waldburger, Pavel Čeleda, Anna Sperotto, and Burkhard Stiller, editors, *Emerging Management Mechanisms for the Future Internet*, Lecture Notes in Computer Science, pages 124–135, Berlin, Heidelberg, 2013. Springer. ISBN 978-3-642-38998-6. doi: 10.1007/978-3-642-38998-6_16.

[31] Eric Merritt. Another Brick in the FrameworkPoS. https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/another-brick-in-the-frameworkpos/, December 2015.

[32] Matthieu Faou, Mathieu Tartare, and Thomas Dupuy. Exchange servers under siege from at least 10 APT groups. https://www.welivesecurity.com/2021/03/10/exchange-servers-under-siege-10-apt-groups/, March 2021.

[33] Greg Farnham. Detecting DNS Tunneling. *SANS Institute InfoSec Reading Room*, 9:1–32, February 2013.

[34] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, June 2006. ISSN 0167-8655. doi: 10.1016/j.patrec.2005.10.010.

[35] Kazunori Fujiwara, Andrew Sullivan, and Paul Hoffman. DNS Terminology. RFC 8499, RFC Editor, January 2019.

[36] Annarita Giani, Vincent H. Berk, and George V. Cybenko. Data exfiltration and covert channels. In Edward M. Carapezza, editor, *Defense and Security Symposium*, page 620103, Orlando (Kissimmee), FL, May 2006. doi: 10.1117/12.670123.

[37] Sahand Hariri, Matias Carrasco Kind, and Robert J. Brunner. Extended Isolation Forest. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, November 2018. ISSN 1041-4347, 1558-2191, 2326-3865. doi: 10.1109/TKDE.2019.2947676.

[38] Nick Hoffman and Jeremy Humble. BernhardPOS. https://securitykitten.github.io/2015/07/14/bernhardpos.html, July 2015.

[39] R. J. Hofstede, Pavel Celeda, Brian Trammell, Idilio Drago, R. Sadre, Anna Sperotto, and Aiko Pras. Flow monitoring explained: from packet capture to data analysis with NetFlow and IPFIX. *IEEE communications surveys & tutorials*, 16(4):2037–2064, November 2014. ISSN 1553-877X. doi: 10.1109/COMST.2014.2321898.

[40] Myles Hollander, Douglas A. Wolfe, and Eric Chicken. *Nonparametric Statistical Methods*. John Wiley & Sons, November 2013. ISBN 978-1-118-55329-9.

[41] Irvin Homem and Panagiotis Papapetrou. Harnessing Predictive Models for Assisting Network Forensic Investigations of DNS Tunnels. In *12th ADFSL Conference on Digital Forensics, Security and Law (2017)*, page 17, Daytona Beach, Florida, 2017.

[42] Irvin Homem, Panagiotis Papapetrou, and Spyridon Dosis. Entropy-based Prediction of Network Protocols in the Forensic Analysis of DNS Tunnels. *arXiv:1709.06363 [cs]*, September 2017.

[43] International Organization for Standardization. Information technology — Identification cards — Financial transaction cards. Standard ISO/IEC 7813:2006, ISO/IEC, 2006.

[44] Steve Jaworski. Using Splunk to Detect DNS Tunneling. *SANS Institute InfoSec Reading Room*, page 118, 2016.

[45] Shachar Kaufman, Saharon Rosset, Claudia Perlich, and Ori Stitelman. Leakage in data mining: Formulation, detection, and avoidance. *ACM Transactions on Knowledge Discovery from Data*, 6 (4):15:1–15:21, December 2012. ISSN 1556-4681. doi: 10.1145/2382577.2382579.

[46] Panagiotis Kintis, Najmeh Miramirkhani, Charles Lever, Yizheng Chen, Rosa Romero-Gómez, Nikolaos Pitropakis, Nick Nikiforakis, and Manos Antonakakis. Hiding in Plain Sight: A Longitudinal Study of Combosquatting Abuse. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 569–586, New York, NY,

USA, October 2017. Association for Computing Machinery. ISBN 978-1-4503-4946-8. doi: 10.1145/3133956.3134002.

[47] Brian Krebs. Home Depot: 56M Cards Impacted, Malware Contained. https://krebsonsecurity.com/2014/09/home-depot-56m-cards-impacted-malware-contained/, September 2014.

[48] Brian Krebs. Deconstructing the 2014 Sally Beauty Breach. https://krebsonsecurity.com/2015/05/deconstructing-the-2014-sally-beauty-breach/, May 2015.

[49] Vitali Kremez. FIN6 “FrameworkPOS”: Point-of-Sale Malware Analysis & Internals. https://labs.sentinelone.com/fin6-frameworkpos-point-of-sale-malware-analysis-internals-2/, September 2019.

[50] Marc Krochmal and Stuart Cheshire. Special-Use Domain Names. RFC 6761, RFC Editor, February 2013.

[51] Marc Krochmal and Stuart Cheshire. Multicast DNS. RFC 6762, RFC Editor, February 2013.

[52] Huaqing Lin, Gao Liu, and Zheng Yan. Detection of Application-Layer Tunnels with Rules and Machine Learning. In Guojun Wang, Jun Feng, Md Zakirul Alam Bhuiyan, and Rongxing Lu, editors, *Security, Privacy, and Anonymity in Computation, Communication, and Storage*, Lecture Notes in Computer Science, pages 441–455, Cham, 2019. Springer International Publishing. ISBN 978-3-030-24907-6. doi: 10.1007/978-3-030-24907-6_33.

[53] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation Forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, December 2008. doi: 10.1109/ICDM.2008.17.

[54] Jingkun Liu, Shuhao Li, Yongzheng Zhang, Jun Xiao, Peng Chang, and Chengwei Peng. Detecting DNS Tunnel through Binary-Classification Based on Behavior Features. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 339–346, August 2017. doi: 10.1109/Trustcom/BigDataSE/ICESS.2017.256.

[55] Martin Lee, Jaeson Schultz, and Warren Mercer. Detecting DNS Data Exfiltration. http://blog.talosintelligence.com/2016/06/detecting-dns-data-exfiltration.html, June 2016.

[56] Sara Marie Mc Carthy, Arunesh Sinha, Milind Tambe, and Pratyusa Manadhata. Data Exfiltration Detection and Prevention: Virtually Distributed POMDPs for Practically Safer Networks. In Quanyan Zhu, Tansu Alpcan, Emmanouil Panaousis, Milind Tambe, and William Casey, editors, *Decision and Game Theory for Security*, Lecture Notes in Computer Science, pages 39–61, Cham, 2016. Springer International Publishing. ISBN 978-3-319-47413-7. doi: 10.1007/978-3-319-47413-7_3.

[57] Luis Mendieta. Three Month FrameworkPOS Malware Campaign Nabs ~43,000 Credit Cards from Point of Sale Systems. https://www.anomali.com/blog/three-month-frameworkpos-malware-campaign-nabs-43000-credits-cards-from-poi, February 2016.

[58] Alessio Merlo, Gianluca Papaleo, Stefano Veneziano, and Maurizio Aiello. A Comparative Performance Evaluation of DNS Tunneling Tools. In Álvaro Herrero and Emilio Corchado, editors, *Computational Intelligence in Security for Information Systems*, Lecture Notes in Computer Science, pages 84–91, Berlin, Heidelberg, 2011. Springer. ISBN 978-3-642-21323-6. doi: 10.1007/978-3-642-21323-6_11.

[59] P. V. Mockapetris. Domain names: Concepts and facilities. RFC 882, RFC Editor, November 1983.

[60] P. V. Mockapetris. Domain names - concepts and facilities. RFC 1034, RFC Editor, November 1987.

[61] Mozilla Foundation. Public Suffix List. https://publicsuffix.org/, June 2020.

[62] Asaf Nadler, Avi Aminov, and Asaf Shabtai. Detection of Malicious and Low Throughput Data Exfiltration Over the DNS Protocol. *Computers & Security*, 80:36–53, 2019. ISSN 0167-4048. doi: 10.1016/j.cose.2018.09.006.

[63] Franco Palau, Carlos Catania, Jorge Guerra, Sebastian Garcia, and Maria Rigaki. DNS Threats Dataset. https://datahub.io/palaufranco12/asai-2019-multiclass/v/1, 2019.

[64] Vern Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31 (23):2435–2463, December 1999. ISSN 1389-1286. doi: 10.1016/S1389-1286(99)00112-7.

[65] Vern Paxson, Mihai Christodorescu, Mobin Javed, Josyula Rao, Reiner Sailer, Douglas Lee Schales, Mark Stoecklin, Kurt Thomas, Wietse Venema, and Nicholas Weaver. Practical Comprehensive Bounds on Surreptitious Communication over DNS. In *Presented as Part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 17–32, 2013. ISBN 978-1-931971-03-4.

[66] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, and Vincent Dubourg. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[67] Richard Preston. DNS Tunneling Detection with Supervised Learning. In *2019 IEEE International Symposium on Technologies for Homeland Security (HST)*, pages 1–6, November 2019. doi: 10.1109/HST47167.2019.9032913.

[68] Philipp Probst and Anne-Laure Boulesteix. To Tune or Not to Tune the Number of Trees in Random Forest. *J. Mach. Learn. Res.*, 18(1):6673–6690, 2017.

[69] Cheng Qi, Xiaojun Chen, Cui Xu, Jinqiao Shi, and Peipeng Liu. A Bigram based Real Time DNS Tunnel Detection Approach. *Procedia Computer Science*, 17:852–860, January 2013. ISSN 1877-0509. doi: 10.1016/j.procs.2013.05.109.

[70] Laura Elena Raileanu and Kilian Stoffel. Theoretical Comparison between the Gini Index and Information Gain Criteria. *Annals of Mathematics and Artificial Intelligence*, 41(1):77–93, May 2004. ISSN 1573-7470. doi: 10.1023/B:AMAI.0000018580.96245.c6.

[71] Paul Rascagneres. New FrameworkPOS variant exfiltrates data via DNS requests. https://www.gdatasoftware.com/blog/2014/10/23942-new-frameworkpos-variant-exfiltrates-data-via-dns-requests, November 2016.

[72] Robert Neumann and Luke Somerville. UDPoS - exfiltrating credit card data via DNS. https://www.forcepoint.com/blog/x-labs/udpos-exfiltrating-credit-card-data-dns, February 2018.

[73] Salvatore Saeli, Federica Bisio, Pierangelo Lombardo, and Danilo Massa. DNS Covert Channel Detection via Behavioral Analysis: A Machine Learning Approach. In *2019 14th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 46–55, Nantucket, MA, USA, October 2019. Malware Conference. ISBN 978-0-578-58383-9.

[74] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition*. Elsevier, fourth edition, 2009. ISBN 978-1-59749-272-0. doi: 10.1016/B978-1-59749-272-0.X0001-2.

