Fragmentation Considered Poisonous

Amir Herzberg and Haya Shulman
Department of Computer Science
Bar Ilan University
Ramat Gan, Israel
Email: {amir.herzberg, haya.shulman}@gmail.com

***Abstract*—We present effective off-path DNS cache poisoning attacks, circumventing all widely-used defenses against poisoning, based on echoing of random challenges from request to response, e.g., port randomisation and query randomisation (0x20). The attacks mainly depend on the use of UDP to retrieve long DNS responses, resulting in packet fragmentation. We show how attackers are often able to cause such fragmented responses, and then abuse them to inject fake, 'poisonous' records, into legitimate DNS responses.**

Proper adoption of DNSSEC would prevent our poisoning attacks; however, we show that the (common) partial and permissive deployments of DNSSEC, actually *facilitate* our attacks. We recommend appropriate steps for DNS deployments to prevent increased vulnerability to off-path poisoning attacks.

We also introduce a new type of DNS attack: *name server (NS) pinning*, forcing resolvers to use name servers of attacker's choice. NS pinning can result from poisoning attacks and in other ways, including by abusing fragmented DNS responses; such attacks can work *even against DNSSEC*. NS pinning can be stepping-stone to other attacks: denial of service, traffic analysis, covert channels - and even, in some scenarios, DNS cache poisoning.

We validated our attacks against popular resolvers (Bind and Unbound), and real DNS name servers on the Internet.

I. Introduction

The correctness and availability of information in the Domain Name System are crucial for the operation of the Internet. DNS poisoning is a significant threat to Internet security; in particular, it may allow weak attackers, without Man-in-the-Middle (MitM) or eavesdropping capabilities, to intercept and modify content, by providing fake destination IP address, thereby circumventing many defense mechanisms such as Same Origin Policy (SOP), domain blacklists and domain-policies (e.g., SPF). Poisoning allows attacks such as phishing, credentials-theft (e.g., XSS), and more.

Defenses against DNS poisoning, can be categorised into two classes: *challenge-response defenses*, which rely on some 'unpredictable challenge' values in requests, which must be echoed in (legitimate) responses; and *cryptographic defenses*, which rely on cryptographic authentication (signature) in responses, most notably the DNSSEC standard [1]–[3]. DNSSEC uses digital signatures to prevent poisoning of the DNS responses, and is hence secure even against man-in-the-middle (MitM) adversaries. DNSSEC had a long and thorough design and evaluation process, and its security is based on extensive evaluation by many experts, as well as on results of formal analysis, e.g., by Bau and Mitchell [4]. (There are also other proposals for cryptographic defenses, designed to protect against MitM, e.g., DNS Curve [5]).

Cryptography is necessary to protect against a MitM attacker, who can easily copy any challenge into a fake response; however, in spite of the awareness that DNS poisoning can allow severe attacks, so far, DNSSEC deployment is progressing slowly. One reason may be that, until Kaminsky's well-known off-path poisoning attack in 2008 [6], off-path poisoning appeared to be a rather impractical and inefficient attack; and MitM attackers are considered rare. Indeed, the efforts to deploy DNSSEC have increased following Kaminsky's attack, although, so far, adoption is still very limited and DNSSEC validation is rarely effective against poisoning; details within.

Following to the publication of Kaminsky's attack, the security of DNS resolvers was enhanced, and most resolvers were 'patched' to defend against it, initially using port and IP randomisation and 'birthday protection'. Additional improvements were soon published and adopted, in particular *query randomisation* by case toggling ('0x20 encoding', [7]) and addition of a *random prefix* to queries [8]; see also RFC 5452 [9]. As a result, there is a somewhat reduced momentum to deploy DNSSEC.

