Computer Networks 195 (2021) 108174

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

DNSxP: Enhancing data exfiltration protection through data plane programmability

Jacob Steadman *, Sandra Scott-Hayward

Centre for Secure Information Technologies (CSIT), Queen's University Belfast, Belfast, BT3 9DT, N. Ireland, United Kingdom of Great Britain and Northern Ireland

ARTICLE INFO

Keywords:
SDN
Data plane programming
P4
eBPF
XDP
DNS
Data exfiltration

ABSTRACT

According to a 2019 Radware report, guarding sensitive data is the highest priority area for investment in cyber security. This is no surprise given the high number of reported data breach incidents annually, and the implication of these on the individuals or organisations targeted. Data exfiltration is a key stage in this form of cyber-attack, and the use of the Domain Name System protocol for data exfiltration is popular due to the essential nature of the protocol for network communication. This paper presents a DNS data exfiltration Protection (DNSxP) security architecture leveraging Software-Defined Networking and Data Plane Programmability. The solution is developed based on analysis of different malicious use cases for transmitting data over the DNS protocol. By performing coarse-grained packet filtering and analysis in the data plane, clear benign or malicious traffic can be identified quickly, while suspicious traffic is passed to additional security controls at the SDN controller for classification. As the results demonstrate, this approach offers the combined benefit of reducing data loss during an exfiltration attack and reducing network resource consumption.

1. Introduction

Obtaining a covert communication channel is an essential step in many cyber security attacks, whether it be for exfiltrating data, tunnelling other protocols, or communicating with malware. Even as the threat landscape changes, this stage in the attack process remains a prominent goal for many exploits, and thus a priority for cyber security defence. As reported by numerous cyber security organisations, guarding sensitive data is a high priority area in cyber security [1–3].

One method for creating *malicious* communication channels, which has remained prevalent over many years, exploits the Domain Name Service (DNS) protocol [4]. Due to the essential nature of the DNS protocol, it has remained an attractive target as networks have failed to prevent it from being exploited. For example, the Unit 42 threat intelligence team at Palo Alto Networks have detailed the activities of the OilRig threat group who have released a number of malicious tools between the period 2016–2019. The malware relies on DNS traffic to communicate between infected hosts and their Command and Control (C&C) servers [5]. Additionally, 2017 saw the rise of fileless malware, which is able to evade many typical anti-virus applications. A notable example of this is DNSmessenger, which used DNS requests to fetch malicious PowerShell scripts [6]. Therefore, if we focus on the detection of the malicious communication channel, then attack tools such as those created by OilRig and DNSmessenger could be detected regardless of the exploits used to gain access.

An issue with detecting covert communication channels that exploit legitimate services is that it requires the analysis of large volumes of benign traffic. Mirroring this traffic to dedicated security middleboxes or applications can have an adverse effect on the required functionality of the network. Furthermore, the communication channel between the security control and the data plane delays the response to malicious activity following detection. We resolve these challenges by introducing detection of DNS data exfiltration attacks at the data plane. The effect is to reduce network monitoring resource consumption and to accelerate the application of mitigation measures to minimise data loss.

In this paper, we present the **DNS** data exfiltration **P**rotection (DNSxP) security architecture utilising Software-Defined Networking (SDN) and Data Plane Programmability (DPP). SDN decouples control from the data plane such that the logically centralised control unit (controller) orchestrates the decision-making process for nodes (network devices) throughout the network, and these nodes effectively become basic packet forwarding devices. DPP allows network operators to directly program how network devices handle network traffic. For example, bespoke forwarding, modification, or inspection can be applied to traffic streams to enable custom behaviour. As a result, DPP can improve network performance and security.

* Corresponding author.
E-mail addresses: jsteadman01@qub.ac.uk (J. Steadman), s.scott-hayward@qub.ac.uk (S. Scott-Hayward).

https://doi.org/10.1016/j.comnet.2021.108174
Received 17 August 2020; Received in revised form 11 May 2021; Accepted 16 May 2021
Available online 24 May 2021

We use the extended Berkeley Packet Filter (eBPF) eXpress Data Path (XDP) [7] and P4 [8] DPP models to achieve a solution that defends against malicious exploitation of DNS traffic in the exfiltration of data. Programming at the data plane has become an attractive option for network security controls due to its ability to move services closer to the wire [9]. We focus on producing a detection-mitigation mechanism that is effective against a range of DNS data exfiltration attack variants. In contrast, existing approaches [10–12] either allow sensitive data to leak even if an attack is successfully detected, or fail to demonstrate the robustness of their solution against a range of threats.

This paper makes the following contributions:

- An analysis of DNS data exfiltration detection features to achieve a detection mechanism that is robust against a range of DNS data exfiltration attack categories.
- A security architecture utilising eBPF XDP to apply threat labelling to traffic in the data plane in order to minimise the impact of security controls on legitimate network functionality.
- A P4 threat-based flow-processing technique to limit unnecessary packet mirroring and controller resource consumption.
- An SDN controller application for DNS monitoring integrated with a dynamic DNS blacklist to block traffic to malicious domains while maintaining legitimate DNS communication.
- An evaluation of the accuracy of the DNSxP detection and mitigation methods, and the impact of the security control on the network performance.

The remainder of this paper is structured as follows. In Section 2, relevant DPP models and related work in detecting DNS data exfiltration attacks are discussed. The feature analysis based on different malicious use cases for transmitting data over the DNS protocol is presented in Section 3. Section 4 provides a description of the DNSxP system architecture. Evaluation results are presented in Section 5. In Section 6, future directions for our work are discussed along with the impact of DNS traffic encryption. Finally, the conclusions are provided in Section 8.

2. Background & related work

2.1. Data plane programming solutions

By moving security controls into the data plane, they can implement actions directly on the traffic as it enters a network interface, which has several positive implications on performance. This removes the reliance on mirroring traffic and waiting for a reply to be sent back to the data plane to implement the security control policies. The only limitation is that the data plane application should not have an adverse effect on legitimate network functionality.

The Extended Berkeley Packet Filter (eBPF) is an upgraded version of the Berkeley Packet Filter (BPF) released in 1992 [7]. eBPF provides low-level packet monitoring and manipulation functionality, which can be attached to network interfaces. Miano et al. [13] provide a summary of the potential and significant aspects of the technology in relation to creating complex network services. eBPF allows for the manipulation of packets as they traverse the network interface. A user-space program can be attached to and run in kernel space via the Connection Tracking (CT), Traffic Control (TC), and eXpress Data Path (XDP) layer hooks. XDP allows access to the packet as it enters a network interface before memory is allocated to it. Scholz et al. [14] provide an analysis of the performance of packet filtering with eBPF XDP and show that XDP is well suited for performing coarse-grained packet filtering, performing up to 4 times faster than filtering at the TC layer by acting on the packet at the earliest instance of it entering the network interface. Chaignon et al. [15] present an XDP based solution for offloading small cloud-based filtering programs into the cloud infrastructure layer. The offload to the Network Interface Card (NIC) CPUs achieves a 4-6x increase in performance compared to performing the same filtering operations within dedicated cloud applications. In [16] Kostopoulos et al. utilise XDP to mitigate DNS water torture Distributed Denial-of-Service (DDoS) attacks. By rapidly analysing domains at the network interface level of authoritative DNS servers, they were able to identify malicious domains without exhausting the server resources. However, to the best of our knowledge, there is no prior work targeting the use of XDP for detecting data exfiltration attacks. In this paper, DNSxP uses XDP to limit the resource consumption at the SDN controller when detecting malicious communication channels, which exploit the DNS protocol.

P4 (Programming Protocol-independent Packet Processors) [8] is a high-level programming language intended to describe the behaviour of the devices that forward, modify, or inspect network traffic. P4 provides the ability to instruct the data plane on how to receive custom protocols, e.g. specifying what to look for at particular offsets within the raw packets.

In [17], Voros and Kiss provide an early example of how P4 can be deployed as a security middleware application. They present a P4 application that can block and ban traffic flows based on defined rules by parsing IPv4, IPv6, TCP, and UDP headers. Their solution blocks users based on packet rate. Ndonda and Sadre [18] take the capabilities of P4 a step further, proposing a two-level Intrusion Detection System (IDS), which implements a flow and Modbus whitelist by leveraging P4, which is updated by a packet inspection application located at the SDN controller.

In our solution, a combination of eBPF XDP and P4 have been applied to the forwarding of traffic within the data plane. The solution, therefore, benefits from the level of control provided by eBPF, the speed of XDP, and the increased flexibility of P4 to control customised protocols. With these DPP models, we can reduce traffic mirroring, achieve greater levels of granularity within our mitigation measures, and remove the communication delay between security control and mitigation measures.

2.2. Detecting DNS data exfiltration

In [4], Bromberger discusses the DNS data exfiltration attack and potential detection features. The research was built upon by Farnham and Atlasis [19] who propose utilising packet and traffic analysis approaches to analyse specific DNS data exfiltration tools that were in use at the time (2013). Jaworski [11] extended this work and presented a method, which utilised the Splunk data analysis engine to collate data on a larger, more robust data set. However, no performance evaluation was provided.

In our earlier work [20], we focused on utilising SDN to detect 3 malicious attacks that exploited DNS traffic; exfiltration, tunnelling, and C&C. Many of the papers discussed in this section focus on the detection of tunnelling, which produces easily identifiable traffic patterns, due to being tightly coupled with the protocol it is attempting to tunnel. However, detecting dedicated exfiltration attacks proves much more difficult as the attackers can obfuscate their malicious packets, and traffic flows to mimic legitimate DNS traffic. Our DNSxD solution [20] was implemented as an OpenDaylight (ODL) [21] controller application, which received all outbound DNS requests. If a flow was deemed to contain malicious domains, a rule was installed at the switch to stop all traffic within that flow, i.e. all DNS traffic from the infected host. Although our solution was successful in detecting each attack category, the method had several limitations; excessive controller load, mitigation delay, and mitigation granularity (i.e. the ability to block specific domains). To address these issues, we have developed DNSxP, as presented in this paper.

