Dynamically Selecting Defenses to DDoS for DNS (extended)

USC/ISI Technical Report ISI-TR-736

December 2019

ASM Rizvi
USC/Information Sciences Institute
asmrizvi@usc.edu

John Heidemann
USC/Information Sciences Institute
johnh@isi.edu

Jelena Mirkovic
USC/Information Sciences Institute
mirkovic@isi.edu

***Abstract*—Distributed Denial-of-Service (DDoS) attacks exhaust resources, leaving a server unavailable to legitimate clients. The Domain Name System (DNS) is frequently the target of DDoS attacks, and its connectionless communication makes it an easy target for spoofing attacks. A large body of prior work has focused on specific filters or anti-spoofing techniques, but DDoS threats continue to grow, augmented by the addition of millions of Internet-of-Things (IoT) devices. We propose two approaches to DDoS-defense: first, we propose having a *library* of defensive filters ready, each applicable to different attack types and with different levels of selectivity. Second, we suggest *automatically selecting* the best defense mechanism at attack start, and re-evaluating that choice during the attack to account for polymorphic attacks. While commercial services deploy automatic defenses today, there are no public descriptions of how they work—our contribution is to document one automated approach, and to show the importance of multiple types of defenses. We evaluate our approach against captured DDoS attacks against a root DNS server, using analysis and testbed experimentation with real DNS servers. Our automated system can detect attack events within 15 s, and choose the best defense within 40 s. We show that we can reduce 23% CPU usage and 63% egress network bandwidth with the same memory consumption and with little collateral damage.**

I. Introduction

Even with more than a decade of research, Distributed-Denial-of-Service (DDoS) attacks are a continuing and growing problem [26], [49]. The memcached attack in February 2018 against Github shows attacks now exceed 1.35 Tb/s [50]. Increasing deployment of Internet-of-Things (IoT) devices means attacks no longer need to exploit amplification, but can attack targets directly [55]. Automated tools make DDoS attacks easier to invoke than ever before [66], and DDoS-As-a-Service [39], [58] allows unsophisticated users to purchase and execute DDoS attacks for as little as $10 per month [33].

The main consequence of a DDoS attack is to exhaust resources of the victim [36], [44]. As servers or their access networks spend more time answering malicious queries, queries from legitimate may be dropped. Different types of attacks may exhaust different resources. We identify four resources that may be exhausted during a DDoS attack: ingress and egress network bandwidth, physical memory and CPU (§III-A2).

This paper focuses on DDoS against the Domain Name System (DNS) and DNS roots. DNS is particularly challenging because most DNS requests use UDP, making spoofing attacks difficult to counter. Moreover, the DNS root service is a high profile, critical service, and so it has been subject to repeated DDoS attacks [46], [56], [57], [63]. Yet defenses are vital, since a DNS outage can prevent users from reaching an otherwise active service [54].

A number of DDoS defense techniques have been proposed, particularly for DNS, including spoof suppression [3], [4], [29], [61], [64], filtering traffic [7], [16], [18], [37], [68], and solutions to specific attacks [8], [9], [11], [37], [70]. To complement such protocol-level defenses, operators provide overcapacity and use anycast to distribute load [14], [47], [52].

Several third-party DDoS Protection Services have arisen to mitigate DDoS attacks [21], [31] including Akamai, Cloudflare, CenturyLink, and F5 networks. These companies normally shift the traffic into their scrubbing system, where they both filter attack traffic and have over-provisioned network capacity and servers. However, DDoS-mitigation services can be expensive, vulnerable [30] and the need to forward traffic through a third party raises privacy concerns [24]. While DDoS-mitigation services have sophisticated defenses, their details are proprietary.

Although large providers have automated defenses and DDoS-mitigation services today [20], [21], [48], and there are many specific defenses in published literature, to our knowledge our paper is the first to describe an automated defense in detail with multiple filters, and to show the importance of matching appropriate filters to attacks.

The contributions of this paper are to show that a *library* of defensive filters is necessary, *automatic selection* can identify good filters, and that there is value in *response-code filtering* as a last line of defense. A library of filters (§III-C) is needed because, while no single method protects against all attack types, different defenses help in different circumstances (§VI-D2), and some defenses may have higher levels of

collateral damage (legitimate traffic that is misclassified and discarded). Automatic selection (§III-B) from a range of filters (§III-C), and the ability to shift defenses during an attack is important to handle polymorphic attacks. Our system detects an attack, evaluates the attack pattern, selects the best filter, deploys it, and continuously evaluates its performance in an automated and quick manner. Finally, we introduce a new filtering approach for DNS based on the response code of the reply (§III-C3)—since a common DDoS attack against DNS is to query random, non-existent names, we consider dropping all negative responses. While this filter will also drop many legitimate queries that happen to be for non-existent domains, it is still a useful last defense. Our approach also includes new implementations of two standard DNS DDoS filters: Query Name Blacklisting (§III-C1) and Source IP Whitelisting (§III-C2).

II. Related Work

DDoS attacks have been a problem for more than two decades, and many research and commercial defenses have been proposed. This section reviews only those solutions that are closely related to our approach.

DDoS exhausts resources which is acknowledged by previous studies [10], [32], [35], [60]. Akamai's kill-bots [32] and Microsoft's FastRoute system [19] rely on server load to get the potential attack signal. Another study highlights the CPU-resource problem in web applications [43]. These studies validate our choice to use resource consumption as the problem indicator.

Several previous studies have focused on specific attacks and defenses on DNS including amplification attacks [11], [37], TCP-SYN flood attacks [16], and volumetric attacks [70]. Each of these studies concentrate on a specific type of attack. In this paper, we show that only a single defense is insufficient to handle the variety of attacks that a service faces. As an example, while query-based blocking mitigates some volumetric attacks, it does not address SYN-flood attacks (§VI-A). Even for a single type of attack, one filter may or may not work depending on the attack characteristics. Attack specific filters show us the necessity of keeping multiple defenses.

Servers can deploy ingress or egress filtering. Ingress filters reject attack traffic with different methods, including hop-counts [29], [64], route traceback [59], path identifier [67], and client legitimacy based on network, transport and application layer information [61]. Keeping a whitelist or blacklist to distinguish legitimate or malicious traffic is also suggested in several studies and RFCs [7], [18], [40], [68]. These solutions and recommendations may work when sources are spoofed, provided upstream routers are not overloaded. However, IoT-based attacks do not require spoofing to reach high bitrates, and many attacks overload upstream routers before filtering. Filtering on DNS reputation helps prevent malware and spam when the attackers originate, repeatedly, from specific domains [1], [73]. DDoS attacks against DNS typically use varying or random names, so reputation does not apply. While each of these approaches addresses a specific threat, none of these solutions provides a complete solution, consistent with our motivation for a library of defenses. Our work describes an automated system that combines a library of ingress and egress defenses to handle a wide range of attacks overcoming the limitation of an attack specific solution.

Servers are recommended to filter egress traffic to avoid resource exhaustion [51] as replies can be larger than the queries. Some RFCs talk [2], [42] about filtering at client egress to avoid spoofing. Egress filters in DNS servers can protect the "spoofed" victims from amplification attacks [37]. Akamai's content delivery network uses DNS filtering to protect the rest of their network [21]. In this paper, we propose a new response-based egress filter which is a useful last level of defense. BIND can also limit the rate of NXDomain replies [28].

Many companies also provide solution for DDoS. Akamai mentions to use dynamic DNS filtering in their content delivery network [21]. Verizon has an automated system named Stonefish [5], [71], and Cloudflare also describes their DDoS mitigation architecture [17]. These ideas are proprietary, hence, we know little about their filter selection policies, detailed design, and performance. The DeDoS system automates replication of parts of applications during DDoS [13] . This defense is useful when an application can be divided into smaller units which is not the case for DNS. We provide a publicly available, automatic anti-DDoS system in this paper.

Lastly, we measure the performance of individual filters and our whole system on DNS server under real attack events (§VI). Some previous studies also report these attack events [15], [56]. The closest related work evaluates the performance of anycast against real DDoS events [46] in DNS.

III. System Design

Our goal is an automated system which can protect DNS servers from resource exhaustion (§III-A2) by selecting from multiple filters (§III-C). Our system needs to select the best filter quickly, and it should be robust to the changes in the attack pattern during an event (§III-B). We describe our prototype in §III-D.

