ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Master's Thesis

Network Security Group, Department of Computer Science, ETH Zurich

Understanding QMIN-enabled DNS Amplification

by Marco Bearzi

Autumn 2022

ETH student ID:	15-823-560
E-mail address:	marco.bearzi@epfl.ch, mbearzi@student.ethz.ch
Supervisors:	Dr. Huayi Duan, Dr. Si Liu Prof. Dr. Adrian Perrig, Prof. Dr. Mathias Payer
Date of submission:	27th of January 2023

Abstract

The Domain Name System (**DNS**) is a core component of the Internet, performing the critical function of resolving human-friendly names to numeric IP addresses. However, DNS poses serious risks to users' privacy, as the information-rich queries can be seen, collected, and analysed by all servers involved in the name resolution process. Query Name Minimisation (QMIN) is a key privacy-enhancing feature for DNS [16]. It reduces information disclosure by having a recursive resolver sending only necessary labels of a queried name to authoritative name servers. Since its introduction in 2016, QMIN has been implemented by all major DNS software and deployed by many large DNS operators.

Whereas QMIN improves users' privacy, it increases the number of "probing" queries sent by a recursive resolver and hence creates new denial-of-service (DoS) attack vectors. While this threat has been briefly discussed in the DNS community, a thorough examination is still lacking. In this project, we aim to fill this gap and investigate the full amplification potential of QMIN, in particular its composibility with other DoS vulnerabilities including different types of chained records, parallel referrals, and unresponsive servers. Our results demonstrate that QMIN is a universal amplifier which can be combined with all known attack vectors to produce multiplicative effect. We validated our attacks on the most popular resolver implementations (BIND, Unbound, and PowerDNS): all of them are vulnerable, and one powerful attack achieves an amplification factor (AF) of 1700 on Unbound. Furthermore, our measurement of 20 common open resolvers show that most of them are susceptible to our attacks, with the worst suffering from an AF over 1200.

These findings are concerning given the scale and importance of DNS. We have been actively working on disclosing these new vulnerabilities to the involved development and operation teams and the DNS community in general.

Acknowledgements

First of all, I want to thank my supervisors Pr. Mathias Payer and Pr. Adrian Perrig for accepting to supervise my thesis and welcoming me in the Network Security group.

I also want to thank my advisors, Dr. Si Liu and Dr. Huayi Duan, who worked closely with me on the project. Our discussions and meetings brought me guidance and helped me prioritise the elements to work on.

Thanks should also go to Jodok Vieli, a fellow student, who worked by me on a related project and who showed great friendship during my thesis.

I would like also to acknowledge my family and especially my parents, Françoise and Paolo, who always encourage me to give my best.

My last thanks will go to my girlfriend Sarah, who supported me along the way and gave me excellent advice on the writing of the thesis.

Contents

1 Introduction

The Domain Name System (**DNS**) is a critical system that helps accessing websites, services and other infrastructures on the Internet. DNS provides users, automated processes and servers with the necessary information to establish a connection to the desired destination endpoint. It does so by mapping names of human-readable addresses to IP addresses which then allows the localization of the service.

DNS plays a crucial role in the Internet as almost any user request to access a resource is preceded by DNS lookups. It is a hierarchical and distributed system, fulfilling the availability condition without which the access to Internet would fall apart. The DNS root domain is at the top of this hierarchy and is served by thirteen logical servers, and several copies of those. When queried, these root servers will delegate the resolution of the queries to nameservers further down the hierarchy.

The servers directly below the root name servers are called the top level domain (**TLD**) nameservers, and are authoritative for domains such as ".com", ".org" or ".net". They manage these subdomains of the root domain and provide information to localize nameservers authoritative for further down domains.

A *zone* is a delimitation between the domain that the nameserver is authoritative of and its subdomains administered by other servers. All data specific to a zone is not contained into a single name server, it is shared between several of them. Multiple zones can co-exist on a nameserver. If one server fails, there should always be at least another one that ensures that users can obtain IP addresses or associated data related to the zones previously served by the failed server.

Due to its importance, DNS is a prey of choice for the attacker. If the servers authoritative for the requested zones are not available or if all their resources are consumed, they cannot answer client queries, which disrupts the resolution process and prevents legitimate users from obtaining or connecting to the destination resources.

There exist several ways to abuse DNS, one of them is a type of attack called Denial-of-Service (**DoS**). Its objective is to either disconnect the targeted machine from the network or to disrupt the communication by overloading the machine's capacity. Some examples are :

- Bombard it with queries which leads to resource exhaustion
- Carefully craft queries and their content so that they excessively consume resources of the victim

- Amplification via reflection, this one may target a victim that is not a DNS nameserver but forces a nameserver to send a considerable number of messages to the victim

Since its creation, DNS has been the target of many attacks such as DoS attacks, and has been compromised a large number of times using certain vulnerabilities in its core specifications. To mitigate those, numerous modifications to the DNS structure and algorithms improvements have been constantly carried out. Most of these additional changes consisted of improving the security of DNS and its communication protocols. And fewer were directed to increase the user privacy. One significant exception is Query Name Minimisation.

Query Name Minimisation (QMIN) is a feature that has been specified in 2016 in the standard RFC 7816 [15], obsoleted by RFC 9156 [16], and whose objective is to enforce and raise privacy in user requests. Its adoption rate has been steadily increasing among open resolvers [19].

When a user would like to access a resource via its name, it contacts a recursive resolver that will take care of querying multiple DNS nameservers in order to resolve the user's requests. The goal of the QMIN mechanism is to prevent the DNS nameserver which the resolver is contacting from knowing the whole query name requested by the user via partial queries with fewer labels sent by the resolver. The user can choose the resolver but has no say in the nameservers that will be queried. This feature is, thus, also a response to the lack of user choice.

The major drawback of this feature is that the resolver will send, for most of the time, more queries to the nameservers as for the conventional name resolution, and such behaviours inherently lead to possible amplification attacks. Furthermore, combining DoS attacks with the fact that targeted resolvers have implemented QMIN could lead to multiplying the effect of these attacks. Nevertheless, some nameservers may have enforced limits on some aspect of query resolution and it might as well reduce the potential amplification of QMIN-enabled attacks. Up to which point does QMIN make the nameserver more vulnerable? What are the amplification factors of attacks enhanced with QMIN? What are the reactions of open resolvers against those attacks? Are those attacks effective in the "wild"?

The objectives of this project are to study more extensively the QMIN mechanism, to observe its vulnerabilities, and to exploit them in attacks against open resolvers to demonstrate its potential amplification factor. To this purpose, attack scenarii exploiting this feature will be constructed in a formal model of DNS. Automatization of the creation of the attack files and the tuning of their parameters will be implemented, serving to evaluate their potential amplification factor. Then a local testbed will be set up and instances of open resolvers will be generated. Previously validated attacks will be reproduced in this environment. They will then be launched against the resolvers and their results will be examined. We also perform measurements of the attacks with controlled authoritative nameservers

against mainstream DNS software over the Internet in order to report their impact over a large number of popular resolvers.

These implementations are modular and allow to study even more open resolvers and their behaviours in details against mounted attacks in a controlled environment and in a more open setting.

1.1 Thesis Structure

The thesis is divided into the following chapters : in the Chapter 2 Background, we discuss the structure and resolution process of DNS and explain which formal DNS model we use for the analysis of theoretical results. We will broaden the description of QMIN in Chapter 3 and explain more in details its specifications.

In Chapter 4, we explain the methodology of our experiments and analyse the amplification factor resulting from the QMIN-enhanced attacks against models and real resolvers instantiated within a local testbed. In Chapter 5, we present measurements conducted "in the wild" about our attacks against open resolvers and whether these are vulnerable. A methodology of the setup will also be explained.

We continue with discussing the results of our study on QMIN in Chapter 6. Related works are described in Chapter 7. And we conclude with ideas for future investigations in the chapter 8.

2 Background

In this chapter, we describe the essential structures and concepts that are used as a basis of the necessary knowledge to understand the project and how it all relates to the Domain Name System.

And upon this basis, we show multiple attacks which exploit some flaws in DNS and can be further enhanced by the Query Name Minimisation feature.

Furthermore, to conduct preliminary experiments, we leverage a formal model of DNS that can create entities triggering a query resolution in a controlled environment.

2.1 Domain Name System

DNS stands for Domain Name System and is a process of uttermost importance in our current daily routine. Whenever we make research, watch videos, want to connect to our accounts or do some online shopping, the flux of all these communications could not be possible in practice without first a call to DNS. To establish a connection to a resource, all that is needed is a numeric IP address.

As machines, devices, and users cannot keep in memory an exhaustive list of all the IP addresses (also owners of IPs may change over time), the necessity of building a system accomplishing this task is fulfilled with DNS. This service provides a mapping of names to the corresponding destination address IP and other associated data.

2.1.1 Hierarchy of DNS

DNS is a hierarchical and distributed system with a common root zone, symbolized by ".". A DNS zone is a representation of a portion of the DNS namespace, which is handled and operated by organisations or individuals. For example, a DNS zone or domain could be "example.com." and one of its subdomains could be "attack.example.com.". Multiple zones can be installed in one server, which is authoritative for them, meaning that the server should answer queries concerning all those zones and their subdomains that they are authoritative of.

One concern in the resolution process is to find the *zone cut* : a zone cut is a way to delimit zones and divides the parent zones (before the cut) and the child

zone (after the cut). For example, with the domain "a.b.c.d.example.com.", one has to determine whether the domain resides in the same zone as "example.com.". If "c.d.example.com." does not belong to the same zone, there is a *zone cut*, delegating authority to this new zone which, in this case, contains data for "a.b.c.d.example.com". The resolver will thus continue the resolution process with this new zone cut. Note the dot at the end of the domains, it illustrates the fact that the common root zone is a parent of "example.com.". This symbol is usually implicit and is most of the time omitted.

The DNS namespace is abstracted as a tree-like graph with "." at the highest level DNS zone, then right below it stand the Top-Level Domains (or TLDs) with zones such as ".com.", ".org." or ".ch.". And further down, we find second-level domains and further subdomains (see Fig 2.1). This structure allows for a fine-grain control of the domains and subdomains.

The root zone is served by 13 root server clusters which are responsible for queries to the top-level domains. Therefore, almost every DNS resolution process starts with a query to one of the root nameservers or uses information previously collected by contacting one of them. As they may fail, due to hardware issues or to attacks, there are various copies of them scattered across the world. Below them are servers authoritative for the TLDs, those name servers contain zones for .com, .org, .ch and can reroute the queries to a closer name server down its domain.

Each authoritative nameserver will either, answer the query and return the IP addresses or the desired resource, or delegate the resolution to another name server.

The goal of the DNS is to obtain a response to the DNS request by translating the query name to an IP address or to provide the requested resources.

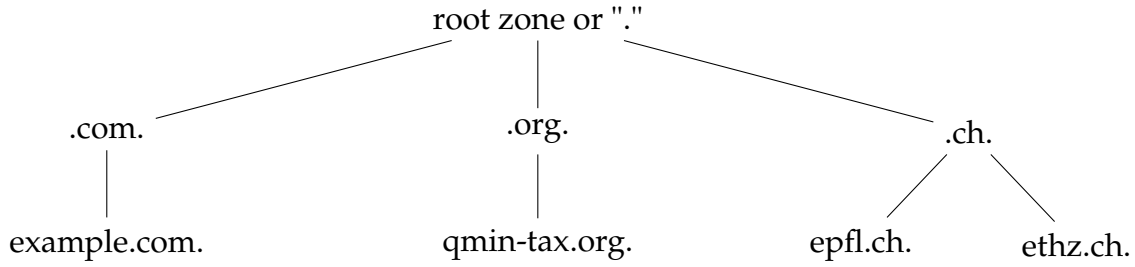

Figure 2.1: Simple diagram of DNS Hierarchy

Resource Records

The DNS data contained in the nameservers is called *Resource Records* (*RRs*) [10]. A resource record is a line in the database of the server constituted of 6 components:

```
1    <NAME> <TYPE> <CLASS> <TTL> <RDLENGTH> <RDATA>
```

- NAME is the name of the node or the owner name of the record. It is built by an ordered sequence of labels (string) divided by dots.
- TYPE can be either of data-type, query-type or meta-type.
- CLASS can simply contain dataclass or be specific to queries. Typically, its value is "IN" for "Internet".
- TTL is the "Time-To-Live" or the time the resource should be cached before being looked up again.
- RDLENGTH is an optional field stating the length of RDATA.
- RDATA is the resource or the target of the record, made up of a string of octets.

A zone file contains all the RRs that are enclosed in the zone. The following resource record (Listing 2.1) shows that, if there is a request for A record (see below) of "www.sub.example.com" to a namserver containing this RR, it will be answered with the IP address 1.2.3.4 and be cached in the resolver for 1 hour (3600 seconds). Note that some fields may be omitted, either because they were defined earlier in the zone file or because they are optional.

```
1    www.sub.example.com     A       IN      3600        2       1.2.3.4
```

Listing 2.1: A simple A resource record in a zone file

2.1.2 Relevant Record Types

Multiple record types are listed in the DNS core specifications and they are used for different purposes. The most common records are "A" and "AAAA" records. Both are QTYPE (or Query Type). The former holds IPv4 addresses while the latter contains IPv6 addresses. We will focus on only three, namely NS, CNAME, and DNAME, as those are the ones we will exploit during our attacks.

Name Server record or NS record is the type of record a name server will return when it specifically informs the resolver that another authoritative name server can answer the query or is closer to the response than itself. In other words, the resolver should continue the querying process by querying this *delegated* name server with the original query. Thus, a NS record does not imply a rewriting. However, the referred name server may not be in the resolver cache and there may be no associated IP address added with the referral (this is called a *glueless delegation*). So the resolver has to send subqueries to resolve first the nameserver's IP address. This increases the workload of this DNS lookup. If there is associated data containing the IP address in the queried nameserver, the additional section of the response may contain this A record and it is called a *glue record*.

CNAME stands for "Canonical Name" and DNAME stands for "Delegation Name". Both those types will induce a rewriting in the resolver query by substituting the whole or parts of the query name. A CNAME record is actually

composed of an owner name (queried name), which is the *alias*, and the target that is the canonical name. The alias will be replaced by the canonical name. Whereas DNAME will substitute a *strict suffix* of the owner name with the target or value contained in the RDATA field.

2.1.3 Resolution Process

In this section, we assume no limits enforced in the resolver and in the name server. This shows the reactions of a plain resolver with almost no specific configuration. The following listing (Listing 2.2) shows the algorithm specified in the RFC 6672 [32] for the DNS nameserver related to the resolution algorithm. This will help to understand how the queried nameserver is functioning and what records it would return when it receives a particular query.

Listing 2.2: Per RFC 6672, 3.2. Server Algorithm

```
1.  Set or clear the value of recursion available in the response depending on whether
    the name server is willing to provide recursive service. If recursive service is
    available and requested via the RD bit in the query, go to step 5; owise, step 2.

2.  Search the available zones for the zone which is the nearest ancestor to QNAME.  If
    such a zone is found, go to step 3; otherwise, step 4.

3.  Start matching down, label by label, in the zone.  The matching process can
    terminate several ways:

    A.  If the whole of QNAME is matched, we have found the node.

        If the data at the node is a CNAME, and QTYPE does not match CNAME, copy the
        CNAME RR into the answer section of the response, change QNAME to the canonical
        name in the CNAME RR, and go back to step 1.

        Otherwise, copy all RRs which match QTYPE into the answer section and go to
        step 6.

   B.  If a match would take us out of the authoritative data, we have a referral.
       This happens when we encounter a node with NS RRs marking cuts along the bottom
       of a zone.

       Copy the NS RRs for the sub-zone into the authority section of the reply. Put
       whatever addresses are available into the additional section, using glue RRs if
       the addr are not available from authoritative data or the cache. Go to step 4.

   C.  If at some label, a match is impossible (i.e., the corresponding label does not
       exist), look to see whether the last label matched has a DNAME record.

       If a DNAME record exists at that point, copy that record to the answer section.
       If substitution of its <target> for its <owner> in QNAME would overflow the
       legal size for a <domain-name>, set RCODE to YXDOMAIN [RFC2136] and exit;
       otherwise, perform the substitution and continue.  The server MUST synthesize a
       CNAME record as described above and include it in the answer section.
       Go back to step 1.

       If there was no DNAME record, look to see if the "*" label exists.

       If the "*" label does not exist, check whether the name weare looking for is the
       original QNAME in the query or a name we have followed due to a CNAME or DNAME.
       If the name is original, set an authoritative name error in the response and
       exit.  Otherwise, just exit.
```

```
42
43          If the "*" label does exist, match RRs at that node against QTYPE. If any match,
44          copy them into the answer section, but set the owner of the RR to be QNAME, and
45          not the node with the "*" label. If the data at the node with the "*" label is a
46          CNAME, and QTYPE doesn't match CNAME, copy the CNAME RR into the answer section
47          of the response changing the owner name to the QNAME, change QNAME to the
48          canonical name in the CNAME RR, and go back to step 1.  Otherwise, go to step 6.
49
50    4.  Start matching down in the cache.  If QNAME is found in the cache, copy all RRs
          attached to it that match QTYPE into theanswer section.  If QNAME is not found in
          the cache but a DNAME record is present at an ancestor of QNAME, copy that DNAME
          record into the answer section.  If there was no delegation from authoritative data
          , look for the best one from the cache, and put it in the authority section.  Go to
           step 6.
51
52    5.  Use the local resolver or a copy of its algorithm to answer the query.  Store the
          results, including any intermediate CNAMEs and DNAMEs, in the answer section of the
           response.
53
54    6.  Using local data only, attempt to add other RRs that may be useful to the
          additional section of the query.  Exit.
55
56    Note that there will be at most one ancestor with a DNAME as described in step 4 unless
57    some zone's data is in violation of the no-descendants limitation in Section 3.  An
58    implementation might take advantage of this limitation by stopping the search of step
59    3c or step 4 when a DNAME record is encountered.
```

Resolution without rewriting

An example of a simple resolution without rewriting is described in RFC 7626 [14]: a client, or stub resolver, wants to know the IP address of "www.example.com". To do so the client will send a DNS query to another server, looking for A records of "www.example.com" which is also the QNAME (or Query Name). This server will be responsible for performing the complete resolution of the query by sending the necessary requests to other name servers. It is called the resolver server or *recursive resolver*, and it is the only server the client directly interacts with during the whole resolution.

Assuming an empty cache, the recursive resolver will first contact the root name servers, whose IP addresses are usually hard-coded in the resolver. One of them will answer with a NS referral to a ".com server", thus, delegating the resolution to a closer server. The resolver will then ask the TLD ".com" server which, again, answers with another NS referral to a server serving the subdomain ".example.com". Once more, the resolver will ask this server which will finally return the IP address or addresses of "www.example.com", assuming it is authoritative for this QNAME, meaning it is responsible of this zone. The resolution has been therefore completed. Now the client knows the IP address of the target and can establish a direct connection.

An important and essential observation here is that all queries to the name servers were identical, namely "looking for A records of www.example.com". The query was not altered during the resolution. That means that all name servers

had access to the full query name, even though unnecessary. We will see in the next chapter how QMIN tries to solve this privacy issue.

Resolution with rewriting

Another case of resolution may imply *rewriting* : this is a mechanism which modifies the query name during the resolution. Instead of directly delegating the resolution, the name server will send a record with a specific QTYPE with which the resolver will rewrite the QNAME. Two types of records trigger this behaviour from the resolver : CNAME and DNAME. CNAME explicitly rewrites the alias with the target, while a DNAME record will induce a synthesis of a new CNAME record at the nameserver side (see 3C in the algorithm of Listing 2.2). It means a new record will be created and sent to the resolver. Those records may then imply that the resolver has to query an authoritative nameserver other than the previously queried one, as the rewriting may change the zone to which the name belongs.

