Optimizing Recurrent Pulsing Attacks using Application-Layer Amplification of Open DNS Resolvers

Jonas Bushart
CISPA, Saarland University

Abstract

Shrew attacks or pulsing attacks are low-bandwidth network-level/layer-3 denial-of-service attacks. They target TCP connections by selectively inducing packet loss to affect latency and throughput. We combine the recently presented DNS `CNAME`-chaining attack [5] with temporal lensing [24], a variant of pulsing attacks, to create a new, harder to block attack. For an attack, thousands of DNS resolvers have to be coordinated. We devise an optimization problem to find the perfect attack and solve it by using a genetic algorithm. The results show pulses created with our attack are 14 times higher than the attacker's average bandwidth. Finally, we present countermeasures applicable to pulsing and `CNAME`-chaining, which also apply to this attack.

1 Introduction

Network-layer or volume-based Denial-of-Service (DoS) attacks are a common attack type in which the attacker tries to overwhelm the network connection of the victim, causing high delay, poor performance, or even total unreachability of the victim. Often they make use of reflectors and amplifiers [26], such as DNS or NTP servers. This creates stronger attacks and provides increased anonymity to the attacker. However, it requires spoofing the source IP address, which might be prohibited by the Internet Service Provider [22] or out of the attacker's control, e.g., when using a botnet.

The second type of volume-based DoS attacks are pulsing ones, which have a much lower average bandwidth [16, 17, 18, 19]. This makes them much harder to detect and defend against. In pulsing attacks the traffic consists of many short but high-bandwidth traffic spikes or pulses. However, for TCP connections traversing the same path as the attack traffic, this can be detrimental. The pulses cause packet loss in the TCP connections, making them adapt their congestion window size and reducing their throughput. An attacker who is synchronized with a TCP connection can even stall it completely.

Attackers benefit from combining pulsing attacks with amplification to create stronger pulses. As shown by Rasti et al. [24], using reflectors allows for a technique called temporal lensing, which uses the difference in latencies for different reflectors to create fewer but much stronger pulses. They showed how recursive DNS resolvers are a good candidate for reflectors and performed measurements with 1201 resolvers.

Recently, we published a new application-layer DNS attack [5]. It uses `CNAME` resource records (RRs) to force recursive resolvers into sending many packets to the victim. `CNAME` records work similar to pointers in programming languages in that they allow for arbitrary redirects and have to be resolved one-by-one to retrieve the desired RRs. Filtering this attack is much harder because it looks more like benign traffic and it can even be launched from a botnet, resulting in more attack bandwidth and increased anonymity for the attacker. `CNAME`-chaining already has periodicity in its traffic pattern, making it suitable for repeated pulsing attacks.

In this paper we draw on the two attacks described above and explore how they can be combined. We develop a methodology to describe a measure of success for pulsing attacks. We measure the path latencies for 60 570 open recursive resolvers, a much larger set compared to Rasti et al. [24], and devise an optimization problem to create an efficient and successful attack. The optimization problem is of high-dimensionality, and we build an evolutionary algorithm which allows us to solve it. With that we show the effectiveness of our attack. Reoccurring pulses, created with this attack, are 14 larger than the attacker's average bandwidth. Lastly, we will cover countermeasures related to all three parts of the attack, namely pulsing, temporal lensing, and `CNAME`-chaining.

The remainder of this paper is organized as follows. We start by introducing the background information for pulsing, temporal lensing, and `CNAME`-chaining

in Section 2. We continue with the attack setup and threat model in Section 3. Section 4 describes our model for optimal pulsing and explains how we use evolutionary algorithms to solve the optimal pulsing problem. The evaluation is contained in Section 5 and followed by a discussion about the limitations of our current implementation. Lastly, we provide an overview over countermeasures applicable to pulsing, temporal lensing, and CNAME-chaining in Section 7.

2 Related Work

This section describes some previous work on pulsing Denial-of-Service attacks and CNAME-chaining attacks, both of which are building blocks for our attack scheme. We explain the basics of each of them, as far as it is relevant for this paper.

2.1 Pulsing Denial-of-Service Attacks

Denial-of-Service (DoS) or Reduction-of-Quality attacks on TCP connections can be performed with short traffic pulses. Named as *shrew attack*, these pulsing attacks were first presented in 2003 by Kuzmanovic and Knightly [16, 17] and further analyzed by Luo and Chang [18, 19].

The attack creates short traffic pulses, which fill the buffer of a router and cause packet loss in concurrent connections passing through this router. Packet loss causes two problems for TCP connections. First, TCP exhibits head-of-line blocking as in-order delivery of data is guaranteed. This increases the end-to-end latency because the receiver has to wait for a retransmission of the lost packet. Even with a fast retransmission strategy, it takes at least one round-trip until the retransmitted packet is received. In the case of flows with few packets in total, such as loading of small resources using HTTP, and long round-trip-times (RTTs), this can be a considerable factor for the overall performance. Similarly, if multiple independent resources are multiplexed over a single TCP connection, for example in HTTP/2, a lost packet stalls the independent resources.

The second problem is a bandwidth reduction due to reduced congestion windows. Common TCP algorithms use packet loss as a congestion signal and are very sensitive to packet loss events, especially on high-speed and long distance links [2, 14]. The common TCP algorithms include TCP Reno [1] and Cubic [25], as used in Linux, and Compound TCP [28] used in Microsoft Windows. Inducing packet loss via pulsing attacks, even if there is no permanent link congestion, will lead TCP to reduce the congestion window size and reduce the bandwidth. Scheduling repeated pulsing attacks, before the congestion window recovers, will lead to continuous reduction until it reaches a low steady state.