A. Threat Model

Our threat model can be described from two standpoints—attackers and target. We next describe our threat model: distributed attackers attempt to exhaust resources at the target running a service like DNS.

1) Attackers: DDoS is a threat because attackers attempt to make a service at the target unavailable, often to extort money, disadvantage a competitor, or simply show their power. Attackers can directly make malicious traffic to a target or the attackers can hide themselves by spoofing their addresses leaving them intractable. DNS is mostly UDP traffic where source validation is not possible. Attackers can also compromise devices, particularly IoT devices, and make a distributed DoS.

DNS is particularly vulnerable to DDoS attacks, since it is an open service where attackers can use fixed, semi-random or

completely random query names. DNS replies are larger than the query size for which attackers use DNS to make reflection attacks.

2) Target: DDoS attacks with DNS can directly affect the DNS server, they can indirectly target another computer by spoofing its addresses and innundating it with traffic returned from DNS servers. In this paper, we focus on direct attacks on the DNS server.

The main threat of a DDoS attack is to exhaust some resource at the target victim. As an example, in 2015 DDoS attacks on the root DNS some operators could reply to all queries, but some failed to receive queries or could not respond to them typically because of bandwidth limitations [41], [46]. Our system watches for resource exhaustion to detect attacks and evaluate defenses, since different attacks threaten different resources:

Ingress network bandwidth: Volumetric attacks like UDP flooding can saturate the ingress network bandwidth [25]. The memcached attack did not even send DNS queries, but exhausted channel capacity [50]. If a root server has a capacity of I Gb/s, then attack traffic of rate I_A (where $I_A > I$) will result in user loss of approximately $(I_A - I)/I_A$ legitimate queries.

Physical memory: Several types of attacks target server memory, forcing kernels to buffer IP fragments [34], [75], or TCP connections from a TCP-SYN flood attack [16]. Today's operating systems are generally hardened to these attacks and drop partially-complete information when resources are limited.

CPU usage: CPU usage increases in proportion to query rates, and while non-DNS traffic may be filtered at a firewall, DNS queries require some application-level processing. Query processing can incur an asymmetric cost; they are cheap for zombies to generate but much more expensive for servers to detect and discard or handle.

Egress network bandwidth: DNS replies are always larger than the requests, and in DNS amplification attacks, an ANY request with signatures can draw an reply that is 100x bigger [49], [62]. As a result, outgoing bandwidth may be a bottleneck even if incoming bandwidth is not.

B. Our System and Its Components

Our automated system has three components: automated problem detection, filter management, and parameter selection.

1) Automated problem detection: Problem detection must collect information about resource status, then recognize when a resource is threatened. Operators care about DDoS when they have degraded performance by the exhausted resources, as listed in §III-A2. We use `collectd` to collect status information from several resources (CPU, memory, bandwidth) every 10 s. Our system consults `collectd` in every 10 s to find an exhaustion.

We detect an attack when some resource exceeds some fraction T_s of capacity. We currently use T_s of 80% of maximum capacity. We then select and deploy a filter, as described next.

After a filter has been deployed, we monitor how much traffic the filter blocks. We declare the attack over when the filter is no longer dropping the usual amount of traffic but still keeping the threatened resource below T_s for at least 20 s.

2) Automated filter management: Automated filter selection has three goals. Our first goal is to select the best filter among several filters. Different attacks show different characteristics with changing their attack pattern during an ongoing attack event. Our system should react accordingly by recognizing the changing attack characteristics. Our second goal is to make a decision quickly. During an attack event, human decisions to deploy filters can be time consuming during which time regular traffic is impeded. In an automated system, as no human is involved, we can expect it to take decisions faster than human operators. Our third goal is to make the system flexible so that we can easily add new filters to handle new attacks without much changes in the decision module.

Filter selection: Our system finds the required parameter for each filter, and then it selects a filter from a set of filters whose parameters are known.

Each filter needs to learn different parameters for functioning (§III-B3). A filter is ready when it knows its required parameter. Our system automatically finds the required parameters. Then the filter selection system selects the possible best filter from the ready filters. Our system learns some parameters during the attack, and some other parameters before the attack, and it does not need to wait to learn all the parameters for all the filters to start filtering.

Our filter selection system is designed to maximize its ability to remove malicious traffic while minimizing any collateral damage. Collateral damage means the false rejection of good traffic. If multiple approaches have the same ability to reduce attack traffic, our system selects one approach based on the likelihood of minimizing collateral damage. We prefer the filter with lowest collateral damage, based on our experiments (§VI). Nevertheless, we can always change the priorities to select the filter and add new filters.

Run-time evaluation of filter effectiveness during attack: Our system reevaluates the deployed filter at run-time. If the deployed filter is not suitable for the ongoing attack, our system dynamically selects other filters.

To see whether the deployed filter is suitable for the attack, we continue to check the resource consumption once every 10 s during the attack, after deploying a filter. If the filter fails to keep the resource below the threshold for three consecutive checks, our system withdraws the ineffective filter and selects some other filter based on the above mentioned priority.

Evaluation of the inactive filters: In addition to on-line evaluation of the active filter, our system also concurrently computes parameters for other filters every 10 s, running in the background. Checking and refreshing parameters of the inactive filters allows filters to be quickly deployed if it is required.

To evaluate the usability of the inactive filters, we cannot use resource consumption as we get resources only for the deployed filter. We estimate how much traffic we could block

using each inactive filter by emulating its operation. We emulate how they would behave by running TCPdump on traffic (more on §III-D), to extract query name, source IP, and response. We then apply inactive filters to these elements and decide whether to block or accept (more about how to block or accept in §III-C). Here, inactive filters only count blocks or accepts using the application program but are not deployed to discard a packet in kernel .

If an inactive filter blocks more than a threshold (how much traffic a filter usually blocks as mentioned in §III-B1), our system considers to deploy the inactive filter again based on the usual priority. Otherwise, our system checks whether the filter requires any parameter updates (as described in §III-B3) and continue to check the usability once every 10 s.

3) Parameters selection: We learn some parameters during the attack, and others ahead-of-time. A filter is ready to be deployed when our system learns the parameter of that filter.

Our system can learn some parameters offline—even before the attack starts. As an example, we can build the source whitelist using normal traffic (more on §III-C2), and use that whitelist during the attack duration. We need to update such offline parameters after a certain duration so that they always remain up-to-date.

Parameters for some filters must be determined at run-time. For example, Query Blacklisting needs to determine attack's common query-name. Each filter has different ways to learn these parameters (described in §III-C1 and §III-C3). Our system needs to find these parameters quickly to deploy the right filter. It uses a time-bin of 100 ms to observe the traffic to learn the required parameters. As our system continuously evaluates the deployed and inactive filters, our approach is self-corrective—incorrect finding of a parameter will be corrected quickly in an automated manner.

When our system looks for the run-time parameters, it deploys Source Whitelisting (§III-C2) as the default filter. This default filter learns the parameter before the attack starts and does not need to evaluate the run-time traffic. Quickly deploying a default filter provides some protection while parameters are computed.

C. Filters

Currently, we have three filters: Query Name Blacklisting (§III-C1), Source IP Whitelisting (§III-C2), Response Blacklisting (§III-C3). Response Blacklisting is new, and to the best of our knowledge the others have not been formally described. We next describe their design, then validate them in §V.

1) Query Name Blacklisting: Often attacks consist of many queries, all with similar domain name (see all the attacks listed in Table I). To stop this class of attack, we can block that *common query name*, and give access to other query names. Although query name filtering is computationally expensive because it requires looking into payload of packets, it drops the queries before making the replies. Due to this early discard of queries, Query Blacklisting can be used to reduce memory, CPU usage, and egress bandwidth.

Most of the time, Query Blacklisting can block a significant portion of the attack traffic with lowest collateral damage (more on §VI-A), our system gives it highest priority.

Query Blacklisting must determine the query to remove. We look at 100 ms of traffic and look for the unusually common queries. As attackers sometimes use randomized prefix names, we distinguish two types of attack queries—fully-matched query names and only suffix-matched query names (Table I). We discard the first domain name component from the left, and consider the rest as the suffix name.