For a CNAME record, the resolver will simply launch a new query with the QNAME being the one contained in the CNAME record, as described on the server side in the step 3.a in Listing 2.2. The resolver then continues the resolution process by querying this new record.

Regarding the other record type, an example of a DNAME record can be found below (Listing 2.3). After a resolver sends a DNS lookup for *www.sub.example.com*, it will synthesize a new record looking for *www.host.attacker.com*. The particularity of this record type is that it only replaces the matching part of the Query Name with the DNAME target.

```
1    sub.example.com.    IN  DNAME   host.attacker.com.
```

Listing 2.3: Example of DNAME RR

A particularly interesting aspect of the rewriting is that, since it creates a new record, this may reset values that were collected during the resolution of the previous query and allow to bypass some limits enforced by resolvers. This will be especially taken advantage of when mounting QMIN-enhanced attacks.

2.2 Popular DoS Attacks against DNS

Due to the availability of DNS and its multiple endpoints, this system is prone to attacks trying to either reroute the traffic of requests to some malicious servers via cache poisoning, to intercept communications, or to crash the name servers due to resource exhaustion. We focus on a specific type of denial-of-service (DoS) attacks. Such attacks target a DNS server (resolver or nameserver) by generating

excessive queries to consume the server's resources. This may prevent legitimate users from accessing the name resolution service.

In the next subsections, we examine existing DoS attacks against DNS infrastructures producing an overhead at specific nameservers. The following attacks can be broken down and their primitive attack vector can be extracted and leveraged to create new attacks. Those attacks had led to the creation of several countermeasures that should now be enforced in resolvers and constitute also the basis of the explored vulnerabilities in this thesis. We have reproduced some of these attacks, extracted some primitives, and enhanced those using QMIN.

2.2.1 Water Torture

The most simple and popular DoS attack is called "Water Torture" [30]. The principle is simple: the attack is triggered by multiple controlled users that send thousands of requests for non-existent subdomains against the same authoritative nameservers. They are *flooded* with this large number of queries and try to answer but, due to the amount, the authoritative nameservers cannot process all of them. As a result, all of their resources are consumed, which leads to the nameservers slowing down and possibly even crashing.

The malicious users can also consist of a botnet. In this case, the attack type would be called Distributed Denial-of-Service (**DDoS**). A famous example of attack is the Mirai botnet [13, 4]. A malware called *Mirai* infected many smart devices and turned them into bots, which, in turn, infected other IoT devices around them by exploiting unchanged default usernames and passwords. This allowed to launch a large-scale water torture attack on the domain registration services Dyn, which disrupted major Internet platforms.

A remark is that the attack focuses on DNS nameservers, but it may also impact the resolvers, as they are constantly trying to resolver non-existent domains.

2.2.2 iDNS

The "Indefinitely Delegating Name Server" [26] (**iDNS**) attack consists of tricking a resolver into continuously sending queries to a specific malicious name server. A malicious client queries a domain name of the attacker-controlled authoritative nameserver via a resolver which carry out the resolution of the query. The authoritative name server will send back to the resolver several glueless delegations whose targets are subdomains managed by the attacker to itself. The resolver will then follow the referrals to the name server that will, once again, answers with glueless delegations to itself. This process is repeated indefinitely if the malicious name server can create dynamically the NS glueless chains. Otherwise, its responses will be cached and the resolver will stop sending queries when detecting that its cache can already answer them and retrieve those cached responses.

There are several variants of this attack :

- Variant 1 : our malicious nameserver only serves one glueless NS delegation to itself at a time. It may result in an indefinite loop, but it does not exhaust the victim resources
- Variant 2 : the contacted nameserver constantly sends back to the recursive resolver a set of NS delegations to itself
- Variant 3 : some nameserver names in the referral response point to sub-domains (existent and/or non-existent) of a victim nameserver and others point to the attacker authoritative nameserver itself (to continue the loop). An example is in Fig 2.2
- Variant 4 (DDoS): similar to Variant 3 but there are many glued delegations that point to other attacker-controlled nameservers that, in turn, reproduce the same attack against the resolvers

The impact of this attack was an increase of the Packet Amplification Factor (**PAF**) of a factor 10. And its disclosure in 2014 led many resolvers to implement some mitigation strategies such as limiting the query depth, breadth and total query count [26].

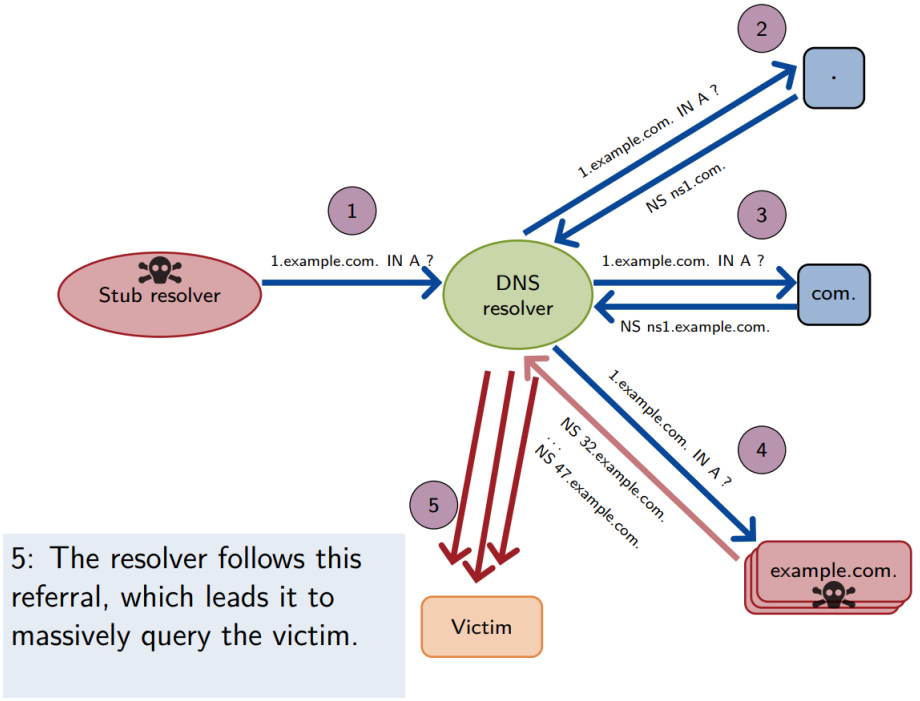

Figure 2.2: A scheme of the iDNS attack on a victim nameserver. This is the Variant 3 of iDNS without having the self-delegations loop shown explicitly. The figure is from [26].

2.2.3 NXNSAttack

Another attack leveraging vulnerabilities related to non-existent domains is the "NoneXistent Name Server Attack" (**NXNSAttack**) [11], which is a more destructive attack than the Mirai attack. To carry out a NXNS attack, the attack must control at least one client and an authoritative name server. The victim can be the resolver in-between or the nameservers that are bombarded with queries by the recursive resolver. In this case, we call it an amplification via reflection attack.

The client will send one query for a subdomain of the attacker-controlled authoritative nameserver to the resolver, which will process them. The authoritative nameserver will reply with a NS referral response containing a large number of nameserver names with glueless records, i.e. no IP addresses corresponding to the names are associated. To bypass the resolver cache and mitigation measures, the requested subdomains should be different every time. As they are not in its cache, the resolver will proactively launch a resolution process for each of them. The NS records can be defined to point to specific authoritative servers, such as a target domain or to fake (non-existent) subdomains controlled by the attacker nameserver itself. This latter strategy will create self-delegations and loops will occur. It accrues the number of queries sent/received. A schema of a variant of this attack can be found in Figure 2.3. We can observe this attack is quite similar to the iDNS attack, except that NXNS exploits the glueless nameservers of the attacker against victim namerservers. This attack was presented but not carried out. A dramatic difference is the number of packet amplification factor : it is 10 for iDSN [26] and 1620 for NXNS [11].

The proposed countermeasure was to improve the resolver resolution algorithm with *MaxFetch(k)* which consists of only resolving k nameserver domains in a referral response per original client request at once. This enhancement has the following impact on the NXNS attack: the resolver, instead of resolving an arbitrary larger number of domains in the received referral response, will only try to resolve k domains at a time. This slows down the resolution processes and reduce the consumption of the resolver's resources. However, all the nameserver domains will eventually be resolved. Therefore, the impact of the attack is still present but greatly weakened by this improvement.

2.2.4 TsuNAME

TsuNAME [28] is an attack exploiting cyclical dependencies between nameservers. The recursive resolver, when under this attack, follows these cycles which will then amplify the effect of the sole original query by launching requests for each element in the cycle, one at a time. This may result in the resolver indefinitely following loops while trying to resolve a domain name.

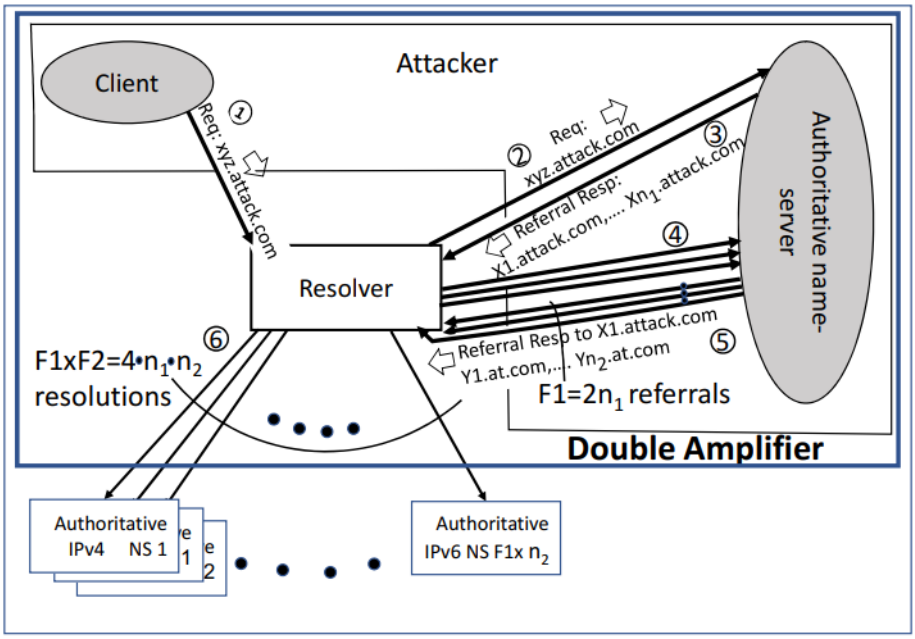

Figure 2.3: Illustration of an NXNS attack using one set of self-delegations responses. The figure is from [11].

These loops can be crafted using NS records. Assume there is a zone that has NS records that point to another zone, and this other zone also contains NS records that redirect to the first zone. The recursive resolver, when trying to make a DNS lookup for the first zone, will be caught up in a NS delegation loop between the authoritative nameservers of those zones. Records of type CNAME can also be employed to mount this attack.

This attack is enabled by having an insufficient cache and no proper configuration checker. The cache could detect that one element along the cycle has already been requested and cached by the recursive resolver, which will then stop the resolution if its Time-To-Live (**TTL**) has not expired.

Note that the resolver queried by the malicious client and all the intermediary nameservers contacted along the cycle will be impacted by this attack since they will send and receive multiple queries. The researchers documented a misconfiguration in a country-level domain that led to the increase of its overall traffic by around 50%. They also demonstrated a query amplification factor of 500.

As a countermeasure, they recommend that resolvers implement negative caching in order to abort the resolution process when a loop has been detected. They also implemented the open-source tool *CycleHunter* [5] which allows operators of authoritative DNS services to check their configuration and detect cyclic dependencies linked to their nameservers.

2.2.5 DNS Unchained

This attack is using the principle of CNAME chained records with the resolution of deep names. It was the first of its kind to exploit application-layer attacks with amplification [17]. Its objective is to amplify the number of queries sent to a particular nameserver, so that it is overloaded with queries. The victim can be either a target nameserver or the resolver used during the DNS resolution. In both cases, the attacker needs to control or at least be able to install malicious zones on two nameservers authoritative for two different zones.

The attack consists of sending a query to a resolver that will follow a chain of CNAME records between two zones. The reason is that, if the CNAME chain was entirely within one zone, the nameserver authoritative for this one would reply the whole chain. The resolver, which has received this answer, would then directly query for the last CNAME record. Therefore, the chain would not be effective.

In short, the attack requires two zones that host each half of the chain. The established halves trigger a "ricochet" of queries sent by the resolver between the two zones. The alias of the first CNAME reply will be used to create a new query or subquery targeting the other nameserver and vice-versa until the complete chain has been queried by the resolver. It will most likely end with a SERVFAIL response from the resolver to the client attacker, as it could not find the requested query name. This error code means that no answer for the query could be successfully found.

Several countermeasures were presented. Implementing a recursion depth limit of the CNAME chain would reduce the potential amplification factor. Bushart and Rossow [17] recommend a limit of 9, as it is the minimal value they observed in tested resolvers and benign chains. This is also the value we observed in the resolver implementation Unbound 1.10.0. Note that restraining the CNAME chain depth to 9 limits the amplification factor of this attack to 5. Another measure is limiting the number of outgoing queries, reducing in general the subqueries sent while following the CNAME chain. Such behaviour was also observed in PowerDNS 4.7.0.

Note that the attacker must define records for each chain it is using. Therefore, if an attack is launched exploiting parallel CNAME chains, the attacker will have to set many records. A variant of this attack makes use of DNAME chains that allows an arbitrary number of subdomains for the chain with a single entry. It is particularly effective against resolvers that do not support DNAME records because they will directly use the synthetic CNAME records provided by the authoritative nameservers, and ignore the additional data (DNAME records). This allows the attacker to create an arbitrary number of chains and to avoid caching, whereas resolvers supporting DNAME implement a cache allowing them to retrieve an entry to answer queries for all subdomains. Thus, the latter kind of resolvers sees its performance increased as only one DNAME has to be cached in comparison to many CNAME records. Note that using DNAME will

consume more resource at the nameserver because DNAME records and their corresponding synthetic CNAME records are exchanged.

2.3 Formal Model of DNS

This thesis will leverage a formal model of DNS resolution [22] to reproduce some of the previously described attacks. The formal analysis results collected will later be compared with implementation results. Moreover, this will give an idea about how effective they are.

This model simulates the DNS resolution process of queries in a network with generated actors, nameservers and custom zones. Here we have full control over those different entities and can design them how we want. Note that we do not cover DNSSEC in this thesis and neither does the model.

In order to be able to predict the amplification factor or the number of queries an attack will produce, we would like first to have an approximation of it. For this reason, this framework will provide this estimation since it can mimic a complete network with crafted DNS name servers and resolvers. Moreover, it can monitor DNS resolution processes of queries, which allows to analyse those in more details. It will help to understand why a resolver behaves a certain way when resolving a particular query, for instance when using DNAME records which implies a synthesis of new records.

This model is the first step on the validation of the attacks. It also represents the basis on which we will build the attacks, as it will quickly show whether a primitive attack setup is worth to extend to a local testbed. The model reproduces most of the DNS core specifications and, if an attack is effective in this controlled environment, it may also be successful "in the wild".

3 Preliminaries

In this chapter, we present the QMIN feature in detail and the general settings for our subsequent analysis.

3.1 QMIN Definition

Query Name Minimisation (QMIN) was first introduced in RFC 7816 [15] in 2016 to allow more privacy for users. This version was experimental and is now obsoleted by RFC 9156 [16], released in 2021.

As recommended by the Internet Engineering Task Force (or **IETF**) in a "Privacy Considerations for Internet Protocols" RFC (6973) [18], protocol designers should try to minimise the amount of data being collected or having the possibility to be collected by the entities reached during the use of protocols. Therefore, with the reduction of the size of the data exchanged, it cuts down on the DNS privacy risks of experiencing data leakage.

The objective of QMIN is to reduce the size of the data exchanged following the principle of minimum disclosure. And this does not include the number of messages. As explained in the subsection 2.1.3, the complete query names of user requests are being seen by all the name servers the resolver contacts in order to resolve the query. The same goes for the original QTYPE. Knowing the original QTYPE and original QNAME is only essential to the nameserver that is directly authoritative for the requested record by the client, i.e. the last nameserver that is contacted. The other nameservers only need some parts of the QNAME to answer the query with a delegation. The original QTYPE is not pertinent for them to learn, as they do not have the authoritative capability to deliver the records requested by the client.

The next subsections shows how the resolver creates QMIN crafted queries. It follows the QMIN Algorithm listed in Listing A.1 in Appendix A.

3.1.1 QMIN design

The RFCs describe how a resolver should implement QMIN by crafting the requests sent to an authoritative nameservers that is not known to be authoritative for the original QNAME:

1. the resolver should obscure the original QTYPE by selecting a certain QTYPE (may be the same)
2. it should also obscure the QNAME, by stripping the original QNAME of its labels until there is only one label more than the domain name for which the nameserver is authoritative

1. QTYPE Selection

The first version of the RFC recommended the QTYPE to be NS, and the latest version relaxes this to any authorized QTYPE. The reason being that the nameserver of the parent zone will send a NS delegation to the child zone, regardless of the QTYPE. Therefore, there is no advantage to use NS over the other QTYPEs.

No information must be inferred between the original QNAME and the obfuscated one. The newest RFC suggests the use of A or AAAA QTYPE, as those types are most likely correctly supported by all DNS nameservers and resolvers.

2. QNAME Crafting

The crafting of QNAME is best described via the following example. Suppose the resolver receives a request for A records of "www.a.b.c.d.example.com" and it already knows the nameserver address of "example.com". Further assume there is a zone cut in the subdomain, for instance between "a" and "b", but the resolver does not know that. So it sends a first request for "d.example.com" to the nameservers of "example.com" and receives a non-referral answer, forcing the resolver to send a second request for "c.d.example.com", effectively adding one label. It will keep adding labels and send "probing" requests until it either receives a CNAME record or NS referral to the nameserver authoritative for the query or authoritative for a longer part of the QNAME, or the IP address requested. In this case, the resolver will receive a NS referral when requesting "a.b.c.d.example.com" to the authoritative nameserver of that zone, which allows the resolver to send the final request and complete the resolution process at this next nameserver.

Here we see an increase in queries sent but the nameservers of "example.com" only learn gradually the QNAME, and even in this case, they only know one more label than the subdomains they are authoritative for. Thus, this process is privacy-preserving.

The inherent problem is that the resolver does not know where the cut zones are. It needs to slowly increment the labels to "advance in the dark". On the side of the resolver, nothing can be done under the current DNS infrastructure to avoid this "extra" label learned by the nameserver.

3.1.2 Recommended Parameters and Mitigations

Since, with QMIN, the resolver sends a query for each label contained in the original QNAME, it can, in the end, send many queries for a single client request (assuming an empty or "cold" cache). This spike in number of queries sent may decrease as the time goes by and the cache fills up.

However, queries for random subdomains with a large number of labels bypass the cache constraints and will force the resolver to build many queries for each long subdomains. Therefore, QMIN definitely creates a new attack vector which leads to the recommendation that resolvers must enforce limits on the number of outgoing queries.

Another measure that can be used to reduce the number of queries is adding more than one label at a time for QNAMEs with a large number of labels, successfully cutting down on the queries sent. Moreover, one could also choose the number of iteration of QMIN where only label must be appended.

From these observations emerged two features (Table 3.1) :

Name	Role	Recommended value
MAX_MINIMISE_COUNT	Maximum number of iterations labels will be added	10
MINIMISE_ONE_LAB	Number of times **only** one label is added	4

Table 3.1: Mechanisms to reduce the amplification of QMIN

Note that MAX_MINIMISE_COUNT does not stop the resolution of QNAME with more than 10 labels, it will simply append more labels at each iteration until the whole QNAME is queried. An observed behaviour is that, once MINIMISE_ONE_LAB has been reached, the rest of the labels are divided almost equally into sets that will be appended one at a time. This means that an attacker does not need to create records with more labels than MAX_MINIMISE_COUNT to exploit QMIN to the maximum of its potential, as the extra labels would simply be added to previous QMIN iterations. Therefore, MAX_MINIMISE_COUNT should be the number of labels in malicious records.