The effectiveness depends on the time between packet loss (the pulsing interval) and the amount of packet loss. Shorter pulsing intervals leave less time for the congestion window to recover. TCP algorithms can have different responses depending on the amount of packet loss. For small amounts the congestion window is only reduced, while for larger amounts the congestion window is fully reset to the initial size.

2.1.1 Pulsing with DNS

Pulsing requires a high-bandwidth from the attacker, because the attacker's and the victim's bandwidth are directly correlated. To overcome this limitation, Rasti et al. [24] presented in 2015 *temporal lensing*, which trades off long low-bandwidth for a short high-bandwidth pulse. The attacker will pick a reflector to use as a relay to send a packet to the target. A reflector is any host, which can be brought to send packets to a third-party. In the simple case these can be servers which respond to spoofed UDP packets, like DNS or NTP [26, 29]. Each reflector has a different path lengths, such that two packets sent at the same time will arrive at different times at the target. The attacker can measure this difference and adapt their timing while sending. They send the packets at different points in time, such that they arrive at the same time. The attacker uses reflectors with a wide range of path latencies to increase the lensing factor, which is the maximal bandwidth at the target compared to the attacker's one.

Rasti et al. used DNS resolvers as reflectors. They are a good fit due to their abundance, providing a wide range of different path latencies, and are easy to use as reflectors. In their paper they cover single-pulse and multi-pulse attacks. They observe how the retransmission behavior of DNS resolvers leads to multiple pulses.

2.1.2 DNS CNAME-Chaining

So far the average bandwidth at the victim mainly depends on the attacker's bandwidth, with temporal lensing allowing for stronger but fewer pulses. We want to extend pulsing with an amplification attack to allow an attacker to create even stronger pulses. For this we will leverage CNAME-chains, that provide high amplification potential with a factor of 8.5 [5].

DNS allows arbitrary redirects during the resolution process through the CNAME record type, that can even point to other CNAME records. This allows for amplifying the number of requests a resolver has to perform and was used by Dagon et al. [10] and Pfeifer et al. [23] to measure resolver behavior. Recently, we [5] showed

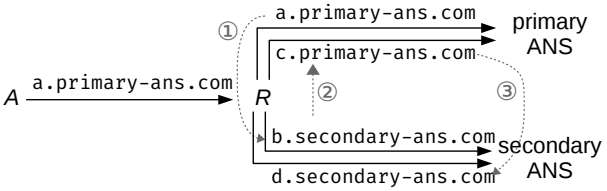

Figure 1: Attacker *A* brings resolver *R* to send multiple queries to each ANS, causing some amplification. The dashed arrows represent the CNAME pointers between the domain, while the circled numbers (①–③) show the CNAME-chain order.

how CNAMEs can be chained for application-level amplification attacks in DNS. In contrast to classical volume-based DoS using DNS the attacker does not require the ability to send spoofed packets. CNAME-chaining is part of the DNS protocol specification and therefore hard to prevent or block. They link two zones together, building CNAME-chains changing the authoritative zone for each redirect. Jumping between different zones prevents resolvers and name servers from optimizing the lookup scheme and forces the resolvers to issue one request for each element in the chain. The whole lookup process is symbolized in Figure 1, where attacker *A* uses a CNAME-chain to bring resolver *R* to send multiple queries to each authoritative name server (ANS). If the chain jumps between two zones, this will create 17 lookups for both zones combined. The 17 lookups hold true for a resolver running a recent version of BIND, other DNS resolvers might have different limits.

In this paper we explore how these two attacks can be combined to increase the effect of pulsing attacks without higher bandwidth requirements for the attacker. The CNAME-chains generate a periodic request pattern at the authoritative name servers, which we leverage to build recurrent pulsing.

3 Threat Model and Attack Setup

We introduced the primitive components used for our attack. We will now describe an attack scenario which will be used for the rest of this paper. On the basis of this scenario we explain our attacker and threat model as well as countermeasures in Section 7.

Our attacker wants to attack a cloud hosted service which uses many long-running TCP connections. The attacker knows or can find out where the servers are physically located. Additionally, they can control a DNS zone file on a server located "close" to the target servers and a DNS zone file on an arbitrary second server. We will refer to them as primary authoritative name server (primary ANS) and secondary ANS. The goal is to cause enough packet loss to severely reduce the throughput of and increase the latency for the target service.

The whole setup is shown in Figure 2. It shows the placement of the primary ANS in relation to the target service. The bottleneck connection will be the congested link which causes the packet loss. As the traffic of the target service has to traverse this link, its TCP flows will be affected. The placement of the secondary ANS is unconstrained. Millions of open resolvers are on the Internet and can easily be found by an attacker.

The attacker has multiple options for finding the location of a server. Often the information can be gained through DNS, e.g., a CNAME to a domain associated with a cloud provider. The IP address of the target service is another good way to find the hosting provider.

For the second requirement, the primary ANS, the attacker could rent a VM from the same hoster. In other cases, buying DNS or web hosting from a provider using the same data center is possible. The important part is the "close" location to the target service, which for this case means that the primary ANS shares part of the path to the Internet with the target service. Sharing a part is important, because this is where the packet loss can be triggered to affect the target.

