On the Feasibility of TTL-based Filtering for DRDoS Mitigation

Michael Backes[1], Thorsten Holz[2], Christian Rossow[3], Teemu Rytilahti[2], Milivoj Simeonovski[3], and Ben Stock[3]

[1] CISPA, Saarland University & MPI-SWS, Saarland Informatics Campus
[2] Horst Görtz Institute for IT-Security, Ruhr University Bochum
[3] CISPA, Saarland University, Saarland Informatics Campus

Abstract A major disturbance for network providers in recent years have been *Distributed Reflective Denial-of-Service* (DRDoS) attacks. In such an attack, the adversary spoofs the IP address of a victim and sends a flood of tiny packets to vulnerable services. The services then respond to spoofed the IP, flooding the victim with large replies. Led by the idea that an attacker cannot fabricate the number of hops a packet travels between amplifier and victim, *Hop Count Filtering* (HCF) mechanisms that analyze the Time-to-Live (TTL) of incoming packets have been proposed as a solution.

In this paper, we evaluate the feasibility of using HCF to mitigate DRDoS attacks. To that end, we detail how a server can use active probing to learn TTLs of alleged packet senders. Based on data sets of benign and spoofed NTP requests, we find that a TTL-based defense could block over 75 % of spoofed traffic, while allowing 85 % of benign traffic to pass. To achieve this performance, however, such an approach must allow for a tolerance of +/-2 hops.

Motivated by this, we investigate the tacit assumption that an attacker cannot learn the correct TTL value. By using a combination of traceroute-ing and BGP data, we build statistical models which allow to estimate the TTL within that tolerance level. We observe that by wisely choosing the used amplifiers, the attacker is able to circumvent such TTL-based defenses. Finally, we argue that any (current or future) defensive system based on TTL values can be bypassed in a similar fashion, and find that future research must be steered towards more fundamental solutions to thwart any kind of IP spoofing attacks.

Keywords: IP spoofing, Hop Count Filtering, Reflective Denial-of-Service

1 Introduction

One of the major hassles for network provides in recent years have been so-called *Distributed Reflective Denial-of-Service* (DRDoS) attacks [5]. In these attacks, an attacker poses as its victim and sends a flood of tiny packets to vulnerable services which then respond with much larger replies to the victim. This is possible

because the Internet Protocol (IP) does not have means to protect against forgery of source addresses in its packets, so-called *IP spoofing*. A variety of different UDP-based protocols have been known to be vulnerable for this category of attacks for long [22], but despite the efforts to locate and shut down vulnerable services, they remain a problem even today.

To ensure that a server does not become unwilling participant in a DRDoS attack, an appealing defense is to detect spoofed packets *at the recipient*. One such technique is to validate certain IP header fields and drop packets that seem unsound. Most promising, Cheng et al. [10] propose a technique called Hop Count Filtering (HCF) to leverage the Time-to-Live (TTL) field encoded in the IP header. The intuition behind a TTL-based filtering approach is that the route of the *actual* source of the traffic and the *claimed* source is likely different, i.e., the spoofing source is in a different network than the spoofed IP address. This is then also reflected in the TTL value, as the attacker's route to the server differs from the one of the spoofed system, and hence the number of hops is different. Thus, it is seemingly possible to filter most spoofed traffic by dropping any traffic which does not correspond to the expected TTL.

In this paper, we evaluate the feasibility of using HCF to defend against DRDoS attacks. To do so, we analyze several means of probing for the TTL of an alleged sender, using different types of probes towards a host in question as well as horizontal probing of its neighbors. We show that this process is prone to errors and frequently tedious in practice, raising the need for a certain tolerance in TTL-based defenses. More precisely, we show that an error margin of +/-2 must be allowed to enable 85 % of benign traffic to pass, while dropping more than 75 % of spoofed traffic.

Any TTL-based defense relies on the tacit assumption that an attacker cannot learn the correct TTL when spoofing a packet. We, however, show that a spoofing attacker can subvert TTL-based filters by predicting the TTL value—without having access to the system or network of either server or impersonated victim. Our idea is to leverage publicly available traceroute data to learn subpaths that an IP packet from IP_A to IP_B will take. We follow the intuition that subpaths from IP_A to any other host on the Internet are quite constant and can be learned by the attacker. Similarly, we show that the attacker can observe that any packet to IP_B traverses a certain subpath. We augment such subpath information with an approximation of how the packet is routed on the higher-tier Internet layers. Given the tolerance required in TTL-based defenses, we can estimate the initial TTL value that the attacker has to set to enable bypassing of such defenses.

These "negative" results prove that TTL-based spoofing filters are unreliable and (if at all) a short-sighted solution only. Rather than attacking existing defense systems, our findings conceptually show that TTL-based defenses cannot work to thwart the outlined attacks. Hence, we see this paper as a valuable contribution to steer future research towards more fundamental solutions, be it alternative defenses against spoofing, or conceptual redesigns of the Internet and its protocols.

To summarize, we make the following contributions:

- We discuss how a server can use active probing to measure the hops to hosts which connect to its services (Section 3).
- We re-evaluate the concept of HCF to determine the necessary level of tolerance required for it to work in practice (Section 4).
- We describe a methodology which leverages previous knowledge about routing and statistical models to estimate the number of hops between an arbitrary victim and an amplifier of the attacker's choosing (Section 5).
- In doing so, we show that TTL-based defenses can be circumvented by an attacker with as little as 40 globally distributed probes (Section 6).

2 Background

In this section, we discuss the background information on routing on the Internet, Distributed Denial of Service attacks, and Hop Count Filtering as a countermeasure against such attacks.

2.1 Relevant Internet Technologies

The Internet is a network of interconnected sub-networks, which route packets between them based on the established routes. These smaller networks are also referred to as *Autonomous Systems* (*AS*). For a host in network A to connect to a host in network B, a route must be found through potentially several different ASes. Traffic between different autonomous systems is routed based on the Border Gateway Protocol, in which routers exchange information about accessible IP ranges and the corresponding AS paths, i.e., routes to these ranges.