We identify when common queries are present with thresholds for full and suffix-matched queries. If any full name is 100 times more than the second highest full query name, and if any suffix is 30 times more than the second highest suffix name, we consider that full name or suffix name as the common query name. Since matching names is computationally expensive, we currently only match one full name or suffix at a time, although in principle we could match several. We find that the CPU usage increases around 15% when we block four query names compared to blocking only one query. We use two different thresholds for full name and suffix name. Suffix names sometimes only have TLDs (e.g. .com or .net) for which we have a high second-most frequent suffix name. Hence, we use a lower threshold for suffix name so that we can get the common suffix name properly. We also consider frequency comparison for randomized suffix name and a fixed prefix name. We have not implemented this option because it has not occured in attacks we have seen.

Adversary: An attacker can defeat Query Blacklisting by randomizing all components of the query. For such an attack, we would automatically select a different filter for defense.

2) Source IP Whitelisting: We consider sources that we have observed prior to attack as known good sources. Since DDoS events are rare, we assume we will capture only "normal" traffic, and include the good sources that are not part of the attack.

For Source Whitelisting, we identify known-good sources by IP address and allow their traffic through, then drop traffic from unknown sources. This filter protects against attacks using spoofed source addresses, and attacks from compromised devices (such as IoT), since DNS traffic typically only comes from a relatively few recursive resolvers. However, depending on how we learn the source addresses, the whitelist may be incomplete or too large. As Source Whitelisting can drop the packets before making the reply, it can potentially reduce memory, CPU and egress bandwidth.

We build the source whitelist offline from service users for some period before the attack starts. We rebuild the source whitelist periodically to identify new sources. Two design questions are: how long a period should we consider to build the whitelist, and how often should we rebuild it. We use 120 minutes of normal traffic to build the whitelist and we build the whitelist once in a day. We explore these questions more in §V.

Adversary: Source Whitelisting can be defeated by a prepared adversary. They might send low-rate queries from

attacking machines ahead of time. We can counter this kind of adversary by trying whitelists from different times, and alerting when a new whitelist shows a large increase of new addresses. Source whitelisting can also fail if the whitelist omits some legitimate clients. We evaluate these effects in §VI-B.

3) Response Blacklisting: DNS replies can be 100x larger than the query size, stressing egress network bandwidth. Some attacks use many queries, all generating the same reply (usually "non-existent domain") We can drop the replies with that response, and pass the replies with other responses reducing egress traffic. This response filtering is useful when the attack cannot be blocked by other filters—large reflection attack from whitelisted sources with completely random query names.

Response Blacklisting is very effective when an attacker floods DNS with many random queries, all of which will fail or succeed, but it is a very imprecise tool (§VI-C), since *normal* traffic often has the same replies (negative replies from Google Chrome's random queries to probe for captive portals [72], or because users mistype URLs). Hence, our system uses Response Blacklisting as a defense of "last resort".

Filtering legitimate packets may cause a legitimate resolver to try again, potentially aggravating an ongoing DoS attack [47]. However, attackers do not typically retry queries, and retries from legitimate clients normally make a small portion of total traffic compared to the huge attack traffic. Also, filtering frees resources at the server end which gives more capacity for the legitimate requests.

We need to know what response should we filter and when. We evaluate the normal and attack traffic characteristics to answer this question. Among the possible DNS responses [45], only two responses are common—response code 0 (no error) and response code 3 (non-existent domain). We get ∼40% response 0 and ∼60% response 3 in the normal DNS replies. During attack, we get significant variations in response ratio. If any response is more than 15% greater than its usual 40%-60% ratio, we consider that response to filter.

Though different attacks make different changes in the response trend, we do not block responses with response 0 (no error). If server blocks response 0, then the legitimate query makers will not receive any response for their valid queries. Our system only selects Response Blacklisting when it needs to block response 3 (NXDomain).

Adversary: Attackers may send queries which have response 0 in the replies or different response for different queries. Response Blacklisting is not usable in these cases. Our automated system is designed to select other filters when a particular filter fails to work.

D. Implementation

We next provide the implementation details to deploy the system.

Resource information: We collect resource information with SNMP. Our implementation of §VI-D uses `collectd`. `Collectd` updates resource information in every 10 s, and our system checks `collectd` for exhaustion in every 10 s. If the system finds any resource exhaustion, it notifies the filter management module to start filtering.

Day	Attack start (UTC)	Duration (mins)	Attack query name
2015-11-30	06:50	150	www.NUMBER1.com www.NUMBER2.com
2015-12-01	05:10	63	www.NUMBERCHAR.com
2016-06-25	22:18	5	N/A (TCP-SYN flood attack)
2017-02-21	06:40	117	RANDOM.FIXED1.com\032 RANDOM.FIXED2.com\032 RANDOM.FIXED3.com\032
2017-03-06	04:43	331	RANDOM.FIXED4.com\032 RANDOM.FIXED5.com\032
2017-04-25	09:54	175	RANDOM.FIXED6.com\032

TABLE I: Common query names in various attack days

Automated decision process: The automated decision process starts filtering when any of the resources gets exhausted. Our system parses Tcpdump output to evaluate the traffic and starts filtering. We describe how the automated system takes decisions in §III-B.

Deploying filters: We deploy all the filters using IPtable rules. For Query Blacklisting, our automated system finds the common query name. It then sets a rule to search the DNS part of the packet using hex-string to match the common query name. If it finds a match, it drops that query. For Source IP Whitelisting, we use IPset to make a set of the whitelisted addresses. We use IPtable rules to give access only to the IPset listed addresses. We drop the queries from other sources. For Response Blacklisting, we check four bytes of the outgoing replies starting from the 28^{th} byte for UDP, and the 54^{th} byte for TCP to get the response code. We use the u32 option of IPtables to check the four bytes. If the response matches 3, we drop those replies. Our automated system deploys one filter at a time. When the traffic becomes normal or decision process changes filter, it flushes all the existing rules.

IV. Evaluation: Data Sources and Approach

We use RSSAC-002 data to find the attack events, and real B-root traces to evaluate our filters and the whole system.

A. RSSAC-002 Data to Find Attack Events

RSSAC-002 is publicly available operational data from DNS root operators that describes the ingress and egress traffic statistics at root servers [38]. Different root servers started to provide RSSAC data at different times, but currently all the roots provide the statistics including traffic volume, number of unique addresses, and response code volume on a daily basis. RSSAC data reflects attack characteristics—increase in the total number of queries and unique addresses or different response code volume.

We identify attacks by looking at RSSAC-002 data [23] from the available root letters starting from January 1, 2015 to February 1, 2018. Each root letter reports a number of statistics; to us the most relevant ones are queries per day and number of unique sources per day.

We compute mean and standard deviation of numbers of queries for each day over an entire month. We consider days where the number of queries is more than one standard deviation above the mean to be potential attack days. We compute the difference between the number of queries in a particular day and one standard deviation above the mean. If the sum of this difference over each root exceeds a threshold and if the potential attack day is common in at least three letters, we consider that as an attack day. After finding the attack days, we find the attack duration using query rates over the whole day using the real B-root capture (§IV-B). If the query rate is unusually high for a particular duration, we suspect that duration as the attack duration.

This approach finds large events, but it will miss small or brief events. Such small events are unlikely to exhaust resources and so they are not our focus. Also, our approach will miss the attacks that target only one or two letters. However, this simple procedure to find the attack events is for evaluation only. Our automated system identifies an attack when the actual resource gets exhausted.

Table I lists the attack events we consider. For privacy reasons, we publish only the anonymized query names. Different attacks last different durations. To get the confusion matrix offline, we need to distinguish the attack traffic. We use attack query name or TCP half-open connections (for SYN flood attack) to distinguish the attack traffic. Our offline evaluation uses the attack characteristics to see the individual performance of the filters.

B. B-Root Data

We evaluate the filters and our automated system with real-world attack data taken from B-Root operators which is also publicly available. These traces are full packet captures, with the IP addresses partially anonymized. These trace files capture all the ingress and egress traffic at B-root.

C. Metrics to Evaluate Defenses

We evaluate the success of DDoS filters using standard information-theoretic metrics. If we consider traffic from bad actors as our detection goal, some traffic will be passed through (accepted). A true accept of legitimate traffic is a *true negative* (TN), while false accepts are malicious traffic that accepted, a *false negative* (FN). For rejected traffic, legitimate traffic that is dropped is a false reject or *false positive* (FP), while dropped malicious traffic is a *true positive* (TP).

