UNIVERSITY OF AMSTERDAM

OS3/SNE – RESEARCH PROJECT 1

Mitigating layer 7-based DDoS attacks in the SURF network

February 14, 2024

Students:
Ferran Tufan
ferran.tufan@os3.nl

Laurens Wijnsma
laurens.wijnsma@os3.nl

Supervisor:
Wim Biemolt

Course:
Research Project 1

Abstract

Layer 7-based DDoS attacks are a more and more prevalent way of executing DDoS attacks. A DNS water torture attack is one of the various categories layer 7-based DDoS attacks, and this attack type is on the rise. This paper aims to provide insights into incorporating DNS zone information in a mitigation strategy against DNS water torture attacks. Experiments were conducted by building an eBPF program, leveraging XDP hooks to inspect DNS queries, in order to match their query names against Bloom filter-type eBPF maps containing DNS zone information. Depending on lookup results, queries will be either let through or blocked. Using legitimate queries and DNS water torture-infected queries, the effectiveness of this program has been tested. Depending on the number of hash functions used in the Bloom filter, the program is able to filter all infected queries, while letting all legitimate queries through. Due to inconsistencies in the results, the overhead of this program cannot be determined. We conclude the use of DNS zone information to filter DNS queries can be effective to protect authoritative name servers against DNS water torture attacks.

Keywords: DNS water torture attacks, layer 7-based attacks, virtual network functions (VNFs), network function virtualisation (NFV), extended Berkeley packet filter (eBPF), eXpress data path (XDP)

Contents

List of Tables

List of Figures

1 Introduction

SURF is the non-profit organisation operating the national research and education network national research and education network (NREN) for educational and research institutions in the Netherlands. Over the SURF network (formerly known as SURFnet), SURF offers various connectivity services for all customers, including IP transit, L2VPN/L3VPN, and high-speed connections to other NRENs. Both large and small institutions are connected to the SURF network. For any institution present on the internet via SURF IP transit, distributed denial of service (DDoS) attacks are part of their threat landscape. An increase has been seen in layer 7-based DDoS attacks (L7-based DDoS attacks). All institutions are prone to L7-based DDoS attacks, but small institutions are especially susceptible; unlike their larger peers, those institutions lack the staff, budget, and protective measures to deflect those attacks.

In the past, border gateway protocol border gateway protocol (BGP) offramping was used in the SURF network to steer traffic away from the regular forwarding path and divert it to a scrubbing appliance (also known as 'washing appliance'). Scrubbed network traffic would be put back on the SURF network for delivery to the institution. Due to cost limitations, this appliance has been retired. In the current generation of the SURF network, two (semi-)automated DDoS mitigation solutions have been kept around. The first solution is specific to the Juniper MX platform and is called Corero SmartWall. Corero performs deep packet inspection (DPI) and is able to apply access control filters automatically or upon request manually on routers. The second solution is a Rate Limiter, applied to the uplinks between institution and SURF network. Each protocol (such as domain name system (DNS)) is an active traffic class and each traffic class is assigned a fixed bandwidth. Bandwidth limits will be enforced unconditionally. Through the SURF Network Dashboard[1], institutions can choose whether they want DDoS mitigations to be enabled or not. In emergencies and/or upon request by the institution, SURFcert[2] can also apply these mitigations.

However, neither solution is suitable to mitigate a L7-based DDoS attack that does not induce excessive bandwidth usage on itself. An example is the DNS water torture attack[1]. In such an attack, an adversary attacks an institution's name servers by sending dozens of DNS queries with the labels `x.y`, where x is a random string (unique for each query) and y the zone of an institution (such as `os3.nl`). Because each label is unique, queries cannot be answered by the recursive resolvers' caches, and therefore all queries will be sent to the institution's name servers. name servers may be overloaded, even if the uplink to the SURF network is nowhere near saturation.

SURF would like to see a proof of concept (PoC) on utilising virtual network functions (VNFs) on SURF's network function virtualisation (NFV) infrastructure to scrub traffic in the SURF network contaminated with DNS water torture attacks based on ready-for-use information[3], after which the scrubbed traffic can be delivered to the institution.

2 Research Questions

The research question has been defined as follows:

`How could SURF mitigate DNS water torture attacks within the current generation SURF network to protect its customers?`

To answer the research question, a number of sub-questions have been defined:

- How can DNS zone information be incorporated in a multi-tier architecture for mitigating DNS water torture attacks?

[1]A self-service interface where institutions can manage their SURF network resources.
[2]SURFcert is the incident response team for their constituency (SURF and connected institutions).
[3]DNS zone information has been chosen as an example for DNS water torture attacks.

- How much overhead does traffic washing using VNFs introduce?

3 Background

3.1 Network scrubbing in the SURF network

SURF used to have a washing appliance based on Netscout Armor software, named the *wasmachine* (English: *washing machine*) [4]. This appliance performed in-line DPI to detect all sorts of DDoS attacks, including L7-based ones, specifically DNS water torture attacks. SURFcert was able to route prefixes up to a size of /27 (IPv4) through this appliance upon request from institution or due to an ongoing attack. While SURFcert was satisfied with the appliance, the solution had to be retired due to spike in cost after the migration to SURFnet8[4]. Furthermore, due to the appliance's position, directly in the forwarding path, and because the appliance receives all institution traffic (all-or-nothing), the link between the SURF router and the appliance was susceptible to bandwidth exhaustion, and performance took a hit (even though the performance hit was acceptable).

NBIP, a Dutch coalition of internet service providers, has invented the NaWas, a scrubbing service. A customer of NaWas can advertise its IP prefixes through BGP originating from NaWas, in order to steer all traffic to the NaWas for scrubbing [35]. Even though we do not want to elaborate on NaWas' ability to scrub traffic contaminated by DNS water torture attacks, the lack of granular advertisements[5] is a dealbreaker for SURF.

For the L7-based DDoS attacks SURF wants to mitigate, the transit and peering capacity is more than sufficient to handle the attacks, hence an in-house scrubbing appliance—behind the routers hosting the transit and peering links—is within reach. The SURF network already allows layer 4-based traffic engineering (L4-based TE) (currently via BGP FlowSpec) to enable granular steering of traffic to other gateway devices, which can be beneficial to reduce the amount of legitimate traffic sent to a scrubbing appliance, hence also reducing bandwidth requirements for the scrubber and improving performance for legitimate traffic.

3.2 Evolution of DNS water torture attacks

The vulnerabilities in the DNS architecture allowing DNS water torture attacks to be successful have been present since the start of DNS, but research in this are has only started a few years ago, presumably because the first attacks were seen no earlier than 2014, according to Bushart and Rossow [5]. In 2016, Luo et al. [32] have performed a 'large scale analysis' on DNS water torture attacks. This is one of the oldest, if not the first paper targeting this DNS attack type. They discovered how DNS water torture attacks are different from other cyber attacks (including those not based on DDoS) involving randomly generated domain names. First of all, the associated traffic levels (# of queries over time) are significantly higher for DNS water torture attacks than for other attacks. On top of that, the frequency of characters in the domain name labels, label lengths, and the number of distinct IP addresses from

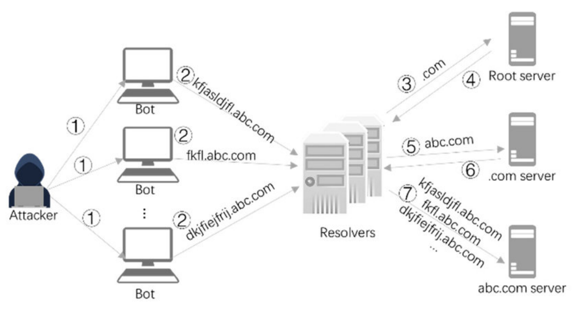

Figure 1: Example of a DNS water torture attack [15]

[4]the latest iteration of the SURF network

[5]Prefixes must be of propagable size in the default-free zone: /24 or greater for IPv4, /48 or greater for IPv6.

clients allegedly performing these queries, make these attacks distinguishable from other attacks. Finally, due to the presence of unique labels, caching recursive resolvers will not help to keep the query flood out of sight for name servers authoritative for the attacked zone.

Even if mitigating DNS water torture attacks is considered to be a tedious task, attack attribution is not hard. Because labels are generated at random, the chance of a query for an existing domain name is slim. As researched by Kaplan and Feibish [24], the rate of `NXDOMAIN` responses by a name server (telling the client the requested domain name does not exist) is a viable metric to detect a DNS water torture attack.[6] However, this only allows a victim to determine they are hit by the attack during or after the attack itself. Distinguishing DNS water torture queries from legitimate queries, to drop those queries before reaching the name servers, is needed to also nullify the victim impact.

Hasegawa et al. [18] have researched the well-known mitigation techniques to combat DNS water torture attacks. They have read a handful of papers where countermeasures were proposed, and they categorised the countermeasures as follows:

1. Applying Rate Limiting
2. Monitoring the `NXDOMAIN` sending rate on the name server
3. Detecting random labels in domain name
4. Using cached, expired name server responses
5. Mapping domain name to zone contents

Based on the paper from Hasegawa et al. [18], we will describe the five categories. The first solution is already used in the SURF network. It may or may not be effective to protect institution name servers from going down under pressure. However, whether to drop a query or not is purely based on traffic levels, hence legitimate queries are also subject to being dropped, just because no other information is available to base the accept or drop decision on.