We show that, under common scenarios, *even an off-path attacker can efficiently circumvent all of these challenge-response mechanisms*. Namely, an off-path attacker can perform effective DNS poisoning, circumventing all the 'patches', and with even a better efficiency than Kaminsky's poisoning technique. While we outline different 'patches' that may defend against our attack, the best solution is obviously to deploy DNSSEC, providing security (even) against a MitM attacker; we hope that our results will, indeed, help speed up deployment of DNSSEC, thereby protecting against attacks by off-path attackers (as in this paper) as well as by MitM attackers.

Note that there *has* been a considerable progress with adoption of DNSSEC, esp. since Kaminsky's attack (2008); in particular, many top-level domains (TLDs) as well as the root already support DNSSEC (i.e., have keys and provide signatures), and most popular DNS resolvers support

DNSSEC validation. However, only a small fraction[1] (about 2%) of the domains are signed, and there are many interoperabity problems and concerns. Indeed, for most of the signed domains (82%), there is *no chain of trust* from the root zone, i.e., the key of the child zone is not signed by its parent zone. In such cases, signatures cannot be validated by DNSSEC-compliant resolvers, unless additional 'trust anchor' keys are installed in the resolver.

As a result, only a minority of the resolvers currently apply *strict validation* of DNSSEC. Based on the measurements reported in [10], about third of resolvers do not apply DNSSEC at all, and only about 1% of them are validating. Apparently, the vast majority of resolvers operate in a *permissive mode*, where they signal support of DNSSEC, but do not reject responses which are not signed properly. Significant additional effort is required to make DNSSEC fully functional, for most of the domain names and resolvers.

Somewhat ironically, in fact, we show that the current situation of partial and permissive deployment of DNSSEC, can actually *facilitate our off-path attacks*. Specifically, our attacks require fragmentation of DNS responses; and the use of DNSSEC results in much longer responses, which are more likely to be fragmented. Note, however, that when properly and fully deployed, DNSSEC will prevent the poisoning attacks, in spite of the fragmentation; also, note that fragmentation is possible even without DNSSEC, and indeed, attackers can induce the required fragmentation (with or without DNSSEC), as we explain within. Still, the partial (or imperfect/permissive) deployment of DNSSEC, makes the attacks easier to deploy. This is undesirable; the introduction of a new security mechanism (DNSSEC), should not result in increased vulnerability and new attack vectors, even during the (often long) adoption period; notice that is stated explicitly as a goal for DNSSEC in RFC 3833 [44].

On the other hand, we identify two attacks, which are possible *even when DNSSEC is fully, and correctly deployed*. Like the DNS poisoning attack, the attacks may be facilitated due to the use of longer, fragmented DNS responses with DNSSEC, although there may be other ways for adversaries to cause the desired fragmentation. These two attacks are *subdomain injection* and *NS pining*.

Subdomain Injection is a special form of poisoning attack, where resolvers accept, cache and provide to clients a mapping for a *non-existing (injected) child domain*, of an existing, potentially DNSSEC-protected, parent domain. Subdomain injection is possible, when a domain uses the *NSEC3 opt-out* option, i.e., refrains from signing separately each of its unsigned subdomains. This was recognised as a potential weakness by Bau and Mitchell [4]; however, there are still many zones using NSEC3 opt-out, for improved efficiency. We show that this attack can be done, even by an off-path attacker, and does not require a MitM attacker (as considered by [4]). The injected subdomains allow attacks on the Same Origin Policy, such as XSS, phishing and cookie stealing.

[1]We computed the statistics for the top 300,000 most popular domains as listed by Alexa.

Zone Resolver mode	DNSSEC Vulnerabilities		
	No chain of trust	NSEC3 opt-out	None
Permissive	*Domain Hijacking*	*Domain Hijacking*	*Domain Hijacking*
Strict	*Domain Hijacking*	*Subdomain Injection*	*NS Pinning*

Table I
DNSSEC DEPLOYMENT VULNERABILITIES THAT ALLOW OUR CACHE POISONING AND NS PINNING ATTACKS (SECTIONS IV AND III RESPECTIVELY).