We evaluate methods using standard information theoretic terms, treating rejects as positives and passes as negative. Sensitivity $TP/(TP + FN)$, identifies how much attack traffic we find. Specificity $TN/(TN + FP)$, identifies the fraction of correctly passed legitimate traffic, and Accuracy $(TP+TN)/(TP+FP+TN+FN)$, shows how often we make the correct decision. Taken together, these metrics define the *confusion matrix*.

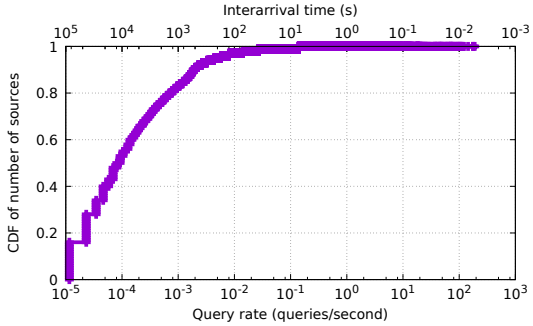


Fig. 1: CDF of source query rates, showing a wide range of rates. Data: 2015-11-29

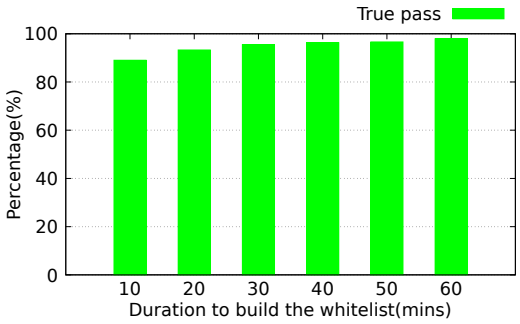


Fig. 2: Impact of the duration to build the whitelist

V. Validation of Filter Parameters

In this section, we validate several parameters that we selected in §III-C. Good parameters are important for an effective system.

A. Source Whitelisting: How Long to Observe?

In §III-C2, we used 120 minutes of traffic to build the whitelist which we validate here.

We want to observe long enough to get most legitimate sources, so we first consider how often each source makes queries. Figure 1 shows that sources place queries at many different rates with a long tail, with about 80% having inter-arrivals of 1000 s or more, while a few place thousands of queries per second. This wide range of rates is also visible in earlier studies [6], [65]. We are certain to capture all frequent queriers in our whitelist, as a relatively short observation period (120 minutes) provides a whitelist that covers the majority of queries. However, a short observation will miss the infrequent queriers. We next quantify how much queries and unique sources we can cover based on an observation duration.

We expect shorter observations for hitlists to observe only the most frequent carriers, hence, most legitimate queries. To evaluate how long we must observe to get the most queries, we examine normal traffic of the day 2015-11-29 and build whitelists from data lasting from 10 minutes to 60 minutes starting from 00:00:00 UTC. We evaluate how many future queries these whitelists can identify from 02:00:00 UTC to

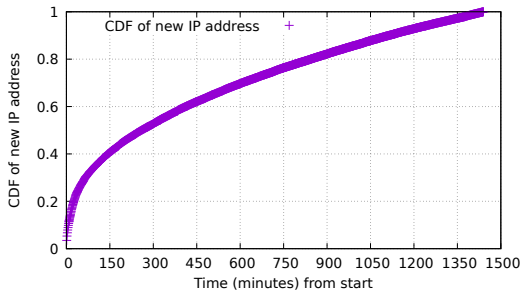


Fig. 3: CDF of new IP address with time

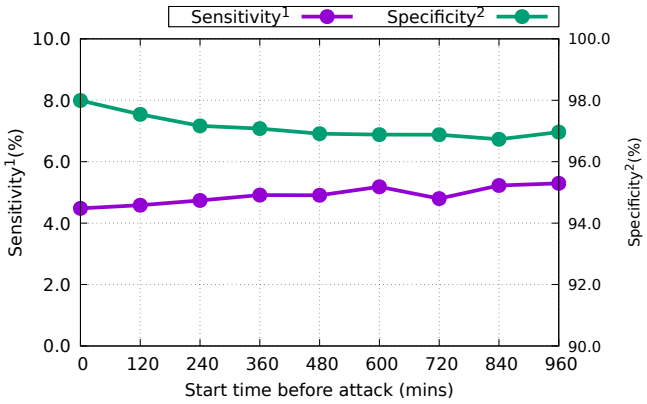


Fig. 4: Impacts of the whitelist creation time based on confusion matrix for 2017-03-06 event considering all queries

23:59:59 UTC traffic. Figure 2 shows only 20 minutes of traffic can cover over 90% of future queries. Hence, only a small duration of traffic can cover the sources which will make most of the future queries of the day.

Though small duration *covers* frequent queriers which make most queries, we still *miss* many non-frequent queriers. From Figure 3, we can see that within a ∼1400 minutes time-frame, we get 50% of unique sources within ∼250 minutes. This implies that even if we create the whitelist with 250 minutes of traffic, we still miss 50% of the future unique legitimate sources (not the number of queries). However, these infrequent sources send very few queries.

Finally, we choose to take 120 minutes of traffic to build the whitelist. According to Figure 3, we expect to miss around 60% of the unique sources of the day (though attacks do not persist the whole day). We choose this value because this value is sufficient to cover most of the legitimate future queries (Figure 2). Also, non-frequent queriers mostly have well-configured cache, and most of the time they get their query response from their own cache.

B. Source IP Whitelisting: How Often to Build?

We next consider how often we need to build a source IP whitelist, confirming that once a day is sufficient (§III-C2).

How often the whitelist is built can influence its success. We must build the list frequently enough that it reflects current queriers, yet list generation has some cost so we cannot build it continuously. To evaluate how list age changes accuracy, we create lists from 120 minutes of data starting 0 to 960 minutes before the attack. We compute the confusion matrix (as discussed in §IV-C) to see how much malicious traffic we can block along with the collateral damage for 2017-03-06 event. From Figure 4, we can see little change in the confusion matrix even if we build the whitelist 960 minutes before the attack event. Sensitivity and specificity values differ by 1% to 2% based on the whitelist creation time.

Since time of list construction has relatively little effect on effectiveness, we build the whitelist once a day.

VI. Evaluation

We next look at the success of each filter method at blocking DDoS and risk of collateral damage, then the latency of filter selecting in our system as a whole.

A. Performance Evaluation of Query Blacklisting

How effective is Query Blacklisting? Query Blacklisting is an ideal filter when attackers attack with a single name or pattern. To evaluate its effectiveness we looked at all 6 Root DNS events from 2015 to 2017 in Table II. We report each event and its confusion matrix after Query Blacklisting. As can be seen, Query Blacklisting detects all or nearly all attack traffic (sensitivity 95% or more) and affects *only* attack traffic (specificity 100% and accuracy 96% or more). However, the challenge is we must determine what the attack query is to deploy a filter. We next look at response time.

Effects of response time: Our system needs some time, RT to find the common query name when the attack begins or if the attacker changes the query. Within RT time, all the legitimate queries are true accepts (true negative) and malicious queries are false accepts (false negative). After this time, assuming we block the actual attack query, we consider all the legitimate queries as true accepts (true negative) and all the malicious queries as true rejects (true positive). As we are blocking a query name, legitimate queries with that query name are false rejects (false positive). Attacks never use an existing query since it would be satisfied by caches. Hence, we do not get false rejects (false positive) by blocking the popular queries.

Our automated system should have a small RT time so that it makes small number of false accepts (false negative). In §III-B3, we set to observe 100 ms of traffic to learn the online parameters. That is why RT value is at least 100 ms. However, selecting a wrong query name causes false acceptance of malicious traffic for a longer duration as our system holds one decision for sometime. We observe 0.16 s to 33 s duration (Table II) to find the right common query name. This small duration allows a small number of false accepts (false negative).

Effects of blocking single query name: We block the most common query using IPtables. DNS queries sometimes use mixed case [12], and although we block the most common

Day	*RT (s)*	Sensitivity (%)	Specificity (%)	Accuracy (%)	Incoming query rate (Gb/s)	Rate after filtering (Gb/s)
2015-11-30	0.16	**100.0**	**100.0**	**100.0**	0.17	**0.001**
2015-12-01	0.17	**100.0**	**100.0**	**100.0**	0.18	**0.002**
2016-06-25	NA	NA	NA	NA	0.67	NA
2017-02-21	32.14	**96.28**	**100.0**	**97.89**	0.09	**0.033**
2017-03-06	1.04	**95.14**	**100.0**	**96.58**	0.10	**0.032**
2017-04-25	0.30	**96.55**	**100.0**	**97.80**	0.08	**0.031**

TABLE II: Confusion matrix and ingress rate with blocking the most frequent query name in different DDoS events

form, others will pass through (false accepts or false negatives). In practice, we find that the most common form covers 95% of malicious traffic (Table II).