The second solution relies on restricting responses based on the rate of `NXDOMAIN` responses being generated by the attacked name server. If the victim name server operator relies on DNS wildcards, allowing an attacker to synthesise labels generating queries for existing domain names, regardless of containing `RDATA` (`NOERROR`) or not (`NODATA`), the rate of `NXDOMAIN` is insufficient to classify DNS water torture attacks.

In the third category, a classification scheme is used to determine whether a query is part of a DNS water torture attack or not. This can be Bayesian-based classification, but classification based on the parameters mentioned by Luo et al. [32] is also possible. However, the effectiveness (specifically: to what extent will legitimate queries be blocked frequently, and to what extent will bogus queries be allowed frequently?) depends greatly on the classification scheme, with some of them being potentially too inaccurate.

In the fourth category, no effort is taken to protect the name server itself from the attacks. Instead, the negative effects of name server downtime were to be mitigated by allowing recursive resolvers to send out stale (expired) data to clients. The effectiveness of this solution depends on two variables: the time-to-live (for what period of time a response may be considered valid) and the presence in common recursive resolvers' caches. It is known the latter is strongly tied to the time-to-live value, because responses are meant to be kept in caches for up to the time-to-live seconds, but potentially for even longer than that, as proposed by Lawrence, Kumari, and Sood [28]. Hasegawa et al. [18] dismissed this option only due to large organisations hosting name servers with resource records containing a very short time-to-live, Lawrence, Kumari, and Sood [28] believe serving stale data also comes with its own set of security risks.

[6]However, as explained later, water torture attacks are also possible on 'existing' domain names.

The last solution, described as using `FQDN-based Allowlist Filter[s]`, relies on generating lists of existing domain names, and dropping a query depending on the existence or non-existence of the domain name. The lists can be generated in various ways: by relying on `NSEC` records (part of `DNSSEC`, used to prove the non-existence of an interval of domain names via a cryptographically signed DNS response[16]) in zones to enumerate a list of existing domain names (known as zone walking), or by initiating zone transfers. Zone walking is only possible if the victim name server operator uses `DNSSEC` with `NSEC` records. But in some cases, `NSEC3` records also allow zone walking [38]. Besides the requirement for `DNSSEC`, the zone walking requires a zone transfer (even if only `NSEC` or `NSEC3` records are transferred) or actively probing the name server beforehand for these records. A zone transfer itself does not rely on `DNSSEC`, but does require authorisation from the name server operator to initiate zone transfers, often not given out freely[30]. Besides, a zone transfer may not be sufficient to determine the final list of existing domain names. In their paper, Hasegawa et al. [18] mention the impact of wildcard records on DNS water torture attack mitigation, primarily because a wildcard record allows the adversary to craft long lists of random domain names that do exist for the name server.[7]

This last solution looks promising to us, because it does not rely on *guessing* the existence of a domain name. As the Dutch NREN, not only does SURF provide IP transit to dozens of institutions, SURF also hosts name servers. An institution can opt to only use SURF name servers as a secondary name server, but relying solely on SURF name servers is also possible. The latter solution may be used in conjunction with a hidden master, hosted on-premise by the institution. In either case, SURF is able to initiate a zone transfer for those zones it's authoritative for. There is no overview of all corporate domain names assigned to all institutions part of the SURF network, and neither does a list of all domain names owned/managed by those organisations, but a sample list exists on the SURF wiki [8]. Judging by this list, for *at least* 75% of the corporate domain names assigned to institutions we would be able to initiate a zone transfer, because SURF hosts at least one of the domain's authoritative name servers.[8]

3.3 Leveraging the extended Berkeley packet filter and eXpress data path for implementing virtual network functions

A few solutions described in Sections 3.2 and 4 leverage eBPF and XDP to implement the filter function (or VNF, explained later in the current section). In the current section, we will explain the technology behind and use cases for eBPF, XDP, and VNFs in the context of this research.

As described by Rice [37], eBPF allows users to develop, run, and control programs meant to be running in the kernel and in kernel mode, without writing kernel modules, and without requiring a reboot. eBPF is the second generation of the Berkeley packet filter, also known as classic Berkeley packet filter (cBPF), which has been present in the Linux kernel since the 90s.[9] Unlike cBPF, originally meant to provide packet filtering only, eBPF's use case is providing arbitrary, so-called eBPF programs to attach to various hooks present in the kernel, allowing the eBPF program to manipulate and monitor system state.[10] Using either the `llvm` or the `gcc`[11] compiler, eBPF programs are compiled to eBPF bytecode. To ensure this bytecode is not able to crash the kernel at any moment, an eBPF verifier performs analysis on the bytecode *before* inserting the code into the eBPF virtual machine.

[7] The maximum size of a label in DNS is 63 bytes, but multiple labels of 63 bytes can be used if the closest encloser [29] is at the apex level, resulting in a maximum DNS water torture label combination size of *approximately* `254 - (apex name size)` bytes.

[8] For example, `ns1.zurich.surf.net` is a SURF name server, authoritative for the zone `os3.nl`.

[9] In kernel source, both cBPF and eBPF elements may be categorised under 'BPF'. We are aware some of the eBPF elements discussed in this document may have their origin in cBPF.

[10] The term 'packet filter' is misleading when talking about the extended version of the Berkeley packet filter.

[11] Support was added later on, in 2019 [33]

Specifically, the eBPF verifier checks the bytecode against unbounded (never-exiting) loops, out-of-bound memory access, and exceeding a certain number of instructions [27]. If the eBPF verifier spots a violation, the program will not be loaded (rejected). If the program passes verification, the bytecode will be sent to a just-in-time compiler for compilation to native machine code. Finally, machine code will be executed in the in-kernel 64-bit eBPF virtual machine.

eBPF programs usually run in kernel space in kernel mode, but eBPF code can also be ran in user space, albeit with reduced functionality. Additionally, software running in user mode can be used to interact with programs running in kernel mode. The `bpf` syscall, exposed in user space, allows a developer to load and stop eBPF programs, and to interact with eBPF maps from user space [2]. For instance, the software `bpftool` is a user space tool—part of the Linux kernel source—used to interact with eBPF programs and maps[36].

An eBPF program can attach to various types of hooks. One of them is the kernel probe (also known as `kprobe`), a breakpoint that can be set on any kernel instruction, except for a few instructions excluded for stability reasons.[22] Kernel probes can be system calls, but also other type of functions executed in the kernel. More relevant to this research, are the networking-related use cases for eBPF, also described by Rice [37]. The program type determines the hook used. There are three program types for socket programming, useful for filtering transmission control protocol (TCP) segments or user datagram protocol (UDP) datagrams. Further down the stack, two program types are available that can be attached to Linux control groups (`cgroups`), allowing network firewall rules to be applied to specific processes only. Another layer down the stack, the traffic control (TC) class and program type can be found, followed by the XDP as the very first hook executed. Both hooks are not specific to control groups and are executed before any frame processing (`IEEE 802.3`) is performed, allowing for raw packet filtering or mangling. However, in the TC hook, the `sk_buff` struct is exposed, compared to the `xdp_buff` struct in XDP. The `sk_buff` struct contains packet metadata only available in TC mode, but injecting the metadata also makes the TC hook slower to execute than the XDP hook[6]. Other than that, the differences between TC and XDP are small, even going as far as sharing offloading opportunities (for instance, `SmartNIC` to run code directly in the network interface controller (NIC), reducing kernel code instructions), except for traffic direction restrictions: XDP only operates on ingress (incoming) traffic, whereas TC hooks can operate both on ingress (incoming) and egress (outbound) traffic.

As explained earlier, eBPF maps can be used to exchange data between kernel mode and user mode. A map is a key-value store of a certain map type, where there exact data type of the key and values may or may not be adjustable by the author of the eBPF program, depending on the map type. If using the `bpf` syscall (in user mode), a file descriptor pointing to the map is used to exchange data and to lookup elements. Within the eBPF program, depending on the map type, one or more of the `peek`, `push`, `pop`, and `lookup` functions are available. In Linux 6.8.0-rc2, 20 map types are available[12]. One of the newest map types is the Bloom filter map type, implemented by Joanne Koong [23], was added to version 5.16 of the Linux kernel (released in January 2022 [31]). This map type implements a Bloom filter, an append-only, probabilistic data structure offering fast lookup times (”is an element part of a set or not”), at the expense of false positives. As seen in Section 4, the potential of Bloom filter maps for mitigating DNS water torture attacks has been evaluated before, before native support was added to the kernel.

To understand the potential role of eBPF within NFV, an overview of NFV must be given. The European telecommunications standards institute (ETSI), an European institute working ”timely development, ratification and testing of globally applicable standards” within the telecommunications, has coined the term NFV, and a number of NFV specifications has been developed by the ETSI. In NFV architectures, network functions (including, but not limited to switching, forwarding, and firewalling/policy enforcement) that typically

[12]`https://docs.kernel.org/bpf/maps.html`

reside on specialised hardware, with their own application-specific integrated circuits, are virtualised on top of commodity hardware, similar to running system virtual machines on hypervisors. Per the ETSI specification 'NFV 001 V1.3.1', the reduced hardware footprint allows more efficient usage of hardware and energy, and enables automation in control plane functions (i.e. the management interface, traditionally used by network engineers) [13] Because VNFs only exist in software and do not depend on the implementation of hardware, they can be deployed on NFV infrastructure in a shorter timeframe than traditional network appliances, which require hardware to be installed in a rack, connected and configured.