[75] Saeed Shafieian, Daniel Smith, and Mohammad Zulkernine. Detecting DNS Tunneling Using Ensemble Learning. In Zheng Yan, Refik Molva, Wojciech Mazurczyk, and Raimo Kantola, editors, *Network and System Security*, volume 10394, pages 112–127. Springer International Publishing, Cham, 2017. ISBN 978-3-319-64700-5 978-3-319-64701-2. doi: 10.1007/978-3-319-64701-2_9.

[76] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27 (3):379–423, July 1948. ISSN 0005-8580. doi: 10.1002/j.1538-7305.1948.tb01338.x.

[77] Stephen Sheridan and Anthony Keane. Detection of dns based covert channels. In *European Conference on Cyber Warfare and Security*, page 267. Academic Conferences International Limited, 2015.

[78] Simon Josefsson. The Base16, Base32, and Base64 Data Encodings. RFC 4648, RFC Editor, October 2006.

[79] Jacob Steadman and Sandra Scott-Hayward. DNSxD: Detecting Data Exfiltration Over DNS. In *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–6, November 2018. doi: 10.1109/NFV-SDN.2018.8725640.

[80] Matija Stevanovic, Jens Myrup Pedersen, Alessandro D'Alconzo, Stefan Ruehrup, and Andreas Berger. On the ground truth problem of malicious DNS traffic analysis. *Computers & Security*, 55: 142–158, November 2015. ISSN 0167-4048. doi: 10.1016/j.cose.2015.09.004.

[81] Dennis Tatang, Florian Quinkert, and Thorsten Holz. Below the Radar: Spotting DNS Tunnels in Newly Observed Hostnames in the Wild. In *2019 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–15, November 2019. doi: 10.1109/eCrime47957.2019.9037595.

[82] The BlackBerry Cylance Threat Research Team. Threat Spotlight: Inside UDPoS Malware. https://blogs.blackberry.com/en/2018/02/threat-spotlight-inside-udpos-malware, February 2018.

[83] Tom van Leijenhorst, Kwan-Wu Chin, and Darryn Lowe. On the viability and performance of DNS tunneling. *International Conference on Information Technology and Applications*, pages 560–566, 2008.

[84] Sophie Walther. WINNTI GROUP: Insights From the Past. https://quointelligence.eu/2020/04/winnti-group-insights-from-the-past/, April 2020.

[85] Bin Yu, Femi Olumofin, Les Smith, and Mark Threefoot. Behavior Analysis based DNS Tunneling Detection and Classification with Big Data Technologies:. In *Proceedings of the International Conference on Internet of Things and Big Data*, pages 284–290, Rome, Italy, 2016. SCITEPRESS - Science and and Technology Publications. ISBN 978-989-758-183-0. doi: 10.5220/0005795002840290.

Acronyms

Acronym	Meaning
APT	advanced persistent threat
DNS	domain name system
EIF	Extended Isolation Forest
FPR	false positive rate
FQDN	fully qualified domain name
i.i.d.	independent and identically distributed
IDS	intrusion detection system
iForest	Isolation Forest
k-NN	k-nearest neighbour
LDA	linear discriminant analysis
LDH	letters, digits and hyphens
PSL	Public Suffix List
RF	random forest
SVM	support vector machine
TLD	top-level domain

A

Malicious Dataset Reference

A.1. Statistics

Tables A.1 to A.5 provide an overview of detailed traffic statistics, per dataset. Query frequency values are expressed in queries per second (*q/s*).

Table A.1: Dataset statistics – dns2tcp

Query type	Compression	Duration	# queries (unique)	Mean q/s	Max. q/s
KEY	no	19m	412,275 (95%)	350	6,671
	yes	19m	490,970 (93%)	409	6,735
TXT	no	19m	422,272 (91%)	356	6,814
	yes	19m	385,532 (95%)	324	5,774

Table A.2: Dataset statistics – Berg2019 (dnscat2)

Tunneled protocol	Duration	# queries (unique)	Mean q/s	Max. q/s
SFTP	43h 46m	994,564 (100%)	6	20
SSH	34h 51m	1,358,727 (100%)	11	24
TELNET	38h 44m	1,148,598 (100%)	8	24

Table A.3: Dataset statistics – simulated malware

Malware variant	Schedule	Duration	# queries (unique)	Mean q/s	Max. q/s
BernhardPOS	Per second	11h 59m	42,945 (100%)	1	2
	Per minute	11h 58m	718 (100%)	0	1
	Per 5 minutes	11h 54m	144 (100%)	0	1
FrameworkPOS	Per second	12h 25m	42,918 (100%)	1	2
	Per minute	12h 25m	720 (100%)	0	1
	Per 5 minutes	12h 22m	146 (100%)	0	1
MULTIGRAIN	Per second	12h 25m	42,871 (100%)	1	2
	Per minute	12h 24m	720 (100%)	0	1
	Per 5 minutes	12h 23m	145 (100%)	0	1
UDPoS	Per second	12h 25m	43,057 (100%)	1	2
	Per minute	12h 24m	838 (100%)	0	2
	Per 5 minutes	12h 21m	258 (100%)	0	2

Table A.4: Dataset statistics – iodine

Qtype	Encoding	Max. len.	Duration	# queries (unique)	Mean q/s	Max. q/s
MX	Base128	100	19m	251,542 (94%)	220	2,079
		255	23m	235,381 (98%)	168	1,745
	Base32	150	28m	399,935 (96%)	232	2,056
		255	25m	268,963 (96%)	173	1,932
	Base64	100	19m	225,960 (90%)	197	2,172
		255	23m	265,898 (97%)	187	1,883
	Base64u	100	19m	234,647 (92%)	205	2,097
		255	23m	258,331 (97%)	186	1,860
NULL	Base128	100	19m	229,372 (95%)	198	2,154
		255	21m	202,654 (97%)	154	1,816
	Base32	150	30m	365,301 (97%)	197	2,165
		255	25m	257,098 (95%)	169	2,057
	Base64	100	19m	201,223 (89%)	173	2,174
		255	24m	235,652 (96%)	162	1,933
	Base64u	100	19m	198,042 (92%)	171	2,218
		255	25m	226,884 (96%)	149	2,004
PRIVATE	Base128	100	17m	277,253 (95%)	268	2,046
		255	22m	320,593 (97%)	236	1,973
	Base32	150	30m	446,138 (97%)	245	2,403
		255	24m	377,482 (96%)	261	2,067
	Base64	100	18m	267,675 (93%)	240	2,119
		255	22m	327,494 (98%)	240	1,914
	Base64u	100	18m	251,619 (95%)	221	2,236
		255	23m	369,417 (97%)	263	2,036
SRV	Base128	100	21m	246,591 (94%)	188	2,097
		255	23m	216,968 (97%)	157	1,777
	Base32	150	28m	384,484 (100%)	226	2,090
		255	24m	261,004 (96%)	176	1,927
	Base64	100	18m	248,645 (92%)	219	2,072
		255	25m	267,422 (97%)	176	1,936
	Base64u	100	19m	248,460 (91%)	216	2,157
		255	25m	272,359 (97%)	175	1,890
TXT	Base128	100	18m	241,593 (93%)	212	2,185
		255	23m	188,039 (99%)	133	1,798
	Base32	150	27m	377,525 (96%)	227	2,139
		255	22m	234,182 (96%)	173	1,964
	Base64	100	18m	242,199 (92%)	212	2,141
		255	24m	213,154 (96%)	144	1,956
	Base64u	100	18m	218,453 (90%)	192	2,204
		255	19m	209,597 (95%)	175	1,952

Table A.5: Dataset statistics – sandbox samples

Malware variant	Duration	# queries (unique)	Mean q/s	Max. q/s
BondUpdater	<1m	23 (100%)	6	12
Carbanak	2m	791 (100%)	6	7
CobaltStrike	3m	262 (100%)	1	17
Denis	<1m	23 (100%)	1	1
DNSpionage	<1m	13 (31%)	1	6
ISMDoor	4m	2,296 (26%)	8	58
Pisloader (1)	2m	13 (100%)	0	1
Pisloader (2)	4m	78 (33%)	0	3
UDPoS	1m	121 (100%)	1	2

A.2. Sample Tunneling Queries

Example connection tunneling queries generated by *iodine* and *dns2tcp* are provided below. Note that these queries have been processed by Zeek v3.1.4, i.e. non-ASCII characters are hex-escaped and the letter case is lost.

iodine

Handshake	`z2lhaa-la-fl\xfbte-na\xefve-fran\xe7aise-est-retir\xe9-\xe0-cr\xe8te[.]tun.lan`
Beacon	`paaahpvy[.]tun.lan`
Data (Base32)	`rbkadzz123twoz1hm3twoz1hm3twoz1hm3twoz1hm3twoz1hm3twoz1hm3twoz1hm3twoz.1hm3twoz1hm3twoz1hm3twoz1hm3twoz1hm3twoz1hm3twoz1hm3twoz1hm3twoz1.hm3twoz1hm3twoz1hm3[.]tun.lan`
Data (Base64)	`0iaa9enpjyobgcgvgsenpdwbwynvrxcxiyatejjtawc-nswcdzwccbuwstm9+m.dawsba3stbxcj1iobibemdkzmya[.]tun.lan`
Data (Base64u)	`02fa3enpjyobgcgvgsei447dgw1b_lhcjixmqm55bxmppokn60bltae9dbop_n.9sygbgzobho5e1bdrf-gbwda-mr[.]tun.lan`
Data (Base128)	`0udaw82\xca2hb\xbe\xeey\xd6ro\xe3\xf1\xc1\xbe\xeewg\xec\xd6\xbeujr\xc2\xe0dz7\xf5\xbf1e4\xe4i\xd9p\xcfd9\xd3w\xe6\xc8z\xfdo\xed\xc4tb\xbe\xe2\xf04.nsk\xdb\xdc\xf5\xe9\xdb\xf6\xc6\xc4k\xc7k\xdf8\xd6ulyvc\xfcrgux[.]tun.lan`

dns2tcp

Handshake	`tnybgaabadjbree5mkvfnzu4rjm0qkuwourenevdmdgwmta0mzg0rdvemky5mtc.=auth[.]tun.lan`
List Resources	`tnxqegj9aa.=resource[.]tun.lan`
Beacon	`ko8agaawba[.]tun.lan`
Data	`ko8amabick2swmj1k/3fwiw5su+iyyaa4z9fex24kvj4r6dthqrdahxfyndautr.5+opnfvgibqttstvktbwngy7geaf/jjybxm/asmc2t/f3q+zjfpgaq+tuqy1ols.olnhsyph8hm1mg71evppkwrhosg9al09mtzugjmnbxb5qb27xlawjyt7qa[.]tun.lan`

A.3. Sample Malware Queries

An overview of the implemented query structures for the PoS-malware variants BernhardPOS, FrameworkPOS, MULTIGRAIN and UDPoS is included below. Refer to `https://github.com/tudelft-cda-lab/dns-storage-channel-detection/` for implementation details.