The secondary ANS can be located almost arbitrarily, except being the same machine as the primary ANS. It will receive considerable fewer queries than the primary ANS as most of them will be cached by the resolver. The ANSs will be configured according to the setup used in our earlier paper [5] by creating two zones with CNAME entries pointing to the zone of the other ANS. This ensures that the attack works on all ANSs by preventing optimizations which would speed up the chain lookup for the resolver.

Besides the ANS setup, a list of reflectors and accurate timing between them and the ANSs is required. The reflectors will be open DNS resolvers or simply *resolvers*. They process queries from any client on the Internet and perform DNS resolution, which forces them to send multiple requests to each ANS. This results in a variable timing between the point a client sends the query and the point when the ANS receives them.

Scanning the Internet on UDP port 53 reveals many open resolvers and can be performed in under an hour [11]. Lists of resolvers can be downloaded [8], if scanning is not viable. Our attacker only needs the ability to send UDP packets. This is in contrast to volume-based reflective distributed DoS attacks [26], which require spoofed source IP addresses for the attack traffic. In our setup the attacker can operate from a single source or even use botnets to increase anonymity and total bandwidth.

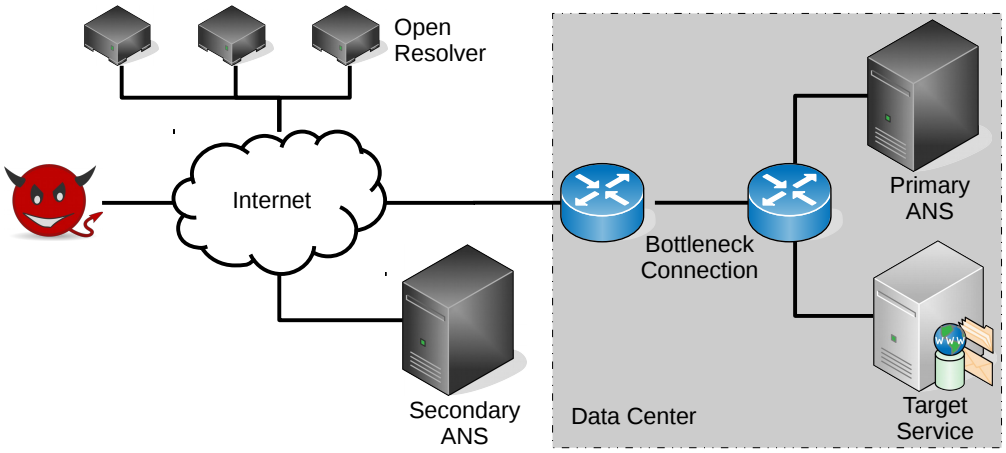

Figure 2: The figure shows how all parts are connected with each other. Important is the placement of the primary ANS, such that it shares the bottleneck connection with the target service. The bottleneck connection will be the congested link causing the packet loss during the pulsing. The placement of the secondary ANS is unconstrained. Open resolvers can be found by scanning the Internet or downloading lists of resolver IP addresses. The icons by VRT Systems are licensed under CC BY-SA 3.0 [9].

3.1 Accurate Timing

For achieving the temporal lensing effect, accurate timing information between all parties, i.e., the attacker, resolvers, and both ANSs is required. They are easy to measure by sending DNS queries to the resolvers and observing the response times. We name the RTT between attacker and resolver $RTT_{A/R}$ and analogously the RTT between resolver and ANS as $RTT_{R/NS}$.

Measuring $RTT_{A/R}$ is accomplished by requesting a cached resource record and measuring the response time. The time between resolver and ANS, denoted as $RTT_{R/NS}$, is measured by sending queries to non-cached resource records, such that the resolver has to fetch the records from the ANS. By subtracting $RTT_{A/R}$ from the measured RTT we gain $RTT_{R/NS}$. The important time is the one between resolver and primary ANS, as these RRs will not be cached and thus will cause traffic. All RRs of the secondary ANS will be cached by the resolver after the first lookup, thus they resolve instantaneously and do not affect the timing calculations.

Halving the RTTs gives the delay from attacker to ANS. This assumes a symmetric delay for the return path, which might not always hold true, but is a good enough approximation [24]. The attacker can now perform a pulse by picking an initial delay d_R depending on the resolvers, such that $\forall R.d_R + (RTT_{A/R} + RTT_{R/NS})/2$ is constant. Sending a packet to resolver R after d_R will result in pulse where all packets arrive at the same time.

4 Pulsing

This section is constructed as follows. We assume the attacker has completed the attack setup which includes measuring the round-trip-times (RTTs) as described in the last section. We then define how an optimal recurrent pulsing attack will look like and describe an optimization problem to create them. Based on this optimization problem of coordinating thousands of reflectors, we develop an evolutionary algorithm which can solve this high-dimensional problem efficiently.

4.1 Optimal Recurrent Pulsing for Chains

Section 3.1 describes how the timing for a single pulse can be calculated. Chaining attacks requires a more complex strategy. Lookup chains started at different resolvers will get out of sync after a few chain elements, due to varying $RTT_{R/NS}$ values. To create several pulses, one can carefully align the chains such that the various request periods add up and exceed the victim's resources. This results in a trade-off between creating pulses (which requires delaying requests) and the average attack bandwidth (which suffers from delays).