How VNFs are implemented depends on the NFV infrastructure. eBPF is one of the many technologies for packet filtering and mangling; assuming a virtual machine running on the Linux kernel, Miano et al. [34] consider Data Plane Development Kit (DPDK) and *FD.io's* vector packet processing (VPP) to be alternatives to eBPF. In fact, any type of software able to mangle or filter packets could be considered for a VNF, as long as the software's instructions can be directly executed on or translated to the NFV infrastructure's instruction set. However, eBPF being built-in into Linux, eBPF is able provide high throughput due to bypassing user mode. It runs on commodity hardware (including `amd64`, which is still a widely-used architecture in the IT landscape), it is proven technology (its predecessor cBPF has been part of the kernel since the 90s), and allows engineers to reuse Linux building blocks for administration and debugging. Not only does it reduce the learning curve for us, but the x86-based NFV infrastructure of SURF already supports Linux virtual machines running eBPF programs.

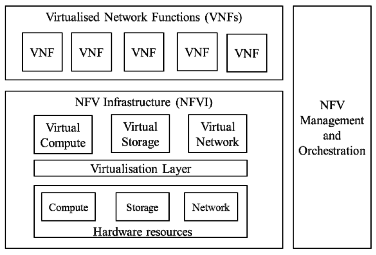

Figure 2: High-level overview of NFV architecture[12]

4 Related Work

In the research from Hasegawa, Kondo, and Tode [17], zone information was used as a parameter for a DNS water torture mitigation solution. Kostopoulos et al. [26] have researched the use of Bayesian filters in eBPF programs, instead of caches to classify the legitimacy of DNS queries, and Kostopoulos, Kalogeras, and Maglaris [25] tried to use Bloom filters in eBPF programs, effectively enhancing the research from Hasegawa, Kondo, and Tode [17] with a different eBPF map. Furthermore, in their research, Cloudflare has shown eBPF and XDP can provide high-performance DDoS mitigation on multiple network layers, even though most of the research was conducted on the transport layer [14]. Hasegawa et al. [18] have published their paper in December 2023, touching upon using zone contents to mitigate DNS water torture attacks, but their solution requires cooperation from the adversary's provider, and does require modification from the victim to implement changes to their name server software. However, none of the papers seem to actually have implemented zone transfers to allow creation of eBPF programs for large internet service providers housing a variety of customers, where zone information is almost freely available for the provider, yet where adjusting name server software for all clients is not a viable option. Additionally, the applicability of these research papers to NRENs has not been determined to date.

Finally, as explained in Section 3.3, support for a Bloom filter type eBPF map has been added in 2022. This research project will validate this option in its proof of concept. This eBPF map type was not leveraged in the aforementioned papers. The new eBPF type may also allow us to overcome a limitation in the research of Kostopoulos, Kalogeras, and Maglaris [25], where the maximum length of domain names was limited by the settings used for the Bloom filter hashing functions, effectively reducing the mafximum length to a size that is lower than the maximum in DNS, known to be 255 bytes, "including separators" (dots) [11].

5 Methods

We will develop a proof of concept on DNS water torture filtering in a programming language that suits the SURF NFV infrastructure. The NFV infrastructure consists of `x86`-based virtual machines, which supports virtual machines running eBPF programs, hence we will focus on writing said programs for the actual VNF.

In the confusion matrix in Table 1, the Cartesian product of the filter actions and the types of traffic have been plotted. In the ideal situation, legitimate DNS queries will be put back on the network (True positive), and DNS water tortures are filtered (True negative). In a Positive case, the domain name has been found in the Bloom filter, hence traffic will be passed on. In a Negative case, the domain name is not present in the Bloom filter, hence traffic will be dropped.

		Filter prediction	
		Legitimate	Water torture
Actual Traffic	Legitimate	True positive: legitimate traffic not dropped	False negative: Legitimate traffic dropped
	Water torture	False positive: Water torture traffic not dropped	True negative: Water torture traffic dropped

Table 1: Legitimacy of filter function plotted in confusion matrix

Measuring the ratios between `True {positive,negative}` and `False {positive,negative}` will help us to understand how effective the filter is.

The ratios can be expressed as:

$$\text{Specificity} = \frac{\text{True Negative}}{\text{True Negative} + \text{False Positive}} \tag{1}$$

The Specificity shows the proportion of DNS water torture traffic that is caught by the filter and is dropped accordingly. Higher is better.

$$\text{Sensitivity} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \tag{2}$$

Sensitivity in this case means how much of the legitimate traffic has been let through. It describes the proportion of rightfully allowed legitimate traffic, compared to the sum of wrongfully dropped legitimate traffic and legitimate traffic rightfully let through. Higher is better. The sum of `Miss rate` and `Sensitivty` must be equal to `1`.

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{True Negative} + \text{False Positive} + \text{False Negative}} \tag{3}$$

Accuracy is the proportion of correctly predicted traffic. Higher is better.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \tag{4}$$

The Precision describes what part of the traffic, that is let through by the filter, has been let thorough rightfully, because the domain name is actually present in the Bloom filter. Higher is better.

Finally, the incurred performance penalties will be graphed to allow SURF to make a cost-benefit analysis. In the test setup, the VNF posing as DNS water torture scrubber—considered a device under test (DUT) [3]—will have its overhead measured by measuring the delay incurred by using the VNF as a gateway for legitimate DNS traffic.[13] The effect on DNS response time by adding the DUT, compared to not using the DUT, will be determined. The overhead can be expressed as:

$$\text{DUT overhead} = \text{response time with DUT} - \text{response time without DUT} \tag{5}$$

6 Experiments

6.1 Filter program

To be able to execute the tests described in Chapter 5, a filter program (VNF) has been written to parse DNS query packets, fetch the `QNAME` from the packet, match this against a list of existing domain names, with the DNS name server it's protecting against a DNS water torture attack containing the master list of existing domain names (in the form of a DNS zone). The VNF should block DNS queries with non-existing `QNAMEs`. The functional diagram for the VNF is attached as Figure 3.

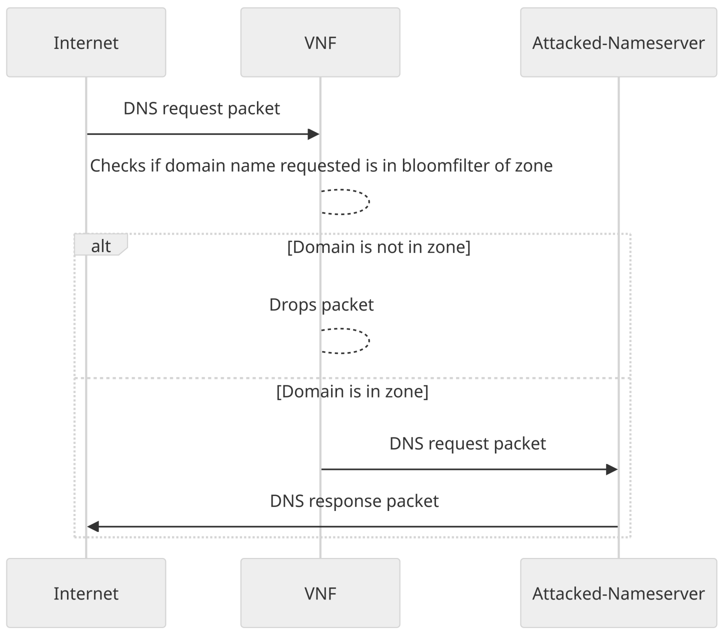

Figure 3: Flow of a DNS packet

As explained in Section 3.3, eBPF has been chosen as the underlying technology. The `C` programming language has been chosen, because most code examples on the Internet are provided for this language. The Bloom filter map type, introduced in Linux 5.16, has been used for low-latency lookups. In accordance with SURF, the program only enforces a policy on UDP port 53, the de-facto port for DNS. Dual-stack support is present, meaning both IPv4 and IPv6 name servers can be protected by the program.

To protect the name server, bogus DNS queries should not be forwarded to the authoritative name server. Fully dropping a packet requires fewer lines of code, but may

[13] In the project proposal, we considered "water torture traffic" to be in scope for the delay tests. This test has been scrapped.

cause time-outs and retries client-side. While this may put extra burden on the VNF, sending `NXDOMAIN` responses on behalf of the authoritative name server effectively abuses a cache poisoning vulnerability, may require the presence of zone-signing keys (private keys in DNSSEC) to send signed responses (if the affected zone uses DNSSEC), and increases the bandwidth usage on the SURF network. We have decided to respect the 'authority' of the authoritative name server by not responding to bogus queries, and we assume the high throughput property of eBPF allows the VNF to handle the burden of client retries.

Because the program only has to consider actions on ingress packets, the XDP program type has been chosen for low-latency query processing. Dropping packets, only possible if a `QNAME` has been parsed and if the element is not present in the Bloom filter (that is, the return code after executing `bpf_map_peek_elem` is equal to `-ENOENT`), is done by returning `XDP_DROP`, in all other cases, `XDP_PASS` will be returned to pass the packet up the TCP/IP stack of the kernel. The full XDP program flow can be found in Figure 4.