Arashloo et al. [22] presented multiple use cases for data plane programmability in the development of SNAP, a stateful network programming model that allows network functions to be programmed into the data plane. They show that their solution is capable of detecting DNS tunnels. However, the work does not explore the suitability of the

Table 1
Features used in DNS data exfiltration Detection (PI: Packet Inspection, TA: Traffic Analysis).

	Farnham [19]	Jaworski [11]	Liu [12]	Arashloo [22]	Infoblox [23]	Ahmed [24]	DNSxD [20]
PI- Statistical analysis		X	X		X	X	X
PI- Query length			X		X	X	X
PI- Record types							X
PI- Tool signatures	X						
PI- Sub-domain count						X	X
PI- Max label length						X	
PI- Avg. label length						X	
PI- Uppercase count						X	
PI- Numerical count						X	
TA- Frequency per domain							X
TA- Orphan DNS requests			X				
TA- Keep-alive				X			
TA- Time interval			X				
TA- Volume per domain		X					X
TA- Domains per IP				X			
TA- IP per domain				X			
TA- Hosts per domain	X						
TA- Unauthorised servers		X					
TA- Blacklist servers		X					X

solution for detecting exfiltration or C&C channels, which are much less predictable.

A number of works have explored the viability of using machine learning (ML) in the detection of DNS tunnels [11,12,25–27]. However, as previously noted, DNS tunnels tend to be more straightforward to detect than other DNS attack categories. Mathas et al. [28] state that their ML approach, which utilised Apache Spot, was unsuccessful in detecting even tunnelling. Nadler et al. [29] have produced a revised edition of their 2017 paper [10], which deploys a ML algorithm that can identify low throughput data exfiltration attacks. However, their approach analyses traffic in 1-hour windows, which means that a successful attack can be executed completely and only detected afterwards without any mitigation taking place.

Ahmed et al. [24] present an ML approach that can detect tunnelling and dedicated data exfiltration attacks in real-time. However, the approach is based on the detection of a single tool, so it is unclear if the results would be replicated with other attacks. The detection method also detects malicious activity on a packet-by-packet basis. While this provides the benefit of being able to block a flow based on a single packet, the reliance on a single packet detection mechanism without considering the entire flow for that domain will make the solution susceptible to evasion through obfuscation techniques, which make the malicious traffic appear benign [30].

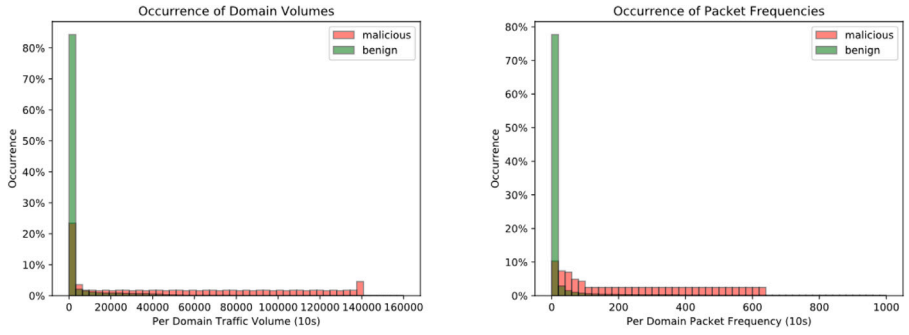
3. DNS data exfiltration techniques and feature analysis

In general, DNS data exfiltration attacks are comprised of a client side application, installed on the victim machine, and a DNS server, controlled by the malicious actor. Once the victim machine is infected with the client side application it will begin transmitting segments of data by sending queries through the DNS protocol. The queries will be redirected through a chain of recursive DNS servers until they reach the server of the top level domain (TLD). Once the requests reach the TLD, the server side application reconstructs the data into its original form. To establish a robust detection mechanism, we categorise DNS data exfiltration attacks by acknowledging variations in the way the attack can be utilised, as follows:

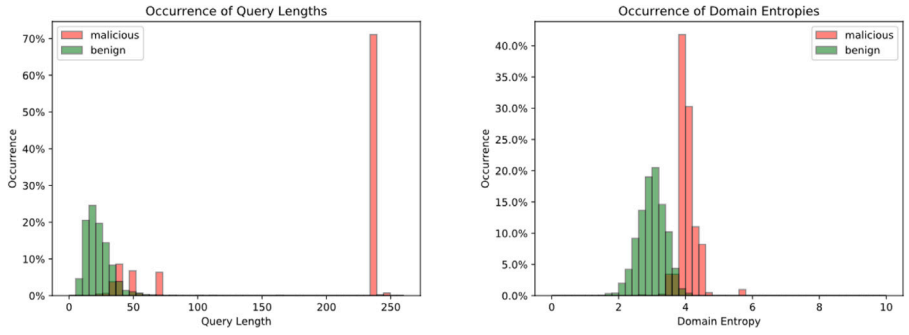
- Tunnelling: A malicious actor can utilise DNS traffic to encapsulate another protocol to bypass network security policies. This form of the attack can be the most straight forward to detect, as the traffic pattern is tightly coupled to the protocol being tunnelled. This generates a high volume and frequency of DNS packets, which can be easily identified from legitimate DNS traffic. The Iodine tool [31] has been selected to represent a tunnelling attack.
- Command & Control: A malicious actor can exploit DNS traffic to create a covert communication channel with their malware on an infected machine. Although the communication channel is not coupled to an existing protocol, the malware must continuously poll the server application to 'request' commands. The polling is necessary as the DNS server cannot initiate communication with clients. To generate an attack similar to potential C&C attacks, we used DNScat [32] which provides the means of issuing commands to the infected host from the DNS server.
- Exfiltration: The most basic form of the attack, pure exfiltration only aims to send data to the malicious DNS server for reconstruction. Unlike tunnelling, the attack does not require a reply and is not linked to another protocol's traffic pattern, and unlike C&C there is no requirement for polling the server to retrieve commands. Two tools have been selected to represent pure exfiltration, DNSExfiltrator [33] and the Data Exfiltration Toolkit (DET) [34]. DNSExfiltrator provides additional functionality to blend the attack traffic with legitimate DNS traffic, e.g. rate and size limiting.

Based on our previous work, DNSxD, discussed in Section 2.2 and presented in Table 1, we analysed each feature by performing packet inspection and traffic analysis on known *benign* and *malicious* data sets. To establish a benign data set comparable to genuine DNS traffic we created a data set that mimicked the statistics for the private DNS traffic datasets collected by DNS OARC [35]. Table 1 shows the final feature set selected for DNSxD, that could successfully identify the 3 categories of attacks by utilising a mix of Packet Inspection (PI) and Traffic Analysis (TA) features. To generate the malicious data set, we use the 4 exfiltration tools previously described (Iodine, DNScat, DNSExfiltrator and DET.) Each exfiltration tool's client application is installed on a VM representing the victim's machine. In addition to the client application, each exfiltration tool has a server-side application, which is installed on a VM in a separate network. The server VM is registered as a DNS server for our malicious domain "dnsexfil.xyz" via a domain hosting platform [36]. DNS queries to "subdomain.dnsexfil.xys" are directed to our malicious DNS server, at which point the server-side application for each tool can reconstruct the data transmitted from the victim VM DNS queries. A single target domain registered to the.xyz top level domain was used throughout the attacks. This allowed the malicious traffic to be exfiltrated from our test environment and received at our malicious server registered as the nameserver for the dnsexfil.xyz domain.

A number of features presented in Table 1 have not been selected for use in DNSxP. **Tool signatures:** rely heavily on the ability to detect known exfiltration tools, and can easily be avoided by making small alterations in the way the tool packages the traffic. **Orphan DNS**



(a) Volume per Domain (b) Frequency per Domain



(c) Query Lengths (d) Query Entropy

Fig. 1. Statistical analysis of features.

requests: rely on the ability to monitor other protocols linked to DNS traffic, e.g. HTTP traffic, which would incur additional performance loss due to the increase in traffic monitoring. **Domains per IP:** the exfiltration tools analysed do not return domain-IP mappings. **Hosts per domain:** as encapsulating the data in each domain name creates a new "host" section, this feature acts the same as monitoring frequency. **Un-authorised servers:** if the attack has been configured to use the network's default DNS servers, this feature will be bypassed automatically. **Blacklist domains:** in [11] blacklisting refers to utilising open-source blacklists to block known malicious domains. This feature can easily be avoided by an attacker altering their TLD. In this paper, we propose using a dynamic blacklist which is generated by the DNSxP application in real-time. However, as the internal DNS server has the ability to blacklist, in addition, it could feasibly utilise open-source blacklists as an extra layer of protection. The most recent related work [24], proposes a method of detection that only utilises PI features. Our classification method does not implement some of these features, as they will not be effective against encrypted DNS traffic. Instead, our solution combines PI and TA features. Our future work will explore protection against data exfiltration over encrypted DNS protocols, as discussed in Section 6.