We define the goal of recurrent pulsing as exceeding the target's estimated maximum resources as long as possible. The intuition behind this goal is that at all times when the attack traffic exceeds the available resources, packet loss will occur and harm the victim. We model this in an optimization strategy that—given a set of initial per-resolver delays to start the first chain—optimizes the overall time the target's bandwidth is over a certain

threshold. More formally, we define an optimization function f

$$f_T(\mathscr{D}) = \int s_T(t, \mathscr{D}) dt \tag{1}$$

with

$$s_T(t, \mathscr{D}) = \begin{cases} 1 & \text{if } b(t, \mathscr{D}) \geq T \\ 0 & \text{else} \end{cases},$$

where $b(t, \mathscr{D})$ is the attack bandwidth reaching the victim at time t and T is the target's available bandwidth. For each resolver $r \in \mathscr{R}$, the attacker can choose an arbitrary delay d_R with $\mathscr{D} = \{d_R | R \in \mathscr{R}\}$ to maximize $f(\mathscr{D})$. They can then estimate the attack bandwidth $b(t, \mathscr{D})$ by computing how many queries would arrive at the victim—given the per-resolver RTTs $RTT_{A/R}$ and $RTT_{R/NS}$.

BIND is one of the most commonly deployed DNS resolvers [5, 15], therefore to simplify matters we assume all resolvers behave like BIND. It follows `CNAME`-chains for 17 elements in total of which nine will be sent towards the victim. We refer to this as the *chain length limit*. Resource records (RRs) can be marked as non-cachable, by settings the time-to-live (TTL) value to zero. BIND caches such RRs until the internal timestamp counter ticks to the next seconds. After the record has expired, fetching it again takes $RTT_{R/NS}$, so chain restarts can happen every $1\,\text{s} + RTT_{R/NS}$ to ensure that they are never cached by the resolver.

Other resolvers have different chain lengths limit or truly never cache a RR. In practice, an attacker could perfectly fingerprint each resolver software and adapt the maximum chain length and request frequency per resolver. There are resolvers with lower chain length limits, which would result in an overall lower amplification, while the ones with higher limits increase it. Resolvers, which never cache a RR, are beneficial, because they can be scheduled more freely, as chain restarts can happen every $RTT_{R/NS}$ for them.

4.2 Evolutionary Model

While the aforementioned method is optimal, solving the optimization is impractical. Choosing optimal values for d_R quickly becomes intractable, as the number of resolvers increases. There is one delay value to choose per resolver, resulting in an $|\mathscr{R}|$-dimensional optimization problem.

To tackle this complexity, we decided to explore evolutionary algorithms for finding a good (yet not optimal) solution to this optimization problem. Evolutionary algorithms borrow ideas from evolution to build algorithms without much expert knowledge. Genetic algorithms [13], a form of evolutionary algorithms, have demonstrated quick convergence to a good solution for many optimization problems [20]. Technically, genetic algorithms choose an input population and then use mutation, crossover, and re-selection to optimize towards a given fitness (i.e., optimization) function. While *mutation* functions take a genome representation and mutate its values, *crossover* takes two parent genomes and combines them to create a new one.

4.2.1 Description

Before describing our implementation, please note that it is common practice to fine-tune genetic algorithm before applying them on a large data set. Different population sizes, selection strategies, or mutation probabilities can have large impacts on quality of the results and the speed in which the algorithm converges. In the following, we thus present configuration parameters that we obtained after a grid search to optimize for a low convergence time.

Figure 3 provides an overview of our genetic algorithm. An *individual* is the sequence of all resolvers' delay values d_R and is represented as an array of length $|\mathscr{R}|$. We represent the delay values as integer multiples of 1 ms. Each delay is in the range of 0 ms to 1500 ms. Larger values have little influence on the performance of the algorithm. This is caused by the way the chain restarts are handled, because they occur in regular intervals of $1\,\text{s} + RTT_{R/NS}$. Offsets which differ by a multiple of $1\,\text{s} + RTT_{R/NS}$ only differ at the initial start but result in the same alignment. 94.2 % of all measured $RTT_{R/NS}$ values are below 500 ms. For these the maximal delay span of 1500 ms is enough to represent all possible alignments.

The *population* is the collection of all individuals which exist at any point in time. The initial population is seeded by randomly choosing each delay value from a uniform distribution (0 to 1500). Each individual can be mutated or mixed with other individuals in the population during crossover. A larger population provides more diversity in the genomes and allows more combination in crossover. Our population has a constant size of 1250 individuals.

In each iteration of the genetic algorithm, 50 % of the population is selected for mating, creating 25 % (312) new individuals overall. The selection is based on a maximizing selector, selecting the fittest individuals. After applying crossover and mutation, some old individuals are replaced with new ones. All individuals in the population are equally likely to be replaced with new individuals.

Figure 3 gives an overview of our crossover and mutation strategies. We implemented a delta-mutation and a reset-mutation. The delta-mutation occurs with a 30 % chance and alters delays by a random value chosen

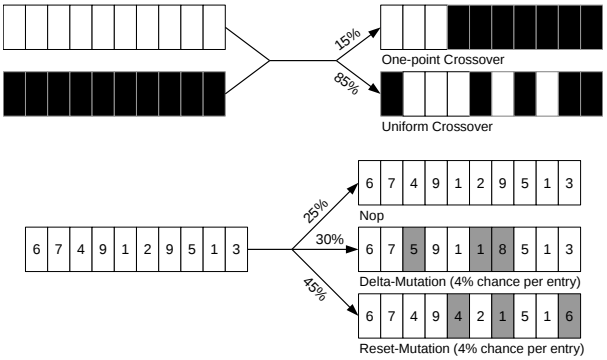

Figure 3: Graphical representation of implemented crossover and mutation strategies. The delta- and reset-mutation have a 4 % chance of mutation per entry, instead of the displayed 30 %.

from a normal distribution with $\mathcal{N}(0, 30^2)$. The reset-mutation occurs with a 45 % chance and replaces delays with a new value chosen uniformly from the range of 0 to 1500. Each individual delay is mutated with a 4 % probability. For the remaining 25 % of cases no mutation is performed.