An interesting aspect is that, even if the resolver sends at some points a query with, for instance, five more labels than zones for which the nameserver is authoritative, the nameserver will actually look for the closest ancestor of the QNAME and return either a delegation or the associated RRs (see Listing 2.2). Thus, one cannot exploit the part of the QMIN algorithm where more than one label are added to the current QNAME.

Moreover, some resolvers implemented two modes of QMIN, "strict" and "relaxed". The former makes the resolver follow the QMIN algorithm of RFC 7816 [15] whereas the latter will make the resolver first try the QMIN algorithm, then fallback on the conventional name resolution.

Until now, we looked at queries that require privacy. But some may not need such privacy as part of domain is already cached by the resolver and the intent of the use of the QNAME is straightforward. In this case, in order to let the resolver know that QMIN does not need to be applied on the query, one can start each non-private label with the special underscore (_). This leads the number of queries sent not to be affected by QMIN in those particular cases.

3.2 Adversary Model

We consider an attacker that aims to exhaust the resources of some victim DNS resolver and/or nameserver. Leveraging some amplification effect, the attack's cost should be less than the resource consumed by the victim. Specifically, each request sent by the attacker should trigger as many queries generated and processed by the victim as possible.

We assume the attcker can install maliciously crafted zone files on one or more nameservers. This is realistic because today's DNS hosting services are pervasive and easy to access [11, 23]. In addition, we assume the attacker can send requests to the target resolver involved in an attack. This is always true for a public resolver. Even if the resolver locates in a closed network, there are still many ways an outside attacker can trigger requests to the resolver, e.g., by injecting URLs pointing to the malicious domain into a webpage visited by a host in the network.

3.3 Amplification Factor

There are different ways of defining the amplification factor of attacks, for example

- the number of messages sent/received
- the size of the messages received
- the number of packets exchanged
- the delay in the response
- the time the victim resources are occupied

We decided to stick with the following definition of amplification factor throughout this document (except in cases where it is explicitly noted) as it is easier to interpret and it abstracts away the complexities introduced by other amplification factors such as TCP fallback. This definition is also used in a precedent project [22]

to measure the impact on other attacks, allowing us to confirm the previous results and setups. This concentrates our focus on application-layer type of attacks.

$$\textit{Message Amplification Factor (MAF)} = \frac{\textit{Number of messages received by the victim}}{\textit{Number of messages sent by the attacker}} \tag{3.1}$$

In our analysis, we focus on the amplification produced by a single request sent by the attacker. In a realistic DDoS scenario, the attacker can generate a large number of requests towards the victim. This requires careful construction of the malicious zone files (e.g., low TTL value) and query names (e.g., random prefix) in order to, for example, bypass caching at the target resolver.

4 Attack Validation and Analysis

QMIN as an amplification dimension is orthogonal to other attack vectors (Section 2.2). It is enforced solely in resolvers and it only concerns the number of labels in QNAME. As such QMIN can enhance the other DoS attacks by multiplying their amplification factor with the number of queries produced by QMIN. In this chapter, we systematically validate these potential attacks and analyse their impacts. Specifically, we will go through three steps:

1. validate the attacks against resolver models to ensure that, with the current DNS formal model, attacks are effective;
2. reproduce the attacks on a local testbed with real implementation of resolver; and
3. test the success of the attacks "in the wild"

To this end, one would like to be able to predict up to some precision the behaviour of resolvers when facing attacks of different flavors. By tuning the parameters of the resolver in the model, we can turn it into a more specific kind of resolver. Explicit limitations and features of some resolvers were previously collected to make the models as close as possible to the real ones [22].

4.1 Methodology

4.1.1 Accuracy of The Formal Model

The model should not be too finely tuned to recreate perfectly a resolver, we want to keep the general behaviours observed formerly. Moreover, it would take more time to modify the configuration of the model resolver to this end. Thus, a general model with some specific aspects of the real resolver is what we are looking for.

Furthermore, this allows us to notice more easily strange performance issues and discrepancies compared to our expectations. It directs then our focus on certain aspects of the resolvers that may be exploitable and that confirms that some attacks are undeniably successful against the targeted resolvers.

An example of such noticeable behaviour was the case of Unbound 1.10.0 [6], which actually retries once unsuccessful queries up to a certain number of delegations. The model resolver of this implementation did not follow this strategy, as this is slightly more complex.

4.1.2 Tested Resolvers

In this project, it has been decided that four open resolvers will be studied in the local testbed : BIND 9.18.4 [7], Unbound 1.10.0 [6], Unbound 1.16.0 [9] and PowerDNS 4.7.3 [8]. These open resolvers will be leveraged in the formal model and in the local testbed. As they had been used as comparison in a previous project, it makes sense to use them again to validate the previous results, consolidating them, and further exploiting the discovered vulnerabilities with them. Moreover, those are popular open resolvers, meaning that successful attacks against them can have a great impact over users and over their other instances on the Internet.

The configuration of all these resolvers can easily be modified to turn on and off QMIN, which is necessary to properly analyse the results of both modes.

4.1.3 Local Testbed

A local testbed (version 1)[33] has been created in order to validate the attacks in a controlled environment with real implementations of resolver. It was built using Docker (version 20.10.21)[2] that it creates multiple containers, representing the nameservers. The testbed sets up a closed network and assigns a distinct known IP address to each of the servers.
This testbed replicates the DNS resolution for a combination of chosen queries and sets of zones installed in the different nameservers.

Zones specific to each attack will have to be written in the database of the nameservers. Some will be converted from the Maude language of the formal model to the correct format and others will have to be created from scratch. An example of such a zone can be found in the Listing 4.1.

```
1  $TTL 0
2  @                        IN      SOA     server.target.com. username.target.com.
       (2006032201 7200 3600 1209600 0)
3                           IN      NS      ns.target.com.
4                           IN      A       172.20.0.7
5  ns.target.com.          IN      A       172.20.0.5
6
7  ; CNAME chain
8
9  fake8.fake7.fake6.fake5.fake4.fake3.fake2.fake1.fake0.a0.target.com.      IN
       CNAME    fake8.fake7.fake6.fake5.fake4.fake3.fake2.fake1.fake0.b0.target.com.
10 fake8.fake7.fake6.fake5.fake4.fake3.fake2.fake1.fake0.b0.target.com.      IN
       CNAME    fake8.fake7.fake6.fake5.fake4.fake3.fake2.fake1.fake0.c0.target.com.
11 fake8.fake7.fake6.fake5.fake4.fake3.fake2.fake1.fake0.c0.target.com.      IN
       CNAME    fake8.fake7.fake6.fake5.fake4.fake3.fake2.fake1.fake0.d0.target.com.
```

Listing 4.1: Zone example for a Scrubbing + CNAME + QMIN attack with a CNAME chain of length 3

This zone resides in the nameserver authoritative for "target.com", where we count the number of queries received. It should approximately be equal to the

number of queries sent by the recursive resolver (minus queries to root servers). A DNS lookup for the following name

```
fake8.fake7.fake6.fake5.fake4.fake3.fake2.fake1.fake0.a0.target.com.
```

launched by the client will trigger a list of CNAME records followed by the resolver, with all the CNAME RRs being contained in "target.com".

The advantage of this testbed is that we can easily launch an attack with a specific list of zones that will be installed in the chosen nameservers. We can as well repeat an attack a defined number of times with, each time, another zone installed. This allows to carry out a large set of attacks and analyse them all at once. Furthermore, instantiating a few containers with a specific configuration of a studied open resolver is straightforward, once all the commands to install and run the particular resolver are known.

Each log file of the attacks is retrieved to see the details of the queries received and the number of queries sent by each resolver is saved into a *csv* file that will facilitate the plotting of the attack results.

4.1.4 Variables in Attacks

When conducting experiments, we want to analyse the amplification factor of the attacks using certain restraints to limit the attack space according to previous observations. Indeed, if we know that a resolver only tries to solve the 15 first queries of a CNAME chain, we don't need to run the test with a CNAME chain of length 100. This can have two advantages : firstly, it reduces the time of experiments; and secondly, it shows that this space is limited by the length and proves that, once we have sufficiently delved into this space, we don't need to conduct further studies along this axis.

On the other hand, we may want to examine the amplification factor a little further than the limit, as we have observed that some of the resolvers may implement retries whose number grows with the length of the chain. Thus, we would like to obtain enough data points around the estimated maximum value to conclude that the limit has been indeed reached and no strange behaviour has occurred.

For our experiments, we are then able to vary the following restraints : the number of NS delegations, the lengths of CNAME chain, DNAME chain and referral chain as well as the number of labels of the records. Those attack vectors are explained more in details in later subsections. Most of the time we will vary the number of NS delegations and set the other variables to their respective limits observed. One of the objectives will be to detect the limit of each of those into each of the resolvers we are studying. This allows first to create more accurate models, and secondly, it gives us the knowledge to maximize the attacks by extending

the range of the attack vectors to their limits. We determine those limits through simple but sufficient enough evaluations in the testbed.

4.2 Implementation in the Formal Model

The main contributions of this thesis concerning the formal model are the automatization of the attack files creation, of the zones files and the retrieval of results of these generated attacks. They are listed below :

- Creation of model resolver class
- Creation of model attacks class, still quite primitive
- Automatization of the creation of zones inside the nameservers
- Automatization of process to create and run a large number of files
- Combined the results of different resolvers in plots
- Conversion zones from Maude to zone format

The formal DNS model is specified in the Maude language, allowing for formalization of the different DNS resolution processes and verification of them. Python will be used to automatically create Maude files that will be run to start the attack. The resulting measures will be stored in text files and retrieved for the plots.

4.2.1 Model resolvers

The way to create and launch an attack using the model is the following : we start with the import of the multiple modules containing the definition of the nameserver, resolver, client and the DNS properties. We then tune parameters related to the resolver : cache, whether QMIN is turned on, what is the timeout after which the resolver stops, whether MaxFetch(k) and CNAME validation are enabled.

Then we create the different zones and fill their database. Finally, we instantiate the resolver and the nameservers and assign them previously defined zones. We also define the query that will launch the attack in the client nameserver. At last, we add commands that run the experiment and retrieve some general data about the attack.

Notice that we do not set other limits for the resolver here such as the depth of a CNAME chain or the maximum number of followed referrals. Those restraints will be artificially enforced when creating the zones for the attack. To facilitate this, we implement the class of model resolver whose instances have fields defining the limits enforced by the resolvers. One example is shown in the Listing 4.2.

```
1 class Unbound1_16_0(ModelResolver):
```

```
2
3     def __init__(self):
4         ModelResolver.__init__(self)
5         self.version = "1.16.0"
6         self.name = "Unbound" + "␣" + self.version
7
8         self.folder = "unbound1_16_0"
9
10        self.cname_chain_validation = True
11        self.cname_limit = 12
12        self.qmin_limit = 10
13
14        self.workBudget = 5000
15
16        self.max_subqueries = 6
```

Listing 4.2: Model resolver of Unbound 1.16.0

The existence of this class of model resolver will permit us to extend attacks to other resolvers, as we only need to find out what their limitations to be able to implement their model. We will then be able to reproduce the attacks on them.

Note that the attack is launched from a single file, which encloses all the different components of the simulated network, their parameters, the original client query and the commands retrieving monitored data such as amplification factor.

4.2.2 Model attacks

Maude files may have the exact same structure for some attacks, such as the variants 1 and 2 of iDNS in chapter 2.2.2. They only differ by the content of the zones. This calls for an abstraction of the attack and leads to, again, a creation of class of model attacks.

The parent class is displayed in Listing 4.3. For simplicity, all its fields consist of strings. The first one "folder" is relevant for the location of the attack folder where all the attack files, results and plots related to the attack will be placed. This allows a neat and organized file structure. The rest of the fields are used to capture the elements of a Maude file discussed in the previous subsection (chapter 4.2.1). And the function gathering every component of the attack for the creation of a Maude file is shown in Listing 4.4. Note that this is an unsophisticated function, but for our experiments, it is sufficient enough.

```
1 class ModelAttackFile():
2
3     def __init__(self):
4
5         self.folder = ""
6
7         self.imports = ""
8
9         self.description = ""
10
11        self.start_text = ""
12
13        self.continue_text = ""
```

```
14
15          self.target_text = ""
16
17          self.intermediary_text = ""
18
19          self.attacker_text = ""
20
21          self.end_text = ""
22
23          self.print_text = ""
24
25          self.victim = "UNDEFINED␣VICTIM"
26
27          self.name_attack = "UNDEFINED␣NAME␣ATTACK"
```

Listing 4.3: Parent class of model attack

```
1  def whole_file(self, PATH_TO_MAIN_DIR, resolver_model, QMIN_DEACTIVATED, target_records,
       ns_records_text_in_intermediary) -> str:
2
3          whole = self.imports.format(PATH_TO_MAIN_DIR=PATH_TO_MAIN_DIR)
4          whole += self.description
5
6          whole += self.start_text.format(resolver_model.config_text())
7          whole += QMIN_text(QMIN_DEACTIVATED)
8          whole += self.continue_text
9          whole += self.target_text.format(target_records)
10         whole += self.intermediary_text.format(ns_records_text_in_intermediary)
11         whole += self.end_text + self.print_text
12
13         return whole
```

Listing 4.4: Function in a model attack class responsible for gathering all the necessary info for a Maude file

4.2.3 Automatization

To avoid errors when building countless records in the zones, we automate the process of their creation. Depending on which attack we focus, we may need a chain of CNAME records between two nameservers or a list of NS RRs with the same owner name but each time a different target. Thus, we require tailored function constructing the database of the nameservers.

All these abstractions via classes and this automatization lead to the automatic creation of a large number of files, divided into their respective attack folder. We are now able to modify the type of RR in the zones, their size, the number of labels of each record and their target. We can now vary those parameters to explore the attack space and produce rapidly a larger number of result files. We also set up a "watcher" object which stops the execution of the current file if it restarts an attack with the exact same parameters as in a previous file. This allows to save time when generating and running the files.

4.3 Analysis and Validation

All of the attacks reproduced here suppose that the attacker controls at least one malicious nameserver or has the ability to compromise it and to insert new resource records into its database. The attacker will be represented by the client which launches the query triggering the attack. We did not assemble multiple users (as in a botnet) for these experiments since we only want to show the amplification factor produced by solely one attacker entity and one client query.

Moreover, we used attack vectors extracted from the attacks depicted in 2.2 : multiple NS delegations of the same record (also called "parallel NS"), chain of CNAME records between nameservers and self-contained CNAME chain.

Each attack constructed from these attack vectors will be briefly explained and resulting plots will be displayed if useful. Some of them are already well-described in the literature but others were more recently conceived. The following plots will contain some results of the formal model and from the local testbed, both with and without QMIN. If QMIN is exploited, 10 labels will be added to each record in the chains as this is the maximum MAX_MINIMISE_COUNT observed. The number of QMIN iterations is thus following the recommendation of the RFC 9156 [16].

Note that we use the term "subqueries" to designate queries that are generated due to the first triggering query. Therefore, all the queries received at the target nameserver following the first one are "subqueries".

4.3.1 Parallel referrals

This attacks consists of a query triggering a list of 30 NS parallel referrals. Here "parallel referrals" means all the NS delegations starting from the same node (here "del.inter.net"), and not the batch of subqueries being resolved (linked to the MaxFetch(k) of NXNS attack). Examples of zone for the unflavoured version and QMIN version of the attack can be found in Listings 4.5 and 4.6.

The attacker installs in a authoritative nameserver a list of NS RRs whose owner name are all the same (here "del.inter.net") and whose targets have all the same parent domain. The query for "del.inter.net" will see the resolver receive one NS response with all those referrals in the additional section of the response. The resolver will then start multiple resolution subqueries for "a1.target.com", "a2.target.com", . . . Thus, this will occupy the resolver resources as it tries to resolve the different referrals. Now an interesting question would be to know whether there are some limits on the amplitude of this "breadth-like" attack.

```
1 ; in inter.net
2 ; Delegations
3 del.inter.net.        IN      NS      a1.target.com.
4 del.inter.net.        IN      NS      a2.target.com.
5 del.inter.net.        IN      NS      a3.target.com.
```

```
...

; in target.com
; Direct results with A records
a1.target.com.          IN      A   172.20.0.20
a2.target.com.          IN      A   172.20.0.20
a3.target.com.          IN      A   172.20.0.20
...
```

Listing 4.5: Zone for the attack Parallel referrals

```
; in inter.net
; Delegations
del.inter.net.         IN      NS      lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.
    lab3.lab2.lab1.a1.target.com.
del.inter.net.         IN      NS      lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.
    lab3.lab2.lab1.a2.target.com.
del.inter.net.         IN      NS      lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.
    lab3.lab2.lab1.a3.target.com.
...


; in target.com
; Direct results with A records
lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.a1.target.com.         IN
          A   172.20.0.20
lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.a2.target.com.         IN
          A   172.20.0.20
lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.a3.target.com.         IN
          A   172.20.0.20
...
```

Listing 4.6: Zone for the attack Parallel referrals + QMIN

Here we report in Table 4.1 the maximum nuomber of parallel referrals followed and the maximum number of subqueries observed for each resolver model and resolver, combined or not with QMIN. A clear distinction has to be doen between the "Model" and "Real" columns. The former is the number of maximum NS referrals followed **concurrently** by the model resolver, while the latter is the total number of **parallel** referrals eventually followed by the resolver.

Resolver	QMIN	Model conc	Real par	Max # subqueries
BIND 9.18.4	No	5	No limit	30
BIND 9.18.4	Yes	5	No limit	30
Unbound 1.10.0	No	6	No limit	60
Unbound 1.10.0	Yes	6	No limit	537
Unbound 1.16.0	No	6	7	7
Unbound 1.16.0	Yes	6	7	60
PowerDNS 4.7.3	No	5	19 / 31	37
PowerDNS 4.7.3	Yes	5	28	37

Table 4.1: Maximum number of referrals followed by the models and resolvers, and the maximum number of subqueries observed.

The "QMIN" column represents whether we exploit QMIN in the attack. We turn on this feature by default as there should not be any difference between the

attack exploiting QMIN and the one that does not. Except with PowerDNS, we observed that the number of parallel referrals followed by this resolver changes if QMIN is enabled or not, regardless of whether QMIN is exploited. PowerDNS has a limit of 19 parallel referrals followed when QMIN is deactivated and 31 if enabled. With QMIN enabled, PowerDNS uses QMIN for the first query then disables it, and has a limit on the number of parallel referrals followed set at 28.

BIND does not seem to have a limit on the number of followed parallel referrals but QMIN does not amplify the attack as it seems to be deactivated. QMIN does not affect any queries sent by the resolver, i.e., no stripping of labels occurs.

Unbound 1.10.0 also does not seem to have a limit on parallel referrals and QMIN is effective. Unbound 1.16.0 has a hard limit at 60 subqueries sent, the resolver tries to resolve a limited number of random NS referrals and sends around 8 subqueries per referral. Both versions perform retries of unsuccessful queries.

Please note that we only report the maximum number of subqueries observed (received at the target) with 30 NS delegations, we did not extend the attack further with more delegations. The maximum numbers of subqueries observed all come from the real resolver as they all have either limits bigger than in the models or retries the subqueries multiple times such as Unbound. We can observe that QMIN does not affect the subqueries of BIND and PowerDNS. Whereas Unbound 1.10.0 has an impressive increase, and Unbound 1.16.0 has slight increase in comparison and is limited to 60 subqueries.