To ensure that a packet is not stuck in a routing loop, the Internet Protocol (IP) header contains a field dubbed Time-to-Live (*TTL*). When handling a packet, "[...] every module that processes a datagram must decrease the TTL" and whenever a packet's TTL value reached zero, the packet must be discarded by the routing device [19]. In practice, the TTL is implemented as a decreasing hop count. The value is initially set by the sending host and depends on the operating system, e.g., Mac OS X uses 64, Windows 128, and while Linux distributions nowadays mostly use 64, some even use 255 [1]. When receiving a packet, analysis of the TTL values therefore allows to approximate the number of routing devices the packet has passed.

The concept of TTLs can also be used to learn the exact route of a packet (*tracerouting*). To that end, the initiator of the tracerouting sends an IP packet towards the intended destination, initially setting the TTL value to 1. When this packet reaches the first hop, the TTL is decreased. According to the RFC, the router must now drop the packet. In such a case, most routers will also send an Internet Control Message Protocol (*ICMP*) error message to the original sender, indicating that the timeout of the packet has been exceeded. This response can be used by the tracerouting machine to learn the IP address of the first hop. By repeating this process with increasing TTL values, this method can be used to learn all IP addresses of routers on the packet's way to its destination.

2.2 Source Spoofing and DRDoS

In its original design, the Internet Protocol does not feature a means of verifying the source of a packet. Since IP packets are only directed based on the *destination*, an attacker may generate an IP packet with a fabricated (or *spoofed*) source address. This design flaw can be abused by an adversary towards several ends. One example are Denial of Service (DoS) attacks, where an attacker tries to either saturate the network link to a server or exhaust resources on the target machine by, e.g., initiating a large number of TCP handshakes. To defend against this, a network administrator may configure a firewall to drop packets from the attacker. The attacker, however, can spoof IP packets from other machines to bypass this defense mechanism.

Moreover, recent years have seen an increase in Distributed Reflective Denial of Service (DRDoS) attacks. These attacks rely on spoofing packets in conjunction with services which respond to requests with significantly larger responses. There are a variety of vulnerable protocols (described in [22,23]), but recently, the most nefarious attacks have been misusing protocols such as DNS, NTP, SSDP, or chargen. As an example, the Network Time Protocol's (NTP) *monlist* feature may generate a response that is more than 4,500 times larger than the request. To abuse this, an attacker generates a flood of *monlist* requests to vulnerable servers while spoofing the source IP address to be that of the victim. Subsequently, a vulnerable NTP server will send the response to the victim's IP. In doing so, the attacker can massively amplify his own bandwidth, while also not revealing his real IP address in the process.

Although this kind of attack has been well-known for long [14,24] and attempts have been made to shut down vulnerable systems used in such attacks (e. g., [12]), they still pose a threat to online services. In order to fight such attacks, several countermeasures dating back to 2001 [17] have been proposed. One obvious defense strategy would be to limit the number of requests a client may issue. However, while such mechanisms may help to protect against excessive abuse of a single amplifier, Rossow's [22] analysis shows that even with rate limiting the aggregated attack bandwidth of some protocols is still an issue. This and many other countermeasures have been evaluated and analyzed by Beitollahi and Deconinck [7], hence we omit to discuss them further and refer the reader to their paper. Instead, we discuss the hop count filtering mechanisms relevant for our work in the following.

2.3 Hop Count Filtering

When a packet is received, its TTL depends on (i) the initial TTL value and (ii) the number of hops the packet has traversed. While it is easy to forge an IP header as such, Cheng et al. [10] propose to use the TTL to detect nefarious packets. More precisely, they assume that an attacker trying to impersonate a specific host cannot ascertain the hop count between the spoofed host and the recipient of the packet. Based on this assumption, they present a reactive defense against DDoS attacks. To detect an attack in which the sender spoofs IP

addresses to conceal his true location, they first require a period of observing the legitimate upcoming traffic (learning state), where the victim builds a mapping between the legitimate clients (IP addresses) and their respective hop count. Once an attack is detected, the victim rejects all packets where the TTL values do not match the recorded hop count. This way, the victim does not have to allocate resources for handling incoming spoofed traffic.

To increase the accuracy of the hop count filtering (HCF), Mukaddam et al. [15] proposed a modified version of HCF that aims to improve the learning phase. Instead of recording only one hop count value per IP, they record a list of all possible hop count values seen in the past. They justify the need for such an extension by arguing that the hop count may change due to the use of different routes. Indeed, such a system decreases the collateral damage by correctly classifying legitimate traffic. On the other hand, however, this mechanism allows an attacker more guesses in evasion attempts by ascertaining the correct TTL value.

3 Re-Evaluating the Feasibility of Hop-Count Filtering

As the previous work by Mukaddam et al. has shown, the original HCF approach may be impaired by routing on the Internet. In addition, such an approach requires a prior learning phase, e. g., through passive TCP handshake analysis, to facilitate detection of spoofing. In the following, we investigate how far the methodology from Cheng et al. can be extended to filter out spoofed traffic used in DRDoS attacks. In contrast to the original HCF, this process cannot rely solely on TCP handshakes from previous connections by the client, as protocols used in DRDoS attacks, such as NTP or DNS, are connection-less. Simply dropping all packets from any host without a previous TCP connection would render any benign use of UDP-based services moot. Therefore, we investigate with what margin of error TTLs for an alleged sender can be learned by the server to evaluate the efficacy of TTL-based filtering on the Internet.

3.1 Protocol-based Probing

The most intuitive way for a server to ascertain a TTL value of a client is to receive an unspoofed packet from that host. This can be done after a successful TCP handshake, as an established connection can only occur if the alleged sender actually initiated the connection. Due to its connection-less nature, we cannot rely on such a process for UDP. Instead, we need to prompt the alleged sender for an unspoofed packet. To achieve this, we can rely on ICMP, TCP, or UDP requests to the system in question. The ports we used in our work for TCP and UDP are derived from the most scanned port discussed by Durumeric *et al.* [8]. We realize that it might not be feasible to send a plethora of probes to an end host whenever a packet to a UDP-based service is received, as this itself would be an amplification attack. Regardless, we want to investigate how different protocols and techniques might be leveraged to learn the TTL.

One way of compelling the probed system to send a packet is to use ICMP. ICMP *echo* can be used to measure the round trip time of a packet to a given host. The TTL of the probe target can be extracted from the IP header of an echo reply. In addition to the echo command, several operating systems also implement the non-mandated *timestamp* command. This can be used in the same fashion to induce a response from the probed system.