Effects on bottleneck resources: To confirm Query Blacklisting reduces resources, the last two columns of Table II shows the bitrate that the server will handle before and after filtering. Except 2016 event, it filters out a significant portion of the incoming traffic confirming our success in blocking most of the malicious traffic. As it has little collateral damage, we infer the right most column as the legitimate traffic bitrate.

B. Performance Evaluation of Source IP Whitelisting

How effective is Source IP Whitelisting? Source IP Whitelisting will work when attackers spoof random addresses, and fail when attack traffic comes from resolvers already in use. Table III shows the confusion matrix for our six events.

We see that whitelisting is very effective for the first two events (high sensitivity, specificity, and accuracy, all above 93%). For the 2016-06-25 event it stops the attack, but at the cost of also discarding legitimate traffic (high sensitivity, 98%, but moderate specificity of 37%). It is not effective for the three attacks in 2017 (fails to identify over 94% malicious users).

For the 2016 event, Source Whitelisting shows collateral damage—identifies only ∼37% legitimate users correctly. To know the reason, we find that three particular IP addresses made a good number of DNS UDP queries. These IP addresses used an empty query name, and they were not in the whitelist. We are not sure whether these IP addresses had any malicious intent or not. If we do not consider these three IP addresses, then we can get over 97% sensitivity value.

Source Whitelisting fails to find the malicious users in 2017 events. To find the reason, Table IV shows that over 66% attack IPs were common when we built the whitelist. We suspect attackers make these per-queries to test their attack or attackers make queries from regular resolvers. From Figure 4, we find that these attack IPs were present even if we build the whitelist 960 minutes before the attack (sensitivity value remains bad). Long observation of these IPs proves that the attackers were behind the resolvers. As a result, these IPs are in the whitelist and Source IP Whitelisting cannot filter them. We also find that the attack sources are not spoofed as the number of unique sources is significantly less than the number that we have for the spoofed cases (2015 and 2016 events). Evidence of non-spoofed attack addresses also supports our claim to have attackers behind the resolvers.

Effects on bottleneck resources: Similar to Query Blacklisting, the last two columns of Table III show the bitrate that the server will serve before and after filtering. We see that the bitrates are high without filtering (up to 0.67 Gb/s), but filtering is effective for the first three attacks (down to 0.012 Gb/s). Source Whitelisting is effective for the 2016 event which was not the case for Query Blacklisting. However, Source Whitelisting fails in the last three events. This suggests to us why we need multiple defenses.

Incoming query rate of Table III under-represents the intensity of the attack events due to the measurement error during the attack duration [46]. However, we believe that the percentage of the blocked queries will remain the same even if we could measure the actual intensity.

C. Performance Evaluation of Response Blacklisting

How effective is Response Blacklisting? Response Blacklisting can reduce prior attacks, but with considerable cost in collateral damage, making it a defense of "last resort". Again, we evaluate all 6 events with Response Blacklisting, showing the confusion matrix in Table V.

We see that Response Blacklisting does very well for the three events in 2017, blocking *all* attack traffic (100% sensitivity). It is so effective because these attacks used random suffixes, which allows legitimate queriers to pass their queries through recursive resolvers, defeating Source IP Whitelisting but making them selected by Response Blacklisting. However, as expected, we see significant collateral damage, with specificity from 42% to 46%—there are a lot of legitimate negative responses. Due to this collateral damage, we see impacts over accuracy.

When a client does not get a response from a server, depending on the implementation, it may continue to send queries to that unresponsive server or it may send queries to other servers [69]. Overall, clients will experience delay due to the unresponsive servers.

Effects on bottleneck resources: To confirm Response Blacklisting reduces egress bandwidth, we show the last two columns of Table V. We can see in the last three events without Response Blacklisting egress bandwidth is as high as 0.75 Gb/s, whereas, response filtering can reduce it to 0.12 Gb/s.

We can see the outgoing reply rate (Table V) is smaller than the incoming query rate (Table III) for the first two events, even though DNS replies are bigger than the query size. This

Day	Whitelist Construction (hours)?	Whitelist size (million)	Sensitivity (%)	Specificity (%)	Accuracy (%)	Incoming query rate (Gb/s)	Rate after filtering (Gb/s)
2015-11-30	-30	1.04	**99.68**	**94.70**	**99.64**	0.17	**0.002**
2015-12-01	-54	1.04	**99.69**	**93.72**	**99.61**	0.18	**0.003**
2016-06-25	-22	1.14	**98.13**	37.48	**97.41**	0.67	**0.012**
2017-02-21	-7	1.28	3.99	**97.46**	38.56	0.09	0.087
2017-03-06	-5	1.29	4.66	**97.32**	34.48	0.10	0.096
2017-04-25	-10	1.32	5.52	**97.51**	41.84	0.08	0.074

TABLE III: Confusion matrix with ingress rate of Source IP Whitelisting in different DDoS events

Day	No. of unique attack IPs	Common IPs when we make the whitelist	% in common
2017-02-21	53424	36699	68.70
2017-03-06	56905	37800	66.43
2017-04-25	52874	36234	68.53

TABLE IV: No. of attack IPs that are common when we build the whitelist in different 2017 DDoS events

is because DNS root servers deployed rate limiting [27] for the first two events.

D. Performance Evaluation of Our Automated System

We already see the performance of individual filtering approach with confusion matrix. Now the question comes, how does our automated system perform when combining multiple filters. To evaluate the performance, we make a controlled experiment in DeterLab testbed (§VI-D1). Using the testbed, we find answers about several research questions (§VI-D2).

1) Experimental setup: We replay traffic with LDplayer [74], replaying captured traffic through 22 clients to get the full original bitrate.

Our attacks require that attackers and legitimate users be on different IP addresses, but prior LDplayer maps all traffic to the same set of IPs. We modified LDplayer to insure that attack and legitimate traffic are on distinct IP addresses for attacks where that was the case. The 2016-05-25 event was a TCP-SYN flood and not a DNS-attack, so in that case we reproduce the attack with the hping3 tool [53].

The attack target is an emulated DNS root server. We implement it with BIND, using the LocalRoot method of providing root service [22]

Reproducing viable events: To show the effects of attacks that drive a production system to resource exhaustion requires many servers to attack and to emulate the service. It is also difficult to perfectly reproduce attacks since the stored traces are often unable to capture the entire attack because of limitations of the capture system. We therefore scale down the server capacity to match the stored traces. We measure the regular traffic resource consumption, and trigger a problem when the resource is double than the regular consumption.

Other testbed implementation: We replay the attack events and emulate a target server using the above mentioned description. For the rest of the implementation, we use the system mentioned in §III-D.

2) Evaluation of our automated system: Our system-level evaluation considers how often and how long it takes to select a good filter, and then how it adapts to polymorphic attacks.

Multiple filters and choosing a good one: We first explore how often and how long it takes for our system to find a good filter. To evaluate filter selection, we first determine the good filters for each event. Table VII describes how effective each filter is for each of our six events, based on off-line analysis (§VI).

We consider a filter "good" if it successfully blocks the malicious traffic (over 80% sensitivity) and accepts the legitimate traffic (over 80% specificity). "Fair" means if a filter successfully blocks malicious traffic, but fails to accept legitimate traffic (less than 80% specificity). "No" means if a filter is not feasible for an event or performs poorly (cannot block over 80% malicious traffic). If there are two "good" filters then the filter which blocks most with least collateral damage is considered as the best one (in bold). For the 2016-06-25 event, Source Whitelisting is best (in bold) as our system does not have any other option.

Table VII also shows whether our system can select the best filter or not. In all the cases, our system successfully converges to the best filter.

Finally, this table shows that *multiple filters are needed*—no single filter is "best" for all six scenario.

Latency to select a good filter: We measure the latency to select the best filter. Latency depends on two factors—latency to detect the problem and latency to find the required parameters (e.g. common query, response). In our system, we measure resources every 10 s and check for overload every 10 s, so we guarantee detection in 20 s, and in our six scenarios we validate that the detection takes 5 to 15 s (see Table VII). Reducing these two timers can give us faster detection at the cost of more measurement overhead.