4.3.2 Referral chain

This type of attack relies on a chain or list of NS records linked to each other. This is a pattern similarly used in the attacks iDSN [26], NXNS [11] and TsuName [28]. Several version of this attack were created in order to cater for varied setups and circumstances.

Lame delegation

A *lame delegation* is a delegation produced by a nameserver whose targeted nameserver is not authoritative of the currently queried zone. This technique will be exploited in this attack that assumes the attacker controls or can insert arbitrary zones to two nameservers.

In Listing 4.7 we can observe a lame delegation in the zone "target.com".
If a client queries A records for "www.sub.target.com", the resolver will first receive the lame delegation from nameserver 1, will follow it to the zone "inter.net", queries the nameserver 2 for "www.sub.target.com" and receives a correct NS delegation back to the first nameserver. This could produce a loop with the resolver querying one nameserver which will point to other one and vice-versa.

```
;zones in target.com. (nameserver 1):
        www.sub.target.com          IN        NS      inter.net.
        inter.net.                  IN        A       172.20.0.6


;zones in inter.net. (nameserver 2):
        www.sub.target.com          IN        NS      target.com.
        target.com.                 IN        A       172.20.0.5
```

Listing 4.7: Zones created for the Referral chain attack

However, all the tested resolvers detect this lame delegation and abort the resolution process. This is also the case for most of the open resolvers over the Internet, since it can be considered more as a configuration error than a deliberate attack. Thus, this attack variant does not generate many subqueries and has a low amplification factor.

Note that enabling QMIN here can slightly amplify the attack. However, the lame delegation is always detected and amplification is therefore limited. The results are shown in the Table 4.2. Moreover, the resolver models of the formal model detect this misconfiguration as well, further confirming that DNS rules were implemented to halt this technique. This attack variant does not bring value from a potential attacker.

Resolver	Subqueries	Subqueries with QMIN
Bind 9.18.4	1	3
Unbound 1.10.0	1	2
Unbound 1.16.0	1	2
PowerDNS 4.7.3	1	2

Table 4.2: Results of the attack referral chain with lame delegation

Glueless version

This variant installs a chain of NS records located inside the same zone (here "target.com"), in contrast to the previous alternative. No lame delegation is used here and multiple *self-probing* queries will be sent by the recursive resolver to the nameserver authoritative for the zone. Zone examples are shown in Listings 4.8 and 4.9.

```
a1.target.com.          IN      NS   b1.target.com.
b1.target.com.          IN      NS   c1.target.com.
c1.target.com.          IN      NS   d1.target.com.
...
```

Listing 4.8: Zone for referral chain attack, glueless variant

```
lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.a1.target.com.          IN
          NS   lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.b1.target.
    com.
```

```
2 lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.b1.target.com.          IN
          NS   lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.c1.target.
    com.
3 lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.c1.target.com.          IN
          NS   lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.d1.target.
    com.
4
5 ...
```

Listing 4.9: Zone for referral chain attack, glueless variant with QMIN

The resolver, triggered with the client query "a1.target.com", will follow each NS delegation until it reaches its maximum NS chain depth allowed. It corresponds to the number of chained NS followed by the resolver. By examining the logs and locating the last NS record queried at "target.com", we can determine this limit and report it in Table 4.3. Moreover, this attack shows that the QMIN version has a reasonable amplification factor of around 8 in comparison to the version without QMIN, except for BIND.

Resolver	No QMIN	NS chain depth	QMIN	NS chain depth
BIND 9.18.4	9	9	13	8
Unbound 1.10.0	10	5	89	5
Unbound 1.16.0	8	5	71	5
PowerDNS 4.7.3	16	16	98	10

Table 4.3: Number of subqueries for version without and with QMIN, and NS chain depth limits from the attack referral chain with glueless records.

BIND experiences a drop of 1 in its NS chain depth and PowerDNS sees a decrease of its NS chain depth limit by 6 when QMIN is enabled and actively exploited. There are no changes of chain depth in Unbound resolvers.
PowerDNS provides the maximum amplification factor with 98 subqueries, followed closely by Unnbound 1.10.0.

Note here that we do not have results from the model resolvers. They do not capture the NS chain depth limit since it is not a parameter that was previously observed and implemented in the formal model.

Glueless version with IP loop

In this attack, the loop between two nameservers that was previously limited by the resolvers is being **bypassed** by exploiting the fact that the IP addresses are not being counted in the loop (in contrast, the names are counted). We can then induce a loop that will not be detected by the resolvers. The strategy consists of the client querying "www.sub.attack.example.com" and the nameserver "example.com" constantly redirecting the resolver's focus between two distinct nameservers, on which different zones are installed (here "attackX.example.com").

The zones that will enable the attack are displayed in Listing 4.10. Notice that the A records in host A and B are in parentheses.

```
; zones in example.com
attack.example.com.     IN      NS      ns1.example.com.
ns1.example.com.        IN      A       1.2.3.4

attack1.example.com.    IN      NS      ns2.example.com.
ns2.example.com.        IN      A       5.6.7.8

attack2.example.com.     IN      NS      ns3.example.com.
ns3.example.com.        IN      A       1.2.3.4

attack3.example.com.    IN      NS      ns4.example.com.
ns4.example.com.        IN      A       5.6.7.8
...

; zone attack.example.com. (host A)
    ns1.example.com. A 1.2.3.4
    ---END OF APEX

    sub.attack.example.com. NS ns.sub.attack1.example.com.
    ;(ns.sub.attack1.example.com. A 5.6.7.8)

; zone attack1.example.com. (host B)
    ns2.example.com. A 5.6.7.8
    ---END OF APEX

    sub.attack1.example.com. NS ns.sub.attack2.example.com.
    ;(ns.sub.attack2.example.com. A 1.2.3.4)

; zone attack2.example.com. (host A)
    ns3.example.com. A 1.2.3.4
    ---END OF APEX

    sub.attack2.example.com. NS ns.sub.attack3.example.com.
    ;(ns.sub.attack3.example.com. A 5.6.7.8)
...
```

Listing 4.10: Zone for referral chain attack, glueless variant with IP loop

When being queried for "www.sub.attack.example.com", the DNS resolver will ask the nameserver of "example.com" and will receive a glued delegation for 1.2.3.4 (host A). Host A will then be queried by the resolver and will respond with either a "glueless" record or a delegation "glued" with the IP address.
In the first case scenario, the resolver will be forced to ask the nameserver of "example.com" for the IP address of "ns.sub.attack1.example.com" as it is authoritative for the parent zone "example.com". As QMIN is enabled, the resolver will first ask for "attack1.example.com" and receive a glued delegation whose target is host B. The resolver will query for "sub.attack1.example.com" and another glued delegation for "ns.sub.attack2.example.com" will be sent to the resolver. The target of this delegation is in the zone hosted by host A. Again the resolver will ask "example.com" for this delegation that will redirect to host A, therefore continuing the loop.
In the case where both hosts send glued delegations, the resolver will not query the nameserver of "example.com" between each host and will directly query the

pointed host. The loop is not detected because IP addresses are used and not names.

For a chain of length 10, we obtain the following Table 4.4 showing the number of subqueries reeceived at the nameserver "example.com" with the fully glueless version of this attack and the number of NS records that the resolver follows. Here the table only includes results from the local testbed as, again, the NS chain limit is not enforced in the model resolvers.

Resolver	Subqueries	NS chain depth
BIND 9.18.4	16	8
Unbound 1.10.0	11	5
Unbound 1.16.0	11	5
PowerDNS 4.7.3	21	No limit

Table 4.4: Number of subqueries resulting of the attack referral chain with glueless records exploiting an IP loop

BIND has a limit on the chain depth of 7. Unbound 1.10.0 and 1.16.0 follow the same pattern : the resolvers follow the chain until the fourth NS record is reached and retry it three times. Then they stop the resolution. PowerDNS does not seem to have a chain limit : it simply adds 2 subqueries each time the chain grows. Note that we did not create experiments with longer referral chain.

This attack is more sophisticated than the previous one since we need to construct two or three nameservers with zones linking one to another. Those zones does not create a loop that can be detected by the tested resolvers. Nevertheless, we still have a limit on the NS chain depth. These limits are equal to the ones observed in the previous variant.

4.3.3 Parallel NS + Unchained

The attack discussed here is the Parallel NS + Unchained attack, which is a variant of *DNS Unchained* [17]. It supposes that the attacker has the control of two authoritative nameservers or that it can insert crafted zones into two legitimate authoritative nameservers.

The principle of the Parallel NS + Unchained attack is to combine NS parallel referrals and CNAME chains using the Unchained-style attack vector. If multiple NS resource records with the same owner name are within the zone served by one nameserver, one client query for this name will trigger all its parallel referrals. This means the resolver will create subqueries to resolve the target NS names. In fact, each NS delegation triggers a chain of CNAME records. The chain is divided in two, one half is installed in the nameserver and the other half in another one. And each CNAME RR in one nameserver has its target name pointing to the other

nameserver's zone. Thus, Parallel NS + Unchained effectively combines both those attack vectors to introduce a larger amplification factor.

Moreover, the type of record used in the chain does not need to be of type CNAME, we can also leverage DNAME records, therefore, leading to two variants of this attack that we reproduced below.

Parallel NS + Unchained + CNAME

The zones needed to start the attack are displayed in the following Listing 4.11. For simplicity, we do not show the zones for the attack combined with QMIN.

```
; zone in one nameserver
del.inter.net.          IN      NS      a1.target.com.
del.inter.net.          IN      NS      b1.target.com.
...
; CNAME chains
b1.inter.net.           IN      CNAME   c1.target.com.
d1.inter.net.           IN      CNAME   e1.target.com.
f1.inter.net.           IN      CNAME   g1.target.com.
...
b2.inter.net.           IN      CNAME   c2.target.com.
d2.inter.net.           IN      CNAME   e2.target.com.
f2.inter.net.           IN      CNAME   g2.target.com.
...

;zones in the other nameserver
; CNAME chains
a1.target.com.          IN      CNAME   b1.inter.net.
c1.target.com.          IN      CNAME   d1.inter.net.
e1.target.com.          IN      CNAME   f1.inter.net.
...

a2.target.com.          IN      CNAME   b2.inter.net.
c2.target.com.          IN      CNAME   d2.inter.net.
e2.target.com.          IN      CNAME   f2.inter.net.
...
```

Listing 4.11: Crafted zones for the attack Parallel NS + Unchained

This attack is triggered by the query "del.inter.net" launching the resolution of the NS delegations and the CNAME chains. Here we can observe on the two following plots (4.1 and 4.2) that enabling QMIN does not introduce impressive change in the formal model using the resolver model. One noticeable difference is the fact that PowerDNS sees its maximum number of parallel referrals changing from 5 to 9 when QMIN is enabled. This was an observation made when experimenting with the local testbed. Other remarks about the model resolvers are reported in the following Table 4.5.

Here we use the maximum possible CNAME chain depth here when installing the zones. As it is an Unchained-style of attack, each resolver model will only follow half of the chain.

We can see that BIND 9.18.4 does not follow the CNAME records in the subqueries created after the NS referrals. This explains that only one query is

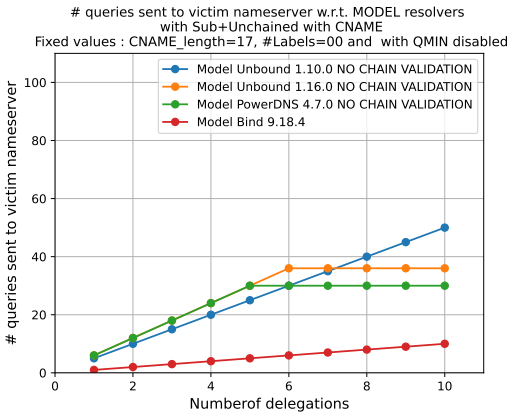

Figure 4.1: Parallel NS + Unchained, model resolver with QMIN disabled

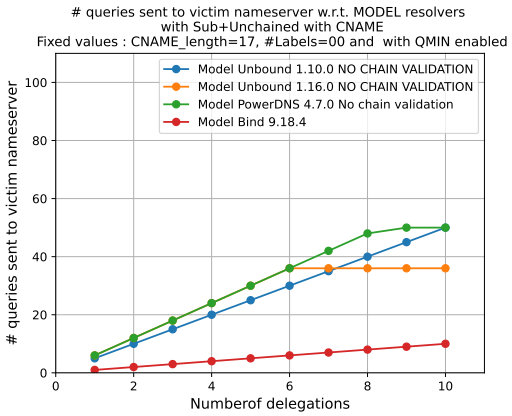

Figure 4.2: Parallel NS + Unchained, model resolvers with QMIN enabled

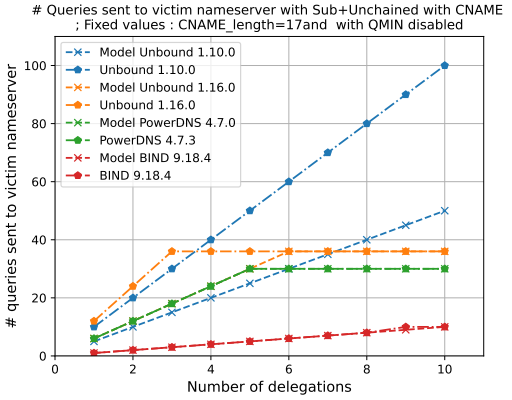

Figure 4.3: A comparison of the Parallel NS + Unchained attack between the model resolvers and the resolvers, with QMIN disabled

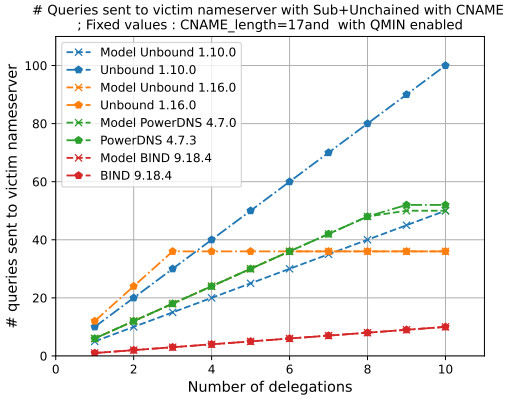

Figure 4.4: A comparison of the Parallel NS + Unchained attack between the model resolvers and the resolvers, with QMIN enabled

Model resolver	Maximum parallel referrals	CNAME chain depth limit	CNAME followed in subqueries
BIND 9.18.4	5	17	No
Unbound 1.10.0	No limit	9	Yes
Unbound 1.16.0	6	12	Yes
PowerDNS 4.7.3	5/9	11	Yes

Table 4.5: Resolver limits and parameters

added per delegation. Concerning Unbound 1.10.0, there is no limit on the number of parallel referrals. We could then add any number of delegation and the resolver will follow them. This is a bug that was disclosed and patched. For Unbound 1.16.0, there is a limit of 6 parallel referrals, defining its total maximum subqueries at 36.

The Figure 4.3 shows a comparison between the results of the model resolvers with the formal model and the ones of the resolvers with the testbed.

One eye-catching difference is the curves of the resolver Unbound 1.10.0. The implementation actually retries another time each query that could not be resolved. This behaviour was not captured by the model. The same goes for the first 3 delegations of Unbound 1.16.0 : replies also occur with this version, but the previously collected parameter of maximum 6 parallel referrals is still valid. Indeed, as the resolver tries twice to resolve three delegations, it amounts to 6 referrals. PowerDNS and BIND follow our predicted measures.

The Figure 4.4 shows the comparison between model resolvers and results from the local testbed, this time with QMIN enabled. Here again the only difference is PowerDNS whose maximum number of parallel referrals has indeed increased to 9. Therefore, the simple fact of **enabling** QMIN does not bring much change to our results.

Remark here that no attack exploiting QMIN has been displayed. The reason being that a subsequent type of attack will actually take advantage of the complete CNAME chain, namely Scrubbing + CNAME attack, which should have a higher amplification factor. Thus, it makes sense to enhance this more powerful attack with QMIN. And this shows that simply enabling QMIN does not amplify the attack, but it demonstrates that resolvers may have different limits when QMIN is enabled.

Parallel NS + Unchained + DNAME

This time, we create a DNAME chain that will be followed by the resolver after the NS delegations. We change the type of the CNAME resource records in the nameservers to DNAME (see Listing 4.12). This attack is also launched using the same query for "del.inter.net".

```
; zone in one nameserver
    ; fake delegation
    del.inter.net.        IN      NS      sub.a1.target.com.
    del.inter.net.        IN      NS      sub.a2.target.com.
    del.inter.net.        IN      NS      sub.a3.target.com.

    ; DNAME chains
    b1.inter.net.         IN      DNAME   c1.target.com.
    d1.inter.net.         IN      DNAME   e1.target.com.
    f1.inter.net.         IN      DNAME   g1.target.com.
    ...
    b2.inter.net.         IN      DNAME   c2.target.com.
    d2.inter.net.         IN      DNAME   e2.target.com.
```

```
14      f2.inter.net.          IN      DNAME   g2.target.com.
15      ...
16  ;zones in the other nameserver
17      ; DNAME chains
18      a1.target.com.          IN      DNAME   b1.inter.net.
19      c1.target.com.          IN      DNAME   d1.inter.net.
20      e1.target.com.          IN      DNAME   f1.inter.net.
21      ...
22      a2.target.com.          IN      DNAME   b2.inter.net.
23      c2.target.com.          IN      DNAME   d2.inter.net.
24      e2.target.com.          IN      DNAME   f2.inter.net.
25      ...
```

Listing 4.12: Zones for Parallel NS + Unchained with DNAME

There is no difference between Figure 4.5, representing the number of queries for this attack against model resolvers, and the Figure 4.1 representing the same variable for the CNAME chain version. However, the fact of enabling QMIN in the formal model amplifies the number of subqueries by a factor 2 (cf. Fig 4.6). This is expected as the first record being queried of the DNAME chain has one more label than the one for the CNAME version.

Thus, the resolver will first perform a DNS lookup that will end up in multiple parallel referrals. Then, for each referral, it asks the partial query name "aX.target.com", which will not return any match. Then it will query for the complete query name which returns a DNAME record. The rewriting occurs and the complete QNAME becomes "sub.bX.target.com". The resolver will then continue by querying, again, a partial query name. Note that this operation is considered to be the same as a CNAME substitution. This process will repeat until the maximum CNAME chain depth is reached.

The Figures 4.7 and 4.6 shows the great contrast of this attack with QMIN disabled and enabled. The plot scale had to be adjusted in the latter case, as it doubles all the results.

In the Figure 4.8 with QMIN enabled, the model resolver of BIND has twice the number of subqueries than the resolver implementation. This suggests that BIND is not affected by QMIN and does not strip labels of the query name. Again, its number of subqueries increment by one with the number of NS delegations. Unbound 1.10.0 and Unbound 1.16.0 follow the same pattern previously explained and retry twice the queries. PowerDNS has an interesting behaviour : as QMIN is enabled, we could expect that its subqueries increase until reaching 9 NS delegations. However, it is not the case, it stops at around 50 subqueries. Therefore, there must be a overall query limit. It was previously noted to be 100 queries, and this is consistent with our results, as the numbers displayed in the graph correspond only to half of the chain.

Here we can conclude the Parallel NS + Unchained attack with DNAME is equivalent to the same attack with its CNAME chain containing one more label per records.