Additionally, the probing server can itself try to establish a TCP connection to the alleged sender. The methodology is independent of the actual application used underneath, since the TCP handshake is conducted by the operating system before handing the socket to the underlying application.

In contrast to TCP, where no application data needs to be sent to the probed host, most UDP-based services require protocol-specific data to be submitted. As an example, DNS and NTP servers only react to datagrams which are conformant to the respective protocol. On the other hand, the UDP-based *chargen* service "simply sends data without regard to the input" [20]. Therefore, we send protocol-conformant packets to DNS and NTP ports, and random data to chargen.

3.2 Interpreting Responses

In any of the cases described above, we may receive a positive or negative response. In the following, we discuss these types of responses and indicate how they can be used to extract the TTL from probed systems.

Positive Responses When using ICMP, an echo or timestamp reply suffices to extract the TTL value from the encapsulating IP packet. For TCP, if a service listens on the probed port, the operating system will follow the three-way handshake process and respond with a SYN/ACK packet. In the case of UDP, the process differs slightly: when a service is listening on the probed port and the incoming packet adheres to the specification of that service, it sends a response back to the requesting system. Analogously to ICMP, the TTL value can be extracted from TCP and UDP responses by simply examining the IP header.

Negative Responses In addition to responses which indicate that the host is up or a service is listening on the probed port, we can also leverage negative responses or error messages to learn the TTL. For example, in cases where a TCP port is not open, the host system may respond with a packet which has the RST flag set. Assuming that the packet is usually generated by the probed system (we discuss exceptions to this rule in Section 3.4), we can extract the TTL value in the same fashion used for positive responses. For UDP, we leverage ICMP *Port Unreachable* replies.

Next to these protocol-specific errors, we may also receive a message indicating that the host is not reachable. For example, the last router on the path can issue an ICMP *Host Unreachable* message. In this case, given the assumption that only the last router will send such a message, we can use the TTL from the incoming packet and decrease it by one (since the original sender would have had one more hop). ICMP also features a more generic *Destination Unreachable* message; this,

however, can be sent by any router on the path and therefore cannot be used to conclusively calculate the TTL value. Next to these, we may receive ICMP *Communication Administratively Prohibited* messages. Such a message can either be sent by a router or the system itself when a packet is rejected by the firewall.

3.3 Horizontal Probing

A probed host may not answer, e.g., because it is firewalled and drops any incoming packets. In these cases, we may still gather valuable information on the path to the host by probing neighboring hosts. A neighbor in this case is a host which is located within the same subnet as the target. Although assuming that each subnet consists of exactly 256 IPs is not correct, this measure can still provide partial insight into the route and give a close estimate of the actual TTL value. Therefore, we probe neighbors by changing the last octet of the IP address +/-1, and use previous knowledge from hosts within the same /24 subnet, as this is the smallest network section generally advertised and accepted via BGP [18].

3.4 Caveats of Active Probing

There are several scenarios which can induce errors in probes. Typically, private customers receive a router for their dial-up account, which uses Network Address Translation (*NAT*) to allow multiple LAN clients access to a single Internet connection. Unless these routers are configured to forward packets to a machine behind the NAT, any response to the previously mentioned probes will be generated by the router. As the router adds an additional hop (and hence decreases the TTL by one) on the way from the NAT client to the server, the TTL values will mismatch in such a case.

For negative responses, additional artefacts may skew the results. Specifically, TCP resets or ICMP error packets may be generated by a firewall located before the intended probe target. In such a case, the firewall itself must spoof the probed IP to send these packets to ensure that the packet is attributed correctly on the system which initiated the connection. Hence, we may assume that negative responses are indeed generated by the probed system. Since we cannot learn the number of hops between the firewall and probed system, using negative responses can yield false results. We discuss the number of false results in Section 4.

As outlined before, the initial TTL value depends on the operating system of the sending host. Considering an example in which a Windows client is located behind a NAT router, which is running a Linux system with an initial TTL value of 255. Even though a packet originating from the Windows machine will only have one additional hop on its way to the probing server, the TTL value received by the probing system will greatly differ depending on whether the Windows or Linux host responded to the probing request. To accommodate for this and for horizontal probing, we normalize all TTL values to values between 0 and 63, i.e., $TTL = TTL\%64$. As the maximum TTL of 255 is not divisible by 64, we first increment TTL values above 128 by one to correct this discrepancy.

4 Probing Analysis

To evaluate how well active probing could be used in the wild to enable the use of HCF, we set up two systems. First, we used a regular NTP server not susceptible to DRDoS to attract benign clients. Second, we set up a honeypot system running a vulnerable version of NTP to attract spoofing attackers. In the following, we describe both data sets, discussing for what fraction of hosts we could learn any TTL value, and comparing this to the TTL values of incoming packets. Although we are using NTP servers for our evaluation, it is out of convenience of getting both spoofed and non-spoofed clients for comparison. In contrast, for protocols like chargen, getting benign traffic would have been significantly harder. We end this section with a discussion on the implications of the results of our analysis.

4.1 Benign Traffic

To capture benign traffic, we set up an NTP server that does not implement *monlist* feature at all, and is therefore not susceptible to amplification vectors. To attract NTP clients, we joined the NTP pool project. Note that the term *client* refers to its role in NTP, i.e., such a host could either be an end user's computer or a server synchronizing its clock with us. Within hours, the server was added to the public pool and started to receive NTP requests. We analyzed the incoming traffic for patterns of suspicious behavior (especially dreaded *monlist* requests). Our analysis showed that such requests were only issued in small numbers by scanners (e.g., operated by research groups). As we did not respond to such amplification requests and did not notice any suspicious activity, it is highly unlikely that an attacker would choose our server for his amplification attack. Hence, we deem this data set to consist exclusively of unspoofed traffic.

In total, we gathered data for 48 hours, in which we received packets from 543,514 distinct IP addresses. In a first step, we probed each of these hosts immediately after their first contact using the different types of probes outlined in Section 3.1. In doing so, we could extract TTL values for 316,012 (58.1 %) for probed systems. The most successful type of probe was ICMP echo, which yielded a result for 257,694 or 47.4 % of the hosts. In comparison, the most successful TCP-based, positive response were SYN/ACKs from TCP port 443 (HTTPS), which accounted for a mere 31,966 (5.9 %) of the hosts. For any UDP-based probes, we only received negligible amounts of positive responses. Among the negative responses, ICMP *Communication Prohibited* for TCP port 4899 (Radmin) was the most frequent message (113,058 or 20,8 %).