The remaining features highlighted in Table 1 are analysed by DNSxP and a comparison for *benign* and *malicious* traffic is generated. The distributions for four individual features are displayed in Fig. 1. With Fig. 2, we provide the receiver operating characteristic (ROC) curve to display the relationship between the true positive rate (TPR) and the false positive rate (FPR) for each of these 4 features. Each curve is generated by decrementing the threshold value from the maximum observed value down to zero i.e. Volume 16,000-0, Frequency 700-0, Query Length 255-0, and Entropy 6-0. Taking query length as an example, the top right-hand corner represents a threshold of 0 bytes, which leads to a TPR of 1 (i.e. 100% malicious packets prevented from leaving the network) and FPR of 1 (i.e. 100% of benign packets blocked from leaving the network). As maintaining the availability of the DNS service is a critical requirement, our target is to achieve the

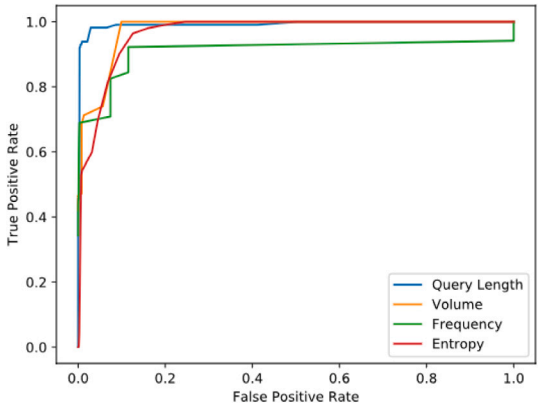


Fig. 2. ROC analysis of features.

highest TPR while maintaining a low FPR. In each case, we set the limit for FPR to 1%. For example, we set the query length threshold value when we reach an FPR of 0.91%, which is achieved at 55 bytes with a TPR of 86%. This process is applied to each of the individual features, and on a per packet/per domain basis, which results in some low TPR values. However, as is shown in Section 5, the classification based on the combination of features together with the effect of the successful detection of a malicious domain blocking all remaining traffic from that domain enables a significant increase in the TPR overcoming the low values observed on individual features and single domain detection.

Returning to the individual feature analysis, we find as follows: **Volume per domain:** total volume of DNS traffic over a 10 s time frame, determined from our earlier work [20]. As shown in Fig. 1a, the upper limit of benign traffic volume is 2000 bytes/10 s (200 Bps), while malicious traffic reaches up to 140,000 bytes/10 s (14,000 Bps). Based on this analysis, a threshold value of 1500 bytes was selected,

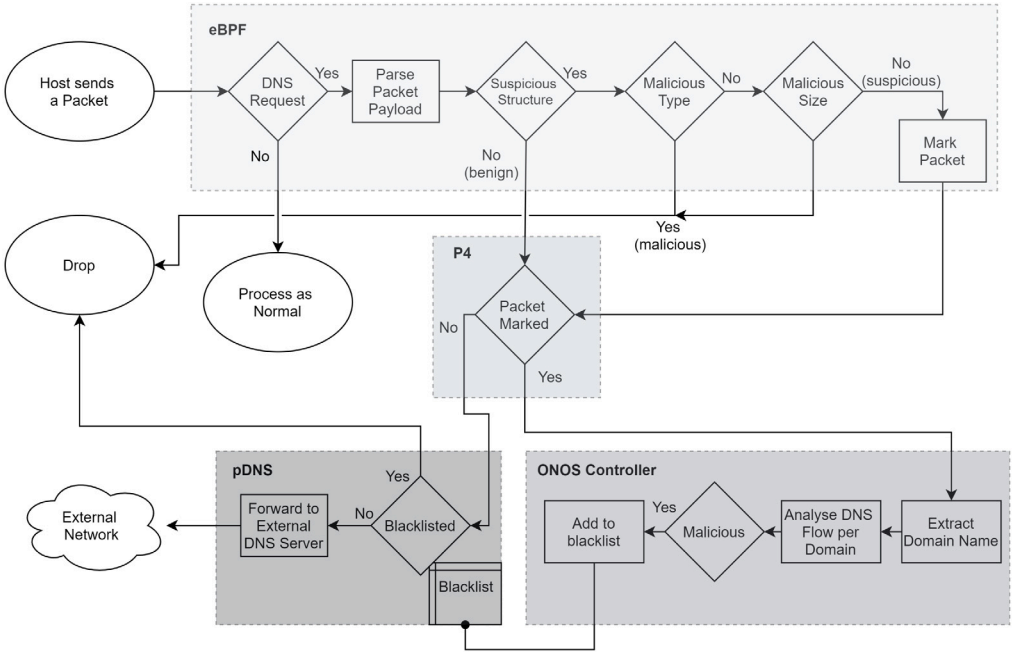


Fig. 3. DNSxP architecture flow model.

based on 0.90% FPR, and 72% TPR. For a threshold of 1400 bytes, the feature produced a 1.35% FPR. **Frequency per domain:** total number of DNS packets per domain over a 10 s time frame. As shown in Fig. 1b, the upper limit of benign traffic frequency is 100 packets/10 s (10 pps), with the majority of values being below 20 packets/10 s (2 pps). In contrast, the frequency of malicious traffic reaches up to 200 packets/10 s (20 pps). Based on this analysis, a threshold value of 100 packets was selected, based on 0.98% FPR, and 68% TPR. With a threshold value of 90, the feature produced a 1.78% FPR. **Query Length:** The DNS protocol reserves a total of 255 bytes for domain name transmission. However, benign traffic rarely utilises the full limit. Fig. 1c shows the contrast between benign and malicious traffic. Based on this analysis, a threshold value of 55 bytes was selected, based on a 0.91% FPR, and 86% TPR. The FPR for query length begins to increase sharply for threshold values less than 55, (e.g. 50 produces an FPR of 3.42%, 40 produces an FPR of 8.39%) **Entropy:** the value of randomness within the domain name string. Data encapsulated in DNS requests generates a higher entropy score than legitimate domains, as shown in Fig. 1d. Based on this analysis, a threshold value of $e > 4$ was selected, based on 0.39% FPR, and 51% TPR. The threshold was selected over an entropy value of 3.9 which produced a 1.42% FPR. **Record Type:** as laid out in DNS traffic statistics by DNS OARC [35]. The most common types of DNS record are A, and AAAA, for the transfer of domain-IP mappings. Various other record types are described in RFC 1035 [37], including TXT for the transfer of additional data back from the DNS server. Ichise et al. [38] present *legitimate* and *malicious* uses for DNS TXT records, often abused for C&C and *legitimately* (ab)used by anti-virus software for updates. As the queries are made to a confined number of domains they can be whitelisted in order to prevent legitimate use being flagged as malicious. Some of these record types, e.g. NULL, have been deprecated for legitimate DNS services and can therefore be blocked by network policies as they are only viewed in malicious traffic. **Subdomain Count:** the number of subdomains can determine if there is space within the DNS request for data to be exfiltrated. For example, queries to TLDs do not have a subdomain field that could carry data. **Keep-alive:** the amount of time recursive DNS resolvers should store replies to DNS requests so that they can respond faster to continuous requests for the same domain.

In addition to the threshold values, there are values below the threshold that would only produce a low number of false positives. By combining these features before flagging a malicious domain, the false positive rate (FPR) can be further reduced. While an attacker might consider using obfuscation to mask a feature, excessive obfuscation of any feature will have the opposite effect on another feature. For example, reducing query length will require an increase in the volume and/or frequency of packets.

As will be discussed in Section 5.2.1, the feature threshold values may vary across network environments. As such, it is anticipated that the thresholds will be configured in DNSxP for a particular network environment by analysing a baseline of the network traffic. To achieve this, DNSxP can be run in analysis mode to collect and store statistics for each feature per-DNS packet from each host in the network. The output of DNSxP in analysis mode is a graphical representation of the traffic statistics, as, for example, in Figs. 1 and 6. The threshold values can then be selected based on the difference in distribution between feature values for the known benign and malicious traffic sets. The process of fine-tuning the threshold values to the specific network security requirements can follow that described with respect to Fig. 2. For example, to set a FPR of <1%.

The threshold values generated from this feature analysis are used in the DNSxP data exfiltration detection process described in the next section.

4. DNSxP architecture

The DNSxP architecture is composed of 4 modules splitting the functionality between the network control and data planes. These modules are illustrated in Fig. 3 and Fig. 4. The data plane elements include the packet classifier, which uses eBPF, and the packet mirror, which leverages P4. At the control plane is an ONOS-based [39] DNS monitoring application, and pDNS [40] DNS server, which instruments blacklisting.

The solution is intended to be installed between the network hosts and an internal DNS server, and assumes the presence of at least one P4 switch with an interface to hook the eBPF program. Ideally, this will be located at the nearest network device to the hosts so that suspicious traffic can be mirrored to the controller application as early as possible. This placement decreases the delay in implementing the blacklisting for malicious domains. In larger networks the cost of deploying a

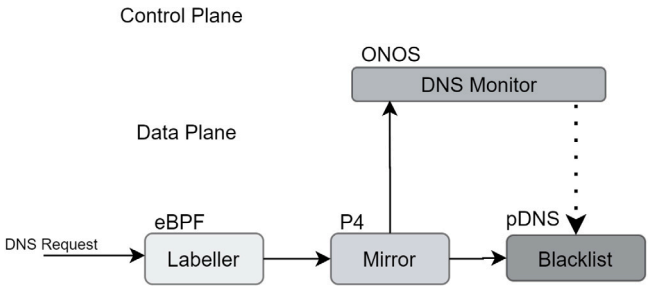


Fig. 4. DNSxP block diagram.

large number of P4 capable switches may become a limiting factor, at which point a trade-off must be made between locating the data plane modules nearest to the hosts at greater expense, or closer to the internal DNS server(s). A second assumption is made that the network restricts all DNS traffic to using port 53 and the internal DNS server. In the open OpenDNS report [41] the authors propose a method to prevent DNS traffic bypassing the internal DNS server. Each of these modules is described in more detail in the following subsections with reference to the flow diagram in Fig. 3.