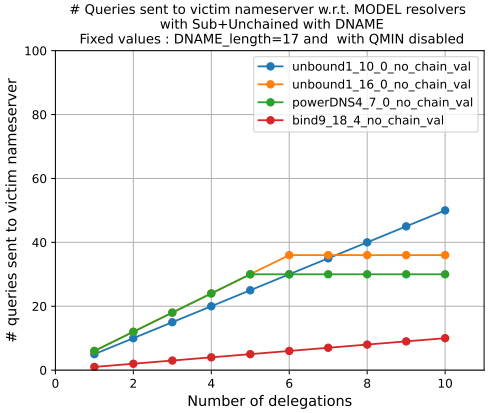

Figure 4.5: Parallel NS + Unchained DNAME using model resolvers, QMIN disabled

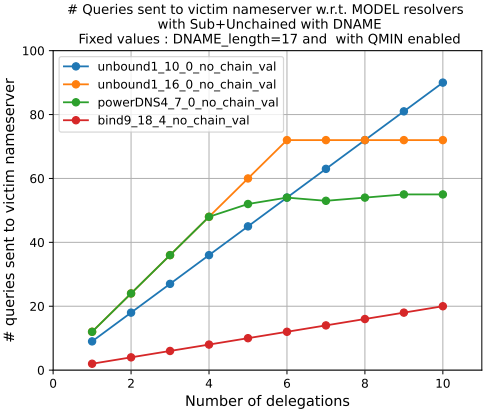

Figure 4.6: Parallel NS + Unchained DNAME using model resolvers, QMIN enabled

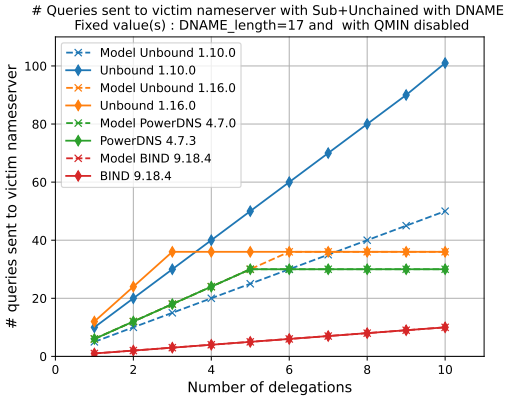

Figure 4.7: A comparison of the Parallel NS + Unchained attack with DNAME between the model resolvers and the resolvers, with QMIN disabled

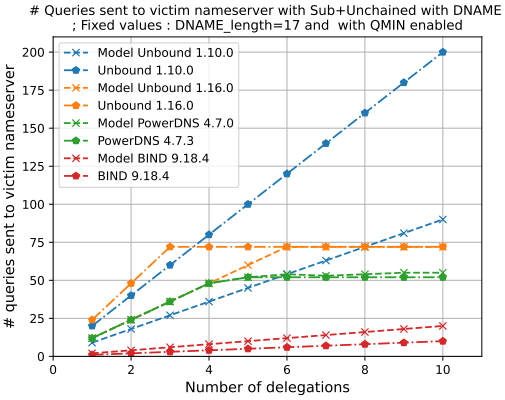

Figure 4.8: A comparison of the Parallel NS + Unchained attack with DNAME between the model resolvers and the resolvers, with QMIN enabled

4.3.4 Parallel NS + Scrubbing + Chain attack

This is the attack which generated the biggest amplification factor observed in the formal model. Here it is also similar to Subqueries Unchained, but more effective because of *CNAME scrubbing* or *CNAME Chain Validation* [22]. This mechanism emerged among popular resolver implementations and consists of following only the first CNAME record of an answer section by querying this canonical name and not considering other data in the answer. The resolvers implementing this feature only trust this first CNAME record. If there is a CNAME chain inside the answer section, it will be simply ignored by the resolvers. This was an response to some cache poisoning attacks.
However, this allows then to install CNAME chained records withing the same single zone, in contrast to the Unchained type of attack where two zones were required. This attack was previously found using a tool generating random zones in the formal model [22]. And it is build the same way as Unchained, the only difference is that the chained records are now inside the same zone. As for the previous attacks, there are two types of chain that can be used.

Parallel NS + Scrubbing + CNAME

We trigger the attacker with a first query for "del.inter.net" which returns several NS records to the starting record of the CNAME chains. An example of the zones for the attack with QMIN is listed in Listing 4.13

```
1  ; zone in inter.net
2  del.inter.net.        IN      NS      lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.
       lab3.lab2.lab1.a1.target.com.
3  del.inter.net.        IN      NS      lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.
       lab3.lab2.lab1.a2.target.com.
4  del.inter.net.        IN      NS      lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.
       lab3.lab2.lab1.a3.target.com.
5  ...
6
7  ; zone in target.com
8
9  lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.a1.target.com.         IN
           CNAME   lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.b1.
       target.com.
10 lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.b1.target.com.         IN
           CNAME  lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.c1.
       target.com.
11 lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.c1.target.com.         IN
           CNAME ...
12
13 lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.a2.target.com.         IN
           CNAME   lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.b2.
       target.com.
14 lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.b2.target.com.         IN
           CNAME   lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.c2.
       target.com.
15 lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.c2.target.com.         IN
           CNAME   ...
16 lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.a3.target.com.         IN
           CNAME   lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.b3.
       target.com.
```

```
17 lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.b3.target.com.          IN
           CNAME   lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.c3.
    target.com.
18 lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.c3.target.com.          IN
           CNAME
```

Listing 4.13: Example Parallel NS + Scrubbing + CNAME + QMIN

By comparing the model plots of this attack and the Parallel NS + Unchained attack with QMIN disabled, both with CNAME version, we can observe that Parallel NS + Scrubbing attack amplifies the number of subqueries received by the target nameserver by a factor two. We expected this behaviour as the resolver now queries all the records of the CNAME chain to the same nameserver.

An interesting aspect that was taken advantage of here is the fact that the CNAME chain limit can be bypassed in the Unbound resolvers. This happens when we query for AAAA records instead of A record. This is not an expected behaviour and this was reported to the developpers. In this attack, we choose the CNAME chain length to be 17, corresponding to the chain limit of BIND, which is the highest of the open resolvers (except for Unbound here).

We can observe a tremendous increase in the model resolvers when QMIN is enabled via the plots 4.9 and 4.10. By activating QMIN, the resolver will use this mechanism for each record in each CNAME chain for each delegation, leading to an astounding amplification factor of 1520 for Unbound 1.10.0. While the other version of Unbound follows at first the same trend, its maximum subqueries is reached at 6 NS delegations, therefore stopping its abrupt increase at 912 . PowerDNS is stable at around 100 subqueries, as this is the limit on the total number of subqueries performed. The results of BIND still do not change, as its implementation strictly forbids to follow CNAME records in subqueries.

When analysing the resolvers in the plot 4.11 with QMIN deactivated, we can arrive to the same previous observations as for Parallel NS + Unchaind : the Unbound resolvers retry twice the queries, Unbound 1.16.0 stops at 3 NS delegations, PowerDNS stops when having reached 100 subqueries (two times the value of Parallel NS + Unchained, due to the scrubbing). BIND resolver simply adds one query per delegation.

Concerning the plot that compares models and resolvers with QMIN activated (Figure 4.12), BIND and PowerDNS are close to their models. While Unbound resolvers and their models follow the same trend, there is quite a difference in the number of subqueries achieved. This is due to the fact that we did not exploit the "bypassing" technique to avoid the CNAME chain limit in Unbound 1.16.0 within the local testbed. For Unbound 1.10.0, the difference can be explained by how we measure the cname chain length : there might be a difference of 1 if we count all the elements of the chain or the number of rewrites. Here there is a difference of one observed.

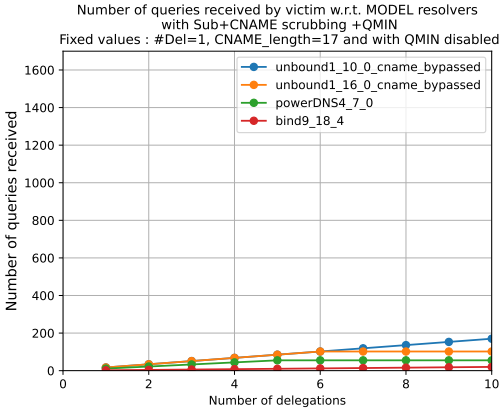

Figure 4.9: Resolver models against Parallel NS + Scrubbing + CNAME with QMIN disabled

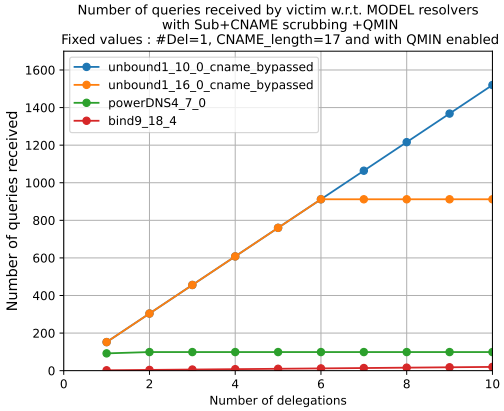

Figure 4.10: Resolver models against Parallel NS + Scrubbing + CNAME with QMIN enabled

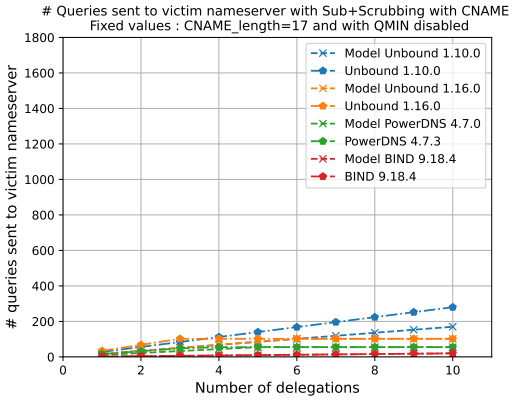

Figure 4.11: Parallel NS + Scrubbing + CNAME with QMIN disabled

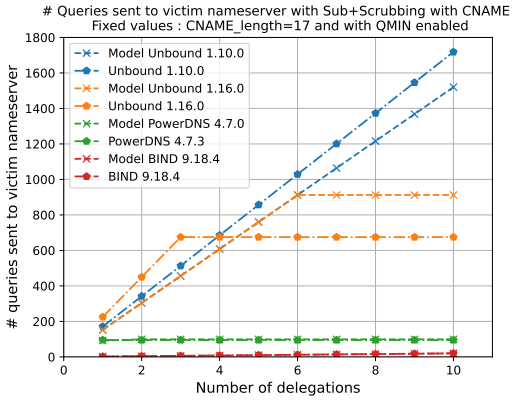

Figure 4.12: Parallel NS + Scrubbing + CNAME with QMIN enabled

Parallel NS + Scrubbing + DNAME

For this variant, the usual substitution from CNAME to DNAME chained records takes place. The target name of the NS records will be modified by prepending another label, e.g., "sub". The substitution of the matching part in the DNAME can then properly be conducted. An exmple of two NS delegations

with a two DNAME chains of length three can be examined in the Listing 4.14.

```
; zone in inter.net
del.inter.net.        IN      NS      sub.lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.
    lab4.lab3.lab2.lab1.a1.target.com.
del.inter.net.        IN      NS      sub.lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.
    lab4.lab3.lab2.lab1.a2.target.com.
...

; zone in target.com

lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.a1.target.com.         IN
          DNAME   lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.b1.
    target.com.
lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.b1.target.com.         IN
          DNAME   lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.c1.
    target.com.
lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.c1.target.com.         IN
          DNAME lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.d1.target
    .com.
...
lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.a2.target.com.         IN
          DNAME   lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.b2.
    target.com.
lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.b2.target.com.         IN
          DNAME   lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.c2.
    target.com.
lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.c2.target.com.         IN
          DNAME lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.lab2.lab1.d2.target
    .com.
...
```

Listing 4.14: Zone for Parallel NS + DNAME + Scrubbing + QMIN

The model resolvers behaviours are reported in Fig 4.13 and Fig 4.14. The first one is exactly the same as the plot from the attack with the CNAME chain with QMIN disabled. The second is similar to the one with QMIN enabled. This is expected, as adding one label to the maximum already used will not generate more queries. Recall that there is a parameter limiting the number of QMIN iterations that is enforced in the model resolvers as well. Even though BIND does not follow CNAME chain in subqueries, we expect this resolver to enforce QMIN since we use a DNAME record, with its MAX_MINIMISE_COUNT set to 5. Thus, a slight increase of the number of subqueries is to occur.

In the plots containing the real resolvers (Figures 4.15 and 4.16, we can observe that they are alike the ones from Parallel NS + CNAME + Scrubbing. In the one with QMIN disabled, the Unbound resolvers experience the retry, while BIND slowly adds one subquery for each delegation. PowerDNS follows its model curve. By enabling QMIN, we can see the tremendous increase induced to the resolvers, similarly to the previous attack. Only BIND shows an unexpected behaviour. Once again, it does not use QMIN, reducing our predicted 5 subqueries per delegations. Moreover, as DNAME record is synthesized into CNAME records, BIND does not follow the chain. However, BIND does retry the failed queries, which increments its amplification factor by 2 subqueries per delegation.

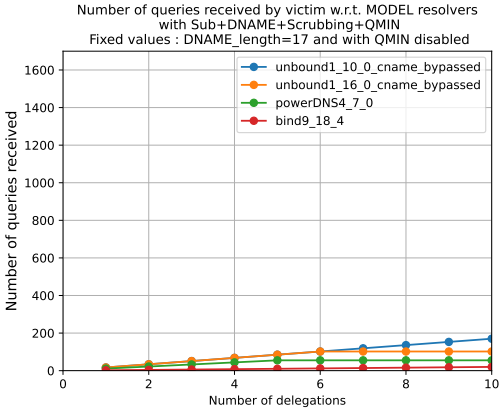

Figure 4.13: Resolver models against Parallel NS + Scrubbing + DNAME with QMIN disabled

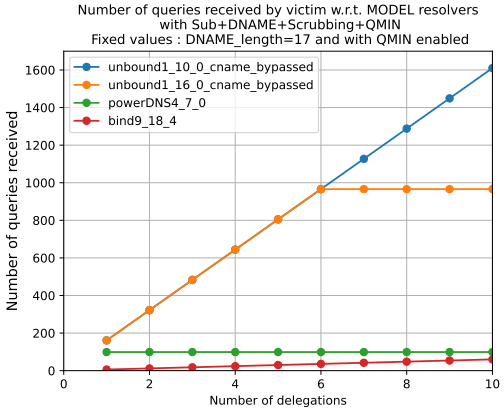

Figure 4.14: Resolver models against Parallel NS + Scrubbing + DNAME with QMIN enabled

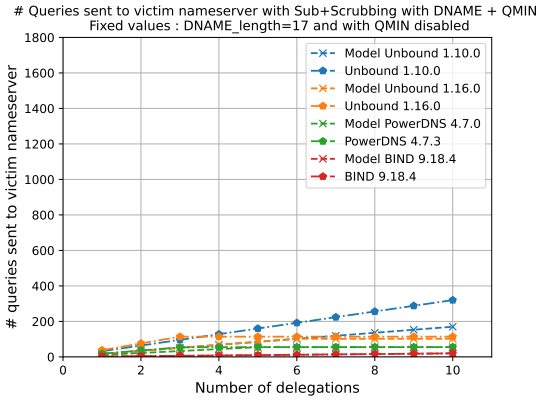

Figure 4.15: Parallel NS + Scrubbing + DNAME with QMIN disabled

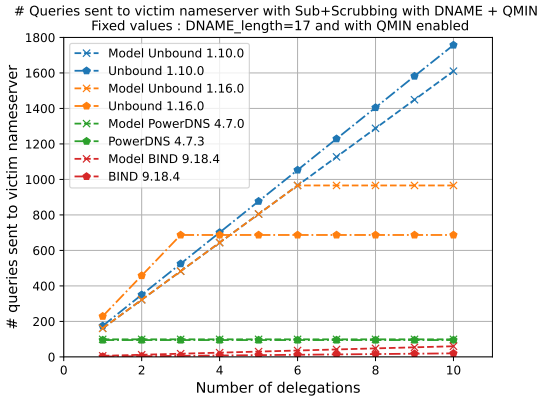

Figure 4.16: Parallel NS + Scrubbing + DNAME with QMIN enabled

4.3.5 Scrubbing attack with delay

This attack exploits the fact that the malicious name servers can arbitrarily delay the responses to the resolver. In order to have the most effective attack, we want to extend the resolution process time as much as possible. This will keep the target resources occupied by our queries. As a resolver usually has a timeout where it either stops the query or retries it, one would like to keep the query "alive" as long as possible taking into account this hard stop time. An appropriate delay would be half the maximum time allotted for a query to stay in the state "being resolved". A longer delay would be accepted by some unsophisticated resolver implementation, smarter resolvers may prevent those seemingly innocuous and slow queries from continuing.

One timeout was observed in the resolvers we focused on : PowerDNS has an overall timeout of around 6 seconds when QMIN is enabled, and 12 seconds

when disabled, meaning that the resolution of a single query will stop after having reached 6 seconds in case of QMIN being activated [22].

This attack leverages the same zones as the Scrubbing attacks. The only adjustment is done inside the attacker nameserver, where we define a range of delays varying from 0 to 1400 milliseconds by increment of 200 milliseconds. The maximum artificial delay was previously chosen because it was the highest value for which all the resolver implementations gave meaningful results. If greater delays were defined, some resolvers would not make any progress with the resolution, mostly due to a timeout. The formal model has the capacity to instantiate "delaying" nameservers and to measure the spent time through its monitoring tools.

Note that we did not plot the variant of this attack with DNAME since, as we have seen with the previous experiments, the results are similar to the CNAME version.

Scrubbing + CNAME + QMIN + Delay (slowDNS)

In the Figure 4.17 are displayed the curves of the models and of the resolvers for the attack Scrubbing + CNAME + QMIN combined with the delayed malicious nameserver.

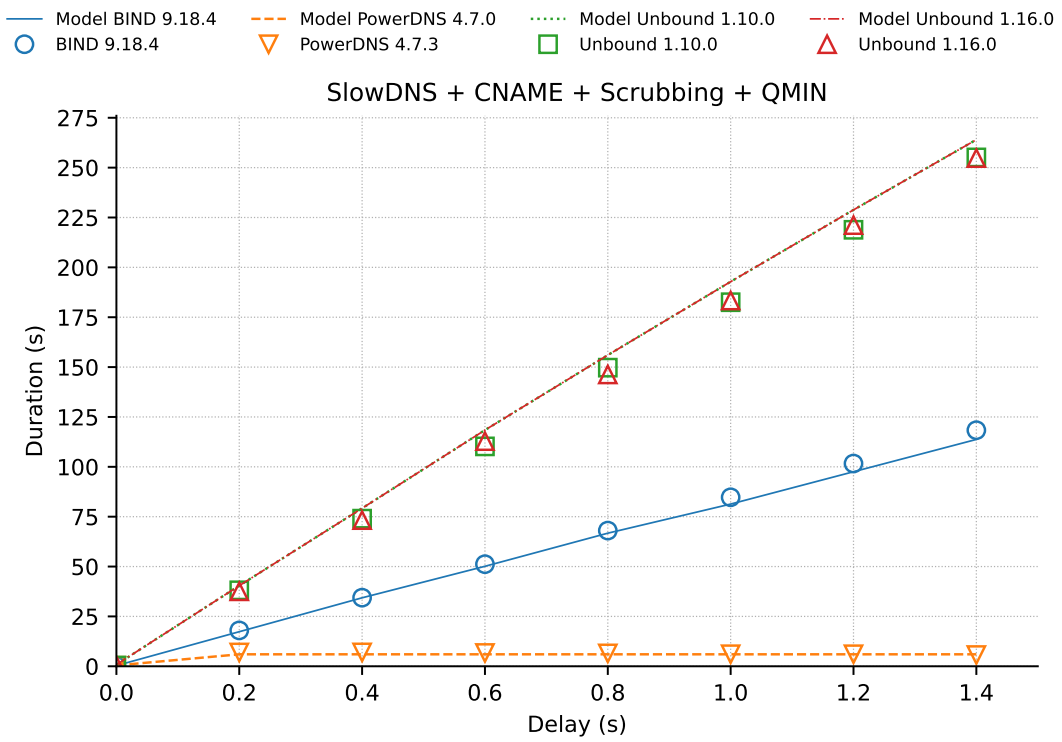

Figure 4.17: SlowDNS + CNAME + Scrubbing with QMIN enabled

First, we can notice the dramatic impact that a small delay on all queries has on the total duration of the query. A delay of 400 milliseconds can result in the interaction between the resolver and the malicious nameserver to last 75'000 milliseconds, or 75 seconds. This attack alone does not exhaust the resources of a resolver, but keeps it busy and waiting for the nameserver responses. In the case of a botnet, this would result in a denial-of-service.