In contrast, *Name Server (NS) Pinning* is a new type of attack, where the attacker tampers with the selection of the DNS servers by the resolver for a particular zone, e.g., forcing requests to be sent to a particular (vulnerable or fake) name server. We show that NS pinning can be stepping-stone to facilitate or improve efficiency of several significant attacks: denial of service, traffic analysis, covert channels, and in some scenarios, even DNS cache poisoning. In particular, NS pining can *facilitate DNS poisoning*, by causing resolvers to use specific name servers, possibly known to be vulnerable; this provides an effective mechanism to deploy the attack of [11], which, so far, was considered impractical. NS pinning attacks can be either *blocking*, i.e., only result in a resolver avoiding a particular name server, or *forcing*, i.e., forcing the resolver to use a particular name server, typically pointing to name servers belonging to the attacker. Blocking is often very easy; indeed, RFC4697 [12] recommends that resolvers do not contact a name server after few successive failures, which makes blocking easy - and many resolvers support this and are hence vulnerable, see [13] (we also confirmed this for some resolvers, e.g., Unbound). It would be desirable to extend the security evaluation of DNSSEC of [4], to cover also NS pinning.

Note that subdomain injection attacks, as well as forcing NS pinning attacks which map to a fake name server, are possible due to the fact that DNSSEC-enabled referral responses contain unsigned delegation NS and A resource records (RRs). Indeed, Bernstein argued already in 2009 [14] that DNSSEC is vulnerable since it does not sign the NS and A delegation records, albeit his argument focused on the fact that signing top-level domains provides no protection to (unsigned) second-level domains.

We summarise all our attacks, with their requirements (discussed above), in Table I. Notice that even the most severe (and demanding) attack of DNS poisoning, is usually possible, since it requires only that either the resolver supports a permissive mode (which currently holds for 99%

of the resolvers, as measured in [10]), *or* that the zone has no chain of trust, holding for 82% of the signed domains.

Fragmentation: Fragmentation is known to be problematic or 'harmful', mainly due to the negative impact on performance; see the seminal paper of Kent and Mogul [15]. As a result, fragmentation is usually avoided, e.g., by use of path MTU discovery [16], [17], mainly for connection-based transport protocol (TCP). However, DNS traffic is usually sent over UDP. Several significant name servers, e.g., com, edu, send long responses over TCP, however, this is a controversial strategy, since the use of TCP for DNS responses results in a significant overhead for the name server, resolver and the network, as well as in additional latency.

Our work builds upon previous disclosures of vulnerabilities due to the design or implementations of the fragmentation mechanism; we next mention few of the most relevant. Zalewski [18] suggested that it may be possible to spoof a non-first fragment of a (fragmented) TCP packet. However, using such non-first-fragment injections to TCP packets seems challenging. Furthermore, currently almost all TCP implementations use path MTU discovery [16], [17] and avoid fragmentation.

Several vulnerabilities related to IP fragmentation, and specifically to predictable fragment identifiers (IP-ID) values, are covered in [19], [20]; and predictable IP-ID values were shown [21] to allow interception and injection of fragments.

Attacker Capabilities: The required attacker capabilities include an arbitrary off-path, spoofing-only adversary, that controls a 'puppet', i.e., a sandboxed script, [22]; see Figure 1. The puppet can trigger DNS requests at the resolver, e.g., by planting images, using HTML IMG or IFRAME tags, which results in DNS requests to the victim domain; this phase initiates the attacks.

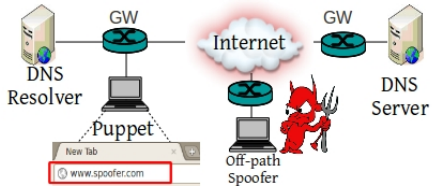

Figure 1. Simplified network topology and attacker capabilities. The spoofing attacker uses a puppet to invoke the query.

Contributions: We next summarise the main contributions of this manuscript.