To find out how accurate these results actually are, we compared the normalized TTL values to the ones from the incoming traffic. As stated before, we assume that the traffic directed to the NTP server is indeed generated by the alleged senders, i.e., the ground truth value for each sending host can be extracted from these incoming packets. Initially, we consider all probes to a specific host for our analysis. In cases where the measured TTL values differ between the probe types, we select the minimum value of any test. The intuition of this is straightforward: whenever a firewall or router answers instead of the probed

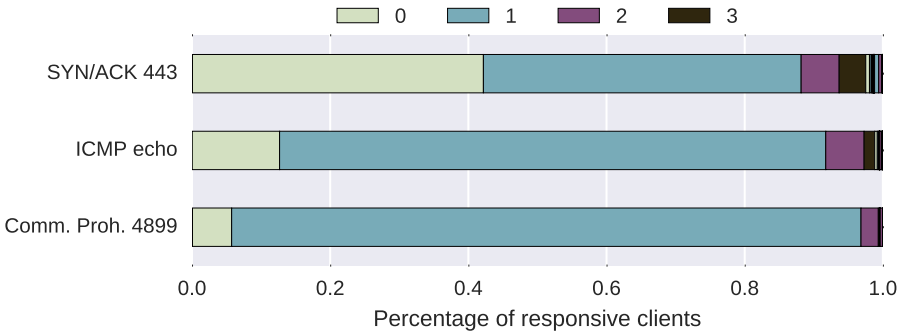

Figure 1. Deviation differences for selected probe types

system, the number of hops between them and our probing server is smaller. Hence, by choosing the minimum TTL value, we ensure that we measure the longest path between us and the host responding to the probe. Therefore, if the probed system answers to one probe whereas all others are responded to by the firewall, we still measure the accurate value for the system in question.

The results of applying this methodology on the data set are shown in Table 1. We observe that, with respect to the total number of responding systems, 26.1 % of the measured TTLs match the ground truth. Moreover, 92.2 % of the values are within a threshold of +/-1, and almost 97 % within +/-2. In the following, we analyze the results for specific tests in more detail, and discuss potential reasons for the observed deviations.

The deviation between the measured and actual values is shown in Figure 1 for ICMP echo, Communication Prohibited to TCP port 4899, and SYN/ACK for TCP port 443. We can observe that for ICMP echo, 12.8 % of measured TTLs were correct, whereas an additional 78.8 % were off-by-one, i.e., 91.6 % of the measured TTLs were within a threshold of +/-1. For *Communication Prohibited* on port 4899, we observe that 96.8 % of the values are within +/-1, whereas 91 % are off-by-one. This appears natural to the scenarios we discussed: ICMP echo requests will often be answered by routers and firewalls due to network address translation. Although SYN on TCP port 443 was only responsive on 5.9 % of the

Deviation	Amount	Fraction	Cumulated Fraction
+/-0	82,629	26.1 %	26.1 %
+/-1	208,891	66.1 %	92.2 %
+/-2	14,623	4.6 %	96.9 %
+/-3	4,684	1.5 %	98.4 %
more	5,185	1.6 %	100 %

Table 1. Accuracy of measured TTLs (direct probes only)

hosts, the results are quite interesting. We observe that for 42.2 % of the hosts which responded to such a probe, the TTL value could be correctly measured. In addition, another 45.9 % were off-by-one, resulting in 88 % of the values being within a threshold of +/-1. We argue that this is caused by nature of TCP, i.e., we only receive a SYN/ACK in case a service is listening on the probed system. This can either occur if the connection directly reached the probed system, i.e., it is not behind a NAT or the corresponding port is forwarded, or there could be a chance that a public-facing administrative interface is being exposed for service needs [2]. Therefore, it is plausible that such routers may respond to HTTPS requests, explaining the high number of our off-by-one measurements.

Next to probing of the target system itself, we can probe neighboring hosts. More specifically, we probe direct neighbors (IP +/-1) and additionally rely on previous measurements aimed towards other hosts within the same /24 network. In doing so, we find that both types of probing increase the coverage. In our experiment, we found that directly probing neighbors increases the number of measurable TTLs by 69,399, resulting in a total coverage of 73.4 %. Taking into account all information from hosts within the same /24 network increases the coverage more drastically (by 168,730 hosts), yielding TTL values for 91.6 % of all hosts. At the same time, the accuracy remains similar, with 27 % of the probed values matching the ground truth. For +/-1, we can correctly measure the TTL in 88.9 % of the cases, and 94.3 % of all measurements are within a threshold of +/-2. Given these results for coverage and accuracy, we note that combining different types of probing towards a single host with horizontal probing of the system's neighbors allows us measure the TTL within a threshold of +/- 2 for 86.4 % of all connecting hosts.

4.2 Spoofed Traffic

Next to the benign data set, for which we can measure the TTL within a small threshold for the majority of the hosts correctly, we wanted to investigate how well HCF would be suited for spoofed traffic. To that end, we set up a honeypot running a vulnerable version of NTP server prone to becoming an amplifier for DRDoS attacks. To avoid unnecessarily harming the spoofed targets while still pretending to be attractive to adversaries, the outgoing bandwidth was limited, i.e., we answered to at most two monlist requests per host per minute. We did not announce the IP address of this machine in any manner and hence assume that no legitimate traffic would be directed to the host. Instead, incoming NTP requests are either due to scanning, or spoofed packets sent by an attacker. In a time-period of 96 hours, we recorded 5,616 distinct alleged sender addresses, for which we could gather direct probe results in 3,983 cases (70.9 %). This slightly higher coverage (compared to the benign data) can be explained by the fact that most attacks are targeting servers, which also are more likely to expose services we actively probe for.