The model resolvers predict with a relatively good accuracy the duration of the delayed queries of the resolver implementations. Only differences of a few seconds can be noticed.

PowerDNS is the resolver with the lowest duration. Its timeout of 6 seconds prevents the attack from keeping the queries alive for a long time. BIND takes the third place with a duration of 113 seconds for a delay of 1.4 seconds. The time amplification factor in this case would be almost of a factor 90. The Unbound resolvers, both bypassing the CNAME chain length, obtain the longest duration with around 300 seconds, resulting in a amplification factor of 210.

The BIND resolver and both version of Unbound do not seem to have a timeout limit. That is problematic as one nameserver could maintain a connection and regular exchanges over an infinite amount of time.

This attack shows that implementing a timeout in the resolver is crucial to avoid a potential resource exhaustion. PowerDNS is the only one of the tested resolvers to enforce a timeout, and that makes it the least vulnerable to such kind of attack.

4.3.6 Parallel referrals + Referral chain

This attack makes use of the "breadth-like" capability of parallel NS delegations and combine it, this time, with NS chains instead of CNAME chains. A simple zone example can be found in the Listing 4.15.

```
; zone in inter.net
del.inter.net.        IN      NS      a1.target.com.
del.inter.net.        IN      NS      a2.target.com.
del.inter.net.        IN      NS      a3.target.com.
...

; zone in target.com
a1.target.com.  IN      NS      b1.target.com.
b1.target.com.  IN      NS      c1.target.com.
c1.target.com.  IN      NS      d1.target.com.
...
a1.target.com.  IN      NS      b1.target.com.
b1.target.com.  IN      NS      c1.target.com.
c1.target.com.  IN      NS      d1.target.com.
...
a3.target.com.  IN      NS      b3.target.com.
b3.target.com.  IN      NS      c3.target.com.
c3.target.com.  IN      NS      d3.target.com.
...
```

Listing 4.15: Zone for Parallel NS + Referral chain without QMIN

We mount this attack with 20 NS delegations with a distinct target name made up of 10 labels. The target name is under the authority of another nameserver. Each target starts a NS chain of length 17. The outcomes and observationsd are reported in the following Table 4.6.

Note that, as NS chain limit were not enforced in the model resolvers, we only show the results of the resolvers in the local testbed.

Resolver	Parallel referrals	NS chain depth	Maximum subqueries
BIND 9.18.4	5	7	35
Unbound 1.10.0	No limit	4	798
Unbound 1.16.0	3	4	133
PowerDNS 4.7.3	1	10	95

Table 4.6: Limits observed in the resolver for the attack Parallel referral + Referral chain + QMIN

Once more, QMIN does not affect BIND. Its maximum number of subqueries is a simple multiplication of the parallel referrals followed and the NS chain depth. Unbound 1.10.0 descend down to 4 NS records and does not seem to have a limit on the number of the parallel referrals followed. Therefore, it obtains the largest number of subqueries at 798. Note that it corresponds to the subqueries observed, the chains could be lengthened.

With the other version of Unbound, its amplification is smaller but still reasonable with a factor of 133. However, it does have a limit of 3 parallel referrals and follows the same chain depth as Unbound 1.10.0. Both versions retry the failed queries, thus increasing the amplification.

PowerDNS only follows one parallel referral but goes down to the 10th record of the NS chain. Moreover, QMIN is very effective with this resolver as each label is queried. Remark that the overall limit of subqueries is still enforced.

4.3.7 NS trees or fan_out times fan_out

This attack is a multiplication of an attack vector : namely the parallel NS. The first (and only) query from the resolver triggers a response with 30 NS delegations (constituting the 1st parallel NS component). When they are queried for by the resolver, it will receive, again, a response with 30 NS delegations (the second use of Parallel NS).

We know from previous observations that some resolvers implement a limit on the number of parallel referrals followed. Nevertheless, it may be enforced only for the first use of parallel referral and not the second level. We only test this combination of vectors with only two components. If there is evidence that resolvers do not enforce a limit on this second level of NS referrals, it would be

also likely the case that they do not enforce on further deep levels. Therefore, testing the multiplication is enough to show the effectiveness and the fact that we may be able to chain more NS delegation levels.

Here we will show two variants of this attack.

4.3.8 Simple NS Trees

This variant consists of recreating the version of the attack presented earlier without exploiting QMIN. The second level of NS delegations has its target inside its **own** zone.

Resolver	1st NS level	2nd NS level	Maximum subqueries
BIND 9.18.4	5	49	59
Unbound 1.10.0	30 (No limit ?)	63	123
Unbound 1.16.0	3	8	11
PowerDNS 4.7.3	1	5	6

Table 4.7: Limits observed in the resolver for the attack fan-out times fan-out

Note that enabling or disabling QMIN only changes slightly the number of subqueries reported, at most by 1 subquery for Unbound resolvers.

BIND only follows 5 NS records from the 1st level and then 49 of the second level. Unbound 1.10.0 queries twicethe target of all the NS records of the 1st level and then queries 63 records of the second NS level. Unbound 1.16.0 randomly chooses 3 NS records of the 1st level and queries between 1 to 3 records of the second level pointed by the first level. PowerDNS only follows 1 NS records of the 1st level and then 5 records that are pointed by the one from the first level.

This attack is quite effective against the Unbound 1.10.0 as it seems not to have a limit on the number of parallel referrals followed at the first level.

4.3.9 NS Trees + QMIN

We construct this attack with 30 NS delegations whose target name contains 10 labels. Another nameserver is authoritative for the zone containing the target name.

BIND is not affected by QMIN and queries the complete query name. It follows 5 referrals of the 1st level two time and then 56 records at the 2nd NS level. Once again, Unbound 1.10.0 follows all the first referrals twice while using QMIN. Then it follows 62 second referrals twice again. However, this time QMIN seems to be deactivated as only the first label and then the complete query name are queried. Unbound 1.16.0 only follows 3 first referrals while using QMIN. Then at the second level, QMIN follows the same pattern as the other version of Unbound. The

Resolver	1st NS level	2nd NS level	Maximum subqueries
BIND 9.18.4	5	56	66
Unbound 1.10.0	30 (No limit ?)	62	783
Unbound 1.16.0	3	8	49
PowerDNS 4.7.3	1	5	17

Table 4.8: Limits observed in the resolver for the attack fan-out times fan-out with QMIN

resolver queries 8 second referrals multiples times. Concerning PowerDNS, the resolver queries one first referral with QMIN then follows 5 second delegatios without using QMIN.

This attack seems to determine that QMIN at the second NS level is deactivated. However, a large amplification factor was found in Unbound 1.10.0.

4.4 Mitigations Observed

During those experiments, several mitigations were noticed due to the resolvers behaviours.

For instance, Unbound 1.16.0 has a limit of 3 parallel referrals followed, whereas BIND has 5 and Unbound 1.10.0 does not seem to be affected by this kind of limitations. We remark also limit in the CNAME chain depth as well as on the number of QMIN iterations. The four tested resolvers deactivated QMIN when querying second level referrals.

One more unveiled mitigation is to completely stop the resolution of the query if it involves an excessive number of queries being sent. PowerDNS has a similar mechanism with a "work budget" that, once it is reached, stops the DNS lookup process. Furthermore, a restraint on the duration of a query was observed in this resolver.

4.5 Conclusion

In this chapter, we build attacks composed of several different attack vectors and compare their results with model resolvers, if relevant. We use the "breadth-like" capability of Parallel NS and the "depth-like" capability of CNAME/DNAME/NS chains as well as the "probing" capability of Query Name Minimisation. By coupling these aspects into different combinations, we can vary the amplification factor to maximise the impact on the victim nameserver.

Here QMIN acts as an orthogonal attack vector that provides a multiplication factor at each possible component of the attack.We observed that simply enabling

QMIN in the resolver may amplify the attack factor or at the worst keeps the same amplification factor. Implementations that do not enable QMIN closes the door to attacks enhanced by this feature.

Delaying the response is conceptually also an independent attack vector, which can be combined to produce surprisingly long lasting queries.

Moreover, the attacks using CNAME can be reproduced with DNAME records. This is expected as, when the resolver receives a DNAME record, it will rewrite the query name and consider the process as a CNAME substitution, thus consituting the equivalence.

5 Measurements

This section will be dedicated to the testing and validation of the attacks using resolvers in a more open environment, namely the Internet.

5.1 Methodology

In order to evaluate the impact of QMIN-enhanced attacks "in the wild", two controlled name servers are setup and serve different chosen domains, namely *qmin-tax.net* and *qmin-tax.org*. Then we launch a query using "dig" tool [1] allowing us to make a DNS lookup for a specific name via a chosen resolver. Note that we focus only on IPv4 addresses. Each query will trigger a distinct attack and should follow a list of domains that we prepared specifically for it beforehand. The nameserver authoritative for the zone *qmin-tax.net* is the target on which we will measure the number of subqueries received for the specific attack.

We set up the following plan for these measurements :

1. Find around 20 popular open resolvers and collect multiple IP addresses for each of them
2. Setup the zones in the controlled name servers
3. Launch the queries and collect the resulting logs
4. Count the number of queries and conclude whether the attack works

5.2 Motivation

20 open resolvers is an arbitrary number and may not be enough to dress a general behaviour of the open resolvers in relation to the attacks. However, it is enough to show and confirm that our attacks are effective against current resolvers over the Internet. The resolvers should be chosen regarding their popularity, as the possible effects of QMIN attacks such as resolver crashes would impact more users and would prove the attack to be urgently mitigated. Two or more IP addresses should be collected to avoid response caching and to double-check our results. Mentioned by the lifewire website [20], some open resolvers and their IP addresses are displayed in the Table 5.1.

Resolvers	IPv4 addresses
Google	8.8.8.8, 8.8.4.4
Control D	76.76.2.0, 76.76.10.0
Quad9	9.9.9.9, 149.112.112.112
OpenDNS Home	278.67.222.222, 208.67.220.220
Cloudflare	1.1.1.1, 1.0.0.1
CleanBrowsing	185.228.168.9, 185.228.169.9
Alternate DNS	76.76.19.19, 76.223.122.150
AdGuard DNS	94.140.14.14, 94.140.15.15
DNS.WATCH	84.200.69.80, 84.200.70.40
Comodo Secure DNS	8.26.56.26, 8.20.247.20
CenturyLink	205.171.3.65, 205.171.2.65
CIRA Canadian Shield	149.112.121.10, 149.112.122.10
SafeDNS	195.46.39.39, 195.46.39.40
OpenNIC	159.89.120.99, 134.195.4.2
Dyn	216.146.35.35, 216.146.36.36
Yandex DNS	77.88.8.8, 77.88.8.1
Hurrican Electric	74.82.42.42
Neustar	64.6.64.6, 64.6.65.6
Freenom World	80.80.80.80, 80.80.81.81
DNS for Family	94.130.180.225, 78.47.64.161

Table 5.1: The list of open resolvers and IPv4 addresses collected.

Concerning the zones, we created specific zones for each attack, so that they don't overlap nor induce some unexpected behaviours. We do so by clearly writing the name of the attacks as a first label of the zone of the targeted domain. An example of zones for the Unchained attack is showed in the code listing 5.1.

```
1  // Zone for Unchained attack in qmin-tax.com nameserver
2  a1.unchained.qmin-tax.net.  IN  A  b1.unchained.qmin-tax.
       org.
3  c1.unchained.qmin-tax.net.  IN  A  d1.unchained.qmin-tax.
       org.
4  e1.unchained.qmin-tax.net.  IN  A  f1.unchained.qmin-tax.
       org.
5  ....
6  // Zones for Unchained attack in qmin-tax.org nameserver
7  b1.unchained.qmin-tax.org.  IN  A  c1.unchained.qmin-tax.
       net.
8  d1.unchained.qmin-tax.org.  IN  A  e1.unchained.qmin-tax.
       net.
9  f1.unchained.qmin-tax.org.  IN  A  g1.unchained.qmin-tax.
       com.
10 ...
```

Listing 5.1: Zones for Unchained attack

Due to the recognizable zones, we are able to easily find which requests correspond to which attacks. Thus, a simple execution of all the "dig" queries one by one should suffice to collect all the desired data, as all queries and zones don't interfere with each other. This should not disrupt the resolvers as we only run each query exactly once, so it should not prevent other users to query them. Moreover, it allows us to use the logs to find, retrieve and possibly split the data among separate files. And a simple count of the queries gives us the amplification factor effective for each combination of resolvers and attack.

For instance, the query used for the Unchained attack for the primary DNS of Google could be :

```
1  dig @8.8.8.8 a1.unchained.qmin-tax.com
```

Listing 5.2: Dig query for Unchained attack

5.3 Implementation details and observations

The two nameservers allocated for this experiment were built up with BIND version 9.18.4 (same version as in the local testbed) as it is a relatively easy server implementation to download and install. They were to accept any queries and log the queries in a specific file (file that should be created beforehand and which

BIND should have the write permissions). DNSSEC was also disabled to avoid more complicated steps to properly setup the nameservers.

We observed in the logs that at least one resolver (Google) uses different addresses along the resolution of a single query, i.e., a subquery is sent by a particular IP address to the controlled nameserver and the related subsequent subquery is sent by other distinct addresses. This may be explained by the fact that the resolver probably chooses the actual nameserver that is the closest to the client location or, as they are a complex system of nameservers, they may choose another nameserver to balance the load on the system. Thus, as a particular resolver may not be easily found in the query logs by looking at the IP address, a simple solution to this is to launch the attacks against a single resolver one at a time after some specific delay, then continue the attacks with another resolver. We can then separate the logs according to the log timestamps and either copy them to a file distinct for each resolver and each attack or directly count the number of queries between two timestamps according to their suffix. Recall that the suffix is made up of a distinct label for each attack.

Another remark was that resolvers may stop our queries if they are launched too close in time. That was the case with Quad9 which stopped responding to our queries after the 5th attack. One way to avoid this was to delay the queries triggering the attacks. Our experiments show that a delay of five seconds between each attack seems to produce good amplification results.

More research showed that resolvers retry the query after some time. To ensure that no query retries are missed in the logs, we add even more delay between the attacks on the same resolver.

One point that was noticed after some iterations of the experiment was that some commands sent were met with a "connection timed out" message, which actually could be expected as some of our attacks may take some time for the resolvers to process. Unfortunately, the dig command does not display the time at which the command has been run. We had previously considered that, when the time has not been found, the attack has been unsuccessful and set the number of queries for this attack and the resolver to zero. That is actually incorrect as we can observe in the logs that some queries were received at our nameservers for the attacks where a timeout occurred. Thus, when this happens, we should properly write the time at which the attack is being launched and add another delay to let all the subsequent queries reached the nameserver without being interleaved with queries of another attack. An additional delay of 25 seconds derived from such observed cases seem to allow enough time in case of timeout.

Moreover, some resolvers may change the capitalization of some letters of the queries. Thus, it may modify as well the first label allowing us to detect which attack has been launched. Therefore, special care should be taken when identifying which lines correspond to a query of an attack.

With all those precautions set up, we are more confident to say that attacks with a result of 0 queries are, indeed, unsuccessful.

5.4 Launched Attacks

We tested all the previous attacks combined with QMIN and collected the number of queries sent by the real resolvers, which actually should approximately correspond to the number of queries received by our controlled name servers. If the domains *qmin-tax.org* and *qmin-tax.net* are not yet cached, it may add a few queries. By looking at the resulting logs, we can observe that some resolver do not implement QMIN at all. The results are listed in the table below (Table 5.2). Note that we implicitly use attacks with QMIN as the objective here was to verify the effectiveness of the attacks exploiting this feature. We did not apply the Subqueries Unchained with QMIN, but simply the Unchained attack with QMIN, the reason being we wanted to verify that this more primitive vector is still effective against recent resolver implementations.

5.5 Results and Conclusion

Here we see that the attacks applied to some resolvers are effective and this shows that QMIN can indeed have a great impact on real resolvers implementations, with up to a factor 1200 in the case of resolvers Hurricane Electric and Control D. These spikes at around 1200 are interesting because we did not use the attack with the highest expected amplification factor. The "parallel-referrals" attack consisted of 30 NS records for the owner "parallel-referrals.qmin-tax.net." with target being each time a variant of "lab12.lab11.lab10...lab1.a1.parallel-referrals.qmin.tax.net" (thus still within the same zone).

We can compare it with the attack "parallel-referrals-v2" where the target is a long QNAME to another nameserver. The resolvers, where the previous attack was the most impactful, handle well the version 2 of the attack with a dividing factor of almost 100.

Other resolvers just do not implement QMIN in their resolution process, which, thus, reduces the potential number of queries but reduces as well the level of privacy of the user requests. This may be detected by the number of queries captured. Given that QMIN is exploited by adding many labels in the query name, we can assume that a relatively small number of queries would mean that QMIN is not enabled or implemented in those resolvers. That makes sense since we prepended at least 10 labels to records in most of our attacks. However, we have to take into account the structure of the attack to determine the limit of queries under which we assume that the resolvers do not employ QMIN. A look at the query

Attack	parallel-referrals	parallel-referrals-v2	referral-chain	referral-chain-qmin	referral-chain-ip-loop	unchained-cname-qmin	unchained-dname-qmin	scrubbing-cname-qmin	scrubbing-dname-qmin	parallel-referral-chain	parallel-scrubbing-cname-qmin	parallel-scrubbing-dname-qmin	ns-trees	ns-trees-qmin
AdGuard DNS-1	8	61	8	43	5	5	5	112	9	5	5	5	5	5
AdGuard DNS-2	24	33	14	18	1	5	5	43	4	5	5	5	5	5
Alternate DNS-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Alternate DNS-2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CIRA Canadian Shield-1	4	5	6	4	0	1	1	16	1	1	1	1	1	1
CIRA Canadian Shield-2	4	5	5	5	0	1	1	4	1	1	1	1	1	1
CenturyLink-1	0	7	5	5	2	2	2	199	5	1	1	1	1	2
CenturyLink-2	0	7	8	12	2	2	2	200	5	1	1	1	1	2
CleanBrowsing-1	36	8	16	50	2	2	2	73	9	1	1	1	1	2
CleanBrowsing-2	36	8	16	45	2	2	2	33	9	1	1	1	1	2
Cloudflare-1	2	11	51	63	2	2	2	13	2	1	1	1	1	2
Cloudflare-2	2	6	50	10	1	1	1	40	1	1	1	1	1	1
Comodo Secure DNS-1	28	61	12	35	0	5	15	75	3	5	5	15	5	5
Comodo Secure DNS-2	26	60	25	20	2	5	5	105	4	5	5	5	5	5
Control D-1	1225	4	66	78	0	1	1	123	1	2	1	2	0	0
Control D-2	1221	1	48	213	0	3	2	131	3	1	2	1	1	1
DNS for Family-1	12	22	15	1	0	1	1	6	1	1	1	1	1	1
DNS for Family-2	13	19	15	15	1	2	1	6	1	1	1	1	2	1
DNS.WATCH-1	132	1	9	12	0	0	0	0	0	0	0	0	0	0
DNS.WATCH-2	0	33	2	38	2	0	0	14	1	6	2	0	0	5
Dyn-1	25	1	11	14	2	5	5	23	2	0	5	5	6	5
Dyn-2	32	4	10	16	2	5	1	25	1	5	5	2	5	5
Freenom World-1	1	223	3	146	1	10	15	306	2	0	2	1	15	15
Freenom World-2	0	201	1	132	1	0	2	72	1	0	2	3	1	3
Google-1	3	20	15	15	0	0	2	6	1	1	1	2	2	1
Google-2	9	21	15	14	1	1	1	6	1	2	2	2	2	1
Hurricane Electric	1199	14	181	23	2	2	2	2	10	1	1	1	1	2
Neustar-1	8	30	13	28	4	5	0	39	1	0	5	5	5	5
Neustar-2	4	0	12	39	1	5	5	81	1	4	5	5	5	5
OpenDNS Home-1	104	4	6	16	2	2	2	12	13	1	1	1	1	2
OpenDNS Home-2	19	17	6	55	2	2	2	12	13	1	1	1	1	2
OpenNIC-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
OpenNIC-2	0	10	4	10	3	1	2	199	4	1	1	1	1	2
Quad9-1	616	24	134	48	2	6	2	0	1	7	1	1	5	1
Quad9-2	10	0	11	208	3	0	1	16	4	1	4	1	1	2
SafeDNS-1	4	35	5	5	0	0	0	5	1	3	2	4	4	3
SafeDNS-2	2	35	1	4	1	0	0	2	1	4	5	5	4	5
Yandex DNS-1	10	504	14	12	2	25	8	17	1	8	2	8	4	10
Yandex DNS-2	4	435	13	19	4	2	7	132	1	10	4	24	3	4

Table 5.2: Number of queries received at target nameserver *qmin-tax.net.* for each attack and each open resolvers.

logs would have shown that QMIN is not used by the resolvers for a certain attack, but this may not be feasible for numerous combinations of attacks and resolvers. Therefore, we may theorize the maximum number of queries by following the whole chain or all the NS referrals encountered. If the number of queries observed is smaller than this limit, it may mean QMIN is not enabled. There is still some uncertainty about this process since it does not take care of retries. However, if we just want to focus on the number of queries reported, it should be a good enough heuristic. Note that this proposed technique is probably supplanted by the one proposed in [19] which is described in chapter 7 and consists in setting different NS responses for QNAMEs according to their length.