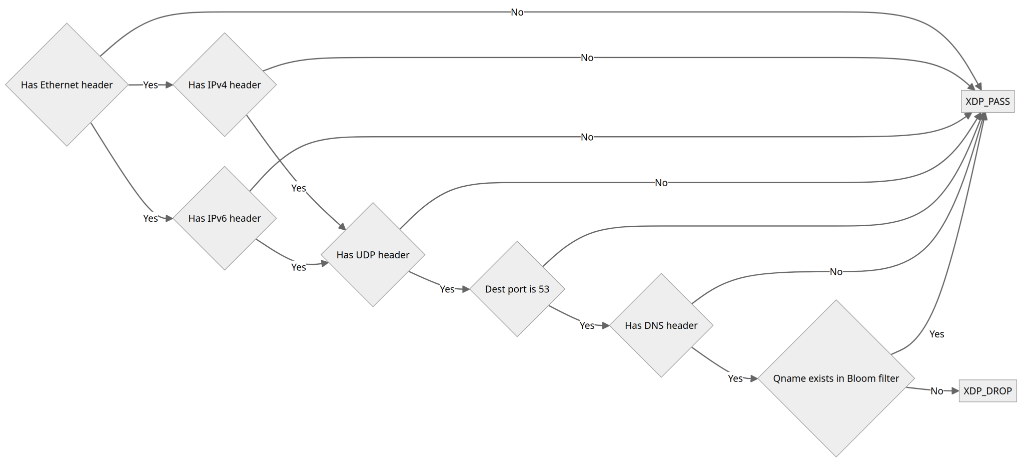

Figure 4: XDP program flow

An inline `parse_dname` function, to parse a domain name in a DNS query and inject it into a `uint8_t[254]` data type pointer, has been written, based on earlier works of PowerDNS. While the maximum size of a `QNAME` is 255 bytes, the data size on the wire is not 255 bytes (including padding bytes, if necessary) [10]. This means parsing a `QNAME` requires a specialised iteration of all characters on the wire. To understand this process, understanding the wire format for regular domain names is key.

As explained in RFC 1035, the original DNS specification, an *absolute* domain name is *"a series of labels, and terminated by a label with zero length"*. A label starts with a length byte `n`, followed by `n` bytes of content (maximum: 63 bytes). Due to the presence of length labels, especially the root label at the end of a domain name, domain names can be formatted without padding up to 255 bytes. An example of an absolute domain name is: `www.os3.nl.`. The `www` and `os3` labels consist of three characters, the `nl` consists of two characters, and the root label is always equal to the length of zero. On the wire, this absolute domain name will be encoded as: `03 77 77 77 03 6f 73 33 02 6e 6c 00`, with a total size of twelve (12) bytes.[14] To put it bluntly, the dots (`0x2e`) are replaced with length labels on the wire, and vice versa to show the domain name to the end user. For debugging purposes, the `parse_dname` function replaces the non-root length labels with dots, and strips the root label, resulting in `.www.os3.nl`.

The `uint8_t[254]` data type is one byte smaller than an array of 255 bytes, which is the maximum size of a `QNAME` in DNS including all labels. Because we do not need the

[14]Domain name compression can be used to reduce repetition overhead in a DNS reply, but domain names will never be sent compressed in queries.

root label (one byte), an array of 254 bytes is sufficient to be compliant to the original DNS specification. Therefore, this program does not suffer from the length limitation experienced by Kostopoulos, Kalogeras, and Maglaris [25].

6.2 Test environment

The test environment for the PoC will consist of three VMs, each with their own role. One VM is designated as a DNS traffic generator for simulating a DNS water torture attack and legitimate DNS traffic. A second VM is designated as the DNS name server. The last VM will run the eBPF-based VNF, and also forwards traffic between the network segments connecting the DNS traffic generator and DNS name server. Each VM will be equipped with four virtual cores, 8 GB memory, and a 20 GB virtual disk. All VMs run on Ubuntu Server 23.10 and are virtualised on a Proxmox 8.1 hypervisor. The hypervisor is a Dell PowerEdge R730xd with two Intel Xeon E5-2650 v4 processors and 128 GB DDR4 memory. The VM disks are stored on a Samsung PM983 960Gb U.2 NVME SSD. The network consists of two network segments, connected by the the VNF VM. The test environment is represented in Figure 5.

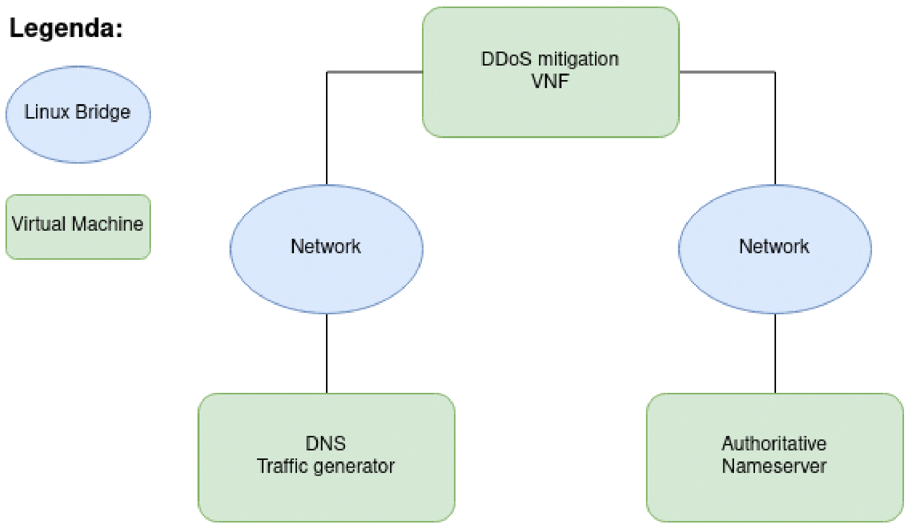

Figure 5: Test environment overview

The NSD name server software (version 4.7.0) will be used to simulate an institution's name server. NSD is developed by NLnet Labs[15], and NSD will be installed from the Ubuntu package repository. The NSD name server software was chosen because it is a purely authoritative name server which is widely used. The name server hosts one zone called `rp1.test`, and the VNF will be allowed to initiate a zone transfer to populate the Bloom filter with the zone contents. The builtin rate limiter of NSD will be disabled to avoid test result skew due to the name server dropping queries. A zone with the apex `rp1.test.` has been defined. This zone contains 3,257 unique resource record sets/domain names.

The VNF VM runs the XDP program written for the PoC (see Section 6.1) on the NIC that is attached to the network segment hosting the DNS traffic generator in native mode. This VM also routes the network traffic between the DNS traffic generator and the name server.

To not skew our experiment results, we tested the maximum amount of queries that can be sent without overloading the traffic generator VM. `flamethrower` is a single-threaded process that can be parallelised by running concurrent processes. In the test environment,

[15]`https://nlnetlabs.nl/projects/nsd/about/`

one process using ten generators can send up to 32 queries per millisecond without overloading the traffic generator VM. This number has been verified by sending 32 queries per millisecond to the name server each time with a varying number of available cores for the name server to use. As expected, reducing the number of cores eventually caused NSD to drop queries due to being overloaded.

6.3 Response times

As noted in Section 5, a performance penalty may have been incurred by forwarding DNS queries through the VNF. The difference in response times with the VNF enabled and without the VNF enabled, allows us to quantify the overhead of the VNF. The response time is considered to be round-trip: the elapsed time period from query transmission on the line to full receipt of the response.

In order to perform analysis on DNS response times, the tool `dnsperf` will be used. `dnsperf` is developed by the DNS Operations, Analysis, and Research Center (DNS-OARC), a consortium for DNS operators, researchers, and developers[16]. `dnsperf` allows us to measure per-query latency for an arbitrary list of domain names and an arbitrary number of queries[17].

An inherent effect of the VNF is that it will drop packets for queries for non-existent domain names. Response times can only be measured if the name server gives a response for a DNS query. Therefore, DNS queries have been adjusted to solely contain existing domain names (i.e. those present in the Bloom filter eBPF map). For the `dnsperf` runs, out of the existing domain names, two existing ones have been chosen:

```
ns1.rp1.test A
existing-1-113.rp1.test A
```

In a single `dnsperf` run, each query is repeated 1,000,000 times, adding up to 2,000,000 queries per run. The resulting command for a run is: `dnsperf -d query_existing_list -s 10.0.10.53 -n 1000000 -v`, where `-d query_existing_list` relates to the file containing the two aforementioned queries, `-s 10.10.10.53` being the IP address of the name server (`UDP/53` is implied), and `-n 1000000` being the 'repeat count'. `dnsperf` shows response time in seconds with six decimals, allowing us to measure response time with a precision of 1 microsecond (= 10^{-3} millisecond and 10^{-6} second).