Before conducting any of our measurements, one property of the spoofed traffic became apparent: more than 99 % of all incoming packets had an assumed initial TTL of 255. This specific feature, however, should not be used solely to

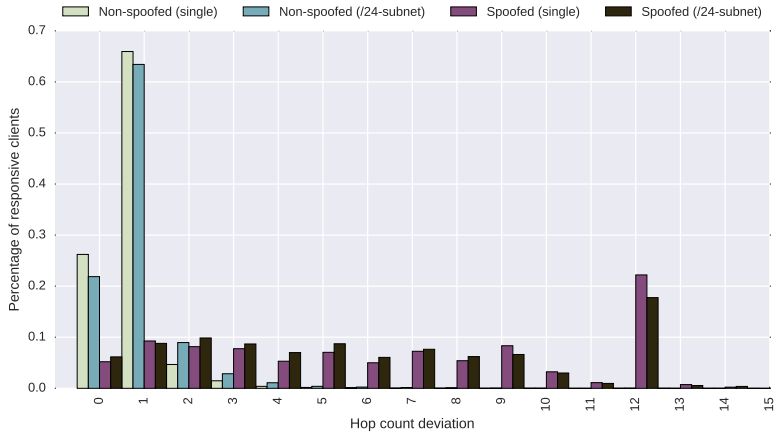

Figure 2. Deviation difference between spoofed and non-spoofed traffic

detect spoofed traffic, since the initial TTL can be changed without much effort by the attacker. Therefore, we normalized the TTL value as outlined before.

Figure 2 shows the comparison between the measured TTL values and the TTL values extracted from incoming packets, for both benign and spoofed data sets. While we can clearly observe that for the majority of benign clients, the TTL can be guessed within a threshold of +/-2, we note that no such trend is visible for spoofed traffic.

4.3 Implications

In this section, we outlined the results of our experiments on benign and spoofed data sets to evaluate a feasible margin of error for HCF. With respect to those data sets, we find that distinguishing between benign and spoofed traffic appears to yield useful results when using a threshold of 2. The reasons for the imprecision of the measurements are manifold, e.g., when a client is behind a NAT or incoming traffic to the machine is filtered by a firewall. Therefore, a TTL-based defense mechanism must make a trade-off between false positives and false negatives, respectively. Based on the data sets we analyzed, if a TTL-based defense mechanism was to be deployed to protect a service against becoming an unwilling actor in an attack, over 85 % of the benign traffic could pass, while more than 3/4 of spoofed packets could be dropped, thus avoiding to harm the targets.

Depending on the type of attacked hosts, this distinction might be even easier to make. Nevertheless, any TTL-based defense relies on one tacit assumption: an attacker can not learn the correct TTL value for an arbitrary victim and an amplifier of his choosing. Therefore, in the following section, we discuss the feasibility of a method in which the attacker can learn the TTL value (within a given threshold).

5 Methodology for Estimating Hop Count Value

So far we showed that deploying a TTL-based filtering at the server side would require some tolerance interval to be functional and avoid collateral damage by incorrectly classifying legitimate traffic. In this section, we assess if an attacker can actually bypass the filtering by predicting the correct hop count value between the hosts and properly adjusting the TTL value. That is, we present a methodology for estimating the hop count value between amplifiers and victims.

5.1 Key Idea and Attacker Model

Our key idea lies on the observation that paths between arbitrary locations to a selected destination share (small) segments of the path. We leverage the fact that such path information can be learned by an attacker to estimate the number of hops of a packet sent from one location to another. To learn subpaths, we (i) probabilistically model known paths obtained via traceroutes, and (ii) combine this knowledge with BGP routing information. Figure 3 shows our idea for estimating the distance (number of hops) between an amplifier (M) and a victim (V). For our methodology, we use the common approach for representing the Internet, which is a graph where nodes are the autonomous systems and edges are the peerings (routing links) between them. Additionally, we assign weights to the nodes to denote the hop count number within the individual AS. One way to build such a graph that illustrates the AS-level topology of the Internet is to use available BGP data to discover the connectivity information for the ASes. Nevertheless, studies have shown that BGP data is only available to a limited extent, therefore the Internet AS-level topology is partially hidden [9,16]. However, our methodology does not primarily rely on the available BGP data, but rather on the traceroute information an attacker can obtain. We use the BGP data, when available, as a complement to the traceroute data in order to discover the missing ASes, and to subsequently calculate the number of hops.

Our attacker (A) aims at evading any TTL-based filter or, at least, reduce its effectiveness in mitigating amplification attacks. His main goal is to predict the TTL value as close as possible to the correct one, such that he can craft requests which are deemed to be legitimate to the server, i. e., amplifier. In theory, there are few approaches that the attacker may follow to learn the correct TTL value. First, he may learn the TTL value by actively or passively monitoring traffic anywhere on the route, and then probe the destination in order to calculate the remaining part of the route. This approach is neither realistic nor practical because the attacker has to be present at every route R_i between M_i and the victim V. Second, if the attacker can position a probe either in the network of M or V, he can easily measure the TTL value by tracerouting to the other host.

For a more realistic scenario, we restrict the attacker's capabilities. Figure 3 illustrates this attacker model. Similar to the reverse traceroute method [11], our attacker is capable of probing from random, distributed locations and can use any publicly available online resources to traceroute to the amplifier and to the

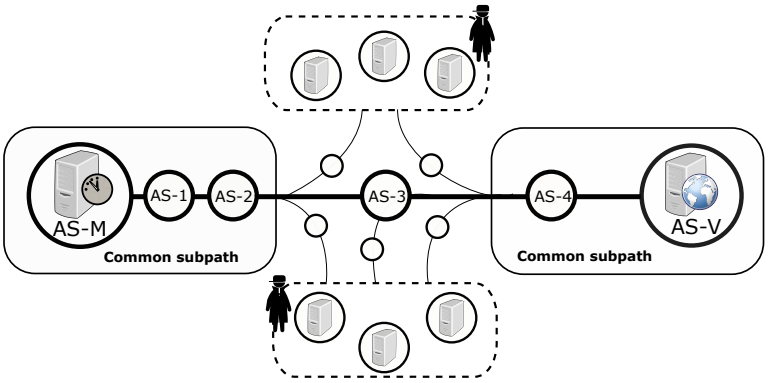

Figure 3. Approach to estimate the hops between amplifier (M) and victim (V)

victim (e. g., RIPE Atlas[3] or looking glass servers). However, he does not have control over the amplifier and not necessarily full control over the probes.

We restrict neither the location of the amplifiers nor the victims, i.e., they can be located at arbitrary network locations. We assume that A can obtain a set of amplifiers (e. g., NTP, DNS), all of which deploy TTL-based filtering and respond to valid requests only[4].