After detection, parameter selection takes a few seconds because of our design to evaluate parameters every 100 ms. We see the best filter selection time is 5 to 16 s (from attack start) in all cases except the 2017-02-21 event. For the 2017-02-21 event, our system selects and deploys ineffective filters in about 7 s, then discovers that problem and deploys a good filter, taking about 40 s in total.

For the 2017-02-21 event, our system makes three incorrect choices before finding the best filter—Source Whitelisting (default filter), Response Blacklisting (ready before query filtering and as Source Whitelisting performs poorly) and

Day	Blocked Rcode	Sensitivity (%)	Specificity (%)	Accuracy (%)	Outgoing response rate (Gb/s)	Rate after filtering (Gb/s)
2015-11-30	0 (not feasible)	100.0	57.22	92.84	0.13	**0.01**
2015-12-01	0 (not feasible)	100.0	54.39	82.72	0.14	**0.02**
2016-06-25	NA	NA	NA	NA	NA	NA
2017-02-21	3	**100.0**	42.0	**77.18**	0.70	**0.13**
2017-03-06	3	**100.0**	43.68	**80.29**	0.75	**0.12**
2017-04-25	3	**100.0**	46.25	**77.15**	0.45	**0.10**

TABLE V: Confusion matrix with egress rate of Response Blacklisting in different DDoS events

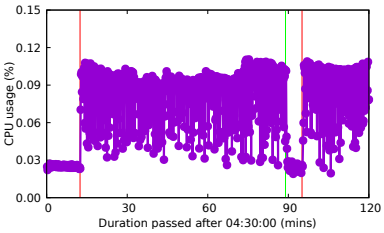
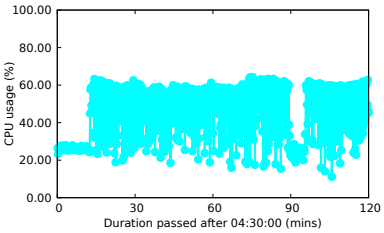
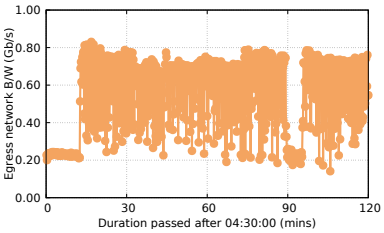
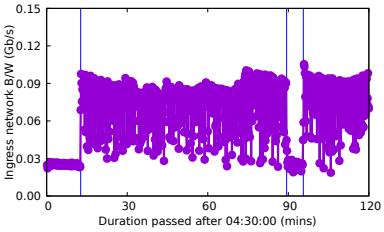
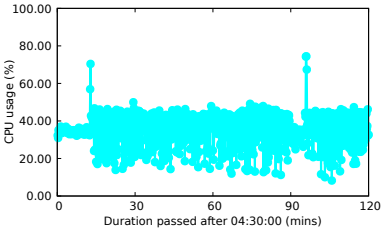
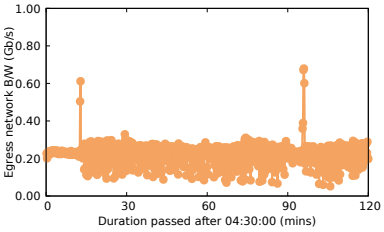
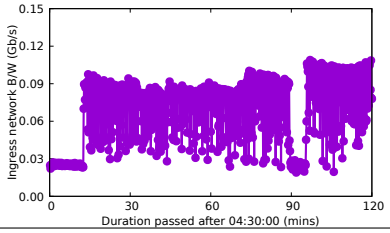
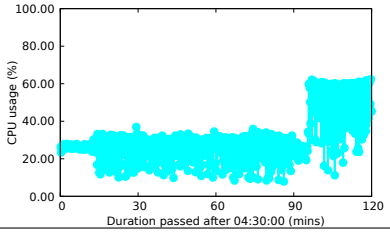
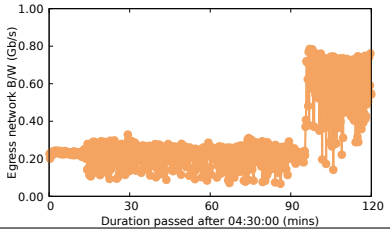
Case	Ingress network B/W (Gb/s)	CPU usage (%)	Egress network B/W (Gb/s)
Without system			
With system			
Non-adaptive			

TABLE VI: Resource consumption comparison for 2017-03-06 event: top row shows resources when we do not deploy the automated system, middle row shows resources when we deploy the automated system, bottom row shows when our system is not adaptive to the changes during an attack. We ignore memory graph as memory remains consistent over experiment.

Event	Source Whitelisting	Response Blacklisting	Query Blacklisting	Converge to the best?	Latency to detect attack (s) (from attack start)	Latency to select the best filter (s) (from attack start)	No. of selected filters before the best choice
2015-11-30	Good	No	**Good**	Yes	13.17	13.33	1
2015-12-01	Good	No	**Good**	Yes	5.05	5.22	1
2016-06-25	**Fair**	No	No	Yes	10.24	10.24	0
2017-02-21	No	Fair	**Good**	Yes	6.67	38.81	3
2017-03-06	No	Fair	**Good**	Yes	14.33	15.37	1
2017-04-25	No	Fair	**Good**	Yes	11.73	12.03	1

TABLE VII: Applicability of each filter in various DDoS events and our selection.

Query Blacklisting with wrong query name. In this event, during the first few seconds of attack, we get a common query name which is sporadic and non-malicious. Our system selects that query as the common query name. In the 2016-06-25 event, our system selects the default filter as the best one for which it does not select any intermediate filter. For the other events, our system chooses the default filter before the best one.

After filtering, does the service function? To answer this question, we examine if resources that the attack would exhaust are preserved after filtering. We consider the resource before and after deploying our system.

From Table VI, we can see the comparison of resource consumption for the 2017-03-06 event. CPU usage reduces from ∼62% (top row, second graph from left) to ∼48% (middle row, second graph from left) using our system. In case of egress network bandwidth, we can reduce the bandwidth from ∼0.8 Gb/s (top row, third graph from left) to ∼0.3 Gb/s (middle row, third graph from left). We see an initial spike before the reduced resource because of the latency to deploy the defense. We cannot reduce ingress network bandwidth as we have to give access before making any filtering. We do not find any effect over memory during the attack and after deploying the system (we ignore that in Table VI).

Defense overhead during problem detection: Our system reduces CPU during the attack by reducing attack traffic, but monitoring has some CPU overhead. We already show the cumulative outcome which is a great reduction in the CPU usage and egress bandwidth.

Monitoring in our current prototype system consumes about 10% of the CPU (the difference between the first two CPU graphs in Table VI considering the first few minutes of non-attack traffic). This overhead is high because our prototype has separate packet extraction and text-based analysis. We are currently working to reduce this overhead by directly counting events in packets without textual conversion and re-parsing.

Responding to polymorphic attacks: Our system periodically evaluates the traffic to address polymorphic attacks that change attack methods during the event. We next look at the 2017-03-06 event to see how our system copes with changing attacks.

The top-leftmost graph of Table VI shows the polymorphic nature of 2017-03-06 event—attack starts (first red line), pause (green line), and then starts again with a new query name (last red line). From the middle row-leftmost graph of Table VI, we can see that our system deploys the best filter within 16 s (first blue line), keeps the best filter until a temporary stop in attack at ∼89 minutes, reacts accordingly to stop filtering within 26 s (middle blue line), and deploys the best filter within 30 s when a different attack starts again (last blue line). This shows our system is adaptive to the polymorphic attack events.

Now, the question comes what will happen if our system is not adaptive. To answer this question, we deliberately disable our system from being adaptive after choosing the initial best filter. We can see the impact from the bottom row of Table VI. The attackers change its common query after ∼95 minutes. As a result, the non-adaptive system gets an increased CPU and egress network bandwidth. Our adaptive system can detect this change, and react accordingly which is shown in the middle row of Table VI.