The size of the Bloom filter (`max_entries`) and the number of hash functions (lower four bits of `map_extra`, maximum of $2^4 - 1$ functions [7]) can have an impact on the lookup speed and the number of false positives. At the very least, using more hash functions will reduce the number of false positives, but will also increase the lookup latency (and therefore decrease lookup speed). For this overhead experiment, the impact on the lookup speed is of interest to us. For every combination of the Bloom filter sizes `{4096,8192,16384}` and `[1,15]` hash functions, six `dnsperf` runs were performed, adding up to $6 \cdot 3 \cdot 15 = 270$ runs. Before each run, whose results will be incorporated in the boxplots, $20,000$ queries will be sent to to 'warm up' the VM hosting the VNF.[18]

Testing the maximum throughput of the VNF is not part of this experiment. Therefore, the `dnsperf` settings above were tuned till *zero* query timeouts were encountered.

6.4 Ratios

In a Bloom filter set, a lookup action can yield two values: "definitely not in set" or "possibly in set". In the ratio measurements, keeping the confusion matrix in mind (Table 1), the only possible values should be: True Positive, False Positive, and True Negative. False Negatives should not occur.

[16] `https://icannwiki.org/DNS-OARC`
[17] `https://www.dns-oarc.net/tools/dnsperf`
[18] Due to an unknown cause, unloading the VNF seems to cause a temporary performance degradation, at least till a few dozen of DNS queries have been sent. By 'warming up', we want to avoid this result skew.

To test the performance of the filter, the tool `flamethrower` will be used. Similar to `dnsperf`, `flamethrower` is developed by the DNS-OARC. `dnsperf` is the successor of `dnsperf` and has a built-in random label generator, allowing us to send randomly crafted DNS queries suffixed with a static fully qualified domain name (FQDN). This will allow us to simulate DNS water torture attacks.

In the tests to send mixed traffic to the DNS server VM trough the VNF VM to be able to filter the generated traffic and test the filter capability's. Creating a real DDoS to overload the VNF or DNS VM is not part of the experiment. Consequently, the `flamethrower` tool will be tuned for maximum DNS questions per seccond (QPS) without overloading either the generator, the VNF, or the DNS VMs.

Similar to the overhead tests, combinations of a Bloom filter size (out of three) and a number of hash functions will be tested to find out what combination gives (close to) 100% Specificity, Sensitivity, and Precision. Three tests will be performed for each combination, and the median of the results will be used.

7 Results

7.1 Ratios

Following up on the test methodology described in Section 6.4, the ratio tests were performed on thee filter sizes: 4096, 8192, and 16384. 1 up to 10 hash functions on the `FQDNs` have been tested.

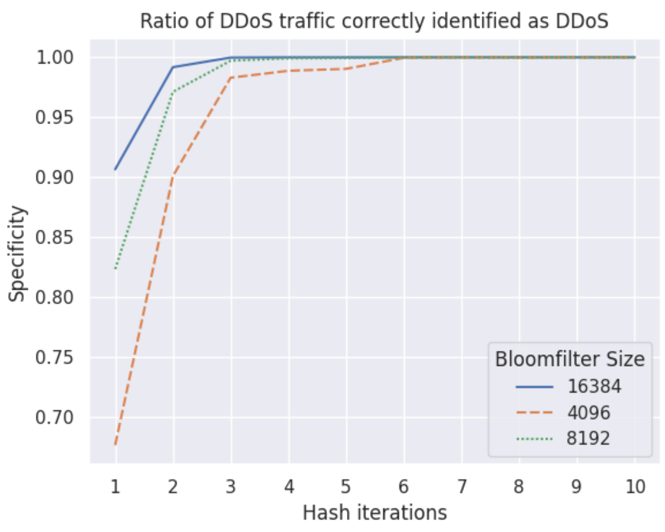

Figure 6: Proportion of DNS water torture traffic correctly identified as such

In Figure 6 the proportion of DNS water torture traffic correctly identified as such has been plotted. Increasing the number of hash functions does increase the Specificity, and at 7 functions, the VNF correctly identifies all DNS water torture traffic. Increasing `max_entries` also increases the Specificity.

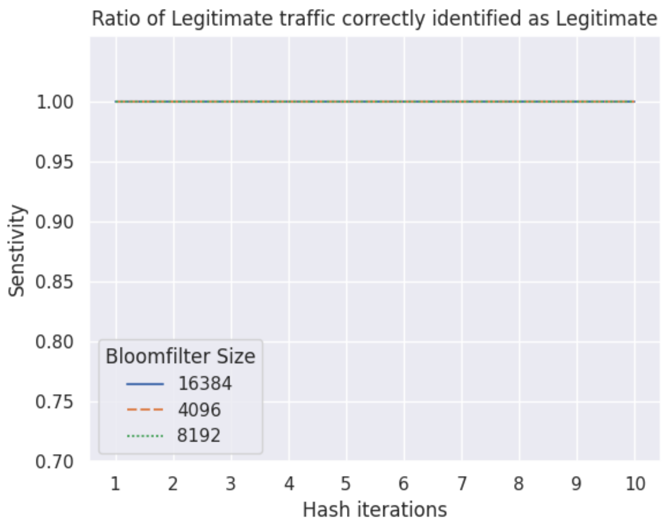

Figure 7: Proportion of legitimate traffic correctly identified as legitimate traffic

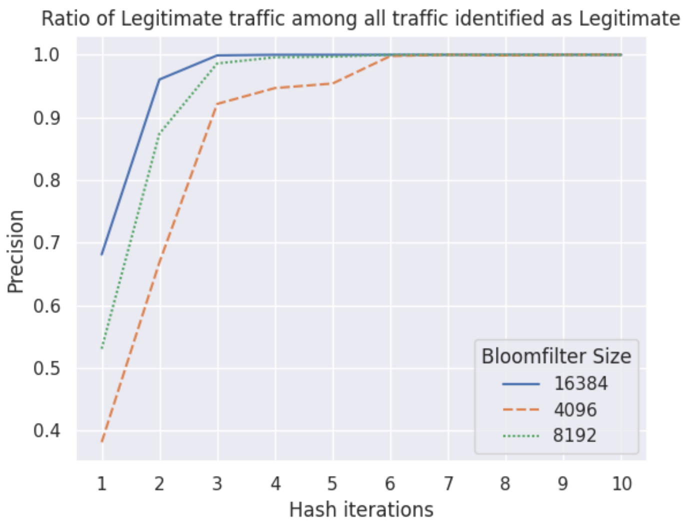

Figure 8: Proportion of legitimate traffic to all traffic identified as legitimate

As displayed in Figure 7, the Sensitivity of the program is 1. This means the Bloom filter is working correctly, because no False Negatives occur (legitimate DNS queries being blocked). It also means the Precision, shown in Figure 8, rises accordingly, because DNS water torture traffic will be blocked more often instead of letting it through.

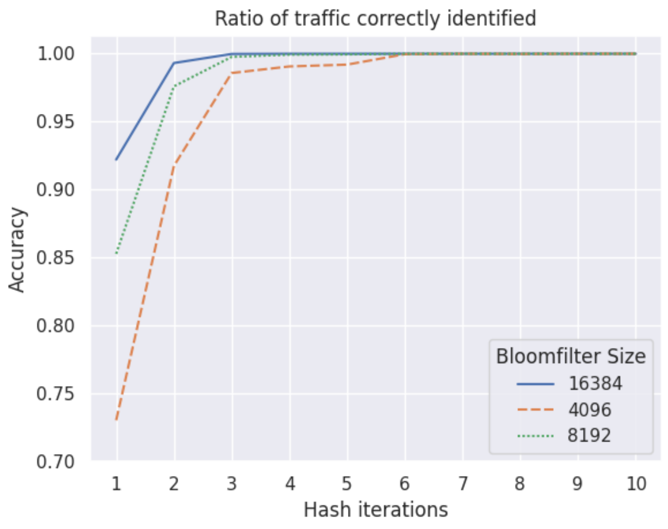

Figure 9: Proportion of traffic correctly identified

Figure 9 shows the proportion of traffic that was correctly mapped by the VNF to one of the four categories in the confusion matrix. As the number of the hash functions increases, the Accurary increases accordingly.

7.2 Response times

The first set of tests has been executed for the test environment where the VNF was *not* loaded. This is the baseline for all other results. As described in Section 6.3, the response time has been measured via `dnsperf` by sending two million DNS queries for existing domain names.

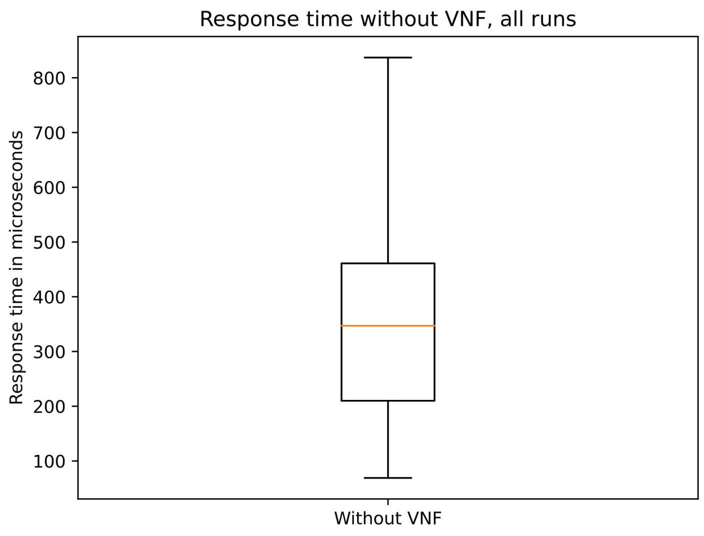

Figure 10: Response times for DNS queries with VNF unloaded

Lowest response time	69
Highest response time	7197
25th percentile (Q1)	210
50th percentile (Q2/median)	347
75th percentile (Q3)	460
Interquartile range (IQR)	251
Mean	348

Table 2: Raw results corresponding to boxplot in Figure 10 (precision: 1 ns)

The lowest and highest response times, and their quartiles and interquartile ranges can be found in Table 2. The corresponding box plot can be seen in Figure 10. As with all box plots in this section, the outliers have been removed to make the box plots easier to read. Outliers are data points—DNS queries—whose response times are lower than $Q1 - 1.5 \cdot IQR$ or higher than $Q3 + 1.5 \cdot IQR$. In Table 2, we have determined DNS query response times do not follow a normal distribution, because the boxplot is positively skewed (median/Q2 is closer to Q3 than to Q1).