5.2 Methodology

We propose a methodology for estimating the distance between hosts on the internet through an Exploratory Data Analysis (EDA)[5]. Our methodology is comprised of three main components, namely, data collection, data processing, and EDA. Figure 4 illustrates the methodology we propose in this paper.

Data Collection First, depicted in the data collection component, the attacker collects traceroute data for the victim and the amplifier(s). The attacker launches traceroutes to the targeted locations from a globally distributed set of hosts on the Internet such as RIPE Atlas [3]. Note that the distribution of the selected hosts is required to be global such that there will be a diversity of the paths, allowing us to predict TTLs for arbitrarily chosen victims.

Data Processing Second, in the data processing component, we have to ensure that the relevant data collected in the previous stage is complete and usable. In an

[4] We assume that the amplifiers have deployed HCF to protect against amplification attacks, therefore "valid" protocol requests are those with matching TTL value.

[5] Exploratory Data Analysis is not a method or a technique, but rather a philosophy for data analysis that employs a variety of techniques.

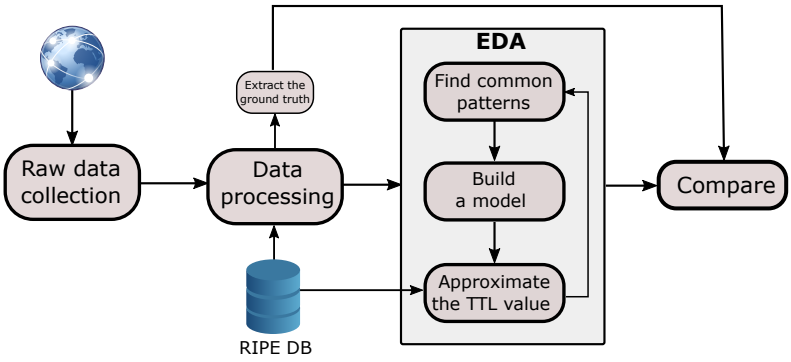

Figure 4. Workflow of the methodology

ideal world, tracerouting returns a complete path including all the IP addresses and ASes on the way up to the destination. In practice, the collected data from the previous phase is usually imperfect, with a plethora of missing connecting hops [13]. Such data can pose difficulties in effective data analysis; therefore, we need to develop certain methods for efficient data scrubbing. First, we discard all the traceroutes that are missing more than a certain percentage (e. g., 50%) of the intermediate hops. Also, we ignore traceroutes that cannot reach at least the AS of the destination. In the case where the destination address belongs to the same AS as the last replying node, we make an intuitive assumption that this is the last AS in the path, and we supplement the route with the AS number of the last replying node. We then continue filling up the gaps of the unknown ASes due to private IP addresses within the traceroute. Private addressing might occur when a packet passes through someone's internal network with implemented Multiprotocol Label Switching (MPLS) routing [21]. In such cases, we assume that the border AS, the one with a public IP address before the MPLS routing, is the correct one, and we fill in the gaps accordingly. Finally, to fill in the remaining missing hops, we apply a technique that employs the publicly available BGP data. The BGP data assists in the discovering of the neighboring AS [6] and helps us to bridge the gap between two known autonomous systems. Note that this technique can only complete the lacking AS numbers, but not the actual hops (and their IP addresses).

Exploratory Data Analysis Once the data is processed, i. e., prepared for analysis, we dissect the data set using the EDA approach. This stage of the methodology repeats for every victim and it involves three subsequent steps.

[6] A neighbor (or peering) autonomous system is the one that the AS directly interconnect with in order to exchange traffic.

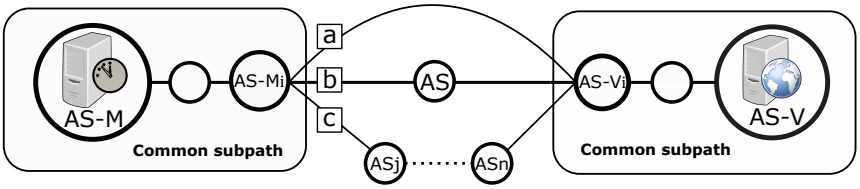

Figure 5. Connecting border ASes (AS-Mi and AS-Vi)

Find common patterns Finding common patterns is the first step in the data exploration. This method transforms the paths from detailed traceroutes with IP addresses of the hops to coarse-grained ones with only AS-level paths and their weights, i. e., the number of hops in each AS for a particular traceroute.

Build a model This method assists in constructing a probabilistic table that identifies the likelihood of an AS to be part of the route between amplifier and victim. If all collected traceroutes pass through a particular AS, say AS-1, on the way to the target location T, the method denotes the probability of 1 that the AS-1 exist as a hop on the way to T. Moreover, this method also considers the average number of hops within the AS and the distance of the AS from the target. The average number is the AS internal hop count value, and it may vary due to routing-related reasons such as load balancing. To identify the border autonomous systems (in the next step), we need to define the distance as a number of hops that a particular AS is distant from the target AS. For example, the AS the target T belongs to always has a probability of 1 and distance 0.

Approximate the TTL value The probabilistic modeling helps in building a partial path between two hosts. Consider the scenario illustrated in Figure 5. The model identifies with a degree of certainty the common subpaths of the target and the source. Furthermore, it estimates the hop count value of these subpaths. To estimate the final hop count value, we need to bridge these two subpaths with the missing intermediate AS(s). To this end, we apply techniques based on the available BGP data such that the final result is a fully connected AS-level path.

Initially, we identify the border autonomous systems (labeled as AS-Mi and AS-Vi in Figure 5), i. e., the last certain (most distant) AS in the common subpaths. With respect to the possible missing hops for connecting these two subpaths, we distinguish three different scenarios (marked with a, b and c in Figure 5):

Direct connection (a) When a direct peering between the border autonomous systems exists, i. e., AS-Mi is in the neighborhood[7] of AS-Vi and vice versa,

[7] Peering ASes are ASes which directly interconnect with each other. We obtain this information from the available BGP data.

and the intersection set of the AS-Mi and AS-Vi neighbors is empty; we assume that the border ASes are directly connected (AS-Mi $\longleftrightarrow$ AS-Vi).