BernhardPOS

Exfil T2 `Pz0lOSwoPjouOikpOTggOicvPzgiPC4uODMjPyktOT8gOSktOgAA[.]29a.de`

FrameworkPOS

Notice `de9186ef.grp3.notice.eec79fc2e9f9ccf9[.]ns.a23-33-37-54-deploy-akamaitechnologies.com`

Install `de9186ef.grp3.ping.adm.cdd2e9d2e9cde9cdd2cd.cdc4cde9cdded2e9fcfce9defe.f0e197ecfdec[.]ns.a23-33-37-54-deploy-akamaitechnologies.com`

Exfil T1 `de9186ef.grp3.tt1.dcdccdc4dcc8e8d2dec4c8d2e8c8e8fc.d8c3ced8c3cec3e792fdeded97cee9e9e9e9e9e9e9e9e9e9e9e9e3d2.defec4fcdec8d0dee8d0fcdce8fec8d2c8cdcdcdfcdedefec8c4ded0dcde95.fee8fefcc8d2e8dcc8cd[.]ns.a23-33-37-54-deploy-akamaitechnologies.com`

Exfil T2 `de9186ef.grp3.tt2.fefec4fec8c4d2defec8dcfcc8dcd0fe.d2e8fecdc4c8d0dedecdfec4fefed2e8c8d2cdc4e9e9e9e9e9e9e9e9e9e9[.]ns.a23-33-37-54-deploy-akamaitechnologies.com`

MULTIGRAIN

Install `install.oa6viscfknevgjtnhuzdiobxge3denjwgathmpjtfyya[.]dojfgj.com`

Exfil `log.gi2dqnzrgyzdknrq.klipzogcfmuw2el7q4uurokievjeprybazrwlhi37tn4mkjzwop6mkpyluhzg3n.euweezoz45mzt2bo2lpqdmeuldrea6gmpjrtyoky4h2t5tmojfh3vn57uctjgpr.o47q5vrl45jssaix5tuziuyu5szjq2f3mshfpzvu3iyjljinh5mmay4tqjke2rd.g37zthe4qidl2lqo[.]dojfgj.com`

UDPoS

Info `e1ccdb67afe76cc.bin.9213523bf7cf221106ae8cc13f4005.bdd9e1fdf110dc5f741d58e289dea4.d104180cfb76dc7a8b39d074de8e8c.bcd7ca60892470c848172950c3d588[.]ns.service-logmeln.network`

Notice `e1ccdb67afe76cc.note.c375522792b747677cd1a8cd3a4d1.6b7d3f2f8fe0dbc42640036abc68a.f78b19435eb3258567df6ad7699b8.585e5889073d16128d5580a1967.ns.service-logmeln[.]network`

Install `e1ccdb67afe76cc.ping.c375522792b747677cd1a8.cd3a4d16b7d3f2f8fe0dbc.42640054f283ceabdc0b15.6ef772d2729b35dd44e9[.]ns.service-logmeln.network`

Exfil T2 `e1ccdb67afe76cc.trp.da60294782b5516e77d7b6cb.324d0ab5d2828d857fa02c0d.614a92fdb7daa47c697d987a.d1758031d571d28b8e8ce0[.]ns.service-logmeln.network`

B

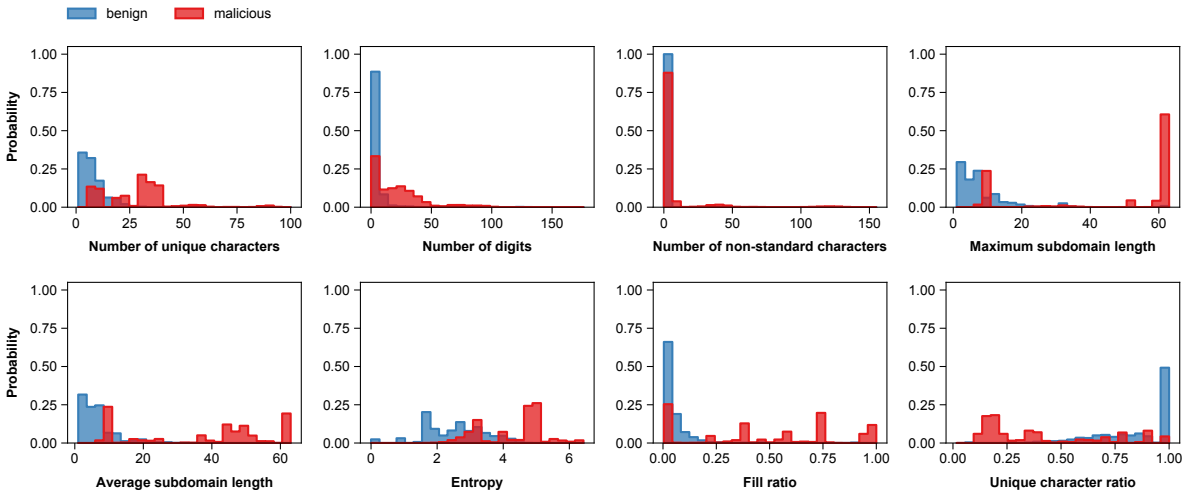
Feature Distribution Visualizations

This appendix contains probability density plots for all feature sets used in this work. The following Figures B.1 to B.6 show for each class, as well as for group within the malicious class, the probability density per feature.

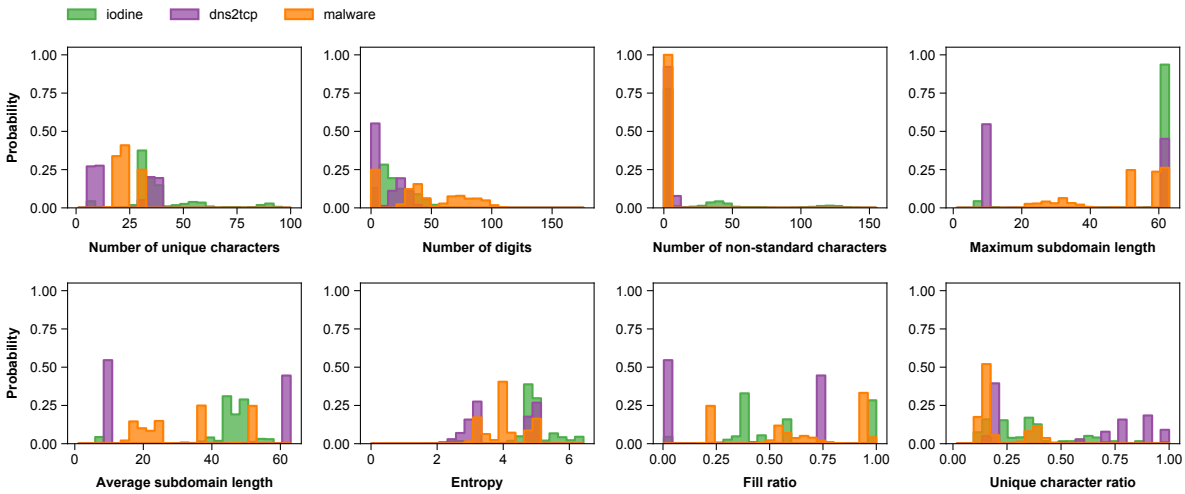
For the benign class, only features extracted from traffic of the second and third days are used, to reduce the effect of novel primary domains and empty sliding windows. For the malicious class, all *dns2tcp* and *malware* datasets are included, and for *iodine* the datasets with query types `TXT` and `NULL`.

B.1. Payload-only

The features visualized in this section are extracted from single query instances (see Section 5.2.2).



(a) Feature distribution, payload-only features

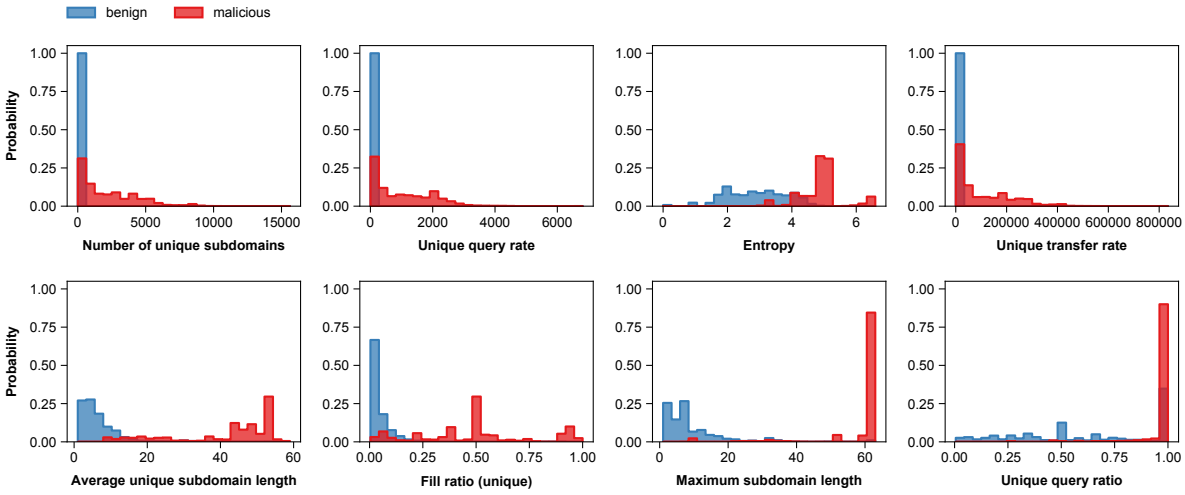


(b) Feature distribution per malicious group, payload-only features

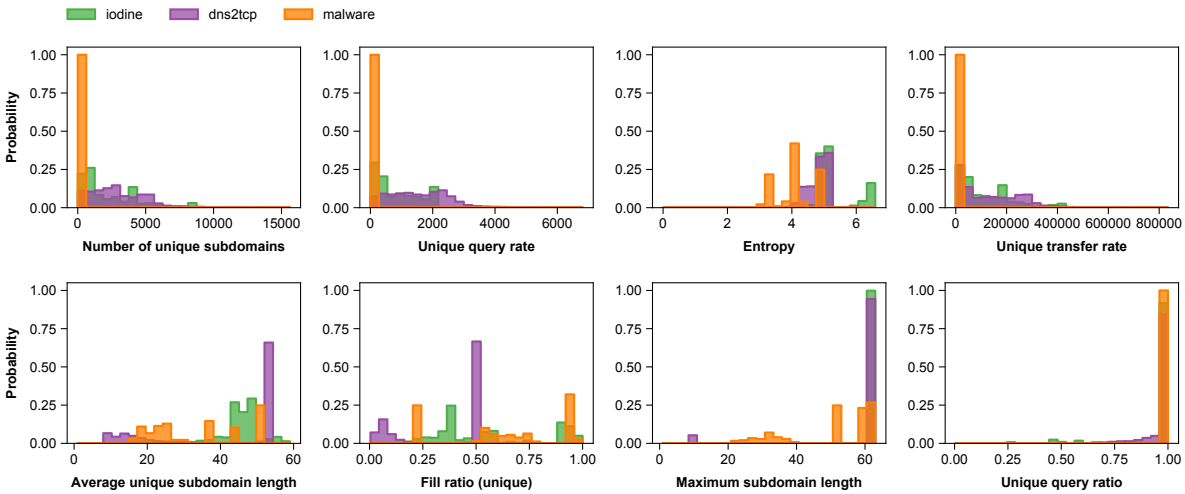
Figure B.1: Feature distribution, payload-only features

B.2. Time-Based Sliding Window

The features in this section are extracted from a time-based sliding window, with lengths $\delta = 1, 2, 5$ (see Section 5.2.3).