	Q1	Q2	Q3
Run 1	137	159	186
Run 2	308	344	378
Run 3	352	389	422
Run 4	454	492	535
Run 5	184	209	238
Run 6	284	526	582

Table 3: Response time quartiles for six individual runs, without VNF (precision: 1 ns)

To find out if the skew can be caused by the round-trip times (RTTs) varying over time, an analysis has been conducted for all six individual runs that were performed to gather the statistics in Table 2. For brevity, only the three quartiles have been plotted. The results can be found in Table 3. For each quartile, the cell with the lowest value for that quartile has been colored. The first run experienced the fastest DNS responses overall, with the fifth run also being marginally faster than the four others. In fact, excluding the fifth run, a trend exists where the $Q2$ and $Q3$ runs induce ever-elevating round-trip times. We would like to note in a previous test, the RTTs were decreasing over time (all quartiles were on their lowest in the fifth run), we were not able to figure out what influencing factor causes this skew.

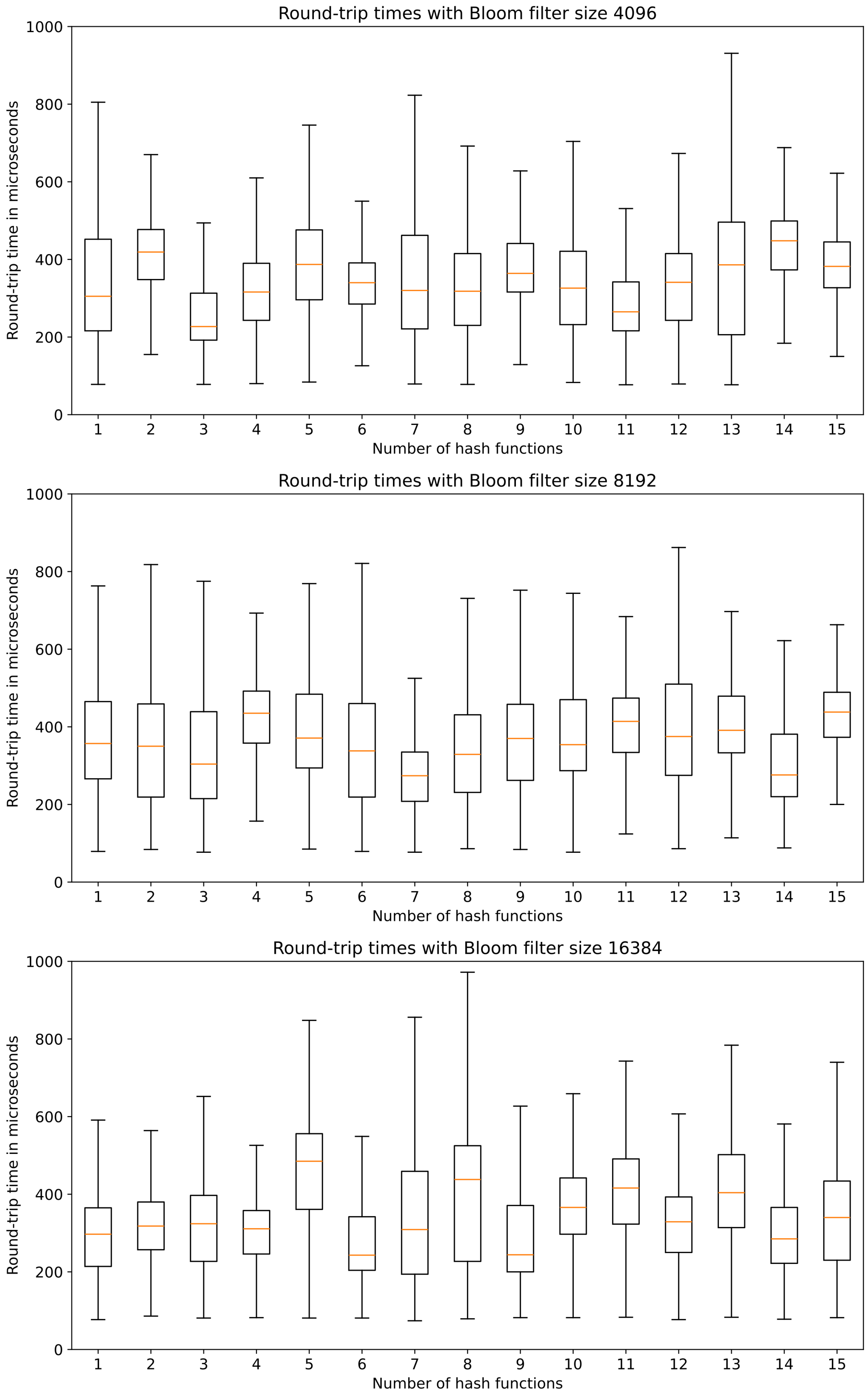

Figure 11: Response times for DNS queries with different hash iterations grouped by Bloom filter size.

The three quartiles of all Bloom filter size and no. of hash functions combinations have been plotted in Figure 11. For each quartile, the lowest value has been colored in blue. In the tests, the test series with `3` hash functions and a Bloom filter size of `4096` produces the fastest response time on all three quartiles, compared to the 44 other combinations. In fact, due to an unknown reason, the RTTs for each quartile are even lower than without the VNF enabled.

1 to 5 hash functions			
	Q1	Q2	Q3
4096	234	338	459
8192	258	370	471
16384	249	331	414
6 to 10 hash functions			
	Q1	Q2	Q3
4096	259	339	422
8192	234	328	432
16384	211	312	447
11 to 15 hash functions			
	Q1	Q2	Q3
4096	257	366	455
8192	295	390	474
16384	251	352	446

Table 4: Response time quartiles for the three tested Bloom filter sizes (precision: 1 ns)

Figure 11 was generated by looking at all RTTs for 45 combinations of Bloom filter sizes and number of hash functions. To know if adjusting the Bloom filter size could affect the RTTs, the results from Figure 11 have been analysed even further. Due to hardware constraints during the data analysis, the data has been divided into three even slices of a number of hash functions: $1, 5$, $6, 10$, and $11, 15$. The results can be found in Table 4. For each quartile, the lowest value has been colored in blue. In two out of three slices, Bloom filter size `8192` produces higher RTTs than `4096` and `16384` on all three quartiles. In most cases, `16384` offers the lowest RTTs, except in $Q1$ for the the first slice, and in $Q3$ in the second slice.

8 Ethics Reflection

In the project proposal, we opted to perform our experiments in the production realm of the SURF network. Specifically, tests would be performed the OS3 hardware and with the IP address ranges under the control of OS3, or on SURF infrastructure explicitly used for testing purposes. Due to hardware constraints, the experiments have been conducted on personal hardware only accessible by the researchers. Not a part of the DNS water torture attacks has been sent to systems other than those in the local bridge domain. This means no targets other than the intended targets could have received the DNS water torture traffic. The impact on non-consenting parties is not present.

Publication of this report, however, could impact the way an adversary is creating and conducting DNS water torture attacks. The adversary could devise a new strategy based on the weaknesses in this PoC, if it were to be deployed in production environments. For that reason, with the help of SURF, confidential information is not included in this report (see the `TLP:CLEAR` classification on the front page). The residual risk is accepted, as part of effective risk management, also taking into account sharing knowledge and code under a free license is part of the core principles of OS3 (Open Standards, Open Software, Open Security).

9 Discussion

9.1 QNAME parsing

In essence, the VNF checks whether a domain name exists in a set of domain names. A domain name found in a DNS query is called a `QNAME`. The set of domain names is constructed based on the results of a zone transfer. To ensure the program is able to determine whether a domain name is part of a set, the format of both the `QNAME` and all domain names in the set must be equal.

In Section 6.1 we touched upon the `QNAME` format used on the wire. In short, domain names can be of variable length, and the `QNAME` field in a DNS query is of variable length as well. The `QNAME` trails the DNS header, consists of label series, and ends with the root label (`0x00`).