Identification of efficient DNS cache poisoning attacks, motivating the deployment of DNSSEC. To ensure security DNSSEC requires adoption by both the zones and the resolvers, and hence, deployment has been - and remains - a significant challenge.

Name Server (NS) pinning - attack and vulnerabilities. We identify the threat of NS pinning, and show how it may help facilitate various attacks, in particular, to force the resolver to stop using a particular name server, and eventually, to query a name server of attacker's choice, e.g., a compromised name server. We also present several NS pinning mechanisms.

Study of IP-ID algorithms. We carried out a study of the IP-ID allocation algorithms supported by the name servers (athoritative for top level domains and those used for 10^6 domains according to Alexa) on the Internet. We also provide techniques, tailored for name servers, for predicting the IP-ID values. Our study can be useful for other purposes, e.g., when designing different mechanisms pertaining to DNS.

Organisation: In the next section, we explain how attackers can abuse the fragmentation reassembly mechanism, to modify DNS responses (replacing the second fragment); we present our study of the IP-ID allocation methods supported by name servers, and provide techiques for efficient IP-ID matching. Then, in Section III, we present the NS pinning attack. Next, in Section IV, we present our DNS cache poisoning attacks. We conclude and discuss defenses in Section V.

II. Altering DNS Responses via Fragment Reassembly Spoofing

In this work we show techniques allowing an *off-path* attacker to modify the DNS responses from name server to the resolver; in following section we use this to launch DNS cache poisoning and name server (NS) pinning attacks. We show how an off-path attacker can accomplish this by exploiting fragmented DNS responses. Specifically the attacker alters the fragment reassembly process by injecting spoofed *second* fragments that are reassembled with authentic *first* fragments, and result in a new modified packet. This technique requires (1) fragmented DNS responses, it also requires (2) matching several fields in the IP-header so that the spoofed fragment is reassembled with the authentic one and (3) it the process of matching the IP header fields may require many queries to the DNS resolver.

In Section II-A we show that often attackers can enforce fragmentation on DNS responses, and we propose different techniques to achieve that, when (otherwise) DNS responses are not sufficiently large and do not get fragmented. Then, in Section II-B we explain the details of fragment reassembly process and the prerequisites to successful fragment reassembly. In Section II-C we provide details related to circumventing the caching of the resolver, which allows issuing multiple queries to some victim domain.

A. Enforcing Fragmentation

Our attacks exploit fragmented DNS responses. Typically DNSSEC-enabled responses get fragmented. However, not all DNSSEC-enabled DNS responses get fragmented. We suggest techniques that can often enforce fragmentation on those responses. Furthermore, we also suggest a technique

which allows to enforce fragmentation on referral responses, for *plain* DNS responses, that support EDNS (but *without* DNSSEC). We next present and discuss the techniques which can assist the attacker in enforcing fragmentation on DNS responses: (1) DNS query size, (2) DNS query type, (3) malicious subdomain, (4) spoofed ICMP fragmentation needed.

1) DNS Query Size: The query field in a DNS packet is limited to 256 bytes, and can be composed of one or more labels, i.e., subdomains, such that each label is at most 63 bytes long. In addition one byte is required to encode the length of a label in a DNS request; if each label is 63 bytes, the query contains 4 labels. Assuming that the size of the domain name which the attacker wishes to poison is l bytes, the attacker can add another $256 - l = x$ bytes to the DNS response, by padding the query field with x bytes. This can be useful to many responses which are almost 1500 bytes long and thus do not get fragmented.

2) DNS Query Type: The attacker may be flexible in the query type that it can issue. A puppet, i.e., *script* (e.g., java script), can issue requests to A type RR. These requests often also trigger requests to NS RRs, DS RRs and to DNSKEY RRs. Each of the responses to those requests can get fragmented. Typically responses to NXDOMAIN ('non-existing' domain) and DNSKEY, get fragmented. Merely sending an email to a mail server, that supports anti-spam mechanisms, e.g., SPF, also triggers DNS requests for SPF and TXT records (in addition to the usual requests to A, NS and DNSKEY RRs). A zombie, i.e., user (as well as root) level malware, can issue requests for additional types, e.g., ANY RR. DNS responses to ANY RR request are almost always fragmented for DNSSEC-enabled DNS responses.