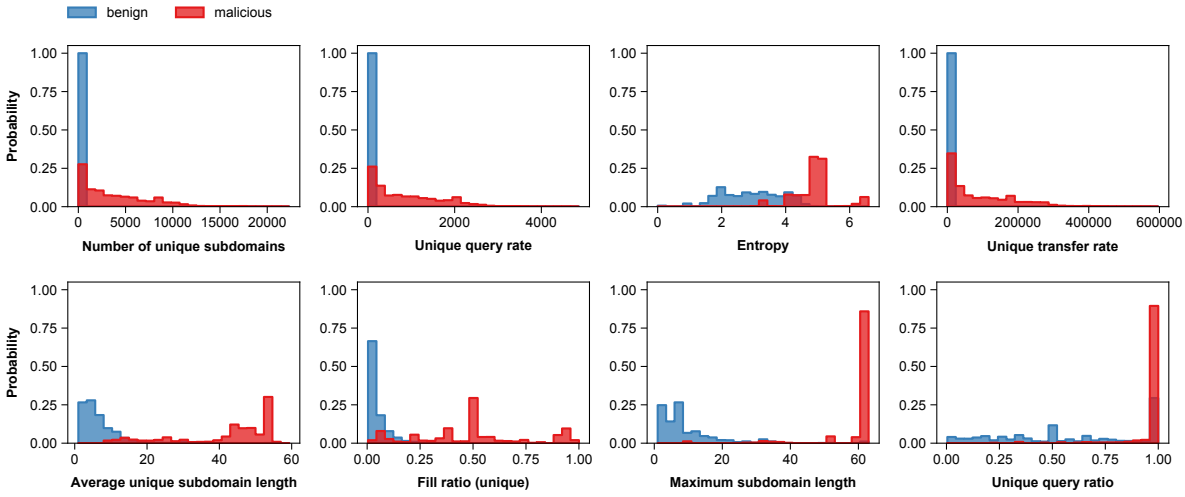


(a) Per class

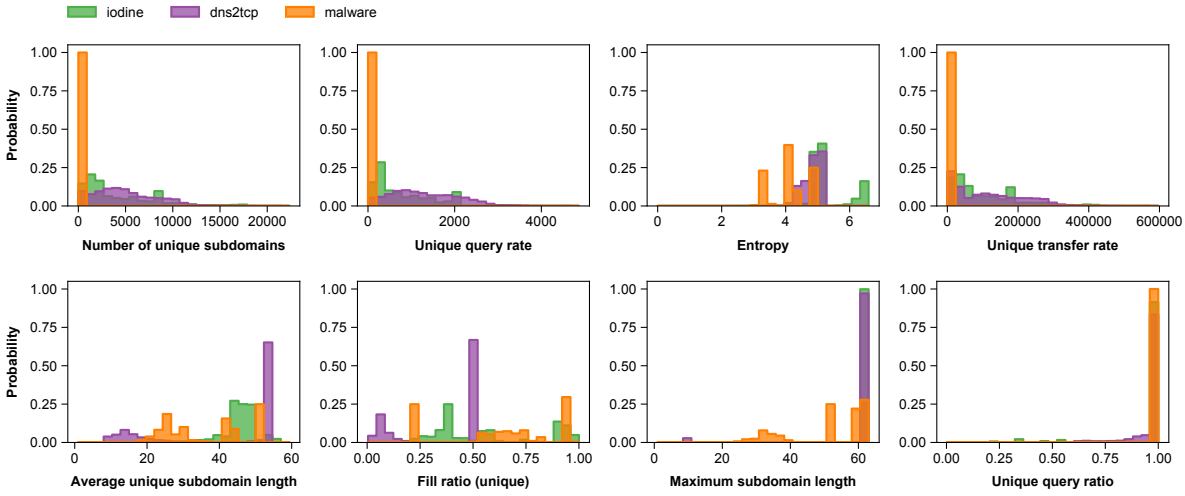


(b) Per malicious group

Figure B.2: Feature distribution, time window (δ = 1 second)

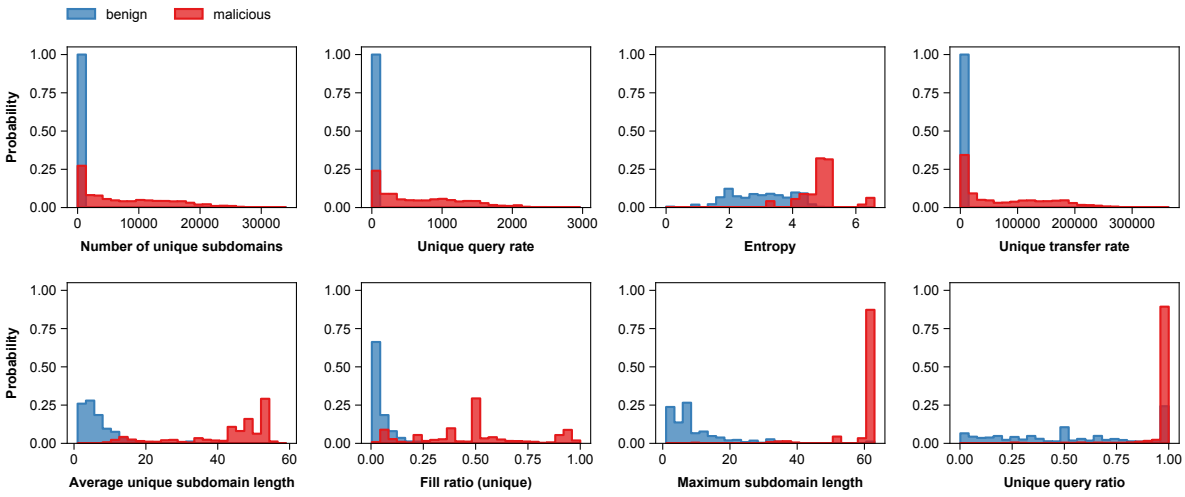


(a) Per class

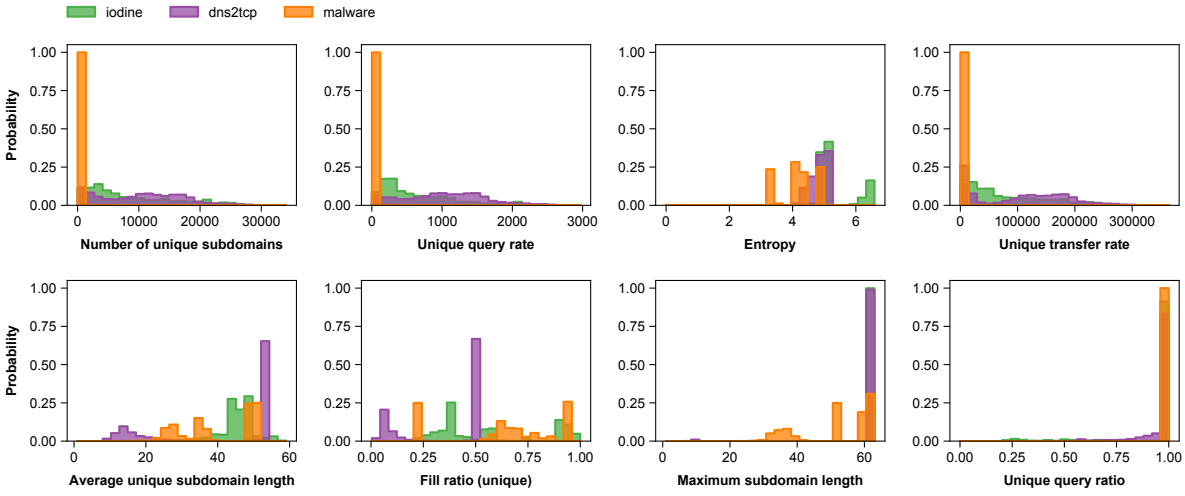


(b) Per malicious group

Figure B.3: Feature distribution, time window (δ = 2 seconds)



(a) Per class

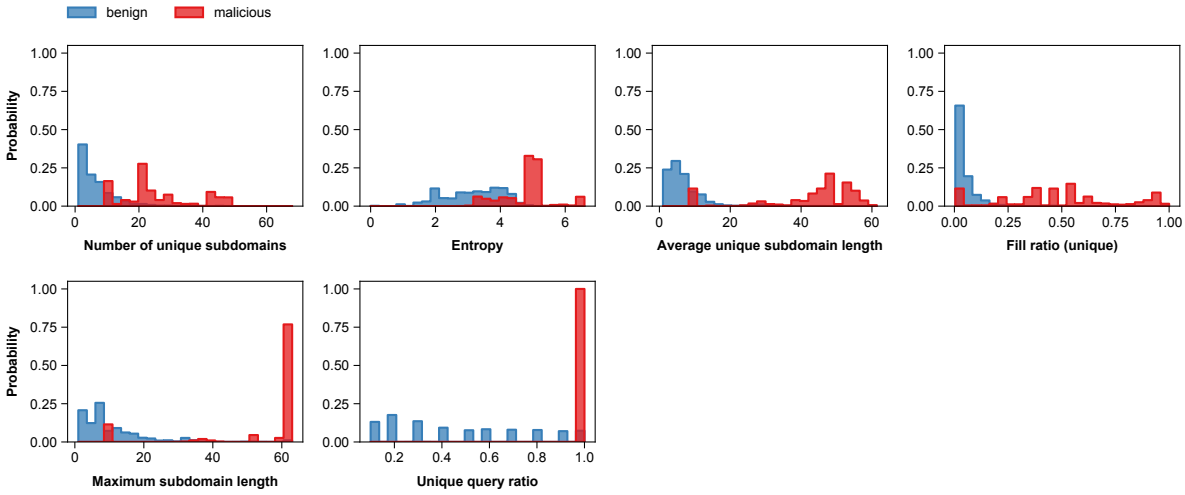


(b) Per malicious group

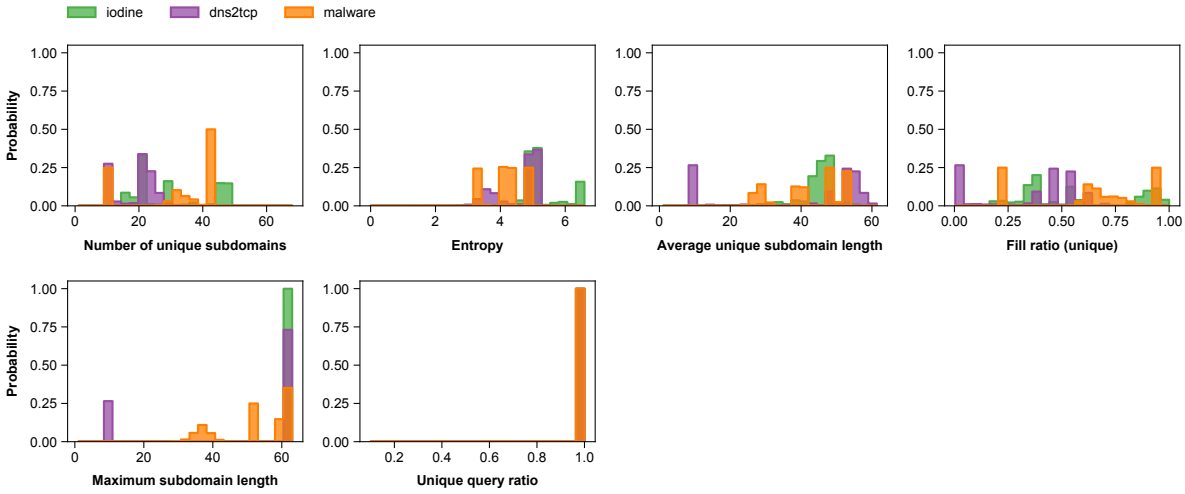
Figure B.4: Feature distribution, time window (δ = 5 seconds)