Working with data of variable length in eBPF can be tedious, because the eBPF verifier will reject any program that *looks like* it will access out-of-bounds data. Explicit bound checks have to be used in the code to pass the verifier checks. As noted by Jeff Dileo [21], implementing loops in eBPF can be a tedious task given the verifier's constraints too, and the eBPF way of padding out a data structure is far from perfect. With 'eBPF C' being a subset of 'C', bluntly copying 'parsing code' from name server software source trees is not an option. Given our lack of prior experience on eBPF and the scarcity of eBPF knowledge at SURF, it took us close to two weeks to get a version of the `parse_dname` function producing 'workable output'. Attempts failed due to out-of-bounds data reads, exceeding the instruction limit of the eBPF verifier, and garbage data from memory being copied as 'padding' to the `QNAME` buffer. `QNAME` lengths have been a source of issues in other eBPF-based DNS experiments as well, such as in the research of Kostopoulos, Kalogeras, and Maglaris [25].

The `bpf_loop` helper function, also introduced by Koong in 2021, allows array traversal without triggering the instruction limit for the eBPF verifier. This function simulates the 'map' higher-order function, where each element of an array is passed through a callback, after which the element will be replaced with the callback return value. We were not able to integrate `bpf_loop` without triggering the eBPF verifier. To date, the cause of this error is unknown.

In the end, the parsing code from PowerDNS has been rewritten to replace the length labels, excluding the root label, with 'dots'. This is not compliant to the absolute domain name format found in zone files, but the domain names are human read-able. Furthermore, before populating the `uint8_t[254]` structure (254 elements array of unsigned one-byte integers[19]), all elements will be zeroed out (padded) through a `__builtin_memset` call. This ensures no garbage data from memory is passed to the eBPF map's lookup function.

9.2 Wildcard records and delegations

A name server authoritative for a zone will craft its DNS responses based on resource record sets (RRsets) defined in name server. The 'set' part is used to identify owner names (domain names) with multiple resource records (RRs). For instance, the RRset for the owner name `www.os3.nl.` consists of at least two resource records, one of the type `A`, and one of the type `AAAA`. The VNF only considers a domain name to be 'existent' if the `QNAME` literally matches the owner name for any defined RRset.

[19] Theoretically, a signed one-byte integer would also be sufficient for `QNAME` parsing.

In the case of 'wildcards', a `QNAME`'s associated RRsets may be found in the zone file under a different owner name. Per Lewis [29], a wildcard is any RR where the left-most label is exactly one asterisk ('*'). At name server side, RRsets with owner names equaling the source of synthesis will be used in DNS responses. A wildcard allows a name server operator to craft RRsets for DNS queries with an arbitrary value in the left-most label(s).

As explained in Section 3.2, wildcards greatly increase name server exposure to DNS water torture attacks. On top of that, adjusting the eBPF program to check if the source of synthesis is defined requires multiple lookups in the eBPF map (starting at the root, one for each label, appended with its consecutive labels [29]). Not only would this decrease the performance of the eBPF program, it also requires a substantial amount of work. For these three reasons, we have decided to exclude wildcard support. If institutions have business-critical applications hosted on domain names whose responses are crafted based on wildcard records, we advise them to explicitly define RRs with the absolute domain name as owner name.

Finally, the scripts used in the PoC assume the queried name server is not only authoritative for the zone requested in the transfer, but also all domain names that are child to the apex. If, however, a subdomain has been delegated to a different name server, indicated by a RRset of type `NS`, the different name server should be queried for zone information. Subdomains that are children of owner names in `DNAME` type RRs are not supported either. Given the current state of the PoC scripts, existing domain names belonging to delegated subdomains will be marked as non-existent, and legitimate traffic will be dropped.

10 Conclusion

In this section, we will reflect on the results from Section 7, and on our research (sub)questions.

For the first sub-question, *"How can DNS zone information be incorporated in a multi-tier architecture for mitigating DNS water torture attacks?"*, we can conclude the following. Zone information is a viable method for filtering DNS water torture attacks, because a filter program (be it in a VNF or on a traditional firewall appliance) is able to predict with good accuracy whether a DNS query is part of a DNS water torture attack or not. It does not rely on name server responses to user queries (other than the response to the `AXFR` query), the concept is not based on predictions for label lengths and characters, and arguably, the concept is easier to explain than for the alternative solutions, because the concept boils down to an "if domain-name in domain-name-list then else" check. Regarding the multi-tier architecture part of the sub question, we think the alternative solutions can be evaluated to filter traffic if the zone information is *not* available, yet alternatives could help the filter program to filter at least a part of the traffic before hitting the filter.

For the second sub-question, *"How much overhead does traffic washing using VNFs introduce?"*, we can conclude the following. The overhead results can be found in Section 7.2. In theory, overhead always exists when comparing to the situation without VNFs, and Bloom filter properties could increase or decrease the overhead. Unfortunately, due to the sheer number of inconsistencies in the results after processing the data, we cannot affirm the hypothesis. Even though Bloom filter properties have influenced the ratio measurements results, changing those properties does not seem to affect the overhead in a predictable way. Furthermore, we were not able to verify the hypothesis that introduce the presence of the VNF itself affects the overhead.

Finally, the main question, *"How could SURF mitigate DNS water torture attacks within the current generation SURF network to protect its customers?"* can be answered. As shown in this research, SURF's access to zone information for institutions can be useful to provide VNFs with data to combat DNS water torture attacks. VNFs offer a viable option for mitigating DNS water torture attacks, and we think other L7-based DDoS attacks can also be filtered using VNFs. Regarding the *current generation* part of the research question:

despite the VNF not being suitable for production roll out yet, primarily due to absence of observability and configuration management, the PoC has been developed with SURF's shift to NFV-based network applications in mind. SURF's NFV infrastructure and L4-based TE capabilities in the network are ready for deploying this VNF.

11 Future Work

First of all, the eBPF program only performs `QNAME` parsing if traffic is sent to UDP port 53[20], as it is the original (and to date, most common) port of DNS. Support for TCP is standard, though, and both legitimate and bogus DNS queries may be sent over TCP [10]. Adding TCP support should not be an issue in eBPF programs of the type XDP. Moreover, inspection of *DNS over HTTPS*[19] and *DNS over TLS*[20] traffic is not possible due to payload encryption. We were not able to find existing research on actual decryption of packets in XDP in kernel space. However, we suggest evaluating if the `AF_XDP` address family could help to decrypt packets with minor overhead in user space, where existing SSL/TLS libraries can be used [9].

Also, in the PoC, the Bloom filter contents are rather static. After filter population, the contents can only be changed by re-running the zone transfer script. The Bloom filter is in no way the single source of truth—the authoritative name server is—but having a copy of the zone is inevitable. If the VNF was to be enabled for a period of time (multiple days, or longer), institutions should be able to update the Bloom filter contents. For redundancy purposes, DNS zones can be hosted on multiple authoritative name servers. One of those implementations is having a number of secondary name servers serving a zone, with the master being the single source of truth. In this setup, `DNS NOTIFY` transactions can be used to notify secondary name servers of updates made to the zone [39]. We advise researching the options of developing a user space program that converts the `DNS NOTIFY` to eBPF map `push` instructions. The researchers should be aware the Bloom filter map type does not support deleting elements [7].

On top of that, we advise researchers to conduct experiments with the `QNAME` length as a variable, to find out if the overhead and/or the Accuracy change linear to the `QNAME` length.

Finally, as seen in Section 6.3, it was not possible to determine the overhead incurred by steering legitimate traffic through the VNF. Future research could include changing the testing methodology, and if reliable results have been produced, we advise the researchers to determine to what extent the additional overhead could cause the RTTs to exceed client timeouts. The effect on the number of query retries may be relevant, because retries produce extra network traffic and load.

12 Acknowledgements

We would like to thank various people (in no particular order) for helping us with guiding the research, writing the report, and developing code:

- Wim Biemolt (SURF), for sharing his thoughts on conducting the research, proofreading our documents, and connecting us with peers at SURF.
- Ronald van der Pol (SURF), for helping us with development of the eBPF program.
- Hajo Haaijer, for providing us with space, internet connection and power for the server the experiments were performed on.
- Hans van der Made (Utrecht University), for proofreading our documents, and sharing his thoughts regarding our experiments.

[20]The port number is checked in the eBPF program, but the check can be omitted.

References

[1] Akamai. *Whitepaper: DNS Reflection, Amplification, & DNS Water-torture.* URL: https://www.akamai.com/site/en/documents/research-paper/dns-reflection-vs-dns-mirai-technical-publication.pdf.

[2] Alexei Starovoitov, Joe Stringer, and Michael Kerrisk. *eBPF Syscall — The Linux Kernel documentation.* URL: https://docs.kernel.org/userspace-api/ebpf/syscall.html (visited on 01/30/2024).

[3] *Benchmarking Methodology for Network Interconnect Devices.* Mar. 1999. DOI: 10.17487/RFC2544. URL: https://www.rfc-editor.org/info/rfc2544.

[4] Wim Biemolt. *Beveiliging van een Netwerk: SURFcert.* May 9, 2020. URL: https://communities.surf.nl/netwerkinfrastructuur/artikel/beveiliging-van-een-netwerk-surfcert (visited on 01/10/2024).

[5] Jonas Bushart and Christian Rossow. "DNS Unchained: Amplified Application-Layer DoS Attacks Against DNS Authoritatives". In: *Research in Attacks, Intrusions, and Defenses.* Ed. by Michael Bailey et al. Cham: Springer International Publishing, 2018, pp. 139–160. ISBN: 978-3-030-00470-5.