One-hop connection (b) To identify the single connecting point in between, accordingly, we have to check the neighbors of the border ASes. In the case where only one intersecting AS exists, we assume that this particular AS is the connecting point. If the intersection set contains more than one common AS, we refer to our probability table. We then accordingly choose the AS with the biggest probability to be a part of the route.

N-hop connection (c) A more complex scenario is when two or more intermediate AS are missing. In such a scenario, we build a tree of possible subpaths by examining additional two levels[8] of neighbors. Upon building up the tree of all possible paths, we test every branch over the database of available BGP routes and the pre-computed table of probabilities. In case the branch is present in the BGP routing database, we deem that particular route to be the accurate one.

Once the bridging subpath is identified, we add up the average hop count of the connecting ASes to the sum of the hop count value estimated for the subpaths.

6 Experimental Setup and Results

In the following, we describe the data set used to evaluate our approach. Subsequently, we present and discuss the experimental results of the evaluation.

6.1 Data Set

To evaluate the proposed methodology, we mainly use services provided by the RIPE Atlas network [3], which is the largest Internet measurement network built by RIPE NCC. Moreover, they provide an API for creating different types of measurements and for collecting the data in a structured format. In the following, we list the services and data sources used for our experimental evaluation.

1. RIPE Atlas probes: To attain a global coverage and also to have a possibility to obtain the ground truth, we use the RIPE Atlas network of probes [3] as a basis for our experiments. We observe that this network has around 9,000 active probes, spread across 181 countries and 3,386 ASes [4]. Such a global coverage fulfils the requirements for our experimental evaluation. Moreover, the platform give us the flexibility for requesting custom measurements, in our case traceroutes, by selecting any of the deployed active probes. This flexibility is of particular importance for our experiments since we can select a subset of nodes with different geographical and logical locations to collect

[8] Statistics [3] show that average length of AS-level paths is 4, therefore we bound the subpath examination to 2 levels, i. e., we can examine paths of at least 6 hops.

the traceroute data. Additionally, when a probe acts as a victim in our leave-one-out analysis (which we outline in the following), we can easily obtain the ground truth by running traceroute measurement from the probes to the amplifiers.
2. BGP data: When the collected traceroute data is not enough for making the final assessment of the connectivity between the ASes, we utilize available BGP data. In order to infer the AS-level connectivity, we use RIPE Atlas as an accurate source for BGP data. Also the BGP data helps to obtain a ground truth of individual ASes.
3. Amplifiers: To investigate the real-world implications of our attack, we scanned for chargen amplifiers on the Internet. In total, we randomly selected 16 such servers.

6.2 Leave-one-out Evaluation

To evaluate the performance of our methodology, we use a leave-one-out (*L-1-O*) evaluation approach, in which every probe acts like a victim at a selected time. Informally, for a data set with P probes, we perform P experiments with $P-1$ training cases and one test case. In other words, for every experiment we temporarily remove one probe from the data set and select that particular probe as our victim. Upon fixing the probe P_i as a victim V, the model is rebuilt upon this newly defined set.

Suppose that $P = p_1, \ldots, p_n$ is a set of probes, $M = m_1, \ldots, m_l$ set of amplifiers, and $R = r_{11}, \ldots, r_{nm}$ set of traceroutes where r_{ij} is a traceroute from p_i to m_j. For ease of exposition, we use the notation $p_i \Rightarrow_R M$ to describe a set of all traceroutes from p_i to every member of the set M. Applying the L-1-0 approach to the methodology works as follows:

1. Collect the traceroute data ($R \bigcup \{p_i \Rightarrow_R P \setminus \{p_i\} | i = 1, \ldots, n\}$).
2. Process the data and extract the ground truth.
3. Remove probe p_i from P ($P \setminus \{p_i\}$) and set $V = p_i$, where V is the victim.
4. Extract the ground truth for p_i to M i. e., the distance from $p_i \Rightarrow_R M$.
5. Run the EDA using the remaining data.
6. Repeat step 3-5 for $i = 1, \ldots, n$

L-1-O in practice. We apply the L-1-O method on a set of 40 random RIPE Atlas probes, located in different ASes, and 16 randomly distributed chargen amplifiers. We first collect the required data, namely, we obtain the path from every probe to all of the 16 amplifiers, and also between the probes within the set. We use the RIPE Atlas REST API to create IPv4 traceroutes using ICMP packets and hops limit of 32. In order to get more precise paths and avoid measurements inconsistencies caused by load balancing routers, we employ the paris traceroute measurement tool [6].

Once the traceroute data is collected and the data set is processed, i. e., cleaned up using the method described in Section 5.2, we pass the data through step 3-6 from the L-1-0 approach. In such experimental setup, L-1-O theoretically

can evaluate 640 TTL predictions, i. e., paths from 16 amplifiers to 40 victims. Unfortunately, because of the incompleteness of the traceroute data as well as instability of some of the probes, the method was able to predict and evaluate around 593 (92.6%) individual paths.

Overall performance Table 2 shows the overall performance of our methodology. The experimental results show that using our methodology, an attacker can predict correctly without any deviation roughly 13% of the paths between the amplifiers and the victims, i. e., 13% of the measured hop counts match the ground truth. However, we showed in Section 4 that, with a tolerance of +/-2, a TTL-based defense could block over 75% of spoofed traffic, while allowing 85% of benign traffic to pass. Therefore, when we take this threshold into consideration, our methodology is effective for 56.3% of the paths.

	Amount	Fraction	Cumulated Fraction
+/-0	78	13.2%	13.2%
+/-1	170	28.7%	28.5%
+/-2	132	22.3%	56.3%
+/-3	49	8.3%	69.1%
more	164	27.7%	100%

Table 2. Overall performance of the methodology

Moreover, we observe that applying our methodology to a set of randomly chosen amplifiers, the attacker can isolate amplifiers for which he can predict the hop count value between the amplifier and any arbitrary victim with higher accuracy. Thus, he can bypass the TTL-based defense running on the amplifier and exploit it for a DRDoS attack. Figure 6 illustrates the average hop count deviation per amplifier and shows that the attacker can, indeed, sample a set of *good* amplifiers. We see several explanations for such a deviation among the amplifiers. The geographical and logical location of the amplifiers and the victims plays an important role. As we discussed before, the limitation of the BGP data makes our methodology not equally precise for all the AS. Also another cause is the inconsistency of the collected data between BGP data and traceroute path caused by Internet Exchange Points and sibling ASes managed by the same institution. However, these results show that even with a low threshold value at the amplifier, by wisely choosing amplifiers to use, an attacker is able to circumvent any TTL-based defense against DRDoS attacks.