B.3. Fixed-Length Sliding Window

The features in this section are extracted from a fixed-length sliding window, with lengths $\lambda = 10, 20$ (see Section 5.2.3).

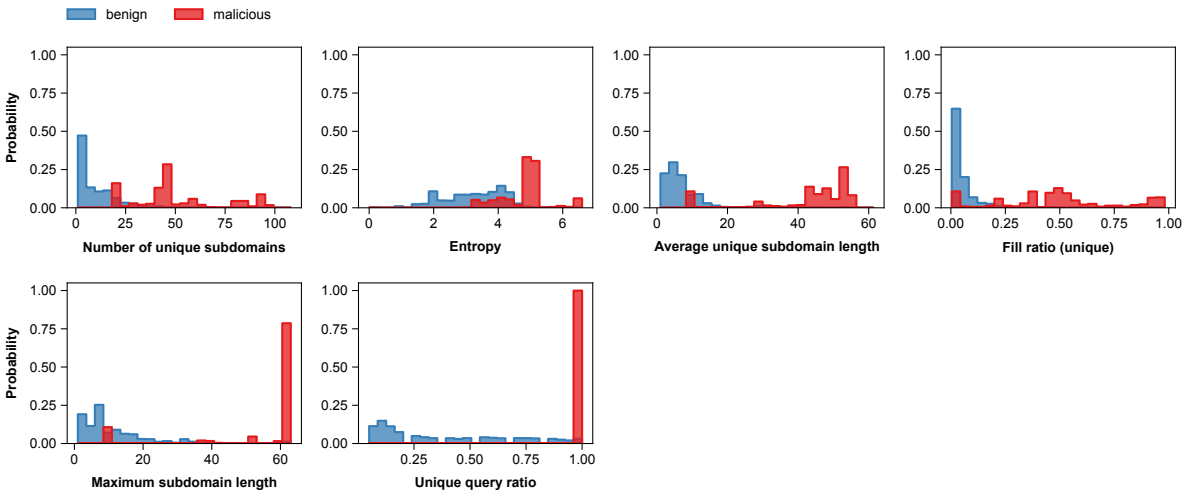


(a) Per class

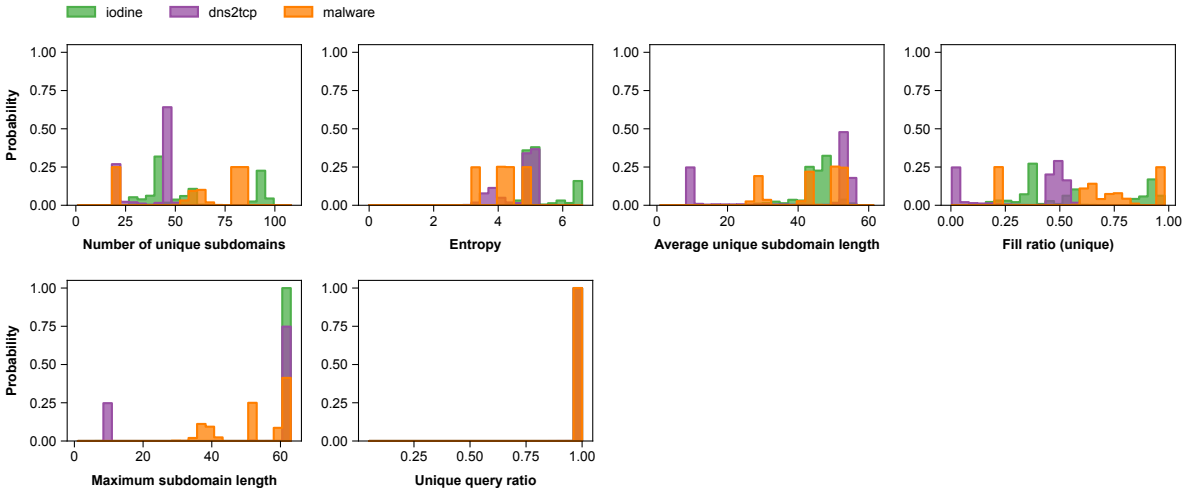


(b) Per malicious group

Figure B.5: Feature distribution, fixed-length window (λ = 10)



(a) Per class



(b) Per malicious group

Figure B.6: Feature distribution, fixed-length window (λ = 20)

C

Optimized Hyperparameters

C.1. (Extended) Isolation Forest

Tables C.1 to C.3 contain the optimized hyperparameter configurations for each Isolation Forest and Extended Isolation Forest experiment.

Table C.1: Final hyperparameters, regular Isolation Forest (Extension Level = 0).

Payload features	Time window	Fixed window	ntrees	sample_size
✗	–	10	30	1,097
✗	–	20	25	8,037
✗	1s	–	25	427,329
✗	1s	10	25	361,472
✗	1s	20	25	351,927
✗	2s	–	25	66,903
✗	2s	10	25	200,436
✗	2s	20	30	382,424
✗	5s	–	25	1,086
✗	5s	10	50	427,329
✗	5s	20	25	391,657
✓	–	–	25	483,091
✓	–	10	25	523,868
✓	–	20	25	474,669
✓	1s	–	30	318,277
✓	1s	10	25	507,876
✓	1s	20	30	411,625
✓	2s	–	25	204,020
✓	2s	10	25	430,030
✓	2s	20	30	461,366
✓	5s	–	25	328,953
✓	5s	10	25	316,279
✓	5s	20	30	281,838

Table C.2: Final hyperparameters, Extended Isolation Forest (Extension Level = 1).

Payload features	Time window	Fixed window	ntrees	sample_size
✗	–	10	25	415,196
✗	–	20	25	372,233
✗	1s	–	25	507,222
✗	1s	10	25	516,682
✗	1s	20	30	240,050
✗	2s	–	25	254,419
✗	2s	10	25	412,481
✗	2s	20	25	271,421
✗	5s	–	40	45,980
✗	5s	10	30	461,366
✗	5s	20	25	183,116
✓	–	–	50	480,132
✓	–	10	25	509,625
✓	–	20	30	460,970
✓	1s	–	25	511,627
✓	1s	10	25	453,433
✓	1s	20	25	405,372
✓	2s	–	25	461,366
✓	2s	10	25	203,238
✓	2s	20	25	435,424
✓	5s	–	25	433,240
✓	5s	10	30	523,790
✓	5s	20	25	503,223

Table C.3: Final hyperparameters, Extended Isolation Forest (Extension Level = 2).

Payload features	Time window	Fixed window	ntrees	sample_size
✗	–	10	25	119,671
✗	–	20	25	1,240
✗	1s	–	25	327,190
✗	1s	10	30	522,649
✗	1s	20	25	196,020
✗	2s	–	25	495,196
✗	2s	10	25	444,436
✗	2s	20	25	380,355
✗	5s	–	25	36,516
✗	5s	10	25	171,134
✗	5s	20	25	254,588
✓	–	–	25	512,697
✓	–	10	30	461,366
✓	–	20	25	288,551
✓	1s	–	25	381,892
✓	1s	10	25	435,424
✓	1s	20	25	306,496
✓	2s	–	25	460,809
✓	2s	10	25	426,848
✓	2s	20	25	523,531
✓	5s	–	25	508,298
✓	5s	10	25	435,424
✓	5s	20	25	468,398

C.2. Random Forest

Table C.4 contains the optimized hyperparameter configurations for each experiment using the Random Forest classifier.

Table C.4: Final hyperparameters, Random Forest.

Payload features	Time window	Fixed window	n_estimators	max_depth	max_samples	min_samples_split	min_samples_leaf	max_features	class_weight
✗	–	10	25	6	514,851	5	5	0.85419	balanced
✗	–	20	30	4	388,782	2	4	0.86989	balanced_subsample
✗	1s	–	25	26	513,722	8	9	0.39394	balanced
✗	1s	10	25	26	485,841	9	4	0.65006	balanced
✗	1s	20	25	20	443,740	7	9	0.48577	balanced
✗	2s	–	25	4	192,137	9	4	0.45191	balanced_subsample
✗	2s	10	25	28	281,135	8	4	0.65804	balanced_subsample
✗	2s	20	40	30	298,768	2	8	0.30877	balanced_subsample
✗	5s	–	30	14	483,644	4	7	0.55994	balanced
✗	5s	10	25	14	431,189	7	3	0.68572	balanced_subsample
✗	5s	20	25	16	246,256	4	7	0.30927	balanced_subsample
✓	–	–	25	8	504,785	7	7	0.66058	balanced
✓	–	10	25	6	345,847	5	8	0.97046	balanced_subsample
✓	–	20	25	6	476,539	3	4	0.56188	balanced_subsample
✓	1s	–	25	20	309,453	9	8	0.18739	balanced
✓	1s	10	25	18	271,319	8	3	0.43102	balanced
✓	1s	20	25	16	410,949	9	4	0.42161	balanced_subsample
✓	2s	–	25	8	129,879	9	2	0.26138	balanced
✓	2s	10	25	30	493,988	6	7	0.32741	balanced_subsample
✓	2s	20	30	24	389,763	5	2	0.48502	balanced_subsample
✓	5s	–	25	12	345,306	2	6	0.27083	balanced_subsample
✓	5s	10	30	8	298,050	9	2	0.49741	balanced
✓	5s	20	25	10	302,631	3	3	0.46623	balanced

D

Experiment Results

D.1. Seen Threat Detection

Tables D.1 to D.4 contain the experiment results for seen threat detection, i.e. the balanced accuracy, macro-recall, false positive query rate and the number of false positive domains on the test set.