[6] Cilium. *Program Types — Cilium 1.16.0-dev documentation.* Documentation. URL: https://docs.cilium.io/en/latest/bpf/progtypes/ (visited on 01/30/2024).

[7] Kernel Development community. *BPF_MAP_TYPE_BLOOM_FILTER.* URL: https://docs.kernel.org/bpf/map_bloom_filter.html (visited on 01/30/2024).

[8] *CSIRTs geregistreerd bij / registered with SURFcert - CSIRTs - SURF Wiki — wiki.surfnet.nl.* URL: https://wiki.surfnet.nl/display/CSIRTs (visited on 01/10/2024).

[9] Bastien Dhiver. *A story about AF_XDP, network namespaces and a cookie.* July 18, 2022. URL: https://blog.cloudflare.com/a-story-about-af-xdp-network-namespaces-and-a-cookie/ (visited on 02/07/2024).

[10] *Domain names - implementation and specification.* Nov. 1987. DOI: 10.17487/RFC1035. URL: https://www.rfc-editor.org/info/rfc1035.

[11] Robert Elz and Randy Bush. *Clarifications to the DNS Specification.* RFC 2181. July 1997. DOI: 10.17487/RFC2181. URL: https://www.rfc-editor.org/info/rfc2181.

[12] ETSI. *Network Functions Virtualisation (NFV); Architectural Framework.* Dec. 2014. URL: https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf (visited on 01/31/2024).

[13] ETSI. *Network Functions Virtualisation (NFV); Use Cases.* English. Mar. 2021. URL: https://www.etsi.org/deliver/etsi_gr/NFV/001_099/001/01.03.01_60/gr_NFV001v010301p.pdf (visited on 01/31/2023).

[14] Arthur Fabre. *L4Drop: XDP DDoS Mitigations.* Nov. 28, 2018. URL: https://blog.cloudflare.com/l4drop-xdp-ebpf-based-ddos-mitigations/ (visited on 01/11/2024).

[15] Lei Fang et al. "A Comprehensive Analysis of DDoS attacks based on DNS". In: *Journal of Physics: Conference Series* 2024 (Sept. 2021), p. 012027. DOI: 10.1088/1742-6596/2024/1/012027.

[16] R. (Miek) Gieben and Matthijs Mekking. *Authenticated Denial of Existence in the DNS.* RFC 7129. Feb. 2014. DOI: 10.17487/RFC7129. URL: https://www.rfc-editor.org/info/rfc7129.

[17] Keita Hasegawa, Daishi Kondo, and Hideki Tode. "FQDN-Based Whitelist Filter on a DNS Cache Server Against the DNS Water Torture Attack". In: *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM).* 2021, pp. 628–632.

[18] Keita Hasegawa et al. “Collaborative Defense Framework Using FQDN-Based Allowlist Filter Against DNS Water Torture Attack”. In: *IEEE Transactions on Network and Service Management* 20.4 (2023), pp. 3968–3983. DOI: 10.1109/TNSM.2023.3277880.

[19] Paul E. Hoffman and Patrick McManus. *DNS Queries over HTTPS (DoH)*. RFC 8484. Oct. 2018. DOI: 10.17487/RFC8484. URL: https://www.rfc-editor.org/info/rfc8484.

[20] Zi Hu et al. *Specification for DNS over Transport Layer Security (TLS)*. RFC 7858. May 2016. DOI: 10.17487/RFC7858. URL: https://www.rfc-editor.org/info/rfc7858.

[21] Jeff Dileo. *eBPF Adventures: Fiddling with the Linux Kernel and Unix Domain Sockets*. Mar. 2019. URL: https://research.nccgroup.com/2019/03/25/ebpf-adventures-fiddling-with-the-linux-kernel-and-unix-domain-sockets/ (visited on 02/07/2024).

[22] Jim Keniston, Prasanna S Panchamukhi, and Masami Hiramatsu. *Kernel Probes (Kprobes) — The Linux Kernel documentation*. Documentation. URL: https://docs.kernel.org/trace/kprobes.html (visited on 01/30/2024).

[23] Joanne Koong. *[PATCH v6 bpf-next 0/5] Implement bloom filter map*. Oct. 2021. URL: https://lwn.net/Articles/874239/ (visited on 01/30/2024).

[24] Alexander Kaplan and Shir Landau Feibish. “DNS Water Torture Detection in the Data Plane”. In: *Proceedings of the SIGCOMM ’21 Poster and Demo Sessions*. SIGCOMM ’21. Virtual Event: Association for Computing Machinery, 2021, pp. 24–26. ISBN: 9781450386296. DOI: 10.1145/3472716.3472854. URL: https://doi.org/10.1145/3472716.3472854.

[25] Nikos Kostopoulos, Dimitris Kalogeras, and Vasilis Maglaris. “Leveraging on the XDP Framework for the Efficient Mitigation of Water Torture Attacks within Authoritative DNS Servers”. In: *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. 2020, pp. 287–291. DOI: 10.1109/NetSoft48620.2020.9165454.

[26] Nikos Kostopoulos et al. “Mitigation of DNS Water Torture Attacks within the Data Plane via XDP-Based Naive Bayes Classifiers”. In: *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*. 2021, pp. 133–139. DOI: 10.1109/CloudNet53349.2021.9657122.

[27] Laura Bauman. *Harnessing the eBPF Verifier*. en-US. Blog. Jan. 2023. URL: https://blog.trailofbits.com/2023/01/19/ebpf-verifier-harness/ (visited on 01/30/2024).

[28] David C Lawrence, Warren ”Ace” Kumari, and Puneet Sood. *Serving Stale Data to Improve DNS Resiliency*. RFC 8767. Mar. 2020. DOI: 10.17487/RFC8767. URL: https://www.rfc-editor.org/info/rfc8767.

[29] Edward P. Lewis. *The Role of Wildcards in the Domain Name System*. RFC 4592. July 2006. DOI: 10.17487/RFC4592. URL: https://www.rfc-editor.org/info/rfc4592.

[30] Edward P. Lewis and Alfred Hoenes. *DNS Zone Transfer Protocol (AXFR)*. RFC 5936. June 2010. DOI: 10.17487/RFC5936. URL: https://www.rfc-editor.org/info/rfc5936.

[31] Linus Torvalds. *Linux 5.16*. Sept. 2022. URL: https://lore.kernel.org/lkml/CAHk-=wgUkBrUVhjixy4wvrUhPbW-DTgtQubJWVOoLW=OOwRKMA@mail.gmail.com/T/ (visited on 01/30/2024).

[32] Xi Luo et al. “A Large Scale Analysis of DNS Water Torture Attack”. In: *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence*. CSAI ’18. Shenzhen, China: Association for Computing Machinery, 2018, pp. 168–173. ISBN: 9781450366069. DOI: 10.1145/3297156.3297272. URL: https://doi-org.proxy.uba.uva.nl/10.1145/3297156.3297272.

[33] Jose E. Marchesi. *[PATCH V6 00/11] eBPF support for GCC*. English. Aug. 2019. URL: https://gcc.gnu.org/legacy-ml/gcc-patches/2019-08/msg01987.html (visited on 01/30/2024).

[34] Sebastiano Miano et al. "A Framework for eBPF-Based Network Functions in an Era of Microservices". In: *IEEE Transactions on Network and Service Management* 18.1 (2021), pp. 133–151. DOI: 10.1109/TNSM.2021.3055676.

[35] Stichting Nationale Beheersorganisatie Internet Providers. *Anti DDoS protection service description*. Jan. 2023.

[36] Quentin Monnet. *Features of bpftool: the thread of tips and examples to work with eBPF objects*. English. Blog. Sept. 2021. URL: https://qmonnet.github.io/whirl-offload/2021/09/23/bpftool-features-thread/ (visited on 01/30/2024).

[37] Liz Rice. *Learning EBPF*. O'Reilly Media, 2023. ISBN: 9781098135096. URL: https://books.google.nl/books?id=dW-yEAAAQBAJ.

[38] SIDN. *Disable advanced features of NSEC3 security technology for DNSSEC | Cybersecurity | SIDN*. English. Nov. 2021. URL: https://www.sidn.nl/en/news-and-blogs/disable-advanced-features-of-nsec3-security-technology-for-dnssec (visited on 01/30/2024).

[39] Paul A. Vixie. *A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)*. RFC 1996. Aug. 1996. DOI: 10.17487/RFC1996. URL: https://www.rfc-editor.org/info/rfc1996.

Acronyms

BGP	border gateway protocol
cBPF	classic Berkeley packet filter
DDoS	distributed denial of service
DNS	domain name system
DNS-OARC	DNS Operations, Analysis, and Research Center
DPDK	Data Plane Development Kit
DPI	deep packet inspection
DUT	device under test
eBPF	extended Berkeley packet filter
ETSI	European telecommunications standards institute
FQDN	fully qualified domain name
L4-based TE	layer 4-based traffic engineering
L7-based DDoS attack	layer 7-based DDoS attack
NFV	network function virtualisation
NIC	network interface controller
NREN	national research and education network
PoC	proof of concept
QPS	questions per seccond
RR	resource record
RRset	resource record set
RTT	round-trip time
TC	traffic control
TCP	transmission control protocol
UDP	user datagram protocol
VM	virtual machine
VNF	virtual network function
VPP	vector packet processing
XDP	eXpress data path