7 Conclusion

In this paper, we evaluated the feasibility of using Hop Count Filtering to mitigate DRDoS attacks. To that end, we detailed how a server can use active probing to

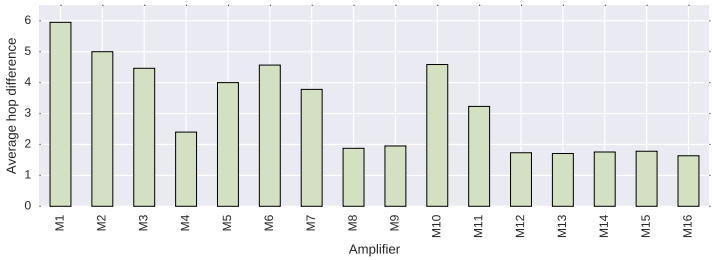

Figure 6. Average hop deviation per amplifier

learn TTLs of alleged packet senders. Based on data sets of benign and spoofed NTP requests, we find that with a tolerance of +/-2, a TTL-based defense could block over 75 % of spoofed traffic, while allowing 85 % of benign traffic to pass. Subsequently, however, we show that an attacker can use a combination of tracerouting and BGP data to build statistical models, which allows him to estimate the TTL for his target within that tolerance level. Hence, by wisely choosing amplifiers to use, he is able to circumvent any TTL-based defense against DRDoS attacks. We therefore argue that any (current or future) defensive system based on TTL values can be bypassed in a similar fashion, and find that future research must be steered towards more fundamental solutions to thwart any kind of IP spoofing attacks.

Acknowledgments

This work was supported by the German Federal Ministry of Education and Research (BMBF) through funding for the Center for IT-Security, Privacy and Accountability (CISPA) as well as through the BMBF grant 01IS14009B ("BD-Sec").

The authors would like to thank Sven Bugiel for his comments on an earlier version of the paper. Additionally, we are grateful for the feedback from our shepherd Roberto Perdisci as well as those of our anonymous reviewers.

References

1. Default TTL Values in TCP/IP. http://www.map.meteoswiss.ch/map-doc/ftp-probleme.htm
2. Functional Requirements for Broadband Residential Gateway Devices. https://www.broadband-forum.org/technical/download/TR-124.pdf
3. RIPE Atlas: Internet data collection system. https://atlas.ripe.net/
4. RIPE Atlas: Statistics and Network coverage. https://atlas.ripe.net/results/maps/network-coverage/
5. Technical Details Behind a 400Gbps NTP Amplification DDoS Attack. https://goo.gl/j7zWEp

6. Augustin, B., Cuvellier, X., Orgogozo, B., Viger, F., Friedman, T., Latapy, M., Magnien, C., Teixeira, R.: Avoiding traceroute anomalies with Paris traceroute. In: Internet Measurement Conference (2006)
7. Beitollahi, H., Deconinck, G.: Analyzing well-known countermeasures against distributed denial of service attacks. Computer Communications (2012)
8. Durumeric, Z., Bailey, M., Halderman, J.A.: An Internet-wide View of Internet-wide Scanning. In: USENIX Security Symposium (2014)
9. Gregori, E., Improta, A., Lenzini, L., Rossi, L., Sani, L.: On the incompleteness of the AS-level graph: a novel methodology for BGP route collector placement. In: Internet Measurement Conference (2012)
10. Jin, C., Wang, H., Shin, K.G.: Hop-count filtering: an effective defense against spoofed DDoS traffic. In: Proceedings of the 10th ACM conference on Computer and communications security. ACM (2003)
11. Katz-Bassett, E., Madhyastha, H.V., Adhikari, V.K., Scott, C., Sherry, J., van Wesep, P., Anderson, T.E., Krishnamurthy, A.: Reverse traceroute. In: USENIX NSDI (2010)
12. Kührer, M., Hupperich, T., Rossow, C., Holz, T.: Exit from Hell? Reducing the Impact of Amplification DDoS Attacks. In: USENIX Security Symposium (2014)
13. Mao, Z.M., Rexford, J., Wang, J., Katz, R.H.: Towards an accurate AS-level traceroute tool. In: Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (2003)
14. Mirkovic, J., Reiher, P.L.: A taxonomy of DDoS attack and DDoS defense mechanisms. Computer Communication Review (2004)
15. Mukaddam, A., Elhajj, I., Kayssi, A.I., Chehab, A.: IP Spoofing Detection Using Modified Hop Count. In: International Conference on Advanced Information Networking and Applications (2014)
16. Oliveira, R.V., Pei, D., Willinger, W., Zhang, B., Zhang, L.: The (in)completeness of the observed internet AS-level structure. IEEE/ACM Trans. Netw. 18(1) (2010)
17. Paxson, V.: An analysis of using reflectors for distributed denial-of-service attacks. Computer Communication Review 31(3) (2001)
18. Pepelnjak, I., Durand, J., Doering, G.: BGP Operations and Security. RFC 7454, RFC Editor (2015), https://tools.ietf.org/html/rfc7454
19. Postel, J.: Internet protocol specification. RFC 791, RFC Editor (1981), https://tools.ietf.org/html/rfc791
20. Postel, J.: Character generator protocol. RFC 864, RFC Editor (1983), https://tools.ietf.org/html/rfc864
21. Rosen, E.C., Viswanathan, A., Callon, R.: Multiprotocol Label Switching Architecture. RFC 3031, RFC Editor (January 2001), http://tools.ietf.org/html/rfc3031
22. Rossow, C.: Amplification Hell: Revisiting Network Protocols for DDoS Abuse. In: NDSS (2014)
23. Ryba, F.J., Orlinski, M., Wählisch, M., Rossow, C., Schmidt, T.C.: Amplification and DRDoS Attack Defense–A Survey and New Perspectives. arXiv preprint arXiv:1505.07892 (2015)
24. Specht, S.M., Lee, R.B.: Distributed Denial of Service: Taxonomies of Attacks, Tools, and Countermeasures. In: International Conference on Parallel and Distributed Computing Systems (2004)