VII. Conclusion

This paper provides the first in-depth design of an automated system to mitigate DDoS in DNS. Our system detects the resource exhaustion problem, selects the best defense from a library of defenses, and deploys the defense automatically. Our system is adaptive to the polymorphic attack events which change attack pattern during an ongoing attack event. We show one defense is not sufficient to handle all the attack events by showing individual defense performance against several real DDoS events captured in B-root. We test our automated system with real DDoS events. We show our system converges to the best defense within a small duration and is adaptive to the polymorphic attack events. Blocking malicious traffic frees resources which makes the server available to more legitimate users. Our system helps to mitigate some DDoS attacks. While we cannot mitigate pure volumetric attacks that completely overwhelm ingress links, our approach improves reliance to medium attacks.

Acknowledgments

This work is partially supported by the project "CICI: RSARC: DDoS Defense In Depth for DNS" (NSF OAC-1739034). This work is partially supported by the DHS HSARPA Cyber Security Division via contract number HSHQDC-17-R-B0004-TTA.02-0006-I (PAADDoS), in collaboration with NWO.

References

[1] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. Building a dynamic reputation system for DNS. In *USENIX security symposium*, pages 273–290, 2010.

[2] BCP-38. Main page. http://www.bcp38.info/index.php/Main_Page, 2018. [Online; accessed 7-May-2018].

[3] Robert Beverly and Steven Bauer. The spoofer project: Inferring the extent of source address filtering on the Internet. In *USENIX Sruti*, volume 5, pages 53–59, 2005.

[4] Robert Beverly, Arthur Berger, Young Hyun, et al. Understanding the efficacy of deployed internet source address validation filtering. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pages 356–369. ACM, 2009.

[5] Jesse Blazina. Stonefish—automating DDoS mitigation at the edge. https://medium.com/@verizondigital/stonefish-automating-ddos-mitigation-at-the-edge-6a2650aeb6af, 2019. [Online; accessed 30-May-2019].

[6] Sebastian Castro, Duane Wessels, Marina Fomenkov, and Kimberly Claffy. A day at the root of the Internet. *ACM SIGCOMM Computer Communication Review*, 38(5):41–46, 2008.

[7] Eric Y Chen and Mistutaka Itoh. A whitelist approach to protect SIP servers from flooding attacks. In *Communications Quality and Reliability (CQR), 2010 IEEE International Workshop Technical Committee on*, pages 1–6. IEEE, 2010.

[8] Yu Chen and Kai Hwang. Collaborative detection and filtering of shrew DDoS attacks using spectral analysis. *Journal of Parallel and Distributed Computing*, 66(9):1137–1151, 2006.

[9] Yu Chen, Kai Hwang, and Yu-Kwong Kwok. Filtering of shrew DDoS attacks in frequency domain. In *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*, pages 8–pp. IEEE, 2005.

[10] Michael Cobb. Using resource allocation management to prevent DoS and other attacks. https://www.computerweekly.com/tip/Using-resource-allocation-management-to-prevent-DoS-and-other-attacks/, 2010. [Online; accessed 4-Feb-2019].

[11] Jakub Czyz, Michael Kallitsis, Manaf Gharaibeh, Christos Papadopoulos, Michael Bailey, and Manish Karir. Taming the 800 pound gorilla: The rise and decline of NTP DDoS attacks. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 435–448. ACM, 2014.

[12] David Dagon, Manos Antonakakis, Paul Vixie, Tatuya Jinmei, and Wenke Lee. Increased DNS forgery resistance through 0x20-bit encoding: SecURitY viA LeET QueRieS. In *ACM Conference on Computer and Communications Security*, Alexandria, VA, USA, October 2008. ACM.

[13] Henri Maxime Demoulin, Tavish Vaidya, Isaac Pedisich, Bob DiMaiolo, Jingyu Qian, Chirag Shah, Yuankai Zhang, Ang Chen, Andreas Haeberlen, Boon Thau Loo, et al. DeDoS: Defusing DoS with dispersion oriented software. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 712–722. ACM, 2018.

[14] John Dilley, Bruce Maggs, Jay Parikh, Harald Prokop, Ramesh Sitaraman, and Bill Weihl. Globally distributed content delivery. *IEEE Internet Computing*, 6(5):50–58, September 2002.

[15] DNS-OARC. Review and analysis of attack traffic against A-root and J-root on November 30 and December 1, 2015. In 24th DNS-OARC Workshop, 2016. Presentation.

[16] Wesley M Eddy. TCP SYN flooding attacks and common mitigations. 2007.

[17] Arthur Fabre. L4drop: Xdp DDoS mitigations. https://blog.cloudflare.com/l4drop-xdp-ebpf-based-ddos-mitigations/, 2018. [Online; accessed 01-Dec-2019].

[18] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. RFC 2267, May 2000. also BCP-38.

[19] Ashley Flavel, Pradeepkumar Mani, David Maltz, Nick Holt, Jie Liu, Yingying Chen, and Oleg Surmachev. Fastroute: A scalable load-aware anycast routing architecture for modern cdns. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 381–394, Oakland, CA, 2015. USENIX Association.

[20] Flexential. Automated DDoS protection. https://www.flexential.com/sites/default/files/Flexential-docs/flexential-ddos-data-sheet-0618.pdf, 2018. [Online; accessed 12-Feb-2019].

[21] David Gillman, Yin Lin, Bruce Maggs, and Ramesh K Sitaraman. Protecting websites from attack with secure delivery networks. *Computer*, 48(4):26–34, 2015.

[22] Wes Hardaker. LocalRoot: Serve Yourself. https://localroot.isi.edu/, 2018. [Online; accessed 11-Jan-2019].

[23] ICANN. RSSAC002 — advisory on measurements of the root server system. https://www.icann.org/resources/files/1209609-2014-11-20-en/, 2014. [Online; accessed 12-June-2019].

[24] Basileal Imana, Aleksandra Korolova, and John Heidemann. Enumerating privacy leaks in DNS data collected above the recursive. In *NDSS: DNS Privacy Workshop*, San Diego, California, USA, feb 2018.

[25] Imperva. Different attack description. https://www.imperva.com/docs/DS_Incapsula_The_Top_10_DDoS_Attack_Trends_ebook.pdf, 2015. [Online; accessed 19-Sept-2017].

[26] Akamai InfoSec. A look back at the DDoS trends of 2018. https://blogs.akamai.com/2019/01/a-look-back-at-the-ddos-trends-of-2018.html, 2017. [Online; accessed 31-May-2019].

[27] ISC. Using the response rate limiting feature. https://kb.isc.org/article/AA-00994/0/Using-the-Response-Rate-Limiting-Feature.html, 2013. [ISC Knowledge Base, Online; accessed 19-March-2018].

[28] Internet Systems Consortium (ISC). Using the response rate limiting feature. https://kb.isc.org/docs/aa-00994, 2018. [Online; accessed 05-May-2019].

[29] Cheng Jin, Haining Wang, and Kang G Shin. Hop-count filtering: an effective defense against spoofed DDoS traffic. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 30–41. ACM, 2003.

[30] Lin Jin, Shuai Hao, Haining Wang, and Chase Cotton. Your remnant tells secret: residual resolution in ddos protection services. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 362–373. IEEE, 2018.

[31] Mattijs Jonker, Anna Sperotto, Roland van Rijswijk-Deij, Ramin Sadre, and Aiko Pras. Measuring the adoption of DDoS protection services. In *Proceedings of the 2016 Internet Measurement Conference*, pages 279–285. ACM, 2016.

[32] Srikanth Kandula, Dina Katabi, Matthias Jacob, and Arthur Berger. Botz-4-sale: Surviving organized DDoS attacks that mimic flash crowds. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 287–300. USENIX Association, 2005.

[33] Mohammad Karami and Damon McCoy. Understanding the emerging threat of DDoS-as-a-service. In *Presented as part of the 6th USENIX Workshop on Large-Scale Exploits and Emergent Threats*, Washington, D.C., 2013.

[34] Charlie Kaufman, Radia Perlman, and Bill Sommerfeld. DoS protection for UDP-based protocols. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 2–7. ACM, 2003.

[35] Yoohwan Kim, Ju-Yeon Jo, H Jonathan Chao, and Frank Merat. High-speed router filter for blocking TCP flooding under DDoS attack. In *Performance, Computing, and Communications Conference, 2003. Conference Proceedings of the 2003 IEEE International*, pages 183–190. IEEE, 2003.