Crossover is implemented using one-point-crossover and uniform crossover. For one-point-crossover, performed at 15 % chance, a crossover point p in the range $p \in [0; |\mathscr{R}|]$ (the length of the genome) is chosen uniformly. The new genome will consist of the first p delays of the first parent and the remaining $\mathscr{R} - p$ delays from the second parent. The uniform crossover is chosen 85 % of all times and selects each delay with equal probability from either parent.

4.2.2 Optimization Goal

We adapt the optimization function defined in Equation (1) to make it suitable for a fitness function in genetic algorithms. First, we use a discrete time model, thus we measure the target's bandwidth in intervals of 16 ms each. Aggregating the bandwidth in intervals is a performance optimization. Second, the fitness function should incrementally improve for bandwidth values which are *close* to the chosen threshold (i.e., the target's available bandwidth) to favor individuals that work towards the final goal. We define the fitness function as

$$f_T(\mathscr{D}) = \sum_{t=0}^{t_{end}} s_T(t, \mathscr{D}) \tag{2}$$

with

$$s_T(t, \mathscr{D}) = min(\frac{b(t, \mathscr{D})^p}{T^p}, 1),$$

where $b(t, \mathscr{D})$ is again the target's bandwidth, but per 16 ms interval, and T is the target's available bandwidth. $\mathscr{D}$ is the genome which is optimized by the genetic algorithm to maximize $f_T(\mathscr{D})$. This fitness function offers more incentives the closer the bandwidth is to the threshold. We set t_{end} to 1875, which corresponds to a 30 s interval. The power p allows specifying how much the genetic algorithm should favor smaller spikes (the lower p, the more it favors also smaller spikes).

We consider two different chain restart strategies, which we term *greedy* and *self-pulsing*. The *greedy* strategy re-starts a chain as soon as possible, which is the aforementioned $1\,\text{s} + RTT_{R/NS}$ delay between restarts. The *self-pulsing* strategy aligns all chain restarts of the same resolver with itself, such that all packets from the same resolver will arrive at the target at the same time. Restarts happen every $\lceil (1\,\text{s} + RTT_{R/NS})/RTT_{R/NS} \rceil * RTT_{R/NS}$, which is the smallest multiple of $RTT_{R/NS}$ larger or equal to $1\,\text{s} + RTT_{R/NS}$. The idea behind this strategy is that due to the self-pulsing of the chain, the alignment between resolvers becomes easier.

5 Evaluation

To evaluate how effective such pulsing might become, we created the following measurement setup. We measured the RTT between resolvers and two authoritative name servers (ANSs) as described in the beginning of Section 4. Our primary ANS was located in Germany, while the less important secondary ANS was located on the US east coast. We primed the resolvers' caches to ensure reliable RTT measurements to the resolvers and set the *recursion desired bit* to false. The recursion desired bit instructs a resolver to only return an answer if one is available in the cache, and the resolver should not fetch it from the ANS. Randomizing the queries to measure the RTT between resolver and ANS ensures resolvers need to forward the query to the ANS. The measurements were repeated five times to gain reliable measurements and to calculate the standard deviation.

For more stable timing measurements, we only considered resolvers that are not forwarders. Forwarders have an additional layer of DNS resolvers to the ANS, making the measured RTTs unreliable. In order to gather such a list, we conducted full IPv4 scans, fingerprinting each resolver during the scan and observing which resolvers contact the ANS. This way we can differentiate between a resolver directly contacting our ANS or via some other resolver. We followed the scanning best practices as outlined by Durumeric et al. [11]. Not all resolvers provided usable data, e.g., because the resolver was not reachable during the whole measurement period. Other resolvers disabled queries with recursion-desired bit set to false and thus our methodology could not determine the RTT from us to the resolver. Excluding those resolvers, overall we gathered reliable RTT data for 60 570 resolvers.

We can evaluate this approach by checking how well the evolutionary algorithm performs while finding a solution. Completing the genetic algorithm is expensive, due to the high number of resolvers. The genetic model parallelizes well and can use all available CPUs. This allows calculating the model using 24 cores of a Xeon E5-2667 server in two hours. Our genetic algorithm stops after 5000 iterations, which we found to be sufficient to converge on a good solution.

After the genetic algorithm terminated, we relate the performance of its output (i.e., the best individual) with the naive attack in which all chains starts immediately (all d_R are 0). Figure 4 compares the packet rates the attacker sends (upper graphs) and the victim receives (lower graphs) for the naive approach (left), the genetic algorithm with $p = 2$ (middle) and $p = 8$ (right).

Estimating absolute bandwidth numbers, such as kbit/s, is very hard, as it depends on the multiple factors we do not control or left undefined in our setup. Different factors are the length of the domain name, EDNS support and options used by the resolvers, and the chosen transport protocol (UDP or TCP). We therefore specify the bandwidth in queries per second (q/s). A short DNS query (no EDNS, using UDP) requires roughly 100 B to transmit including the inter-packet gap of Ethernet.