There is some small variability between the first IP address and the second of the list of resolvers. By looking at the specific hostnames ("security", "family", etc. . .), we understand there are various filterings of the addresses and maybe other checks performed which can explain those differences. However, it seems that in general the amplification vector between both are similar.

Moreover, some resolvers do not respond to our queries or timeout without sending any queries to our nameservers. That is the case for Alternate and one resolver of OpenNIC. Another explanation could be that IP addresses of the resolver were updated, meaning that we sent queries to invalid or unresponsive servers.

We compiled the histogram of the amplification factors collected in Figure 5.1 (on the next page) to have a better overview of the impact of these attacks.

This figure shows that there are many attacks resulting in a low number of subqueries received at the nameserver. That could be explained by the fact that the resolvers implement a combination of countermeasures that reduce greatly the amplification factor.

The most **consistent** and powerful attacks across all the resolvers are the attacks involving the chain of NS referrals, many parallel referrals leading to long target names and the scrubbing CNAME chain. They are all combined with QMIN. The other attacks do not show strong and steady amplification factors.

The highest amplification factor observed is around 1200 when using the open resolvers "Hurricane Electric" and "Control D".

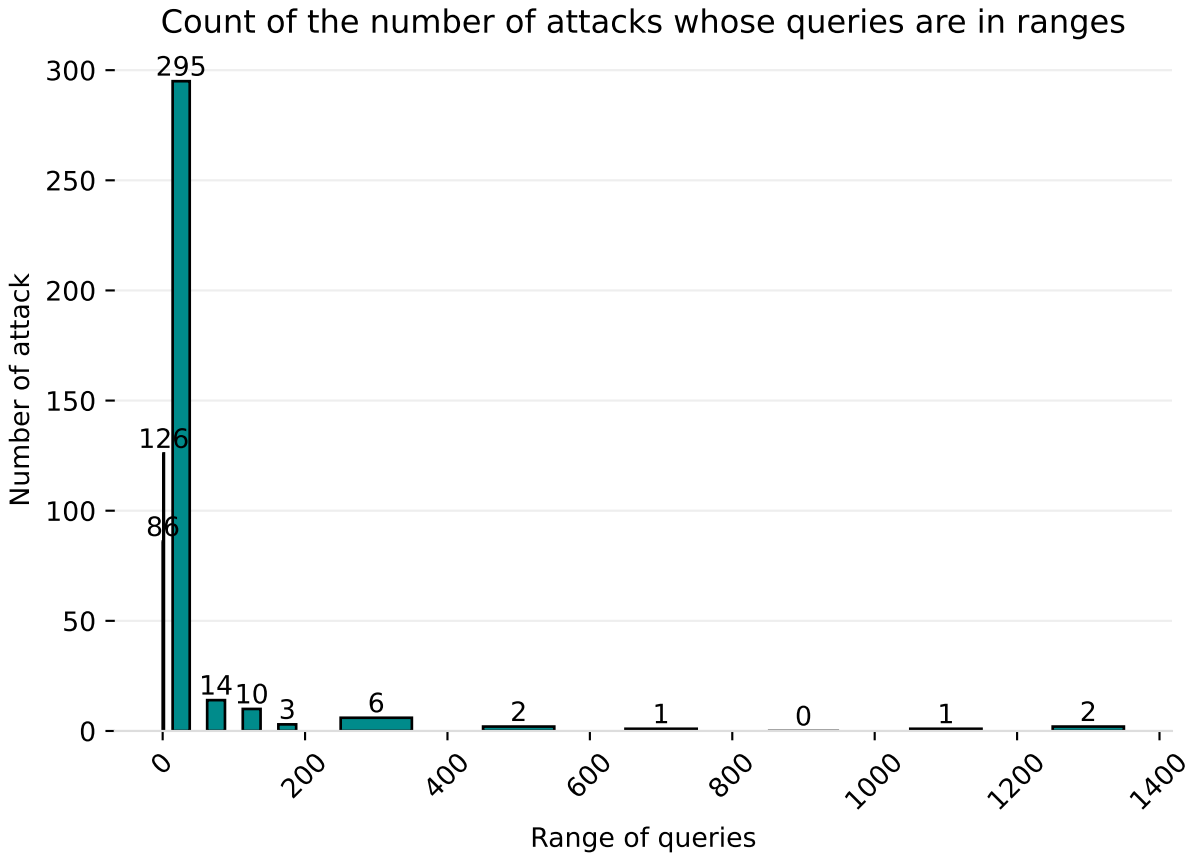

Figure 5.1: Counts of the different attack subqueries

6 Discussion

6.1 Benefits of QMIN

As explained in chapter 2.1.3, the DNS resolution without QMIN implies that most of the name servers contacted will receive the whole original QNAME, despite the fact that they may simply redirect to another name server along the list of labels. Thus, they do not need to know the full query name but receive more information than necessary.

QMIN was designed to prevent this leakage to happen, by effectively reducing the number of labels in the Query Name that will be sent to the name servers. This concept also rejoined the idea that the size of data exchanged should be minimised to improve the users' privacy levels.

The adoption of QMIN has been steady and was even smoothen because the nameservers cannot infer zone cuts from the queries, thus to improve privacy, resolvers need to probe other nameservers until the zone cuts are discovered (if any). This proves that QMIN is indeed an important privacy-enhancing feature, and it is recommended to be implemented in all resolvers.

6.2 Flaws and Direct Impacts

On the other hand, the QMIN mechanism directly increases the number of subqueries sent by the resolver. Indeed, this implies that all the attacks that were previously effective with a QMIN-free resolver configuration can now be amplified. This inherent behaviour makes the resolvers implementing Query Name Minimisation particularly vulnerable to attacks that exploit this feature in addition to their initial attack vectors.
Most of them have already enforced different limitations on the number of total subqueries, the depth of the subqueries, or the number of retries to handle previously discussed attacks such as NXNS, iDNS or Unchained. Those mitigations can be effective up to some level against QMIN.

In particular, the resolver BIND deactivates completely QMIN in its implementation, forcefully removing the multiplicative effect of QMIN over the attacks, at the expense of removing any user request privacy.

Moreover, as we have seen, partial countermeasures do not prevent attacks to occur. The most powerful amplification factor observed in the local testbed is oscillating between the one from the Parallel NS + Scrubbing + CNAME + QMIN attack and the one from the Parallel NS + Scrubbing + DNAME + QMIN attack, with the maximum subqueries found being 1718 and 1757, respectively. Whereas the highest amplification factor of the measurements is around 1200 with a unintended attack that is parallel-referral with the NS delegations lying within the same zoe. Both attacks exploited the QMIN mechanism to amplify their impact.

6.3 Possible Mitigation

A particularly interesting aspect that is exploited in combination with QMIN is the reset due to query rewriting. This allows to continue abusing the QMIN mechanism as long as another limit is not reached. One solution would be to stop the querying process when an unreasonable number of queries are requiring multiple rewritings. A more specific study should be conducted to defined the proper number of rewriting allowed.

If the focus is only on QMIN, another study should be carried out to determine the new optimal value for the two QMIN parameters defining the maximum iterations and times only one label should be sent. Because the current values are enabling these amplification factors.

One thought to keep in mind is that only one countermeasure is probably not enough to take care of all those attacks, thus a cocktail of mitigations would be the most likely solution to refrain those attacks. However, to solely hinder QMIN amplification, we could directly disable it. But that is not a solution as this does not improve the user's situation regarding their privacy. So except by lowering its parameters values (mentioned above), we should focus on other aspects of DNS that could cut down on QMIN amplification potential.

6.3.1 Zone Cuts

We could for instance focus on an improvement concerning the zone cuts. Note that this has not been tested nor implemented, it is a simple proposition for improvement.

Assuming that a resolver is trying to resolve a long QNAME, when it first tries to contact a nameserver authoritative for a zone in the stripped QNAME, the queried nameserver could add in its response the next closest number of labels needed to go to a zone cut. If the resolver sees that it is bigger than the number of left labels of its QNAME, it just continues adding label per label (or following its respective algorithm). Otherwise, the resolver could simply try to add the necessary number

of label to get to the zone cut directly, thus preventing the resolver to send too many probing requests.
This supposes that there is not an excessive number of zone cuts inside the zone for which the nameserver is authoritative of. Otherwise, this proposed mechanism would not bring any gains as the resolver would actually simply follow its conventional QMIN implementation. It does even add a slight overhead to the messages since the number of labels until the next zone cut would be incorporated in the response.
Another disadvantage is that it would require to modify DNS protocols to incorporate this new mechanism to the messages. Thus, a rapid deployment would probably not be possible. Moreover, we assume that a non-malicious nameserver is sending correctly the next number of labels. In the case of a malicious nameserver, the resolver may not distinguish between a genuinely-run nameserver and one controlled by an attacker, since the two types of behaviour would be legitimate with this proposal. However, this algorithm would reveal the path taken by the resolver if we analyse the number of queries sent to each contacted nameservers. We could then infer what is the original QNAME or at least a considerable part of it. Thus, even though we could perform the same analysis on the current DNS, this may not be an ideal solution for privacy but it may reduce the amplification resulting from the use of QMIN.

6.3.2 Another Variant

Another variant to consider would be one where the contacted nameserver send a NODATA rcode, meaning that the current QNAME does not have any valid of its own, but it has one descendant who does (see chapter 7). The nameserver would additionally incorporate in its response the next records whose owner name is a descendant of the current query name. The resolver would then be able to see whether one of the received records corresponds to the original query name. It would then either get the response, or would continue querying the nameserver with a longer QNAME than the ones in the response.

This has the advantage of avoiding sending an unusable response to the resolver, since the latter would more than likely query again the nameserver. The resolver would then gain more knowledge about how many labels it can directly add to the current query name if the records of the response do not answer its query. Once again, it creates more load on the nameserver, since it has to look for the descendant of the request name. Optimized search should be implemented in the nameserver in case of a large record database to avoid that the nameserver is being overwhelmed by search, which could open up a new vulnerability to DoS attacks.

We assume that the number of attached records is reasonable since the query name should put the focus on a smaller part of the database of the nameserver. We further suppose the attached records have the same number of labels, but we

could imagine an optimized strategy that would incorporate more data if there are a few records attached. Nevertheless, if an excessive number of records would be attached to the response, the nameserver could simply respond without additional section and wait for a longer query name.

7 Related Work

7.1 Root Servers and Privacy

With a cold cache and without QMIN, root nameservers would be the first nodes to receive the request by the recursive resolver. They would be the initial ones to know the full query name of the user request. A study by Hardaker [21] shows that high level analysis techniques could be used on data collected by the root nameservers to derive useful information about the structure of a particular network and its users. This could allow to know the different devices connected, their operating systems and applications that they use. All helpful and practical data it takes to mount a tailored attack.

Hardaker proposed "LocalRoot" to entirely avoid the privacy risks enabled when interacting with the root nameservers of DNS. It is a recursive resolver containing a copy of the root nameserver following the RFC 7706 [24], which was originally written to decrease the access time to DNS root servers and to prevent third parties from snooping the requests. That means that the recursive resolver, when resolving a request, can, therefore, avoid asking the root servers and leaking unnecessary data by "running" and updating a local instance of a root server.

While pre-caching the entire root zone data may store unnecessary records and needs frequent updates, this solution takes care of completely mitigating the data leaked to root nameserver and TLDs. Whereas it is partially protected by QMIN, as the root nameserver will know only one more label than the domain name it is authoritative of. In spite of reducing the time to access root information, LocalRoot does not consider data disclosure further down to servers authoritative of subdomains. That is when QMIN enables more privacy gains. Therefore, a combination of both would be beneficial for the resolver to improve the user's privacy.

7.2 Slow but Steady Adoption of QMIN

The paper "A First Look at QNAME Minimization in the Domain Name System" [19] showed the adoption rate of this feature in open resolvers starting in April 2017.

De Vries et al.[19] studied during a one-year experiment the adoption rate of QMIN amongst open resolvers. They conducted two types of Internet-wide measurements. The first one is using RIPE Atlas probes. RIPE Atlas is a large measurement network with around ten thousands of small hardware devices, also called probes, that report the Internet connectivity and its reachability in a particular zone of the globe [3]. The probes were utilized to query a specific subdomain that was under the researchers' control. As each probe has its own list of resolvers, it will result in a extended study of the attack. For the second type, they replaced the probes by open resolvers and used the same queries.

Their goals were to report QMIN adoption over time and discover the different implementations of this privacy-preserving tool carried out by the resolvers.

7.2.1 Detection of QMIN resolvers

They set up a mechanism to detect which resolvers were applying QMIN on their received requests through the following method. A QMIN-enabled resolver will gradually add labels to the query name and will follow the first NS record found between the truncated initial query name and the full one. If one of the labels before the last one contains a delegation to another nameserver with a specific response, QMIN resolvers would follow it and return this response, whereas non-QMIN resolvers would directly query the full QNAME which will return another distinct pre-defined response. The responses here were TXT records containing either "qmin enabled" or "qmin not enabled".

They set the TTL of the delegation records to 10 seconds and scheduled the experiments to launch the queries and to collect those responses every hour. And from April 2017 to October 2018, there has been a 8.8 % rise in the adoption of QMIN, which is not a tremendous increase. A spike was however noticed in April 2018 and the main possible reason for this is the adoption of the Cloudflare resolver 1.1.1.1, announced as the fastest, privacy DNS recursive resolver [31], which implements QMIN.
They noted that some resolvers actually use a QMIN-enabled forwarding resolver. This introduces another resolver standing between the original resolver queried by the client and the other nameserver that are authoritative for the asked domains. This other resolver receives the complete QNAME from the original one and processes the query using QMIN, effectively preventing the nameservers from learning the whole QNAME. This behaviour was noticed in most of the QMIN-supporting open resolvers, which forwarded their queries to larger DNS resolvers. Therefore, the adoption of QMIN should start with those larger public DNS providers to enhance user privacy at large scale.

As the RFC algorithm for QMIN [16] was a simple suggestion, several different implementations emerged amongst the resolvers. De Vries et al.[19] managed to fingerprint those types of QMIN implementations by assigning them specific

signatures whose goal was to map the behaviour of the query along the whole resolution process. This allowed them to find some advantageous behaviours in resolvers when encountering QMIN. Those are part of the recommendations De Vries et al.[19] mentioned in their conclusion and repeated in the next subsection.

7.2.2 Conclusion

They concluded that QMIN was helpful to provide more privacy to the user, but that it can also lengthen the number of queries needed for the resolution of a name. This results in a penalty performance of up to 26% (with BIND 9.13.3). Furthermore, they mentioned that operators of open resolvers must be careful when incorporating this feature to avoid increasing vulnerabilities. The authors of this paper recommend the following points: QMIN should be implemented in all resolvers to improve privacy in spite of its performance decrease on the resolvers; resolvers should fallback on the conventional name resolution upon errors ("relaxed" mode); resolvers should stop QMIN after a configurable number of labels; replace NS and other QTYPE with A queries; heuristics should take care of cases where privacy is not a concern and thus reduce QMIN negative amplification impacts.

Note that some of those recommendations already belong to the specification of QMIN in RFC 9156[16].

7.3 Zone Cuts in QMIN and NXDOMAIN Optimization

Zhen Wang investigated the usage and impacts of zone cuts leading to NXDOMAIN in his paper "Understanding the Performance and Challenges of DNS Query Name Minimization" [35]. He measured the performance of QMIN over real domains on the Internet, while also evaluating its combination with an NXDOMAIN optimization proposed by Vixie et al. [34]. Broken empty non-terminals (ENT) were found to be the reason why a transition to Query Name Minimisation coupled with NXDOMAIN optimization is difficult.

Wang retrieved the query logs of a one week period of a recursive resolver at the NIST campus and examined those in details. He reported 9.56 as the average number of labels of queries captured, which could correspond to the recommended value of QMIN parameter MAX_MINIMISE_COUNT. Moreover, his measurements showed that a few long query names resulted in a large amount of queries. If those queries are repeated (and most were in this experiment as he clearly marked the unique queries), an aggressive cache policy could considerably reduce the performance penalty of QMIN.

He also reported the distribution of number of zone cuts in the captured queries, which is dominated by two zone cuts per query. The number of zone cuts largely

influences the number of queries that the resolver sends due to QMIN, as well as the zone cuts cached in the resolver.
If the resolver cache contains several zone cuts of the current query, it may optimize the resolution by "jumping" to the closest zone cut and start sending queries from this starting point. However, the remaining labels still need to be gradually queried by the probing process of the resolver.

The maximum potential increase of queries sent with QMIN, in comparison to the version without QMIN, is the number of non-zone-cut-points labels in the QNAME. Remember that, in-between two zone cuts, the resolver will send probing queries to the nameserver label by label or more, until it has reached a zone cut. Thus, the number of queries in this zone is directly dependent on the number of labels between the two zones, or the *non-zone-cut-point labels*. For unique query names, the average of maximum query increase is 1.28 whereas for all query names it ascends to 6.44, meaning that the repeated queries for some long query names engender largely this QMIN-performance penalty.

7.3.1 NXDOMAIN Optimization

There exists another optimization which truncates the trailing labels. It makes use of ENT or *empty non-terminal* [12] : this is a name within a domain that has no valid records of its own but has at least one descendant name that does.
For instance, a nameserver can be authoritative for the domain "example.com". The subdomain "lab2.lab1.example.com" must be added within the zone and there are several ways to do that. If the record for "lab2.lab1.example" is simply added to the database, this leads the name "lab1.example.com" to become an empty non-terminal node : it does not have valid records but its descendant "lab2.lab1.example.com" does. It is therefore empty, and non-terminal, since it is has a descendant.

For this type of node, the response will contain a RCODE with NODATA, informing that the name is valid, but no RRsets will be present since there are no data for this name. In case of an invalid name, the RCODE would be NXDOMAIN, which shows the clear distinction with valid names. The resolver is then able to stop the resolution process if it encounters a NXDOMAIN code, as it knows that the currently queried QNAME is invalid.