4.1. eBPF threat labelling

When a DNS request is sent from a host, it is received at the first network device i.e. the P4 switch. Before P4 parses the packet, the packet is passed through an eBPF filter program; the labeller/eBPF packet classifier, as described in Algorithm 1. As eBPF programs are loaded directly into the kernel, a number of strict checks are performed to restrict the application. These restrictions are applied to ensure that the application terminates, is within the maximum number of instructions (1 million) and does not contain unbounded loops that could cause the kernel to lock up. This prevents complex features being applied at this stage of the solution, i.e. volume and frequency per domain over a given period, and entropy. Therefore, using the remaining features discussed in Section 3, each packet is given a classification. The output of the classifier is one of 3 types:

- *Benign*: A DNS request (identified by TCP/UDP port 53) is determined as *benign* if it is querying a TLD. This is because exfiltrated data would replace the sub-domains of a query, which would not be present in a query to a TLD. If the packet is categorised as *benign*, the eBPF program forwards the packet without further analysis. Instead, the packet is passed to the P4 switch pipeline for standard packet processing.
- *Malicious*: There are two light-weight *malicious* policy checks that can be performed inline to determine if the packet is definitely *malicious*; domain length (MDL), and query type (MQT). If the domain length is greater than 55 Bytes or query type is NULL, the packet is dropped without further analysis or forwarding.
- *Suspicious*: if the packet is not confirmed to be *malicious*, but does contain sub-domains which could hide data, a label is applied. To store the label, we use the *Z space* within the 2 byte *Flags* location of the DNS header which is reserved for future use as defined in RFC1035 [37].[1] The packet is then passed to the P4 switch pipeline, which will detect the labelling, and mirror the packet to the controller so that more detailed analysis can take place.

This process is also illustrated in Fig. 3.

[1] Note that if the *Z space* was required at some stage, the eBPF algorithm could be altered to extend the DNS header to add the threat label, and the P4 algorithm could remove the additional information before the packet leaves the network to maintain the necessary DNS structure.

Algorithm 1 eBPF Packet Classifier

Input Network Packet (p), Malicious Length (MDL), Malicious Query Type (MQT)
Output Classified Traffic

```
if p.port = 53 then
  if p.size > MDL OR p.type = MQT then
    p → DROP
  end if
  for byte in p.query do
    if byte =' .' then
      subdomainCount++
    end if
  end for
  if subdomainCount < 2 then
    p → allow
  else
    p → p.suspicious
  end if
end if
```

4.2. P4 flow processing

Based on the eBPF threat labelling step, DNS requests reaching the P4 switch pipeline are identified as either *benign* (unmarked) or *suspicious* (marked), as shown in Algorithm 2.

The P4 algorithm determines if the packet has been marked by checking for DNS requests at the offset 45 bytes into the packet. Unmarked/*benign* requests are processed as they would be without the security controls in place i.e. using reactive flow installation, the first packet is sent to the controller, which installs a flow rule to determine where the switch should route the rest of the packets within that flow. Reactive flow installation has been selected over proactive to ensure the DNS requests are forwarded to the correct destination after being processed by the internal DNS server. Depending on the network configuration, it would be possible to deploy a mixed reactive/proactive processing approach by limiting the use of reactive flow installation to the switches between the internal DNS server and the network gateway. Marked/*suspicious* packets are both processed through the normal pipeline and mirrored to the controller for further analysis. This ensures that they reach their intended destination uninterrupted, until they are determined to be *malicious*.

Algorithm 2 P4 Flow Processing

Input Network Packet (p)
Output Network Packet (p), Mirrored Packet (m)

```
if p.port = 53 then
  if p.suspicious = True then
    p → DNSSERVER
    m → CONTROLLER
  else
    p → DNSSERVER
  end if
else
  p → CONTINUE
end if
```

4.3. ONOS DNS monitor application

Having transferred elements of the DNSxD [20] functionality into the data plane, an optimised version of the controller application could be developed. As shown in Algorithm 3, the ONOS [39] application's

Algorithm 3 ONOS DNS Monitor

Input Mirrored Packet (m), Domain Log (d), Malicious Frequency (MF), Malicious Volume (MV), Malicious Entropy (ME), Benign Frequency (BF), Benign Volume (BV), Benign Size (BS)
Output Blacklist Domain (bl)

```
if m.port = 53 then
  if d.contains(m.domain) then
    d.add(m.domain)
  else
    d.update(m.domain)
  end if
end if
if d.getFrequency(m.domain) > MF then
  m.domain → bl
else if d.getVolume(m.domain) > MV then
  m.domain → bl
else if d.getEntropy(m.domain) > ME then
  m.domain → bl
end if
if d.getFrequency(m.domain) > BF then
  malicious + 1
else if d.getVolume(m.domain) > BV then
  malicious + 1
else if d.getEntropy(m.domain) > BS then
  malicious + 1
end if
if malicious > 1 then
  m.domain → bl
end if
```

main function is to monitor for domain-specific traffic patterns, and to detect the string entropy for each query. The application maintains a log of each domain currently in use within the network during a 10 s window. A mapping of the frequency of packets, and the total volume of data for each domain, for which the thresholds are set as follows: if in a 10 s window, the frequency is greater than 100 packets/domain (MF), the volume is greater than 1500 B (MV), and the entropy score is greater than 4 (ME), the domain is classified as malicious. In addition to these malicious values, there are values that are rare for *benign* traffic that can be combined to reduce false positives. An accumulation of values higher than *benign* values will trigger a malicious detection (BF <10 packets/domain, BV < 300 B, BS < 40). These thresholds can be adjusted using the DNSxP architecture based on analysis of *benign* traffic patterns. If the domain is determined to be malicious, it is written to a blacklist stored at the pDNS server. In addition to the DNS monitoring application, the ONOS controller also implements a P4 handling application to manage the P4 switches.

4.4. pDNS mitigation

Within this architecture, pDNS (Power DNS) [40] is an open-source DNS server, which acts as the endpoint for outbound DNS traffic. Within the network, all DNS traffic is routed to this service, which then sends DNS requests through a recursive chain of external DNS servers until the request reaches the authoritative server for the domain being queried. pDNS has been selected for its support for DNS security measures, including blacklisting malicious domains. Whereas many existing solutions rely on third party static domain blacklists, such as the NCSC's Protective DNS [42], DNSxP provides dynamic blacklisting provided by the ONOS application specified in Section 4.3. Due to the logically centralised nature of the SDN controller, implementing the blacklist in a network with multiple internal DNS servers is a simple matter. This can be achieved by extending the ONOS application's write function to include blacklist files held at each server. Regardless of whether a packet has been labelled as either *benign* or *suspicious*, all packets, that are not dropped by eBPF, are received by the pDNS server [40]. At this point, pDNS will check it's blacklist to determine if the current DNS request should be allowed to pass out of the network, as shown in Algorithm 4.

Algorithm 4 pDNS Blacklist

Input DNS Packet (p)
Output DNS Packet (p)

```
if p.domain = blacklist.domain then
  p.drop
else
  p.continue
end if
```

5. Evaluation

To evaluate DNSxP, the topology depicted in Fig. 5 was created using the Mininet network emulation tool [43]. The network consists of 16 mininet hosts running within an Ubuntu 18.04 VM 16 Gb, 4 CPU, and 5 virtual switches running grp_switch and the P4 BMv2 [44]. Host 16 acts as an internal DNS server, running pDNS, which is the visible endpoint for DNS requests within the network, which are then forwarded through the chain of external DNS servers to be resolved at the authoritative server for that domain. ONOS 2.3.0 [39] was used as the SDN controller due to its support for P4. As a baseline, the solution uses 25%–30% CPU and 4.7% RAM to process benign DNS traffic at 5 Mbps.

5.1. Test traffic

The traffic used to test the architecture is presented in Table 2 and is constructed from malicious DNS, benign DNS, and background network traffic packet captures.

***Malicious* DNS:** The *malicious* DNS traffic sample was generated from 4 DNS data exfiltration attack tools, as described in Section 3; DNScat [32], Iodine [31], DNS Exfiltrator [33], and DET [34]. The traffic was captured from each tool attempting to exfiltrate two separate files. The two different file types were produced to create variation in the attacks:

- File Size: a 50 kB file containing credentials and a 1029 kB PDF file. The datasets produced by each attack are described in Table 2.
- File Compression: Changing the type of data in the file effected how some of the tools compressed the data prior to sending it. This is most noticeable with the DNSExfiltrator rows in Table 2.

The smaller malicious sample was used to generate the threshold values and compare the mitigation efficiency with our previous work [20], and the larger malicious sample was used to validate the performance of the threshold values. This also provided a *malicious* dataset, which is comparable to the size of the *benign* dataset used. Beyond the size of the file being transmitted, there is little discernible difference in the traffic as the data is encoded or encrypted by the tool, which means that the structure of the original data is lost. The number of packets for each attack was also dependent on the volume of data encapsulated in each query as Iodine, DNSExfiltrator, and DNScat make use of the full 255 byte range available. DNSExfiltrator Stealth and DET choose to limit the data to blend with *benign* traffic. In contrast, DNScat, which provides C&C capabilities, continuously polls the server for commands, which generates a high volume of DNS requests that contain no data.

Table 2
Evaluation traffic details.