While the naive approach also shows some pulsing behavior (due to resolvers sharing similar $RTT_{R/NS}$), (i) the pulses decay after a few seconds, and (ii) the naive solution requires the attacker to start all chains virtually simultaneously—which is unrealistic in practice. In contrast, the solutions found by the genetic algorithm are practical, i.e., require a reasonably constant sending rate from the attacker. This is a positive side effect of this approach, even though the fitness function actually does not favor a uniform attacker bandwidth. Furthermore, the genetic algorithm creates slightly larger and significantly more steady spikes than the naive approach. This effectively means that an attacker can create reoccurring pulses that are 14 times higher than its average sending rate.

The original temporal lensing experiment and results are quite different. We showed that temporal lensing is viable with a high attacker bandwidth and very low per-resolver bandwidths. Rasti et al. [24] only achieved high (> 10) amplification rates if both (i) the attacker's maximum bandwidth is severely restricted ($< 10\,000$ q/s) and (ii) the maximum bandwidth to any resolver is at least 100 times larger compared to our setup. Given an identical per-resolver bandwidth of 1 q/s in both experiments, we achieve a factor of 14 while Rasti et al. report no amplification. As such our experiment improves results given many more resolvers and lower per-resolver bandwidths.

Finally, we compared the performance of the greedy restart compared to self-pulsing. Self-pulsing has longer pauses between restarts, reducing the overall average bandwidth by 5.4 %. We noticed little influence on the genetic algorithm between those two strategies. The naive approach profited slightly from this change, as the naturally occurring pulses are amplified, due to the consistent alignment between chains of different resolvers. In the greedy approach, the alignment between chains of different resolvers becomes distorted over time and flattens the occurring pulses.

6 Discussion

We discuss the limitations and how our model could be extended to improve the shortcomings. Our first simplification concerns the time model we use. We assumed all measured times are constant and based on that decision chose a discrete time model. Network packets experience jitter resulting in varying RTT values. This can be modeled by using a probabilistic model. Instead of assigning a single time to each packet, the packet is represented as a probability distribution of when it arrives.

A second limitation regards redundancy in DNS. DNS requires at least two authoritative name servers (ANSs) configured for a zone and resolvers are free to choose between all available ANSs. If the primary ANS is under full control, both configured ANSs could use the same IP, thus the same machine, and reduce this uncertainty.

If the primary ANS is a hosted service, the hoster will have multiple physical machines configured. Measurements of the server selection algorithm [21, 31] have shown that most resolvers prefer the most responsive (i.e., lowest RTT) ANS but most will use even the slower ANSs for some fraction of all queries. Resolvers use it to re-check the RTT of all available ANSs. This makes it impossible to predict exactly when a packet arrives at the target ANS, as the resolver might use a different ANS.

A general solution to combat the uncertainty of timing is to have re-synchronization points. At these points, the attacker stops sending queries to the resolvers and waits. This ensures that in-flight queries of resolvers will be received and cache entries can time out. After some time, the attacker can restart the attack, which will be identical to the initial attack. Additionally, this allows an attacker to re-use the computed delay values, as the timings simply repeat.

Lastly, our genetic algorithm implementation so far has only optimized the initial delays. We have discussed greedy and self-pulsing strategies to restart the chains. Ideally, the timing between restarts of the chain could be optimized as well. Other values might yield better results, especially if the wait time is calculated for every restart. However, optimizing the restart times would

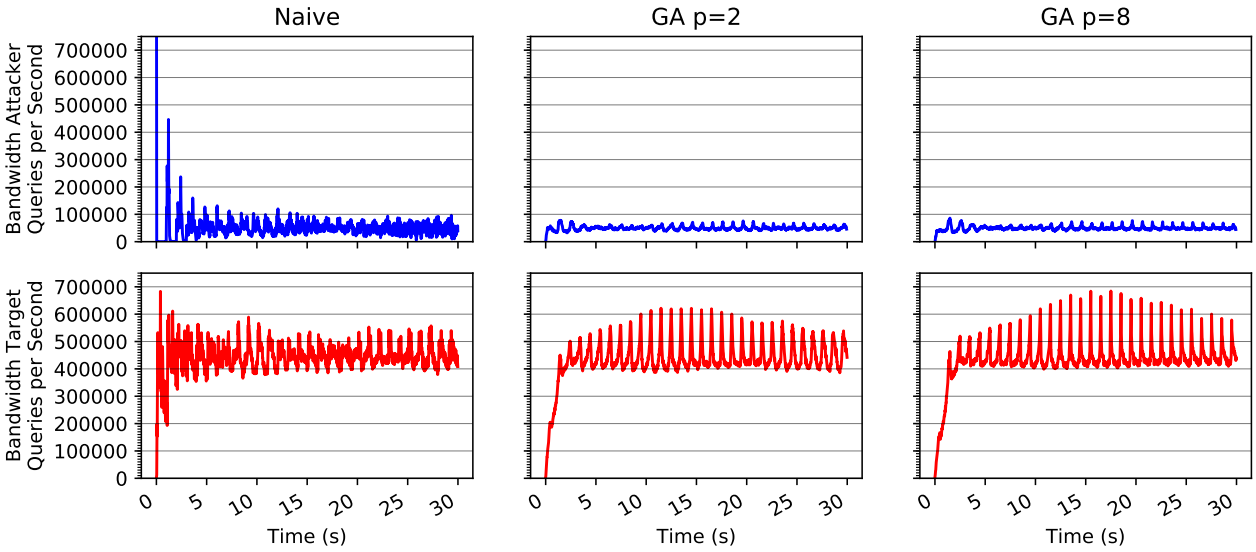

Figure 4: Bandwidth comparison of the genetic algorithm's solutions compared to a naive one, which starts all chains as soon as possible. The upper graphs are the outgoing bandwidth of the attacker, while the lower graphs are the incoming bandwidth of the target. The time values between attacker and target graphs are independent and start at zero on the first packet sent or received.

multiply the dimensions in our search space and would require more advanced strategies to scale.