In this work we restrict ourselves to the weakest attacker, that controls a puppet on the victim network, and can send requests for A RR (which also trigger requests to other records, e.g., DNSKEY); all our attacks in this work are conducted using a puppet. As evident from Figure 2 a zombie can almost always inflict fragmentation, by triggering ANY type DNS requests.

See Figure 2, for a study of the DNS responses' sizes to A and ANY RR DNS requests to subdomains in gov TLD.

3) Malicious Sub-Domain: We suggest a technique which can expand the DNS *referral* responses so that they are fragmented. Let tld be the top level domain that the attacker wishes to spoof; the same idea applies to second (and lower) level domains. The attacker creates a new subdomain of tld, e.g., attacker.tld, and inserts a maximal number of delegation records into the parent domain, i.e., tld. The maximal number of delegation records depends on the registrar, e.g., *Network Solutions* (servicing many top level domains, e.g., *de*) as well as other registrars, e.g., of com, net, limit the number of NS, A and DS records, that the child can add to the parent zone, to 13 each; (this includes domains such as, com, net, de).

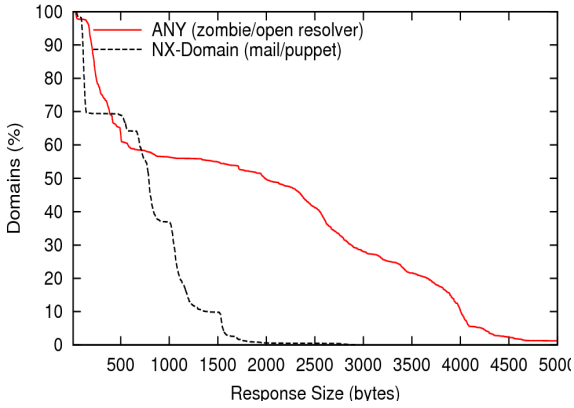

Figure 2. Length of ANY and NXDOMAIN responses of gov domains. Domains taken from [32]

We next provide calculations for the DNS response size, based on the standard and typical values. First, every DNS responses contains headers, and the size of the headers (in all TCP/IP layers) including DNS, is 54 bytes. Then the DNS payload contains four sections: (1) `question`, (2) `answer`, (3) `authority` and (4) `additional`; the additional section also contains the EDNS RR which is typically 11 bytes long.

The `question` section can be up to 260 bytes, and currently only a single question, per DNS packet, is supported. The `answer` section is not present in referral type responses. The `authority` section contains NS, DS (or NSEC/NSEC3) and RRSIG records; and the `additional` section contains A (IPv4) and AAAA (IPv6) records (and typically these delegation records are not signed therefore no RRSIG records will be present in `additional` section in referral responses).

In Table II we present calculations for different sizes of (DNSSEC-enabled as well as plain) DNS referral responses, ranging from average to maximal, i.e., (each) containing six and thirteen NS, A and AAAA records respectively. As can be seen from the table, the responses are above 1500 bytes and thus will get fragmented. Note that we assumed that only two DS records are used in a response, and one signature; more DS records (as well as more signatures) are common in referral DNS responses, e.g., some of gov domains use more than ten DS records. In this attack the domain name that the attacker creates is not required to be large (although a large domain name will further exacerbate the problem). The main factor that induces the expansion factor is the name of the NS records, i.e., name servers. Specifically, by adjusting the names of DNS servers the attacker can inflict fragmentation on the referral DNS responses (supporting EDNS). There is no limitation (beyond 256 bytes) that is recommended or enforced on the name field in resource records.