	File size	Exfiltration tool/Dataset	No. of DNS requests
Malicious sample 1 - credentials file (50 kB)	50 kB	DNScat	639
		Iodine	369
		DNS Exfiltrator	11
		DNS Exfil. Stealth	110
		DET	56
Malicious sample 2 - PDF file (1029 kB)	1029 kB	DNScat	11,467
		Iodine	5994
		DNS Exfiltrator	5071
		DNS Exfil. Stealth	26,163
		DET	30,493
Benign DNS traffic		Top 100k Domains	100,126
		Internet Browsing	7074
		FTTH traffic	44,000,000
Background traffic		iperf clients-servers	Hosts 3, 4, 7–8, 11–14, each running 4 TCP and 4 UDP clients

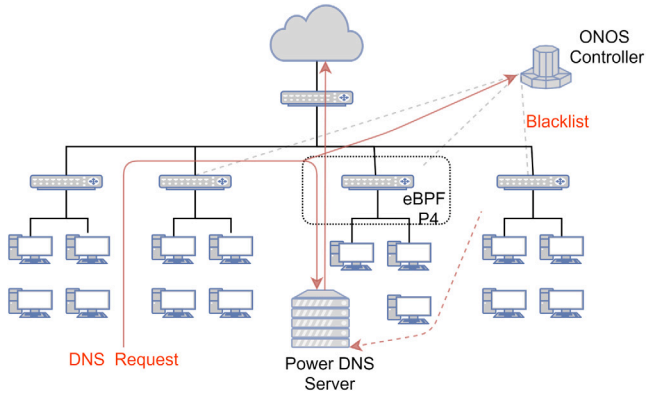


Fig. 5. Test network topology.

***Benign* DNS:** The *benign* DNS traffic sample was collected from two sources: the top 100,000 domains queried in 2018 on the Cisco Umbrella global network [45], and DNS traffic generated from browsing the Internet on the university network. The top 100,000 domains traffic was generated using the *dig* command to query each domain. The final sample contained 100,126 DNS requests due to an additional 126, generated from background processes.

A further dataset containing 44 million DNS requests collected from 8000 fibre-to-the-home (FTTH) customers over 1 day in 2017 [46] was used to validate the DNSxP FPR on a large live DNS traffic sample.

Background Traffic: To test the tool under varying network loads, benign traffic was generated using iperf to automate several client and server processes for a range of ports between each of the VMs. Each iperf client creates 4 TCP and 4 UDP connections to unique server hosts within the test network topology. In addition to this, benign traffic from the UNB dataset [47] has been incorporated to simulate benign network activity. The traffic was captured over a 1-hour period and totalled 10 GB. To map to the experiments, the traffic was divided into 10 1 GB files and replayed simultaneously from 10 hosts across the test topology. The traffic was replayed at 1 Mbps to record the effect of increasing the network load on the DNSxP performance.

5.2. Evaluation results

In the evaluation, we study both the accuracy of the DNSxP detection and mitigation methods, and the impact of the security control on the network performance.

5.2.1. Feature threshold analysis

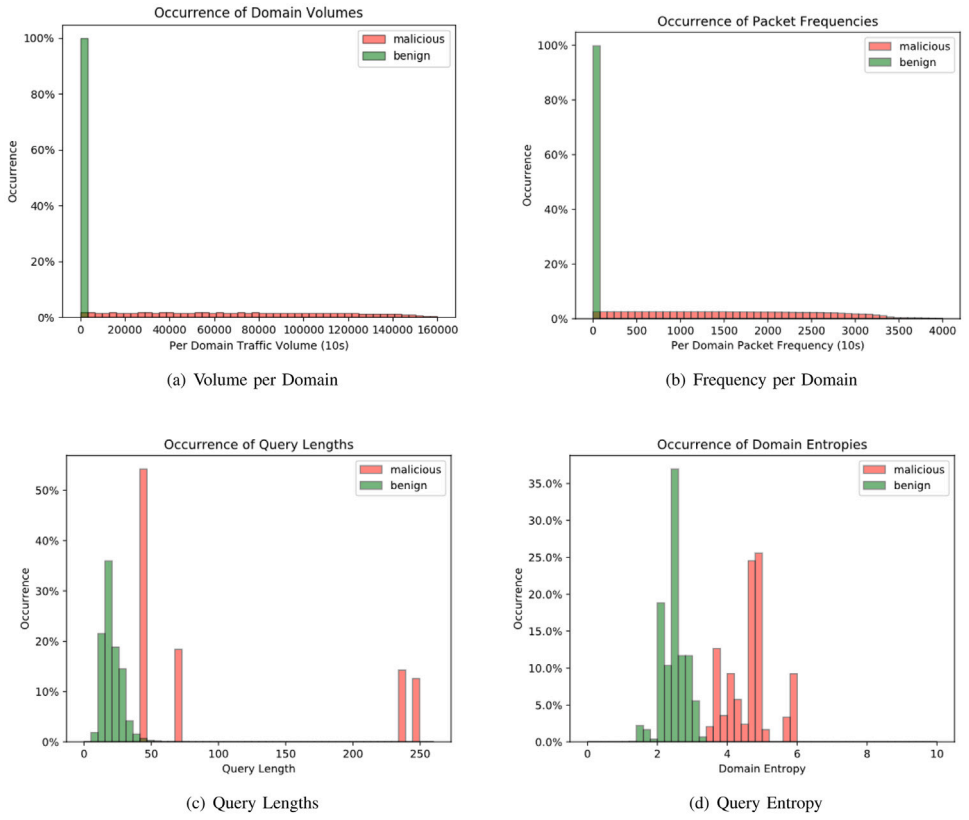
To provide more insight into the detection accuracy against the evaluation data sets presented in Table 2 (i.e. datasets not used in the feature analysis and threshold setting presented in Section 3) we study the distribution of feature values for the benign FTTH dataset and the malicious PDF data exfiltration attacks, as shown in Fig. 6.

The distribution of traffic features followed a similar pattern to the data sets analysed in Section 3. The **volume per domain** of the benign FTTH traffic is within the range of 10 to 2000 bytes/10 s, as observed in the QUB dataset, while the malicious traffic exceeds 150,000 bytes/10 s (15 kb/s). This is higher than observed during exfiltration of the credentials file, which can be attributed to the larger PDF file size transmitted during the attack. The **frequency per domain** of the benign FTTH traffic is also within the range of 0 to 100 packets/10 s, as observed in the QUB data set. In this case, the malicious traffic reaches > 3500 packets/10 s due to the larger volume of data being exfiltrated in the PDF. The **query length** of the benign FTTH traffic ranges from 10 to 60 bytes, similar to our observation on the QUB data set. However, over 50% of the malicious traffic now falls within this benign traffic range. We note that the four prominent query lengths in the malicious dataset are attributed to the payload size used by each attack tool for encapsulating segments of data in the DNS query. Finally, the **domain entropy** of both benign and malicious datasets show a similar pattern to the data sets used for feature analysis and threshold setting, with slightly lower entropy values (e<5) for the benign FTTH traffic compared with e<6 for the QUB dataset.

In summary, this analysis confirms that the data sets analysed to produce the DNSxP feature threshold settings are representative of "unseen" benign/malicious network traffic. We note that it is possible to adjust the thresholds for a particular network environment, which could be done by analysing a baseline of the network traffic and configuring DNSxP with the appropriate thresholds. For example, based on the FTTH Dataset, the query length threshold might be configured to 45 bytes and domain entropy to 3. Based on these settings, we can expect an improved detection performance. However, for the purposes of our evaluation, we maintain the threshold settings defined in Section 3 to produce a conservative set of results without tuning.

5.2.2. Mitigation efficiency

To compare the mitigation efficiency to our earlier work [20], exfiltration of a 50 kB file was attempted with each attack tool (malicious sample 1 in Table 2). In Table 3, the volume of data that was successfully exfiltrated from the sample file for each of the five attacks is presented. As previously noted, the attacks span the different variations of threat including: exfiltration, tunnelling, C&C, and obfuscation techniques. It is clear that the current system outperforms the controller-only solution by detecting and mitigating the full range of attacks with no data lost for 3 of the attacks. Only a small quantity of



(a) Volume per Domain (b) Frequency per Domain

(c) Query Lengths (d) Query Entropy

Fig. 6. Statistical analysis of the FTTH data set features.

Table 3
Comparison of volume of data exfiltrated during each attack.

	DNSxD	DNSxP
Iodine	1.37	0.00
DNScat2	0.46	0.00
DET	0.76	0.09
DNS Exfiltrator	3.65	0.00
DNS Exfil. Stealth	0.32	0.05

Table 4
Detection performance confusion matrix.

		Detected	
		Malicious	Benign
Actual	Malicious	80,238	135
	Benign	1161	98,965

data, 0.05 and 0.09 kB, is exfiltrated during the DNS Exfiltrator stealth and DET attacks, respectively. The data exfiltration can be attributed to the latency of the detection algorithm, as a delay is incurred in the detection between an attack being initiated and a threshold value being reached. In addition, the current architecture only blocks specific domains, allowing *benign* DNS traffic to continue, whereas our earlier work blocked all DNS traffic from the infected host. By integrating the security controls within the data plane devices, the solution allows for fine-grained control of the traffic beyond what was possible with the ODL-OpenFlow implementation in our earlier work; DNSxD. The DNSxD solution mirrored all outbound DNS traffic to the controller application introducing latency to the detection. In contrast, DNSxP implements a multi-level protection against attacks; dropping malicious packets directly from the data plane and blocking specific domains at the internal DNS server. Note that there is a linear relationship between number of hosts and the volume of data/packets exfiltrated. As the detection algorithm works on a per host basis (analysing DNS traffic per domain from each host), the data exfiltrated by multiple hosts is a multiple of the data exfiltrated by a single host. As a result, we present only the results for single host exfiltration. Next, we evaluate the detection performance of the larger data sets.