Table D.1: Experiment results, iForest

Payload	Time	Fixed	Balanced Acc.	Macro-Recall	FPR (queries)	FP (domains)
✗	–	10	0.92678	0.91649	11.15%	74,186
✗	–	20	0.94341	0.92845	8.66%	69,402
✗	1s	–	0.99237	0.99906	1.39%	3,191
✗	1s	10	**0.99556**	0.99927	**0.82%**	2,866
✗	1s	20	0.99481	0.99948	1.00%	4,222
✗	2s	–	0.98747	0.99938	2.40%	**1,345**
✗	2s	10	0.99361	0.99943	1.21%	2,325
✗	2s	20	0.99488	0.99962	0.99%	3,873
✗	5s	–	0.97661	0.99958	4.60%	4,619
✗	5s	10	0.99539	0.99962	0.86%	3,479
✗	5s	20	0.99397	**0.99970**	1.17%	2,116

Payload	Time	Fixed	Balanced Acc.	Macro-Recall	FPR (queries)	FP (domains)
✓	–	–	0.88983	0.78924	2.05%	20,925
✓	–	10	0.96097	0.89897	1.51%	13,878
✓	–	20	0.97693	0.91984	1.61%	13,102
✓	1s	–	0.99382	0.99899	1.10%	**1,907**
✓	1s	10	0.99468	0.99903	0.98%	2,157
✓	1s	20	**0.99575**	0.99920	**0.77%**	2,328
✓	2s	–	0.99362	0.99922	1.17%	2,726
✓	2s	10	0.99486	0.99938	0.96%	2,921
✓	2s	20	0.99519	0.99945	0.91%	2,721
✓	5s	–	0.99355	0.99949	1.21%	3,672
✓	5s	10	0.99390	0.99952	1.16%	3,160
✓	5s	20	0.99465	**0.99964**	1.03%	2,375

Table D.2: Experiment results, Ext. iForest (level=1)

Payload	Time	Fixed	Balanced Acc.	Macro-Recall	FPR (queries)	FP (domains)
✗	–	10	0.97302	0.90795	1.49%	31,139
✗	–	20	0.97602	0.91604	1.66%	33,976
✗	1s	–	0.99676	0.98702	**0.48%**	495
✗	1s	10	**0.99711**	0.99895	0.48%	883
✗	1s	20	0.99571	0.99943	0.81%	2,682
✗	2s	–	0.99527	0.99919	0.84%	682
✗	2s	10	0.99684	0.99920	0.54%	758
✗	2s	20	0.99610	0.99942	0.73%	368
✗	5s	–	0.99197	0.99901	1.47%	**231**
✗	5s	10	0.99693	**0.99951**	0.56%	1,207
✗	5s	20	0.99578	0.99947	0.78%	2,009

Payload	Time	Fixed	Balanced Acc.	Macro-Recall	FPR (queries)	FP (domains)
✓	–	–	0.89423	0.78725	1.03%	7,291
✓	–	10	0.97632	0.91453	0.86%	2,693
✓	–	20	0.96226	0.90466	0.89%	6,064
✓	1s	–	0.99691	0.99889	0.47%	389
✓	1s	10	0.99645	0.99875	0.60%	993
✓	1s	20	0.99673	0.99904	0.55%	935
✓	2s	–	0.99651	0.99907	0.56%	600
✓	2s	10	0.99652	0.99892	0.56%	**388**
✓	2s	20	0.99654	0.99943	0.64%	1,320
✓	5s	–	0.99654	0.99784	0.51%	414
✓	5s	10	**0.99748**	0.99936	**0.42%**	456
✓	5s	20	0.99733	**0.99956**	0.48%	470

Table D.3: Experiment results, Ext. iForest (level=2)

Payload	Time	Fixed	Balanced Acc.	Macro-Recall	FPR (queries)	FP (domains)
✗	–	10	0.97167	0.90785	1.76%	37,119
✗	–	20	0.95067	0.92650	7.12%	57,024
✗	1s	–	0.99654	0.98857	0.46%	429
✗	1s	10	0.99716	0.99874	0.44%	635
✗	1s	20	0.99645	0.99902	0.63%	752
✗	2s	–	0.99636	0.99882	0.55%	**198**
✗	2s	10	**0.99776**	0.99929	**0.38%**	723
✗	2s	20	0.99730	0.99913	0.45%	748
✗	5s	–	0.99117	0.99951	1.69%	417
✗	5s	10	0.99606	**0.99954**	0.73%	749
✗	5s	20	0.99627	0.99918	0.65%	2,341

Payload	Time	Fixed	Balanced Acc.	Macro-Recall	FPR (queries)	FP (domains)
✓	–	–	0.89496	0.78733	0.90%	6,358
✓	–	10	0.95385	0.89464	0.63%	1,907
✓	–	20	0.95126	0.89627	0.78%	5,188
✓	1s	–	0.99717	0.99870	0.39%	422
✓	1s	10	0.99706	0.99869	0.45%	268
✓	1s	20	0.99691	0.99909	0.53%	522
✓	2s	–	0.99711	0.99877	0.42%	**264**
✓	2s	10	0.99723	0.99899	0.42%	340
✓	2s	20	**0.99753**	0.99901	**0.38%**	332
✓	5s	–	0.99670	0.99853	0.48%	470
✓	5s	10	0.99711	**0.99918**	0.47%	299
✓	5s	20	0.99722	0.99909	0.46%	317

Table D.4: Experiment results, Random Forest

Payload	Time	Fixed	Balanced Acc.	Macro-Recall	FPR (queries)	FP (domains)
✗	–	10	0.99394	0.99785	0.74%	109
✗	–	20	0.99185	**0.99943**	1.60%	24,785
✗	1s	–	0.99946	0.99926	0.03%	452
✗	1s	10	0.99965	0.99854	**<0.001%**	85
✗	1s	20	**0.99984**	0.99939	**<0.001%**	8
✗	2s	–	0.99249	0.99849	1.38%	147
✗	2s	10	0.99699	0.99624	**<0.001%**	16
✗	2s	20	0.99957	0.99918	**<0.001%**	**2**
✗	5s	–	0.99972	0.99903	**<0.001%**	16
✗	5s	10	0.99799	0.99732	0.02%	16
✗	5s	20	0.99962	0.99940	**<0.001%**	3

Payload	Time	Fixed	Balanced Acc.	Macro-Recall	FPR (queries)	FP (domains)
✓	–	–	0.99725	0.99887	0.30%	1,539
✓	–	10	0.99473	0.99743	0.50%	505
✓	–	20	0.99783	0.99936	0.40%	200
✓	1s	–	0.99931	0.99875	0.01%	246
✓	1s	10	0.99984	0.99964	**<0.001%**	77
✓	1s	20	**0.99991**	**0.99972**	**<0.001%**	10
✓	2s	–	0.99676	0.99867	0.52%	75
✓	2s	10	0.99985	0.99971	**<0.001%**	**9**
✓	2s	20	0.99974	0.99927	**<0.001%**	21
✓	5s	–	0.99961	0.99924	0.02%	26
✓	5s	10	0.99752	0.99931	0.42%	27
✓	5s	20	0.99882	0.99927	0.18%	16

D.2. Unseen Threat Detection

Tables D.5 to D.8 contain all experiment results regarding unseen threat detection, i.e. detection rate of every threat in the unseen storage channel set.

Table D.5: Experiment results, unseen threat recall, iForest

Payload	Time	Fixed	dnscat2 SFTP	dnscat2 SSH	dnscat2 Telnet	Custom Plain	BondUp-dater	Carbanak	Cobalt Strike	Win32. Denis	DNSpio-nage	ISMdoor	Pisloader (1)	Pisloader (2)	UDPoS
✗	–	10	0.631	1.000	1.000	**1.000**	**1.000**	0.580	**1.000**	**1.000**	0.769	0.989	0.385	**0.974**	**1.000**
✗	–	20	0.870	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**0.998**	**1.000**	0.077	**1.000**
✗	1s	–	0.034	0.999	0.990	0.010	0.826	0.000	0.000	**1.000**	0.000	0.901	0.000	0.000	**1.000**
✗	1s	10	0.034	1.000	0.992	0.339	0.957	0.000	0.000	**1.000**	0.077	0.893	0.000	0.000	**1.000**
✗	1s	20	0.059	1.000	0.992	0.102	0.913	0.000	0.027	**1.000**	0.231	0.916	0.000	0.000	**1.000**
✗	2s	–	0.063	0.999	0.991	0.094	0.913	0.000	0.954	**1.000**	0.000	0.918	0.000	0.000	**1.000**
✗	2s	10	0.041	1.000	0.992	0.102	0.957	0.000	0.000	**1.000**	0.077	0.817	0.000	0.000	**1.000**
✗	2s	20	0.063	1.000	0.992	0.195	0.913	0.000	0.912	**1.000**	0.000	0.882	0.000	0.000	**1.000**
✗	5s	–	**0.999**	1.000	0.991	0.268	0.870	0.000	0.858	**1.000**	0.000	0.780	0.000	0.000	**1.000**
✗	5s	10	0.134	1.000	0.992	0.058	0.957	0.000	0.592	**1.000**	0.077	0.954	0.000	0.000	**1.000**
✗	5s	20	0.134	1.000	0.992	0.816	0.870	0.000	0.392	**1.000**	0.077	0.978	0.000	0.000	**1.000**

Payload	Time	Fixed	dnscat2 SFTP	dnscat2 SSH	dnscat2 Telnet	Custom Plain	BondUp-dater	Carbanak	Cobalt Strike	Win32. Denis	DNSpio-nage	ISMdoor	Pisloader (1)	Pisloader (2)	UDPoS
✓	–	–	**0.220**	0.999	0.981	**1.000**	**1.000**	0.000	0.000	**1.000**	**0.077**	0.808	0.000	0.000	**1.000**
✓	–	10	0.030	1.000	0.992	0.926	**1.000**	0.000	0.004	**1.000**	**0.077**	**0.952**	0.000	0.000	**1.000**
✓	–	20	0.063	**1.000**	**0.992**	1.000	**1.000**	0.000	0.000	**1.000**	**0.077**	0.844	0.000	0.000	**1.000**
✓	1s	–	0.034	0.999	0.990	0.851	0.957	0.000	0.000	**1.000**	0.000	0.925	0.000	0.000	**1.000**
✓	1s	10	0.033	1.000	0.991	0.606	0.957	0.000	0.000	**1.000**	0.000	0.598	0.000	0.000	**1.000**
✓	1s	20	0.026	1.000	0.992	0.128	0.913	0.000	0.000	**1.000**	0.000	0.870	0.000	0.000	**1.000**
✓	2s	–	0.062	0.999	0.991	0.820	0.870	0.000	0.000	**1.000**	0.000	0.605	0.000	0.000	**1.000**
✓	2s	10	0.063	1.000	0.992	0.563	0.913	0.000	0.000	**1.000**	**0.077**	0.796	0.000	0.000	**1.000**
✓	2s	20	0.043	1.000	0.992	0.759	0.957	0.000	0.000	**1.000**	**0.077**	0.915	0.000	0.000	**1.000**
✓	5s	–	0.134	1.000	0.991	0.741	0.957	0.000	0.177	**1.000**	0.000	0.860	0.000	0.000	**1.000**
✓	5s	10	0.034	1.000	0.992	0.865	0.957	0.000	**0.262**	**1.000**	**0.077**	0.949	0.000	0.000	**1.000**
✓	5s	20	0.116	1.000	0.992	0.703	0.957	0.000	0.000	**1.000**	**0.077**	0.925	0.000	0.000	**1.000**