4) Spoofed ICMP Fragmentation Needed: In contrast to other techniques, which attempt to increase the DNS response size, this technique reduces the MTU to the tar-

Records	NS	DS	RRSIG	A	AAAA	DNS Payload (w/EDNS) Size (bytes)	IP Packet Size w/Headers (bytes)
RR size (bytes)	268	(20 - 30)	(160 - 260)	16	28		
Average response w/DNSSEC	$6 \cdot 268$	$(20 + 30)$	260	$6 \cdot 16$	$6 \cdot 28$	2193	2247
Maximal response w/DNSSEC	$13 \cdot 268$	$13 \cdot 30$	260	$13 \cdot 16$	$13 \cdot 28$	4717	4771
Average response size	$6 \cdot 268$	0	0	$6 \cdot 16$	$6 \cdot 28$	1883	1937
Maximal response size	$13 \cdot 268$	0	0	$13 \cdot 16$	$13 \cdot 28$	4067	4151

Table II
RESOURCE RECORDS AND DNS RESPONSE SIZES.

get domain. The attacker utilises the existing path MTU discovery procedure in a malicious way, by sending *ICMP destination unreachable fragmentation needed* packets to the name server with a spoofed source IP address. The idea of spoofing ICMP fragmentation needed packets is not new and was used as an attack vector for denial of service attacks, [33]–[35]. We suggest to apply ICMP fragmentation needed packets to enforce fragmentation. Note that such behaviour of host fragmentation is recommended, e.g., [17]. However, the ICMP fragmentation needed packets may be filtered by firewalls, therefore this technique will not always work.

We tested the PMTU reduction by sending spoofed fragmentation needed packets to a name server running on a linux OS, and successfully enforced fragmentation on DNS responses. We could force the name server to fragment responses to fragments of 500 bytes. This technique did not always work (well) when attempting to fragment to smaller sizes than 500 bytes.

B. Fragments Reassembly Conditions

In this section we review the basics of the fragment reassembly process. We then perform a study of the name servers IP-ID allocation methods, which is required for our attacks on DNS resolvers.

Let x be the payload in the original IP packet (containing a DNS response) sent by the name server, then after fragmentation, it is sent as two IP packets y_1, y_2, such that their respective lengths are smaller than x. On inputs $< y_1, y_2 >$, the defragmentation process, running at the resolver[2] reproduces x, by reassembling y_1 and y_2, and passes it to the transport layer.

To overwrite the second fragment, the attacker sends a fake second fragment y_2' so that it arrives at the defragmentation module *before* the authentic fragments y_1, y_2 of the DNS response[3]. The defragmentation mechanism at the IP layer caches y_2', in anticipation of the rest of the packet[4].

To ensure that the spoofed second fragment is matched with the authentic first fragment the attacker must predict certain fields in the IP header.

1) Matching IP Header Fields: According to [24], [25] the fragments of a datagram are associated with each other by four parameters in the IP header: (1) the transport layer protocol number, (2) the value in their IP-ID field, and (3-4) by the source/destination IP address pair. Thus both the first authentic fragment y_1 and the second spoofed fragment y_2' must have the same destination IP address (of the resolver that sent the query), the same source IP address (of the responding DNS server), the same protocol field (UDP) and the same fragment identifier (IP-ID). In addition, the spoofed second fragment should have the correct offset (as in the authentic second fragment); but the second spoofed fragment is *not* required to be of the same length as the authentic second fragment. The fragment reassembly process, applied to the pair $< y_1, y_2' >$, either returns a failure or a different packet $x' \neq x$. Matching most of these parameters is easy, as they can be anticipated by the attacker.

Maximal Transmission Unit: Path MTU changes infrequently, and can be found by the attacker easily, e.g., via trace-route; thus the attacker can compute the offset in the spoofed fragment correctly.

IP Addresses: In many scenarios, the resolver has a single, known IP address. Zones have 6 IP addresses on average and typically up to 13 IP addresses[5].