This NXDOMAIN optimization enables the resolver to stop the probing granted by QMIN when it runs into the first NXDOMAIN. Two cases were detailed in the paper [35] : the first with the closest zone cut cached by the result, and the second with no caching. The former can abort the resolution by processing the limited labels under the last zone cut, while the latter has the ability to avoid a potential larger number of queries by querying labels starting from the root zone.

Therefore, NXDOMAIN optimization is crucial to avoid that DDoS attacks using querie for invalid randomly generated deep names are amplified by abusing

QMIN. If this optimization technique is used, the resolver would probably send minimised queries and gradually add labels until sending the whole QNAME resulting in a NXDOMAIN rcode.

The paper showed that NXDOMAIN optimization makes the QMIN overheads similar to the conventional name resolution as the name resolution stops at the first NXDOMAIN label of the QNAME. However, the supposition is that nameservers handle correctly ENT nodes by sending NODATA RCODE. Some nameservers may send NXDOMAIN responses, triggering the stop of the resolution and wrongly indicating that the name has no valid descendant name. This behaviour was discovered in the paper and named "ENT brokenness". It concerns around 10& of the unique queries they captured, which shows the amount of the ENT brokenness.

The paper concludes that the privacy gains with QMIN is counter-balanced by the performance decrease in name resolution due to additional queries. And they recommended that QMIN-enabled implementation should at least adopt NXDOMAIN optimization and must be robust enough against the ENT brokenness of the nameservers.

8 Conclusion

8.1 Summary

Since the Domain Name System was implemented in 1980's and as its core specification were not studied extensively at the time, DNS had had and still has a lot of flaws, and those are fixed progressively by discovering new attacks and engraving their countermeasures in updated RFCs.

Each open resolver has a different behaviour regarding the attacks, some do not enable certain features and some abruptly stop when encountering some queries, such as a delegation to a subdomain that the nameserver is not authoritative of. But all resolvers follow generally (or at least loosely) the DNS core specifications. Meaning that, as DNS is vulnerable to countless of attacks, resolvers have to set some limits to remove or reduce the possibility of attacks to occur.

Some attacks that were discovered in the previous years (NSNX, TsuNAME, iDNS) are conceptually quite simple, and I am surprised that they were not discovered earlier because of their simplicity. Even the initial algorithms did not take those attacks into account.

Most of the attacks discussed in this thesis exploit the fact that chained records can be followed up to some depth, and, if indefinite, can create loops, which is the most essential way to exhaust server resources. Furthermore, launching the same attack in parallel with NS referrals may increase the final amplification factor.
As observed, some resolvers have implemented mitigation measures against those. However, the chained style attack can still be leveraged by combining them with other attack vectors to result in powerful amplification vectors.

As demonstrated, an impressive and interesting feature to take advantage of is QMIN and its reset through rewriting. This thesis shows its flaws and its potential for greatly increasing the amplification factor of Denial-of-Service attacks against DNS. It also proves that several open resolvers are vulnerable to these QMIN-enhanced attacks.

More studies are needed to properly encompass and construct mitigation measures against QMIN amplification and QMIN-enhanced attacks. Some future work are described in the next section.

8.2 Future Work

8.2.1 Wildcard

An interesting aspect that has not been thoroughly analysed in this project is the wildcard DNS record, defined in RFC 1034 [27]. This special record allows the nameserver authoritative for a domain to match requests of non-existent subdomains (NXDOMAIN) and reroute to the target of that record. For instance, a wildcard record in the zone *target.net* could be written as :

```
1 *.target.net.        IN      MX      1.2.3.4
```

Listing 8.1: Wilcard record

The symbol "⋆" represents all the other subdomains of *target.net*. Thus, any request for non-existent subdomains will be redirected to the address *1.2.3.4*. An attacker who knows this record belongs in the database of the nameserver could abuse this feature by sending a large number of queries to non-existent subdomains that will trigger a reflection attack from the nameserver to the address pointed by the resource records.

Another attack involving QMIN could also have a powerful impact : the attacker client can create multiple distinct queries with the maximum number of labels for a non-existent subdomain of "target.net". This will force the resolver to use QMIN for each query.

Due to a previously vague description of the behaviour of a wildcard record, its use was confusing. But it was finally clarified after 20 years in RFC 4592 [25].

A study of the updated algorithm could show whether QMIN-enhanced attacks can be improved by the insertion of this record in victim zones.

8.2.2 A More Extensive Study of Open Resolvers

In our Internet-wide measurements, we chose an arbitrary number of 20 open resolver to test our QMIN-attacks on. This gives a somewhat imprecise description of the reaction of open resolver over the Internet, given that there are more than 3 millions open resolvers according to a recent study[29]. Thus, increasing the number of tested resolvers would show the currents trends and implemented limits of DNS resolvers over the Internet.

Moreover, only a brief study of the attacks against the list of open resolver in Chapter 5 was carried out in this project. Researching more in depth the exact behaviours of the resolvers may allow to find out and exploit more vulnerabilities. For instance, looking into the time needed for a resolver to process and answer a query could lead to an attack vector leveraging timeout and time used for consumption of this particular resolver.

8.2.3 A Focus on Other Amplification Aspects And Mitigations

In this project, we decided to focus on the number of messages sent or received by the nameservers during the resolution process. Counting the exchanged messages is the most high-level abstraction of defining the amplification factor. If we go down the application layer, we could investigate the time required by the nameservers to process the requests or the size of the messages. Those two elements could be influenced by the packet size or by custom message content that trigger a large processing time or response. Long DNS names could also be exploited to increase the size of the messages. This study would not be limited by QMIN-only attacks but would concern all attacks. It would be a subpart of the larger project mentioned in the next subsection.

8.2.4 Taxonomy of DNS Amplification

As the preceding subsections point to, there is a need to provide concrete and complete studies about the vulnerabilities of the DNS features. And that is why this thesis is part of a more global ongoing project, which has the goal to systematize attack vectors that could be exploited against the Domain Name System and provide knowledge about the different approaches that attackers could follow. By determining the way the attacks are created, the scientific community will also have all the cards in their hands to be able to find countermeasures to those.

On the one hand, this research looks for the most primitive attack vectors that are enabled due to the core DNS specifications. By closing the gap on the most simple yet effective attack vectors, it will set and define the attack vector space by concentrating our focus on those fundamental axes. On the other hand, it will allow to create all the possible combinations of the attack vectors and ensure that no attack vector is missing.

Thus, this will provide an extensive way to learn about, discover and combine multiple attack vectors. Moreover, this ongoing project will contribute to the design of new countermeasures tailored to each attack vectors, and not to a specific attack. Therefore, while slightly mentioned in this thesis, more fundamental mechanisms alleviating the amplification factor of QMIN-enhanced attacks will be constructed and tested in this next work.

Bibliography

[1] dig(1) - linux man page. https://linux.die.net/man/1/dig. Accessed: 2022-12-30.

[2] Docker. https://docs.docker.com/get-docker/. Docker version 20.10.21, build baeda1f, Accessed: 2023-01-10.

[3] RIPE Atlas. https://atlas.ripe.net/about. Accessed: 2023-01-21.

[4] What is the mirai botnet? https://www.cloudflare.com/learning/ddos/glossary/mirai-botnet/. Accessed: 2023-01-08.

[5] Cyclehunter. https://github.com/SIDN/CycleHunter, 2013. Accessed: 2023-01-10.

[6] Unbound 1.10.0 released. https://www.nlnetlabs.nl/news/2020/Feb/20/unbound-1.10.0-released/, February 2020. [Online; posted 20-February-2020].

[7] BIND 9.18.0 - A New Stable Branch. https://www.isc.org/blogs/bind918/, January 2022. [Online; posted 26-January-2022].

[8] PowerDNS Authoritative Server 4.5.5, 4.6.4 and 4.7.3 Released. https://blog.powerdns.com/2022/12/09/powerdns-authoritative-server-4-5-5-4-6-4-and-4-7-3-released/, December 2022. [Online; posted 09-December-2022].

[9] Unbound 1.16.0 released. https://www.nlnetlabs.nl/news/2022/Jun/02/unbound-1.16.0-released/, June 2022. [Online; posted 02-June-2022].

[10] Donald E. Eastlake 3rd. Domain Name System (DNS) IANA Considerations. RFC 6195, March 2011.

[11] Yehuda Afek, Anat Bremler-Barr, and Lior Shafir. Nxnsattack: Recursive DNS inefficiencies and vulnerabilities. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 631–648. USENIX Association, 2020.

[12] Cathy Almond. What is an Empty Non-Terminal? https://kb.isc.org/docs/what-is-an-empty-non-terminal. Accessed: 2023-01-20.

[13] Manos Antonakakis, Tim April, Michael Bailey, Matthew Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *Proceedings of the 26th USENIX Conference on Security Symposium*, SEC'17, page 1093–1110, USA, 2017. USENIX Association.

[14] Stéphane Bortzmeyer. DNS Privacy Considerations. RFC 7626, August 2015.

[15] Stéphane Bortzmeyer. DNS Query Name Minimisation to Improve Privacy. RFC 7816, March 2016.

[16] Stéphane Bortzmeyer, Ralph Dolmans, and Paul E. Hoffman. DNS Query Name Minimisation to Improve Privacy. RFC 9156, November 2021.

[17] Jonas Bushart and Christian Rossow. DNS unchained: Amplified application-layer dos attacks against DNS authoritatives. In Michael Bailey, Thorsten Holz, Manolis Stamatogiannakis, and Sotiris Ioannidis, editors, *Research in Attacks, Intrusions, and Defenses - 21st International Symposium, RAID 2018, Heraklion, Crete, Greece, September 10-12, 2018, Proceedings*, volume 11050 of *Lecture Notes in Computer Science*, pages 139–160. Springer, 2018.

[18] Alissa Cooper, Hannes Tschofenig, Dr. Bernard D. Aboba, Jon Peterson, John Morris, Marit Hansen, and Rhys Smith. Privacy Considerations for Internet Protocols. RFC 6973, July 2013.

[19] Wouter B. de Vries, Quirin Scheitle, Moritz Müller, Willem Toorop, Ralph Dolmans, and Roland van Rijswijk-Deij. A first look at QNAME minimization in the domain name system. In David R. Choffnes and Marinho P. Barcellos, editors, *Passive and Active Measurement - 20th International Conference, PAM 2019, Puerto Varas, Chile, March 27-29, 2019, Proceedings*, volume 11419 of *Lecture Notes in Computer Science*, pages 147–160. Springer, 2019.

[20] Tim Fisher. The Best Free and Public DNS Servers (December 2022). `https://www.lifewire.com/free-and-public-dns-servers-2626062`, 2022. Accessed: 2022-12-30.

[21] Wes Hardaker. Analyzing and mitigating privacy with the DNS root service. In *Proceedings of the ISOC NDSS Workshop on DNS Privacy*, San Diego, California, USA, February 2018. The Internet Society.

[22] Lukas Heimes. A Formal Framework for End-to-End DNS Resolution. Master Thesis at ETHZ.

[23] Aqsa Kashaf, Vyas Sekar, and Yuvraj Agarwal. Analyzing third party service dependencies in modern web services: Have we learned from the mirai-dyn

incident? In *Proceedings of the ACM Internet Measurement Conference*, IMC '20, page 634–647, New York, NY, USA, 2020. Association for Computing Machinery.

[24] Warren "Ace" Kumari and Paul E. Hoffman. Decreasing Access Time to Root Servers by Running One on Loopback. RFC 7706, November 2015.

[25] Edward P. Lewis. The Role of Wildcards in the Domain Name System. RFC 4592, July 2006.

[26] Florian Maury. The "Indefinitely" Delegating Name Servers (iDNS) Attack. `https://indico.dns-oarc.net/event/21/contributions/301/attachments/272/492/slides.pdf`, 2015. Accessed: 2022-11-03.

[27] Paul Mockapetris. Domain names - concepts and facilities. RFC 1034, November 1987.

[28] Giovane C. M. Moura, Sebastian Castro, John Heidemann, and Wes Hardaker. Tsuname: Exploiting misconfiguration and vulnerability to ddos dns. In *Proceedings of the 21st ACM Internet Measurement Conference*, IMC '21, page 398–418, New York, NY, USA, 2021. Association for Computing Machinery.

[29] Jeman Park, Rhongho Jang, Manar Mohaisen, and David Mohaisen. A large-scale behavioral analysis of the open dns resolvers on the internet. *IEEE/ACM Transactions on Networking*, 30(1):76–89, 2022.

[30] Raymond Pompon. The DNS Attacks We're Still Seeing. `https://www.f5.com/labs/articles/threat-intelligence/the-dns-attacks-we-re-still-seeing`. Accessed: 2023-01-08.

[31] Matthew Prince. Announcing 1.1.1.1: the fastest, privacy-first consumer DNS service. `https://blog.cloudflare.com/announcing-1111/`. Accessed: 2023-01-21, posted on 01.04.2018.

[32] Scott Rose and Wouter Wijngaards. DNAME Redirection in the DNS. RFC 6672, June 2012.

[33] Jodok Vieli. A Taxonomy of DoS attacks. Master Thesis at ETHZ (in progress).

[34] Paul A. Vixie, Rodney Joffe, and Frederico Neves. Improvements to DNS Resolvers for Resiliency, Robustness, and Responsiveness. Internet-Draft draft-vixie-dnsext-resimprove-00, Internet Engineering Task Force, June 2010. Work in Progress.

[35] Zheng Wang. Understanding the performance and challenges of dns query name minimization. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 1115–1120, 2018.

A QMIN Algorithm

Listing A.1: Per RFC 9156, 3. Algorithm to Perform QNAME Minimisation

```
(0)
    If the query can be answered from the cache, do so; otherwise, iterate as follows:
(1)

    Get the closest delegation point that can be used for the original QNAME from the
        cache.

(1a)
    For queries with a QTYPE for which the authority only lies at the parent side (like
        QTYPE=DS), this is the NS RRset with the owner matching the most labels with
        QNAME stripped by one label. QNAME will be a subdomain of (but not equal to)
        this NS RRset. Call this ANCESTOR.

(1b)
    For queries with other original QTYPEs, this is the NS RRset with the owner matching
         the most labels with QNAME. QNAME will be equal to or a subdomain of this NS
        RRset. Call this ANCESTOR.

(2)
    Initialise CHILD to the same as ANCESTOR.
(3)
    If CHILD is the same as QNAME, or if CHILD is one label shorter than QNAME and the
        original QTYPE can only be at the parent side (like QTYPE=DS), resolve the
        original query as normal, starting from ANCESTOR's name servers. Start over from
         step 0 if new names need to be resolved as a result of this answer, for example
        , when the answer contains a CNAME or DNAME [RFC6672] record.
(4)
    Otherwise, update the value of CHILD by adding the next relevant label or labels
        from QNAME to the start of CHILD. The number of labels to add is discussed in
        Section 2.3.
(5)
    Look for a cache entry for the RRset at CHILD with the original QTYPE. If the cached
         response code is NXDOMAIN and the resolver has support for [RFC8020], the
        NXDOMAIN can be used in response to the original query, and stop. If the cached
        response code is NOERROR (including NODATA), go back to step 3. If the cached
        response code is NXDOMAIN and the resolver does not support [RFC8020], go back
        to step 3.
(6)

    Query for CHILD with the selected QTYPE using one of ANCESTOR's name servers. The
        response can be:

(6a)
    A referral. Cache the NS RRset from the authority section, and go back to step 1.

(6b)
    A DNAME response. Proceed as if a DNAME is received for the original query. Start
        over from step 0 to resolve the new name based on the DNAME target.
(6c)
    All other NOERROR answers (including NODATA). Cache this answer. Regardless of the
        answered RRset type, including CNAMEs, continue with the algorithm from step 3
        by building the original QNAME.
(6d)
```

```
33      An NXDOMAIN response. If the resolver supports [RFC8020], return an NXDOMAIN
            response to the original query, and stop. If the resolver does not support [
            RFC8020], go to step 3.
34  (6e)
35      A timeout or response with another RCODE. The implementation may choose to retry
            step 6 with a different ANCESTOR name server.
```

B Attack Triggers

```
1     1st attack : parallel referrals + QMIN
2       Trigger : dig @[resolver] parallel-referrals.qmin-tax.net
3
4     2nd attack : referral chain without QMIN
5       Trigger : dig @[resolver] a1.referral-chain.qmin-tax.net
6
7     2nd attack : referral chain with QMIN
8       Trigger : dig @[resolver] lab12.lab11.lab10.lab9.lab8.lab7.lab6.lab5.lab4.lab3.
            lab2.lab1.a1.referral-chain-qmin.qmin-tax.net
9
10    2nd attack : referral chain with QMIN, second version, loop IPs
11      Trigger : dig @[resolver] www.sub.referral-chain-ip-loop.qmin-tax.net
12
13
14    3rd attack : unchaind cname + qmin
15      Trigger : dig @[resolver] fake8.fake7.fake6.fake5.fake4.fake3.fake2.fake1.fake0.a0
            .unchained-cname-qmin.qmin-tax.org.
16
17    3rd attack : unchaind dname + qmin
18      Trigger : dig @[resolver] sub.fake8.fake7.fake6.fake5.fake4.fake3.fake2.fake1.
            fake0.a0.unchained-dname-qmin.qmin-tax.org.
19
20    4th attack : Scrubbing + CNAME + QMIN
21      Trigger : dig @[resolver] fake8.fake7.fake6.fake5.fake4.fake3.fake2.fake1.fake0.a0
            .scrubbing-cname-qmin.qmin-tax.net.
22
23    4th attack : Scrubbing + dNAME + QMIN
24      Trigger : dig @[resolver] sub.fake8.fake7.fake6.fake5.fake4.fake3.fake2.fake1.
            fake0.a0.scrubbing-dname-qmin.qmin-tax.net.
25
26    5th attack : Paralell NS + Referral Chain + QMIN
27      Trigger : dig @[resolver] parallel-referral-chain.qmin-tax.org.
28
29    6th attack : Paralell NS + SCRUBBING  + CNAME + QMIN
30      Trigger : dig @[resolver] parallel-scrubbing-cname-qmin.qmin-tax.org.
31
32    6th attack : Paralell NS + SCRUBBING  + DNAME + QMIN
33      Trigger : dig @[resolver] parallel-scrubbing-dname-qmin.qmin-tax.org.
34
35     % Our 7th and 8th attacks are Parallel NS + Scrubbing + QMIN and Referral Chain +
           QMIN, both enhanced with voluntarily slowing the responses of the controlled
           nameservers. Due to the lack of time to set them up, those attacks are not
           examined here
36
37    9th attack : NS trees (fan_out * fan_out)
38      Trigger : dig @[resolver] ns-trees.qmin-tax.org.
```

C Configuration nameservers

```
// Write this to the named.conf.options :
      options {
        directory "/var/cache/bind";
        querylog yes;
        recursion no;
        //dnstap { all; };
        //dnstap-output unix "/etc/logs/dnstap.sock";
        //dnstap-identity "tiggr";
        //dnstap-version "bind-9.18.4";
        allow-query { any; };
      };

      logging {
        channel query {
          file "/etc/logs/query.log" versions 3 size 100M;
          print-time yes;
          print-severity yes;
          severity debug 3;
        };
        channel general {
          file "/etc/logs/general.log" versions 3 size 100M;
          print-time yes;
          print-category yes;
          print-severity yes;
          severity debug 3;
        };

        category queries { query; };
        category cname { general; };
        category query-errors { general; };
      };


\\ Add the zones to the nameserver in named.conf.local
        zone "qmin-tax.net." {
            type master;
            file "/etc/bind/zones/zones-qmin-tax.net";
        };
```

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Understanding QMIN-enabled DNS Amplification

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Bearzi	Marco

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Neuchâtel, 13th of January, 2023	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.