Table D.6: Experiment results, unseen threat recall, Ext. iForest (level=1)

Payload	Time	Fixed	dnscat2 SFTP	dnscat2 SSH	dnscat2 Telnet	Custom Plain	BondUp-dater	Carbanak	Cobalt Strike	Win32. Denis	DNSpio-nage	ISMdoor	Pisloader (1)	Pisloader (2)	UDPoS
✗	–	10	0.031	1.000	0.992	0.766	**1.000**	0.010	**1.000**	**1.000**	**0.615**	0.637	0.000	**0.013**	**1.000**
✗	–	20	0.059	**1.000**	**0.992**	**1.000**	**1.000**	**0.374**	0.062	**1.000**	0.385	0.730	**0.231**	0.000	**1.000**
✗	1s	–	0.034	0.999	0.990	0.001	0.826	0.000	0.000	**1.000**	0.000	0.683	0.000	0.000	**1.000**
✗	1s	10	0.034	1.000	0.992	0.010	0.826	0.000	0.000	**1.000**	0.000	0.452	0.000	0.000	**1.000**
✗	1s	20	0.041	1.000	0.992	0.346	0.957	0.000	0.000	**1.000**	0.000	0.892	0.000	0.000	**1.000**
✗	2s	–	0.063	0.999	0.991	0.000	0.826	0.000	0.942	**1.000**	0.000	0.883	0.000	0.000	**1.000**
✗	2s	10	0.063	1.000	0.992	0.089	0.826	0.000	0.000	**1.000**	0.000	0.481	0.000	0.000	**1.000**
✗	2s	20	0.063	1.000	0.992	0.055	0.826	0.000	0.073	**1.000**	0.000	0.478	0.000	0.000	**1.000**
✗	5s	–	0.134	0.999	0.991	0.000	0.826	0.000	0.885	**1.000**	0.000	**0.928**	0.000	0.000	**1.000**
✗	5s	10	0.074	1.000	0.992	0.001	0.826	0.000	0.562	**1.000**	0.000	0.731	0.000	0.000	**1.000**
✗	5s	20	**0.134**	1.000	0.992	0.012	0.826	0.000	0.000	**1.000**	0.000	0.683	0.000	0.000	**1.000**

Payload	Time	Fixed	dnscat2 SFTP	dnscat2 SSH	dnscat2 Telnet	Custom Plain	BondUp-dater	Carbanak	Cobalt Strike	Win32. Denis	DNSpio-nage	ISMdoor	Pisloader (1)	Pisloader (2)	UDPoS
✓	–	–	0.006	0.999	0.977	**0.993**	0.957	0.000	0.000	**1.000**	**0.077**	0.390	0.000	0.000	**1.000**
✓	–	10	0.018	1.000	0.992	0.800	0.957	0.000	0.000	**1.000**	**0.077**	0.327	0.000	0.000	**1.000**
✓	–	20	0.056	**1.000**	**0.992**	0.920	**1.000**	0.000	0.000	**1.000**	**0.077**	0.317	0.000	0.000	**1.000**
✓	1s	–	0.031	0.999	0.990	0.524	0.826	0.000	0.000	**1.000**	0.000	0.342	0.000	0.000	**1.000**
✓	1s	10	0.024	1.000	0.991	0.397	0.870	0.000	0.000	**1.000**	0.000	0.219	0.000	0.000	**1.000**
✓	1s	20	0.033	1.000	0.991	0.162	0.870	0.000	0.000	**1.000**	0.000	0.614	0.000	0.000	**1.000**
✓	2s	–	0.049	0.999	0.991	0.436	0.826	0.000	0.000	**1.000**	0.000	0.531	0.000	0.000	**1.000**
✓	2s	10	0.007	1.000	0.991	0.033	0.870	0.000	0.000	**1.000**	0.000	0.183	0.000	0.000	**1.000**
✓	2s	20	0.061	1.000	0.992	0.679	0.957	0.000	0.000	**1.000**	0.000	**0.874**	0.000	0.000	**1.000**
✓	5s	–	0.005	0.999	0.991	0.664	0.826	0.000	0.000	**1.000**	0.000	0.664	0.000	0.000	**1.000**
✓	5s	10	0.009	1.000	0.992	0.308	0.826	0.000	0.000	**1.000**	0.000	0.529	0.000	0.000	**1.000**
✓	5s	20	**0.102**	1.000	0.992	0.391	0.826	0.000	0.000	**1.000**	0.000	0.694	0.000	0.000	**1.000**

Table D.7: Experiment results, unseen threat recall, Ext. iForest (level=2)

Payload	Time	Fixed	dnscat2 SFTP	dnscat2 SSH	dnscat2 Telnet	Custom Plain	BondUp-dater	Carbanak	Cobalt Strike	Win32. Denis	DNSpio-nage	ISMdoor	Pisloader (1)	Pisloader (2)	UDPoS
✗	–	10	0.031	1.000	0.992	0.344	**1.000**	0.001	0.004	**1.000**	0.231	0.613	0.000	0.000	**1.000**
✗	–	20	**0.198**	**1.000**	**0.998**	**1.000**	**1.000**	**0.006**	0.023	**1.000**	**0.462**	**0.988**	**0.692**	**0.013**	**1.000**
✗	1s	–	0.034	0.999	0.990	0.000	0.870	0.000	0.000	**1.000**	0.000	0.560	0.000	0.000	**1.000**
✗	1s	10	0.033	1.000	0.992	0.001	0.826	0.000	0.000	**1.000**	0.000	0.133	0.000	0.000	**1.000**
✗	1s	20	0.034	1.000	0.992	0.000	0.826	0.000	0.000	**1.000**	0.077	0.323	0.000	0.000	**1.000**
✗	2s	–	0.063	0.999	0.991	0.000	0.826	0.000	0.000	**1.000**	0.000	0.444	0.000	0.000	**1.000**
✗	2s	10	0.062	1.000	0.992	0.000	0.826	0.000	0.000	**1.000**	0.000	0.378	0.000	0.000	**1.000**
✗	2s	20	0.061	1.000	0.992	0.017	0.826	0.000	0.000	**1.000**	0.000	0.406	0.000	0.000	**1.000**
✗	5s	–	0.134	0.999	0.991	0.002	0.826	0.000	0.000	**1.000**	0.000	0.791	0.000	0.000	**1.000**
✗	5s	10	0.065	1.000	0.992	0.005	0.826	0.000	**0.388**	**1.000**	0.000	0.664	0.000	0.000	**1.000**
✗	5s	20	0.092	1.000	0.992	0.183	0.826	0.000	0.085	**1.000**	0.000	0.746	0.000	0.000	**1.000**

Payload	Time	Fixed	dnscat2 SFTP	dnscat2 SSH	dnscat2 Telnet	Custom Plain	BondUp-dater	Carbanak	Cobalt Strike	Win32. Denis	DNSpio-nage	ISMdoor	Pisloader (1)	Pisloader (2)	UDPoS
✓	–	–	0.005	0.999	0.977	**0.981**	0.957	0.000	0.000	**1.000**	**0.077**	0.572	0.000	0.000	**1.000**
✓	–	10	0.019	1.000	0.992	0.579	0.957	0.000	0.000	**1.000**	**0.077**	0.217	0.000	0.000	**1.000**
✓	–	20	0.053	**1.000**	**0.992**	0.806	**1.000**	0.000	0.000	**1.000**	**0.077**	0.587	0.000	0.000	**1.000**
✓	1s	–	0.006	0.999	0.990	0.436	0.826	0.000	0.000	**1.000**	0.000	0.177	0.000	0.000	**1.000**
✓	1s	10	0.014	1.000	0.991	0.096	0.826	0.000	0.000	**1.000**	0.000	0.182	0.000	0.000	**1.000**
✓	1s	20	0.016	1.000	0.992	0.431	0.826	0.000	0.000	**1.000**	0.000	0.301	0.000	0.000	**1.000**
✓	2s	–	0.009	0.999	0.991	0.379	0.826	0.000	0.000	**1.000**	0.000	0.379	0.000	0.000	**1.000**
✓	2s	10	0.019	1.000	0.991	0.175	0.826	0.000	0.000	**1.000**	0.000	0.317	0.000	0.000	**1.000**
✓	2s	20	0.026	1.000	0.991	0.008	0.826	0.000	0.000	**1.000**	0.000	0.243	0.000	0.000	**1.000**
✓	5s	–	**0.057**	0.999	0.991	0.245	0.826	0.000	0.000	**1.000**	0.000	**0.661**	0.000	0.000	**1.000**
✓	5s	10	0.023	1.000	0.991	0.072	0.826	0.000	0.000	**1.000**	0.000	0.608	0.000	0.000	**1.000**
✓	5s	20	0.028	1.000	0.992	0.047	0.826	0.000	0.000	**1.000**	0.000	0.473	0.000	0.000	**1.000**

Table D.8: Experiment results, unseen threat recall, Random Forest

Payload	Time	Fixed	dnscat2 SFTP	dnscat2 SSH	dnscat2 Telnet	Custom Plain	BondUp-dater	Carbanak	Cobalt Strike	Win32. Denis	DNSpio-nage	ISMdoor	Pisloader (1)	Pisloader (2)	UDPoS
✗	–	10	0.031	1.000	0.992	0.000	**0.826**	0.000	0.000	0.000	0.000	0.067	0.000	0.000	0.983
✗	–	20	**1.000**	**1.000**	**1.000**	0.258	**0.826**	0.000	0.000	**0.353**	**0.077**	0.053	**1.000**	**0.013**	0.975
✗	1s	–	0.000	0.000	0.044	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**1.000**
✗	1s	10	0.013	0.999	0.990	0.000	0.652	0.000	0.000	0.000	0.000	0.070	0.000	0.000	0.983
✗	1s	20	0.014	0.999	0.990	0.000	0.696	0.000	0.000	0.000	0.000	0.051	0.000	0.000	0.975
✗	2s	–	0.062	0.999	0.991	**0.468**	**0.826**	0.000	0.000	0.000	0.000	**0.270**	0.000	0.000	**1.000**
✗	2s	10	0.006	0.999	0.763	0.000	0.652	0.000	0.000	0.000	0.000	0.061	0.000	0.000	0.983
✗	2s	20	0.012	0.999	0.771	0.000	0.522	0.000	0.000	0.000	0.000	0.049	0.000	0.000	0.975
✗	5s	–	0.000	0.003	0.084	0.000	0.783	0.000	0.000	0.000	0.000	0.009	0.000	0.000	**1.000**
✗	5s	10	0.000	0.999	0.750	0.000	0.783	0.000	0.000	0.000	0.000	0.070	0.000	0.000	0.992
✗	5s	20	0.000	0.980	0.696	0.000	0.565	0.000	0.000	0.000	0.000	0.051	0.000	0.000	0.967