5.2.3. Detection performance

To test the detection accuracy, we used the test data presented in Table 2 to assess how many packets were incorrectly labelled by our classification method. As shown with the confusion matrix in Table 4, for the 100,126 *benign* domains, the DNSxP architecture incorrectly identified 1161 queries as *malicious* (i.e. False Positives (FP)), with 98,965 True Negatives (TN). For the 80,373 *malicious* domains, 135 were incorrectly identified as benign (i.e. false negatives (FN)), with 80238 True Positive (TP). The system accuracy (percentage of correctly classified DNS packets, both benign and malicious) and precision (ratio between the correctly detected malicious DNS packets and all the detected DNS packets, true and false) are calculated using Eqs. (1) & (2), producing an accuracy rate of 98.52% and a precision rate of 98.57%.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$PREC = \frac{TP}{TP + FP} \tag{2}$$

Table 5 provides a more fine-grained analysis of the detection identifying at which points in the solution architecture the traffic was handled. **Drop** refers to packets that were intentionally dropped in the data plane by the eBPF application without further monitoring. **Allow** refers to packets that were marked as benign by the eBPF application and forwarded to the internal DNS server without further monitoring. **Monitor** refers to packets that were mirrored to the controller application for further analysis. **Blacklist** refers to packets that were monitored and then blacklisted at the internal DNS server.

Table 5
DNSxP detection results.

	Sample	Drop	Allow	Monitor	Blacklist	Total
Malicious sample 1&2	DNScat	99.23%	0%	0.77%	0.62%	12,106
	Iodine	100%	0%	0%	0%	6363
	DNSExfiltrator	100%	0%	0%	0%	5082
	DNSExfil. Stealth	0%	0%	100%	99.98%	26,273
	DET	97.53%	0%	2.46%	2.12%	30,549
Benign	Top 100k Domains	0.91%	27.56%	71.53%	0.25%	100,126
	FTTH DNS	0.47%	16.28%	83.25%	0.84%	44 million

Looking at the DNScat entry in Table 5, 99.23% of the *malicious* traffic was correctly identified and handled within the data plane, the remaining 0.77% of the malicious traffic was mirrored to the controller, and 0.67% was identified as malicious by the ONOS application and blocked by the blacklist. Similar results were achieved with the DET tool. The full volume of malicious DNS traffic generated by Iodine and DNSExfiltrator was correctly identified as malicious by the data plane and dropped from the network without further analysis. Stealth/Throttled variant of the DNSExfiltrator attack was able to evade detection in the data plane. However, the full volume of traffic was successfully mirrored to the controller for further analysis, and 99.98% of the *malicious* traffic was blocked by the blacklist. These results reflect the variation in sophistication of the attacks, specifically highlighting the benefit of the multi-level DNSxP architecture, which can quickly block basic attacks such as Iodine and DNSExfiltrator, and apply further controls to successfully detect stealthier attacks such as represented by DNSExfil. Stealth.

Analysing the two large benign datasets, we observe a similar pattern of detection for each dataset. For example, for the top 100k domains, 0.97% of the benign traffic was incorrectly dropped within the data plane, with 27.56% being correctly classified as benign and allowed to pass through the network without further analysis. The remaining 71.53% was mirrored to the controller, with 0.25% being incorrectly blacklisted and blocked at the DNS server. The relatively low volume of benign traffic (28%/16%) that is allowed at the data plane without requiring further monitoring reflects the trade off between protecting the network from data exfiltration and managing the controller load. This will be further discussed later in this section.

Using the FTTH dataset [46], we demonstrate how the solution would perform if it was positioned at the ISP level, and how it deals with a large volume of live DNS traffic. The dataset was used to confirm the FPR producing a slightly reduced FPR of 1.04% compared to 1.16% for the top 100k domains [45]. There were a number of repeat domains that produced a high volume of FPs. For example, *nist.gov, amazonaws.com, and ntp.org* were responsible for 32.77% of all FPs. Implementing a whitelist within the eBPF module, which will be a focus in our future work, will account for *benign* domains which repeatedly show malicious indicators. This would reduce the volume of FPs significantly, and allow for legitimate (ab)uses of the DNS protocol to continue within the network.

As the rate of false positives is low, commonly used domains that are incorrectly flagged as *malicious* can be manually added to a whitelist within the ONOS application to prevent further blacklisting. Our classification method is slightly more accurate than the strictly PI approach presented in [24] (ACC = 98.23%) and our tests have been produced with a greater degree of variance in the *malicious* attack traffic by using 4 attack variations.

5.2.4. Impact on benign traffic

Due to the separation of operations into 4 distinct modules, the security controls are spread throughout the network architecture, which means that no single point becomes a bottleneck for the DNS requests. This decreases the impact on legitimate traffic by allowing processes to run in parallel. Whereas our previous solution [20] relied on OpenFlow flow rules to mirror all DNS traffic to the SDN controller and then block malicious DNS traffic, this architecture utilises P4 to redirect suspicious traffic, and a blacklist at the pDNS server to block malicious flows. This prevents the security controls from exhausting resources across the network, lowering the impact the security controls have on legitimate network functionality. To test the performance implications of the security control, we replayed the benign DNS traffic sample with and without the security controls installed. The time for each packet to traverse the network was recorded. Note that as malicious DNS traffic is rare and gets blocked once it reaches high traffic rates, the greatest potential for performance impact comes from benign traffic. Fig. 7 shows the comparison between the time it took to make each DNS request with and without the DNSxP security controls in place for increasing network traffic volumes. The traffic volumes for each test are shown in Table 6. Each host creates 8 client connections to servers running within the network. The difference between request time with and without the security controls for the tests with 0 and 2 clients generating background traffic shows no effect on the DNS traffic. This outcome is to be expected as only the eBPF packet labelling phase acts on packets in-line. The use of the XDP hook during the eBPF program is intended to reduce impact on performance degradation and the rest of the monitoring is performed on mirrored traffic at the controller. The results for the tests show that implementing DNSxP while increasing network traffic does affect the throughput of DNS traffic. However, as illustrated in Fig. 7, where variation has been measured (e.g. wmon-1/woMon-1), the difference is only a fraction of a microsecond, 0.02–0.1 µs. These results suggest that the additional DNS security controls have no measurable impact on the benign DNS traffic flows and that any variation measured is caused by the other factors in the network, or test environment, and not the security controls.

Table 6
Composition of background traffic during Fig. 7 tests.

Test run	No. of clients	Packets (pps)	Vol. (kbps)
wMon-0/woMon-0	0	0	0
wMon-1/woMon-1	8	1125	1478
wMon-2/woMon-2	16	2251	2966
wMon-4/woMon-4	32	4155	4928

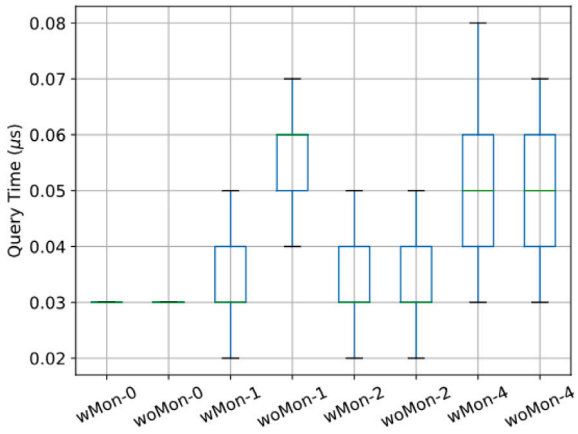


Fig. 7. DNS query time with increasing network load.

Table 7
Impact of increasing traffic rate on network reliability.

	OvS	P4-BMv2	P4-BMv2 DNSxP	P4rt-OvS
Max throughput (zero packet loss)	>10 Gbps	5 Mbps	5 Mbps	50 Mbps
Packet loss	N/A	10% at 6 Mbps 37% at 10 Mbps	10% at 6 Mbps 37% at 6 Mbps	8% at 60 Mbps 25% at 100 Mbps

As can be seen in Table 6, the max. traffic volume tested is <5 Mbps. This is explained with respect to Table 7. In Table 7, we present the results of our tests with different software switches to illustrate the achievable throughput. The OvS, P4-BMv2, and P4-BMv2 DNSxP tests were run on an Ubuntu 18 VM with 16 Gb of memory, 4 CPUs, using Mininet v 2.3.0d6 with 2 hosts connected to a single switch. P4-BMv2 and P4rt-OvS tests were run on an Ubuntu 20 VM with 7 Gb of memory, 1 CPU, using Mininet v 2.2.2 with 2 hosts connected to a single switch. The first test was carried out with the iperf tool to determine the maximum throughput achievable in each network between two hosts. Then, a sample of 100k DNS packets was replayed, using tcpreplay, at a throughput higher than the maximum reported by iperf. For these tests, the packet loss was observed at the destination host, via tcpdump. The results show that increasing the traffic rate beyond 5 Mbps had a measurable impact on network stability, with or without the DNSxP modules installed.