[36] Lukas Krämer, Johannes Krupp, Daisuke Makita, Tomomi Nishizoe, Takashi Koide, Katsunari Yoshioka, and Christian Rossow. AmpPot: Monitoring and defending against amplification DDoS attacks. In *International Workshop on Recent Advances in Intrusion Detection*, pages 615–636. Springer, 2015.

[37] Marc Kührer, Thomas Hupperich, Christian Rossow, and Thorsten Holz. Exit from hell? reducing the impact of amplification DDoS attacks. In *USENIX Security Symposium*, pages 111–125, 2014.

[38] Warren Kumari. Rssac002 - rssac advisory on measurements of the root server system. https://ai.google/research/pubs/pub43973, 2015. [Online; accessed 08-Aug-2019].

[39] Jason Lackery. A new twist on denial of service: DDoS as a service. https://blogs.cisco.com/security/a_new_twist_on_denial_of_service_ddos_as_a_service, 2010. [Online; accessed 6-June-2019].

[40] J. Levine. Dns blacklists and whitelists. RFC 5782, February 2010.

[41] Kieren McCarthy. Internet's root servers take hit in DDoS attack. https://www.theregister.co.uk/2015/12/08/internet_root_servers_ddos/, 2015. [Online; accessed 29-Janurary-2019].

[42] Mcconachie. Anti-spoofing, BCP 38, and the tragedy of the commons. https://www.internetsociety.org/blog/2014/07/anti-spoofing-bcp-38-and-the-tragedy-of-the-commons/, 2018. [Online; accessed 7-May-2018].

[43] Wei Meng, Chenxiong Qian, Shuang Hao, Kevin Borgolte, Giovanni Vigna, Christopher Kruegel, and Wenke Lee. Rampart: Protecting web applications from CPU-exhaustion Denial-of-Service attacks. In *27th USENIX Security Symposium*, pages 393–410, 2018.

[44] Jelena Mirkovic and Peter Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[45] P. Mockapetris. Domain names—implementation and specficiation. RFC 1035, November 1987.

[46] Giovane Moura, Ricardo de O Schmidt, John Heidemann, Wouter B de Vries, Moritz Muller, Lan Wei, and Cristian Hesselman. Anycast vs. DDoS: evaluating the november 2015 root DNS event. In *Proceedings of the 2016 ACM on Internet Measurement Conference*, pages 255–270. ACM, 2016.

[47] Giovane C. M. Moura, John Heidemann, Moritz Müller, Ricardo de O. Schmidt, and Marco Davids. When the dike breaks: Dissecting DNS defenses during DDoS. In *Proceedings of the ACM Internet Measurement Conference*, October 2018.

[48] Netscout. Automation puts time on your side during a DDoS attack. https://www.netscout.com/blog/ddos-attack-automation, 2018. [Online; accessed 12-Feb-2019].

[49] Arbor Network. NETSCOUT arbor's 13th annual worldwide infrastructure security report. https://pages.arbornetworks.com/rs/082-KNA-087/images/13th_Worldwide_Infrastructure_Security_Report.pdf, 2019. [Online; accessed 31-May-2019].

[50] Lily Hay Newman. Github servived the biggest DDoS attack ever recorded. https://www.wired.com/story/github-ddos-memcached/, 2018. [Online; accessed 19-March-2018].

[51] NTP. NTP: The network time protocol. http://www.ntp.org/, 2018. [Online; accessed 7-May-2018].

[52] Jeffrey Pang, James Hendricks, Aditya Akella, Roberto De Prisco, Bruce Maggs, and Srinivasan Seshan. Availability, usage, and deployment characteristics of the Domain Name System. In *Proceedings of the ACM*

Internet Measurement Conference, pages 123–137, Taormina, Sicily, Italy, October 2004. ACM.

[53] Salvatore Sanfilippo. hping3(8) - linux man page. https://linux.die.net/man/8/hping3/, 2019. [Online; accessed 29-May-2019].

[54] Bruce Schneier. Lessons from the Dyn DDoS attack. https://www.schneier.com/blog/archives/2016/11/lessons_from_th_5.html, 2016. [Online; accessed 21-June-2018].

[55] Alex Scroxton. Major DDoS attacks see huge increase, says akamai. https://www.computerweekly.com/news/450412933/Major-DDoS-attacks-see-huge-increase-says-Akamai, 2017. [Online; accessed 29-Jan-2018].

[56] Root server operators. Events of 2015-11-30. http://root-servers.org/news/events-of-20151130.txt, 2017. [Online; accessed 17-Jan-2018].

[57] Root server operators. Events of 2016-06-25. http://root-servers.org/news/events-of-20160625.txt, 2017. [Online; accessed 17-Jan-2018].

[58] Daniel Smith. Antithe growth of DDoS-as-a-service: Stresser services. https://blog.radware.com/security/2017/09/growth-of-ddos-as-a-service-stresser-services/, 2017. [Online; accessed 18-June-2018].

[59] Minho Sung and Jun Xu. IP traceback-based intelligent packet filtering: a novel technique for defending against internet DDoS attacks. *IEEE Transactions on Parallel and Distributed Systems*, 14(9):861–872, 2003.

[60] Alexander Sword. Preparing for DDoS: How to fight off resource exhaustion. https://www.cbronline.com/cybersecurity/protection/preparing-ddos-fight-off-resource-exhaustion/, 2016. [Online; accessed 4-Feb-2019].

[61] Roshan Thomas, Brian Mark, Tommy Johnson, and James Croall. Netbouncer: client-legitimacy-based high-performance DDoS filtering. In *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, volume 1, pages 14–25. IEEE, 2003.

[62] Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. DNSSEC and its potential for DDoS attacks: a comprehensive measurement study. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 449–460. ACM, 2014.

[63] Paul Vixie, Gerry Sneeringer, and Mark Schleifer. Events of 21-Oct-2002. web page http://c.root-servers.org/october21.txt, November 2002.

[64] Haining Wang, Cheng Jin, and Kang G Shin. Defense against spoofed IP traffic using hop-count filtering. *IEEE/ACM Transactions on Networking (ToN)*, 15(1):40–53, 2007.

[65] D. Wessels and M. Fomenkov. Wow, That's a lot of packets. In *Passive and Active Network Measurement Workshop (PAM)*, San Diego, CA, Apr 2003. PAM.

[66] Curt Wilson. Attack of the Shuriken: Many hands, many weapons. https://www.arbornetworks.com/blog/asert/ddos-tools/, 2012. [Online; accessed 31-Jan-2018].

[67] Abraham Yaar, Adrian Perrig, and Dawn Song. StackPi: New packet marking and filtering mechanisms for DDoS and IP spoofing defense. *IEEE Journal on Selected Areas in Communications*, 24(10):1853–1863, 2006.

[68] MyungKeun Yoon. Using whitelisting to mitigate DDoS attacks on critical Internet sites. *IEEE Communications Magazine*, 48(7), 2010.

[69] Yingdi Yu, Duane Wessels, Matt Larson, and Lixia Zhang. Authority server selection in DNS caching resolvers. *ACM SIGCOMM Computer Communication Review*, 42(2):80–86, 2012.

[70] Saman Taghavi Zargar, James Joshi, and David Tipper. A survey of defense mechanisms against Distributed Denial of Service (DDoS) flooding attacks. *IEEE communications surveys & tutorials*, 15(4):2046–2069, 2013.

[71] Tin Zaw. DDoS is coming: A story of DD4BC and the copy-cat squalls. https://www.rsaconference.com/writable/presentations/file_upload/cct-w06_ddos_is_coming_final.pdf, 2016. [Online; accessed 03-May-2019].

[72] Bojan Zdrnja. Google Chrome and (weird) DNS requests. https://isc.sans.edu/diary/Google+Chrome+and+weird+DNS+requests/10312/, 2011. [Online; accessed 13-June-2019].

[73] Yury Zhauniarovich, Issa Khalil, Ting Yu, and Marc Dacier. A survey on malicious domains detection through DNS data analysis. *ACM Computing Surveys (CSUR)*, 51(4):67, 2018.

[74] Liang Zhu and John Heidemann. LDplayer: DNS experimentation at scale. In *Proceedings of the Internet Measurement Conference 2018*, pages 119–132. ACM, 2018.

[75] Liang Zhu, Zi Hu, John Heidemann, Duane Wessels, Allison Mankin, and Nikita Somaiya. T-DNS: Connection-oriented DNS to improve privacy and security. *ACM SIGCOMM Computer Communication Review*, 44(4):379–380, 2015.