Payload	Time	Fixed	dnscat2 SFTP	dnscat2 SSH	dnscat2 Telnet	Custom Plain	BondUp-dater	Carbanak	Cobalt Strike	Win32. Denis	DNSpio-nage	ISMdoor	Pisloader (1)	Pisloader (2)	UDPoS
✓	–	–	0.001	0.000	0.005	0.000	0.783	0.000	0.000	0.000	**0.077**	0.000	0.000	0.000	0.992
✓	–	10	0.027	0.001	0.023	0.000	**0.826**	0.000	0.000	0.000	0.000	0.070	0.000	0.000	0.983
✓	–	20	0.056	**0.001**	**0.024**	0.192	**0.826**	0.000	0.000	0.000	0.000	0.053	0.000	0.000	0.983
✓	1s	–	0.000	0.000	0.002	0.000	0.043	0.000	0.000	0.000	0.000	0.002	0.000	0.000	0.992
✓	1s	10	0.026	0.000	0.019	0.000	0.696	0.000	0.000	0.000	0.000	0.060	0.000	0.000	0.992
✓	1s	20	0.030	0.001	0.019	0.000	0.783	0.000	0.000	0.000	0.000	0.049	0.000	0.000	0.975
✓	2s	–	0.002	0.000	0.022	**0.232**	**0.826**	0.000	0.000	0.000	0.000	**0.113**	0.000	0.000	**1.000**
✓	2s	10	0.000	0.000	0.007	0.000	0.739	0.000	0.000	0.000	0.000	0.061	0.000	0.000	0.992
✓	2s	20	0.059	0.000	0.008	0.000	0.391	0.000	0.000	0.000	0.000	0.049	0.000	0.000	0.975
✓	5s	–	0.000	0.001	0.014	0.000	0.696	0.000	0.000	0.000	0.000	0.007	0.000	0.000	**1.000**
✓	5s	10	**0.131**	0.001	0.022	0.086	**0.826**	0.000	0.000	0.000	0.000	0.081	0.000	0.000	0.992
✓	5s	20	0.056	0.001	0.020	0.000	**0.826**	0.000	0.000	0.000	0.000	0.051	0.000	0.000	0.975

D.3. Experiment Ranking

Table D.9 (next page) contains an overview of all experiments results, ranked by macro-recall on seen threats.

Table D.9: Performance metrics for all experiments, ranked by macro-averaged recall of groups in the malicious class of the test set. Payload-dropper malware (Carbanak, Cobalt Strike) is excluded from the unseen macro-recall scores.

Algorithm	Payload	Time	Fixed	Macro-Recall (seen)	Macro-Recall (unseen)	Detected (unseen)	FPR (queries)	FP (domains)
Random Forest	✓	1s	20	**0.9997**	0.1689	6	**<0.001%**	10
Random Forest	✓	2s	10	0.9997	0.1635	6	**<0.001%**	9
iForest	✗	5s	20	0.9997	0.6242	10	1.17%	2,116
Random Forest	✓	1s	10	0.9996	0.1630	6	**<0.001%**	77
iForest	✓	5s	20	0.9996	0.6153	9	1.03%	2,375
iForest	✗	2s	20	0.9996	0.5495	9	0.99%	3,873
iForest	✗	5s	10	0.9996	0.5610	10	0.86%	3,479
iForest	✗	5s	–	0.9996	0.6279	9	4.60%	4,619
EIF-1	✓	5s	20	0.9996	0.5459	8	0.48%	470
EIF-2	✗	5s	10	0.9995	0.5047	9	0.73%	749
iForest	✓	5s	10	0.9995	0.6249	10	1.16%	3,160
EIF-1	✗	5s	10	0.9995	0.5112	9	0.56%	1,207
EIF-2	✗	5s	–	0.9995	0.5222	8	1.69%	417
iForest	✓	5s	–	0.9995	0.6074	9	1.21%	3,672
iForest	✗	1s	20	0.9995	0.5648	10	1.00%	4,222
EIF-1	✗	5s	20	0.9995	0.5133	8	0.78%	2,009
iForest	✓	2s	20	0.9995	0.6129	9	0.91%	2,721
iForest	✗	2s	10	0.9994	0.5441	9	1.21%	2,325
EIF-1	✗	1s	20	0.9994	0.5661	8	0.81%	2,682
EIF-1	✓	2s	20	0.9994	0.5965	8	0.64%	1,320
Random Forest	✗	–	20	0.9994	0.5959	11	1.60%	24,785
EIF-1	✗	2s	20	0.9994	0.4921	9	0.73%	368
Random Forest	✗	5s	20	0.9994	0.2963	6	**<0.001%**	3
Random Forest	✗	1s	20	0.9994	0.3387	6	**<0.001%**	8
iForest	✗	2s	–	0.9994	0.5435	9	2.40%	1,345
iForest	✓	2s	10	0.9994	0.5820	9	0.96%	2,921
EIF-1	✓	5s	10	0.9994	0.5149	8	0.42%	456
Random Forest	✓	–	20	0.9994	0.1941	7	0.40%	200
Random Forest	✓	5s	10	0.9993	0.1944	7	0.42%	27
EIF-2	✗	2s	10	0.9993	0.4780	7	0.38%	723
Random Forest	✓	2s	20	0.9993	0.1349	6	**<0.001%**	21
iForest	✗	1s	10	0.9993	0.5719	9	0.82%	2,866
Random Forest	✓	5s	20	0.9993	0.1753	6	0.18%	16
Random Forest	✗	1s	–	0.9993	0.0950	4	0.03%	452
Random Forest	✓	5s	–	0.9992	0.1561	7	0.02%	26
iForest	✓	2s	–	0.9992	0.5769	8	1.17%	2,726
iForest	✓	1s	20	0.9992	0.5389	8	0.77%	2,328
EIF-1	✗	2s	10	0.9992	0.4954	8	0.54%	758
EIF-1	✗	2s	–	0.9992	0.5238	8	0.84%	682
Random Forest	✗	2s	20	0.9992	0.3025	6	**<0.001%**	**2**
EIF-2	✗	5s	20	0.9992	0.5309	9	0.65%	2,341
EIF-2	✓	5s	10	0.9992	0.5018	8	0.47%	299
EIF-2	✗	2s	20	0.9991	0.4820	8	0.45%	748
EIF-2	✓	5s	20	0.9991	0.4877	8	0.46%	317
EIF-2	✓	1s	20	0.9991	0.5060	8	0.53%	522
EIF-1	✓	2s	–	0.9991	0.5302	8	0.56%	600
iForest	✗	1s	–	0.9991	0.5236	8	1.39%	3,191
EIF-1	✓	1s	20	0.9990	0.5155	8	0.55%	935
Random Forest	✗	5s	–	0.9990	0.1708	7	**<0.001%**	16
iForest	✓	1s	10	0.9990	0.5623	8	0.98%	2,157
EIF-2	✗	1s	20	0.9990	0.4774	9	0.63%	752
EIF-2	✓	2s	20	0.9990	0.4631	8	0.38%	332
EIF-1	✗	5s	–	0.9990	0.5344	9	1.47%	231
EIF-2	✓	2s	10	0.9990	0.4843	8	0.42%	340
iForest	✓	1s	–	0.9990	0.6141	8	1.10%	1,907
EIF-1	✗	1s	10	0.9990	0.4830	8	0.48%	883
EIF-1	✓	2s	10	0.9989	0.4621	8	0.56%	388
EIF-1	✓	1s	–	0.9989	0.5193	8	0.47%	389
Random Forest	✓	–	–	0.9989	0.1688	6	0.30%	1,539
EIF-2	✗	2s	–	0.9988	0.4838	8	0.55%	198
EIF-2	✓	2s	–	0.9988	0.5075	8	0.42%	264
EIF-1	✓	1s	10	0.9988	0.5001	8	0.60%	993
Random Forest	✓	1s	–	0.9987	0.0945	6	0.01%	246
EIF-2	✗	1s	10	0.9987	0.4531	8	0.44%	635
EIF-2	✓	1s	–	0.9987	0.4940	8	0.39%	422
EIF-2	✓	1s	10	0.9987	0.4645	8	0.45%	268
Random Forest	✓	2s	–	0.9987	0.1997	7	0.52%	75
Random Forest	✗	1s	10	0.9985	0.3370	6	**<0.001%**	85
EIF-2	✓	5s	–	0.9985	0.5255	8	0.48%	470
Random Forest	✗	2s	–	0.9985	0.4197	7	1.38%	147
Random Forest	✗	–	10	0.9978	0.3544	6	0.74%	109
EIF-1	✓	5s	–	0.9978	0.5591	8	0.51%	414
Random Forest	✓	–	10	0.9974	0.1755	6	0.50%	505
Random Forest	✗	5s	10	0.9973	0.3267	6	0.02%	16
Random Forest	✗	2s	10	0.9962	0.3149	6	**<0.001%**	16
EIF-2	✗	1s	–	0.9886	0.4957	8	0.46%	429
EIF-1	✗	1s	–	0.9870	0.5030	8	0.48%	495
iForest	✗	–	20	0.9285	**0.9041**	**13**	8.66%	69,402
EIF-2	✗	–	20	0.9265	0.7591	**13**	7.12%	57,024
iForest	✓	–	20	0.9198	0.6342	9	1.61%	13,102
iForest	✗	–	10	0.9165	0.8862	**13**	11.15%	74,186
EIF-1	✗	–	20	0.9160	0.6724	12	1.66%	33,976
EIF-1	✓	–	10	0.9145	0.5608	9	0.86%	2,693
EIF-1	✗	–	10	0.9080	0.6412	12	1.49%	31,139
EIF-2	✗	–	10	0.9079	0.5646	11	1.76%	37,119
EIF-1	✓	–	20	0.9047	0.5783	9	0.89%	6,064
iForest	✓	–	10	0.8990	0.6343	10	1.51%	13,878
EIF-2	✓	–	20	0.8963	0.5922	9	0.78%	5,188
EIF-2	✓	–	10	0.8946	0.5309	9	0.63%	1,907
iForest	✓	–	–	0.7892	0.6442	9	2.05%	20,925
EIF-2	✓	–	–	0.7873	0.5970	9	0.90%	6,358
EIF-1	✓	–	–	0.7872	0.5816	9	1.03%	7,291