5.2.5. Controller resource usage

The DNSxP architecture reduces the amount of traffic that needs to be mirrored to the controller by classifying traffic as *benign*, *suspicious* or *malicious* within the data plane. The traffic can then be routed based on this labelling. Traffic labelled as *malicious* is dropped by the eBPF program without causing further strain on the network. *Benign* traffic is forwarded as usual without any further monitoring, and only traffic labelled as *suspicious* is mirrored to the controller so that the domain in question can be monitored for traffic patterns that can be identified as *malicious*. In Fig. 8, a comparison of the variation in controller load between DNSxP and our earlier work, DNSxD, is provided. As shown, with this approach, there is a 27% decrease in the number of *benign* DNS requests that needed to be monitored at the controller. However, there is a small increase in the number of *malicious* requests sent to the controller. This is a consequence of the placement of the blacklisting function within the architecture; checking the blacklist is the final process before packets are forwarded externally. To avoid overwhelming the switch flow table with individual flow rules per blacklisted domain, all DNS packets are processed through the system with *suspicious* packets arriving at the controller DNS monitoring application. However, once the controller application has blacklisted a domain, it will keep a local blacklist to remain aware that it no longer needs to monitor the domain and will not waste further resources on it.

5.2.6. DNSxP resource requirements

The individual modules of the DNSxP architecture (eBPF, P4, ONOS, and pDNS) each absorb memory resources on their host device. As noted in Section 4, the eBPF compiler imposes strict limits to ensure that resources are not exhausted within the kernel. These restrictions include the amount of memory that can be used and an upper limit of 1 million instructions. Our solution for classifying DNS traffic requires 12,387 instructions, which occupies only 1.24% of this capacity and enables additional security controls to be added, as required.

The P4 switch has been modified from the standard ONOS-P4 tunnel base application to add functionality to parse DNS packet headers, check for the classification labels, and mirror suspicious packets to the controller application. This additional functionality increases the switch program size from 10.1 kb to 13 kb, well within the limits of a typical P4 switch[2] that will process a wide range of traffic depending on the network.

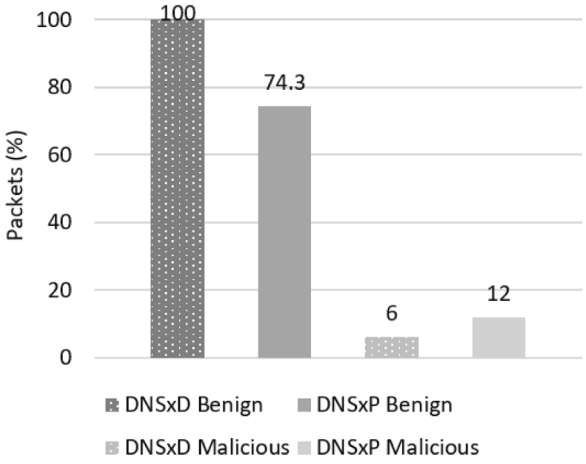


Fig. 8. Comparison of controller load between DNSxP and DNSxD.

To support DNSxP, the ONOS SDN controller implements the DNS monitoring application (124 kb) and a P4 handling application to manage the P4 switches (300 kb). To place this in context, the core ONOS system requires 1.1 Gb, excluding logs.

pDNS is designed to implement blacklists. With DNSxP, at initialisation, the blacklist is empty. The maximum memory required to blacklist a domain is 126 bytes (as the blacklist does not include subdomains). For the malicious traffic blacklisted in the DNSxP evaluation, the total memory required was 891.46 Kb, without implementing any blacklist cleaning policy. This dynamic blacklist is maintained at the internal pDNS server, which by default requires 1 Gb to store DNS logs and cached domains. The DNSxP blacklist from our evaluation occupies <1% of this capacity.

Each component of DNSxP implements a lightweight solution that falls well within the standard capacity of the host device enabling additional network and security services to be provided, as discussed in Section 6.

6. Discussion

While detection methods in related works have been presented in Section 2, they use a limited range of DNS data exfiltration attack variants and lack in their evaluation of how implementing mitigation measures would effect network performance and DNS availability. The evaluation in Section 5 demonstrates that DNSxP is accurate in detecting DNS-based data exfiltration attacks and efficient in mitigating the attacks both through preventing data exfiltration and maintaining the availability of the DNS service on the host from which the malicious traffic is originating. Furthermore, by utilising DPP to integrate the security controls into the data plane, the solution reduces the load on a single resource, e.g. the controller or middle-box applications. We

[2] The Barefoot Tofino P4 compatible switch has models up to 128 GB SSD - https://www.stordis.com/wp-content/uploads/2019/12/STORDIS_BF2556X-1T-A1F.pdf.

demonstrated that the security controls also have no measurable impact on legitimate network functionality under increasing network load. This is supported by the set of simple but effective algorithms providing a tiered approach to identifying malicious traffic that is scalable and resource efficient.

For this reason, the solution is designed to be extensible. For example, we can anticipate that future iterations of exfiltration attacks may become stealthier as attackers adapt to improved detection mechanisms. To provide deeper insight into this, we present an extensibility case study.

6.1. DNSxP extensibility case study

There is an ongoing discussion within the community regarding the adoption of DNS over Encryption (DoE), i.e. DNS over TLS (DoT) [48] and DNS over HTTPS (DoH) [49], driven by a demand for greater privacy. Both proposals aim to encrypt DNS traffic between the user and the target authoritative domain server. However, DoH blends the DNS traffic with all other HTTPS (port 443) traffic. In contrast, DoT allows DNS traffic to remain on its own dedicated port (port 853). This means that DoT provides more opportunity for network administrators to use existing port filtering-based security controls to monitor the DNS traffic for anomalies. So far, DoH has presented itself as the more prominent service and has been implemented by a number of service providers, with Google and Cloudflare receiving the majority of DoE requests [50], and six major browsers [51] offering the option for users to opt in to using the service. The service is not used by default as it is slower and, as discussed, presents security concerns due to the reduction in monitoring. There is a clear trade-off between the demand for increased privacy over DNS to tackle issues such as internet censorship and surveillance, and the necessity for security specialists to access network traffic for monitoring and analysis. Setting aside the requirement to address these conflicting requirements, we anticipate that DNSxP should be extended to handle DoE traffic.

The ability to efficiently update DNSxP is supported by the modularity of the architecture and the inherent benefits of an SDN-based design. We describe here the relevant updates and update times for each individual module to monitor DoE traffic.

Starting with the eBPF module, in the case of DNS over HTTPS (DoH), encrypted DNS traffic is carried in HTTPS traffic. This requires an extension to the eBPF module to classify HTTPS packets as benign, malicious, or suspicious based on the features we identify from our statistical analysis of the DoH traffic. Updating the eBPF program requires reinstallation of the switch module that intercepts packets, which takes 2.54 s in our system. During this period, the DNS classifier would be inactive. However, other network functionality would be maintained.

The P4 program should also be updated to check the classification label of the DoH traffic. In contrast to the eBPF module, updating the P4 module incurs network down time with reinstallation of the P4 switch with the new program taking 39.21 s in our system. This length of downtime is, of course, a serious limitation but it is not unique to our system. The ability to dynamically update P4 switches is a separate programme of research beyond the scope of our work.

Finally, the SDN controller application can either be updated or a new application added to monitor the DoH traffic classified as suspicious. Integrating the DoH monitoring with the DNS monitoring requires the application to be reinstalled, which incurs a controller downtime of 12.9 s in our system. During this period, traffic mirrored to the controller would not be analysed by the DNS monitoring application. However, a separate application can be added with no downtime incurred. A new application is installed in 0.36 s in our system, after which the mirrored traffic is analysed, as required. No changes are required to the pDNS module.

The encrypted DNS data exfiltration detection example requires an update to three of the four DNSxP modules. However, we can anticipate changing attacks or new detection algorithms requiring an update to only one module. For example, the eBPF classifier could be extended to introduce a new packet feature. Similarly, a new SDN controller application could be added to support additional monitoring functionality. The network/monitoring downtime varies from approx. 3 s to 55 s requiring update scheduling by the network operator.

7. Limitations

We identify a number of limitations with the proposed solution. While the solution is effective against the attack tools demonstrated in this work, it is still possible for an attacker with knowledge of the system design to evade detection. For example, an attacker could use knowledge of the DNS policies (features and thresholds used for classification) to create a specifically crafted attack to circumvent the security controls. However, the separation of security controls across the data plane, controller, internal services (e.g. pDNS server), and local firewall makes such an attack difficult. For many attacks, for example botnets that use DNS as a C&C channel and target a large number of hosts across multiple targets, it would be very resource intensive for an attacker to craft specifically configured attacks for each target.

We also note that our evaluation is constrained by the capacity of the P4 software switch, BMv2. While the reference OpenFlow software switch implementation, Open vSwitch, can achieve Gbps throughput in a Mininet network, BMv2 has a significantly lower limit at less than 5 Mbps, as discussed in Section 5.2.4. As a result, although there are example implementations of security solutions achieving Gbps on a P4 hardware switch (e.g. [52]), we have not been able to validate a similar performance for DNSxP in our virtual testbed. Future work should include this performance validation in a physical testbed (i.e. with a P4 hardware switch).

8. Conclusion

The threat of data exfiltration via network protocols such as DNS continues to challenge business. In this work, we have presented DNSxP, an architecture for detecting and mitigating malicious data exfiltration attacks that exploit the DNS protocol. The architecture makes use of the software-defined network control plane and data plane programmability to distribute security functionality across the network to make best use of the network resources. We have demonstrated the benefit of this model to provide effective protection against data exfiltration attacks (tunnelling, C&C, and exfiltration) while minimising the impact on the network performance for legitimate users.

CRediT authorship contribution statement

Jacob Steadman: Data curation, Writing, Visualization, Methodology, Validation, Formal analysis. **Sandra Scott-Hayward:** Supervision, Reviewing, Editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The research was funded by the UK Department for Education.

References

[1] C. Herberger, et al., Radware global application & network security report 2018-2019, Tech. Rep., 2019, [Online]. Available: https://www.radware.com/ert-report-2018/.