7 Countermeasures

Our attack consists of the three parts (i) pulsing (ii) temporal lensing and (iii) `CNAME`-chains. There are different countermeasures applicable for each of the parts.

7.1 Pulsing and Packet Loss

Common TCP variants, like TCP Reno [1] and Cubic [25], use packet loss as the main congestion signal. Previous work [12, 16, 30] proposed randomizing the retransmission timeout (RTO) and extending buffer sizes [27].

Pulsing has the most detrimental effect, if the attacker is synchronized with the victim. Causing packet loss on every retransmission effectively stalls the connection. Randomizing the RTO prohibits the attacker from being synchronized, thus reducing the effects.

Larger buffers will cause less packet loss, all else being equal. Extending buffers requires hardware changes to the network routers which makes it costly. The additional buffer space might be used by the existing flows, thus negating the effect. It can even increase the RTT for flows, thus having a negative effect under normal operations.

There are TCP algorithms, which use other feedback mechanisms as their congestion signal; an older variant of it being TCP Vegas [4]. However, TCP Vegas is being suppressed by algorithms with loss-based congestion control. Recently TCP BBR [6, 7], short for bottleneck bandwidth and RTT, was presented, which uses RTT information as congestion signal. TCP BBR ignores packet loss up to a threshold, thus being more resilient to slight packet loss.

The second problem of TCP is the head-of-line blocking it creates after packet loss. This does not necessarily affect the throughput by much, but the end-to-end latency. Where applicable, for example with HTTP requests, connections with true multiplexing help. One candidate for this is QUIC [3]. It can use UDP and multiplexes requests and responses through the UDP connection. Packet loss here will only affect the data where a packet was lost.

7.2 Temporal Lensing

Temporal lensing requires precise timing information between all involved hosts. Any disturbance, such as network jitter, will make the attack less effective. Jitter could be introduced by the network routers. Care needs to be taken, that packets of the same flow do not overtake each other, as such, having a constant jitter per flow 5-tuple would work [24].

Another approach would be to introduce the jitter on the DNS servers. This would not be a generic solution, as other kinds of reflectors can be used, but it would not have the problem of packets overtaking each other.

The problem with artificial jitter is, that it unnecessarily slows down all applications on the Internet. It forces higher memory and CPU consumption as packets have to be stored and re-scheduled for later processing.

7.3 DNS and CNAME-Chains

Countermeasures against CNAME-chains as presented in [5] also apply here. These can be summarized into preventative measures and mitigation ones. Authoritative name servers could prevent loading zones with too long chains. Since this is not forbidden by the DNS specification, this is more a solution for managed DNS hosting providers.

Resolvers should not follow CNAME-chains too long. This does not prevent the amplification, but at least reduces its impact. Benign chains are only up to nine elements, which is the shortest supported length of common resolvers [5, 23]. In contrast, some resolvers support lengths of up to 33 elements. These resolvers provide a huge amplification potential which can be reduced without impacting deployed DNS setups. Resolvers might also consider minimal Time-to-Live (TTL) for DNS records, such that caching can take effect for reducing the number of queries. However, there are legitimate reasons to use low TTL values, such as load balancing, so this is also not a general purpose solution.

8 Conclusions

Recurrent pulsing provides a new way of performing distributed denial-of-service attacks against TCP connections. Leveraging a DNS application-layer amplification attack allows for traffic pulses 14 times higher than the average attacker bandwidth, while creating thousands of low bandwidth and hard to block flows.

We developed a genetic algorithm to solve the challenge of coordinating an attack between tens of thousands of reflectors, while still being efficiently computable. Our implementation shows the feasibility of launching this attack and the high amplification ratio achievable.

We present a wide range of potential countermeasures ranging from changes to networking equipment, like larger buffer sizes, over protocol design, such as different TCP algorithms, to DNS specific countermeasures.

In the future, we want to extend our implementation by building a more realistic network model, which can factor in jitter. By giving the attacker more control over when chain restarts happen, we likely can improve the pulsing factor.

Acknowledgment

We thank our anonymous reviewers whose useful comments helped us improving the quality of our paper. This work was supported by the German Federal Ministry of Education and Research (BMBF) through funding for the BMBF project 16KIS0656 (CAMRICS).

References

[1] ALLMAN, M., PAXSON, V., AND BLANTON, E. TCP congestion control. Tech. rep., RFC Editor, 2009.

[2] ALLMAN, M., PAXSON, V., AND STEVENS, W. R. TCP congestion control. Tech. rep., RFC Editor, 1999.

[3] BISHOP, M. Hypertext transfer protocol (http) over quic. Internet-Draft draft-ietf-quic-http-12, IETF Secretariat, May 2018.

[4] BRAKMO, L. S., O'MALLEY, S. W., AND PETERSON, L. L. TCP vegas: New techniques for congestion detection and avoidance. In *Proceedings of the ACM SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications* (1994).

[5] BUSHART, J., AND ROSSOW, C. DNS Unchained: Amplified Application-Layer DoS Attacks Against DNS Authoritatives. In *Proceedings of the 21th International Symposium on Research in Attacks, Intrusions and Defenses* (Sept. 2018).

[6] CARDWELL, N., CHENG, Y., GUNN, C. S., YEGANEH, S. H., AND JACOBSON, V. BBR: congestion-based congestion control. *Commun. ACM* (2017).

[7] CARDWELL, N., CHENG, Y., YEGANEH, S., AND JACOBSON, V. Bbr congestion control. Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-00, IETF Secretariat, July 2017.

[8] Censys DNS lookup full IPv4, Apr. 2017. https://censys.io/data/53-dns-lookup-full_ipv4.

[9] Creative Commons Attribution-ShareAlike 3.0 Unported, May 2018. https://creativecommons.org/licenses/by-sa/3.0/.

[10] DAGON, D., ANTONAKAKIS, M., DAY, K., LUO, X., LEE, C. P., AND LEE, W. Recursive DNS architectures and vulnerability implications. In *Proceedings of the Network and Distributed System Security Symposium* (2009).

[11] DURUMERIC, Z., WUSTROW, E., AND HALDERMAN, J. A. Zmap: Fast internet-wide scanning and its security applications. In *Proceedings of the 22th USENIX Security Symposium* (2013).

[12] EFSTATHOPOULOS, P. Practical study of a defense against low-rate tcp-targeted dos attack. In *Proceedings of the 4th International Conference for Internet Technology and Secured Transactions* (2009).

[13] EIBEN, A. E., AND SMITH, J. E. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 2015.

[14] FLOYD, S. Highspeed TCP for large congestion windows. Tech. rep., RFC Editor, 2003.

[15] KÜHRER, M., HUPPERICH, T., BUSHART, J., ROSSOW, C., AND HOLZ, T. Going wild: Large-scale classification of open DNS resolvers. In *Proceedings of the 2015 ACM Internet Measurement Conference* (2015).

[16] KUZMANOVIC, A., AND KNIGHTLY, E. W. Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants. In *Proceedings of the ACM SIGCOMM 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (2003).

[17] KUZMANOVIC, A., AND KNIGHTLY, E. W. Low-rate TCP-targeted denial of service attacks and counter strategies. *IEEE/ACM Trans. Netw.* (2006).

[18] LUO, X., AND CHANG, R. K. C. On a new class of pulsing denial-of-service attacks and the defense. In *Proceedings of the Network and Distributed System Security Symposium* (2005).

[19] LUO, X., AND CHANG, R. K. C. Optimizing the pulsing denial-of-service attacks. In *2005 International Conference on Dependable Systems and Networks* (2005).

[20] MAN, K., TANG, W. K., AND KWONG, S. Genetic algorithms: concepts and applications [in engineering design]. *IEEE Trans. Industrial Electronics* (1996).

[21] MÜLLER, M., MOURA, G. C. M., DE OLIVEIRA SCHMIDT, R., AND HEIDEMANN, J. S. Recursives in the wild: engineering authoritative DNS servers. In *Proceedings of the 2017 Internet Measurement Conference* (2017).

[22] P. FERGUSON, D. SENIE. BCP 38 on network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing. `http://tools.ietf.org/html/bcp38`, May 2000.

[23] PFEIFER, G., MARTIN, A., AND FETZER, C. Reducible Complexity in DNS. In *Proceedings of the IADIS International Conference WWW/Internet* (2008).

[24] RASTI, R., MURTHY, M., WEAVER, N., AND PAXSON, V. Temporal lensing and its application in pulsing denial-of-service attacks. In *2015 IEEE Symposium on Security and Privacy* (2015).

[25] RHEE, I., XU, L., HA, S., ZIMMERMANN, A., EGGERT, L., AND SCHEFFENEGGER, R. CUBIC for fast long-distance networks. Tech. rep., RFC Editor, 2018.

[26] ROSSOW, C. Amplification hell: Revisiting network protocols for ddos abuse. In *21st Annual Network and Distributed System Security Symposium* (2014).

[27] SARAT, S., AND TERZIS, A. On the effect of router buffer sizes on low-rate denial of service attacks. In *Proceedings of the 14th International Conference On Computer Communications and Networks* (2005).

[28] SRIDHARAN, M., TAN, K., BANSAL, D., AND THALER, D. Compound tcp: A new tcp congestion control for high-speed and long distance networks. Internet-Draft draft-sridharan-tcpm-ctcp-02, IETF Secretariat, November 2008.

[29] US-CERT. Alert (TA14-017A) UDP-based amplification attacks, Mar. 2018. `https://www.us-cert.gov/ncas/alerts/TA14-017A`.

[30] YANG, G., GERLA, M., AND SANADIDI, M. Y. Defense against low-rate tcp-targeted denial-of-service attacks. In *Proceedings of the 9th IEEE Symposium on Computers and Communications* (2004).

[31] YU, Y., WESSELS, D., LARSON, M., AND ZHANG, L. Authority server selection in DNS caching resolvers. *Computer Communication Review* (2012).