[2] T.C. Group, Crypsis releases 2020 Incident Response and Data Breach Report, [Online]. Available: https://www.securityinfowatch.com/cybersecurity/press-release/21141149/the-crypsis-group-crypsis-releases-2020-incident-response-and-data-breach-report.

[3] C. Point, Cyber SEcurity Report, [Online]. Available: https://www.ntsc.org/assets/pdfs/cyber-security-report-2020.pdf.

[4] S. Bromberger, DNS as a Covert Channel Within Protected Networks, National Electronic Sector Cyber Security Organization (NESCO), 2011, (Jan., 2011).

[5] R. Falcone, DNS Tunneling in the Wild: Overview of oilrig's dns Tunneling, [Online]. Available: https://unit42.paloaltonetworks.com/dns-tunneling-in-the-wild-overview-of-oilrigs-dns-tunneling/.

[6] C. Talos, Spoofed sec Emails Distribute Evolved DNSMessenger, [Online]. Available: https://blog.talosintelligence.com/2017/10/dnsmessenger-sec-campaign.html.

[7] M. Flemming, A thorough Introduction to ebpf, [Online]. Available: https://lwn.net/Articles/740157/.

[8] P. Bosshart, et al., P4: Programming protocol-independent packet processors, ACM SIGCOMM Comput. Commun. Rev. 44 (3) (2014) 87–95.

[9] M. Bertrone, S. Miano, F. Risso, M. Tumolo, Accelerating linux security with eBPF iptables, in: Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos, ACM, 2018, pp. 108–110.

[10] A. Nadler, A. Aminov, A. Shabtai, Detection of malicious and low throughput data exfiltration over the DNS protocol, 2017, arXiv preprint arXiv:1709.08395.

[11] S. Jaworski, Using Splunk to Detect DNS Tunneling, SANS Institute InfoSec Reading Room, 2016.

[12] J. Liu, S. Li, Y. Zhang, J. Xiao, P. Chang, C. Peng, Detecting DNS tunnel through binary-classification based on behavior features, in: Trustcom/BigDataSE/ICESS, 2017 IEEE, IEEE, 2017, pp. 339–346.

[13] S. Miano, M. Bertrone, F. Risso, M. Tumolo, M.V. Bernal, Creating complex network service with eBPF: Experience and lessons learned, in: High Performance Switching and Routing (HPSR), IEEE, 2018.

[14] D. Scholz, D. Raumer, P. Emmerich, A. Kurtz, K. Lesiak, G. Carle, Performance implications of packet filtering with linux ebpf, in: 2018 30th International Teletraffic Congress (ITC 30), 1, IEEE, 2018, pp. 209–217.

[15] P. Chaignon, D. Adjavon, K. Lazri, J. François, O. Festor, Offloading security services to the cloud infrastructure, in: Proceedings of the 2018 Workshop on Security in Softwarized Networks: Prospects and Challenges, ACM, 2018, pp. 27–32.

[16] V.M. Nikos Kostopoulos, Leveraging on the xdp Framework for the Efficient Mitigation of Water Torture Attacks within Authoritative dns Servers, in: Proceedings of the 6th IEEE International Conference on Network Softwarization, 2020.

[17] P. Vörös, A. Kiss, Security Middleware Programming Using P4, Vol. 9750, 2016, pp. 277–287.

[18] G.K. Ndonda, R. Sadre, A two-level intrusion detection system for industrial control system networks using p4, in: 5th International Symposium for ICS & SCADA Cyber Security Research 2018 5, 2018, pp. 31–40.

[19] G. Farnham, Detecting DNS Tunneling, SANS Institute InfoSec Reading Room, 2013, https://www.sans.org/reading-room/whitepapers/dns/detecting-dns-tunneling-34152.

[20] J. Steadman, S. Scott-Hayward, Dnsxd: Detecting data exfiltration over dns, in: Proceedings of the IEEE Conference on Network Functions Virtualization and Software-Defined Networking, Institute of Electrical and Electronics Engineers (IEEE), 2018.

[21] OpenDaylight, OpenDaylight Nitrogen SDN Controller, [Online]. Available: https://www.opendaylight.org/what-we-do/current-release/carbon.

[22] M.T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, D. Walker, Snap: Stateful network-wide abstractions for packet processing, in: Proceedings of the 2016 ACM SIGCOMM Conference, ACM, 2016, pp. 29–43.

[23] Infoblox, Infoblox, [Online]. Available: https://www.infoblox.com/.

[24] J. Ahmed, H.H. Gharakheili, Q. Raza, C. Russell, V. Sivaraman, Real-time detection of dns exfiltration and tunneling from enterprise networks, in: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), IEEE, 2019, pp. 649–653.

[25] Y. Bubnov, Dns tunneling detection using feedforward neural network, Eur. J. Eng. Res. Sci. 3 (11) (2018) 16–19.

[26] C.-M. Lai, B.-C. Huang, S.-Y. Huang, C.-H. Mao, H.-M. Lee, Detection of dns tunneling by feature-free mechanism, in: 2018 IEEE Conference on Dependable and Secure Computing (DSC), IEEE, 2018, pp. 1–2.

[27] P. Walsh, Comparing the Effectiveness of Different Classification Techniques in Predicting DNS Tunnels, Dublin Institute of Technology, 2018.

[28] C.M. Mathas, O.E. Segou, G. Xylouris, D. Christinakis, M.-A. Kourtis, C. Vassilakis, A. Kourtis, Evaluation of apache spot's machine learning capabilities in an sdn/nfv enabled environment, in: Proceedings of the 13th International Conference on Availability, Reliability and Security, ACM, 2018, p. 52.

[29] A. Nadler, A. Aminov, A. Shabtai, Detection of malicious and low throughput data exfiltration over the dns protocol, Comput. Secur. 80 (2019) 36–53.

[30] S. Sheridan, A. Keane, Improving the stealthiness of dns-based covert communication, in: ECCWS 2017 16th European Conference on Cyber Warfare and Security, Academic Conferences and publishing limited, 2017, p. 433.

[31] Yarrick, Iodine, [Online]. Available: https://github.com/yarrick/iodine.

[32] iagox86, dnscat2, [Online]. Available: https://github.com/iagox86/dnscat2.

[33] Arno0x, DNSExfiltrator, [Online]. Available: https://github.com/Arno0x/DNSExfiltrator.

[34] Sensepost, DET, [Online]. Available: https://github.com/sensepost/DET.

[35] DNS Operation Analysis and Research Center, https://www.dns-oarc.net/.

[36] Buy a domain and everything else you need, Namecheap, [Online]. Available: https://www.namecheap.com/.

[37] P. Mockapetris, Domain Names - Implementation and Specification, Network Working Group RFC 1035, 1987, [Online]. Available: https://www.ietf.org/rfc/rfc1035.txt.

[38] H. Ichise, Y. Jin, K. Iida, Analysis of DNS txt record usage and consideration of botnet communication detection, IEICE Trans. Commun. 101 (1) (2018) 70–79.

[39] Open Networking lab, ONOS, [Online]. Available: https://github.com/opennetworkinglab/onos.

[40] Power DNS, Opensource DNS Software, [Online]. Available: https://www.powerdns.com/.

[41] Thien Huynh, How to prevent users from circumventing OpenDNS using firewall rules, [Online]. Available: https://support.opendns.com/hc/en-us/articles/227988027-How-to-prevent-users-from-circumventing-OpenDNS-using-firewall-rules.

[42] NCSC, Protective DNS, [Online]. Available: https://www.ncsc.gov.uk/information/pdns.

[43] Mininet, Mininet, [Online]. Available: http://mininet.org/.

[44] p4lang, Behavioural Model, [Online]. Available: https://github.com/p4lang/behavioral-model.

[45] Cisco, Cisco Umbrella Top 1m Domains, [Online]. Available: http://s3-us-west-1.amazonaws.com/umbrella-static/index.html.

[46] M. Trevisan, D. Giordano, I. Drago, M.M. Munafò, M. Mellia, Five years at the edge: Watching internet from the isp network, IEEE/ACM Trans. Netw. 28 (2) (2020) 561–574.

[47] I. Sharafaldin, A.H. Lashkari, A.A. Ghorbani, Toward generating a new intrusion detection dataset and intrusion traffic characterization., in: ICISSP, 2018, pp. 108–116.

[48] Z. Hu, et al., Specification for DNS over Transport Layer Security (TLS), RFC 7858, 2016, [Online]. Available: https://tools.ietf.org/html/rfc7858.html.

[49] P.M. P. Hoffman, DNS Queries over HTTPS (DoH), RFC 8484, 2018, [Online]. Available: https://tools.ietf.org/html/rfc8484.

[50] C. Lu, B. Liu, Z. Li, S. Hao, H. Duan, M. Zhang, C. Leng, Y. Liu, Z. Zhang, J. Wu, An end-to-end, large-scale measurement of DNS-over-encryption: How far have we come?, in: Proceedings of the Internet Measurement Conference, 2019, pp. 22–35.

[51] Catalin Cimpanu, How to enable DoH in browser, [Online]. Available: https://www.zdnet.com/article/dns-over-https-will-eventually-roll-out-in-all-major-browsers-despite-isp-opposition/.

[52] D. Scholz, S. Gallenmüller, H. Stubbe, G. Carle, SYN flood defense in programmable data planes, in: Proceedings of the 3rd P4 Workshop in Europe, 2020, pp. 13–20.

Jacob Steadman is a Computer Science B.Sc. and Cyber Security M.Sc. graduate currently enrolled in a Software-Defined Network security Ph.D. at the Centre for Secure Information Technologies, Queen's University Belfast.