Internet Protocol Identifier (IP-ID): It remains to ensure a match between the value of the fragment identifier (IP-ID) field in the fake fragment y_2', and the IP-ID in the authentic fragment y_1 of the original response x. Specifically, the attacker has to craft a fragment containing the same IP-ID value as the one that was assigned by the DNS server to its DNS response to the resolver. The success probability of the attacker, as well as the strategy that the attacker applies, depend on the following factors:

1) the version of the IP protocol (IPv4 or IPv6).

[2]Alternatively, defragmentation may happen at an intermediate device such as a firewall or a network address translator (NAT); this does not impact the attack.

[3]Note that it is easy to adjust this technique to the (less common) case where fragments are sent in a reverse order: attacker removes the authentic second fragment y_2 from the reassembly buffer by sending an arbitrary y_1' (whose validation fields match those of the y_2), and the rest is the identical to the description above.

[4]By default, an unmatched fragment is kept in the cache for 30 seconds or so, hence, 'planting' such fake fragments is easy, e.g., the attacker can send it even before triggering the DNS request.

[5]The restriction to 13 IP addresses predates back to the pre-EDNS, [26], period, where the DNS responses had to fit in a 512 byte UDP datagram.

2) the size of the fragment reassembly buffer at the receiver (i.e., resolver or NAT/FW behind which the resolver is located).
3) the IP-ID assignment algorithm implemented by OS of the DNS server to allocate the IP-ID to outbound packets.

IP VERSION. In the most common scenario the communication is over IPv4, where the IP-ID field is 16 bits, for communication over IPv6, the IP-ID field in IPv6 is 32 bits long. In this work we focus on IPv4 since support of IPv6 by DNS servers is limited.

REASSEMBLY BUFFER SIZE. The host performing the defragmentation process, i.e., the resolver or firewall, allocates a reassembly cache for fragments per particular <source IP, destination IP, protocol> combination. The reassembly cache is typically limited to n fragments, and when more than n fragments for the same tuple arrive, the oldest fragments are evicted from cache and replaced with the new ones.

In linux, this limit n on the number of cached fragments per source, destination and protocol tuple is imposed by the *ipfrag_max_dist* parameter, and is 64 by default; see [27]. Legacy kernel versions of Linux OS allow buffering of up to few thousands of fragments; assuming a cache size of 256KB, and assuming that the attacker sends minimal size fragments, e.g., 21 bytes in IPv4 or 49 in IPv6, then the attacker can send up to $12,400$ fragments in IPv4 or 5300 in IPv6. In Windows the default value of n is 100 fragments. The limitation on the fragment reassembly buffer does not exist in IPv6.

IP-ID ASSIGNMENT METHODS. The efficiency of our attacks depends on the ability to predict the IP-ID value assigned by the name server to the DNS responses.

We carried out a study[6] of the IP-ID allocation methods implemented by the name servers authoritative for the top level domains (TLDs), i.e., a total of 271 TLDs[7]. The total number of DNS servers, authoritative for TLDs, is comprised of 1545 name servers, which constitutes 1139 distinct IP addresses (since some name servers, 407 to be precise, serve more than one domain). Figure 3 summarises our findings[8] related to the IP-ID allocation algorithms supported by servers authoritative for TLDs: 0.51% servers could not be reached, 8.99% of the name servers assign a constant *zero* IP-ID value[9] in the IP header of DNS responses; 56.63% of the servers assign per-destination incrementing IP-ID; 13.85% of the servers assign globally incrementing IP-ID, and 0.26% use some *other* allocation (under 'other' allocation we include (a seemingly) random IP-ID assignment); 19.74% support a 'mixed' IP-ID allocation. The 'mixed' allocation is an indication of name servers that support load balancing, [29]. When using DNS server-side load balancing, several physical machines are located behind the same IP address, and each physical machine may be implementing a different IP-ID allocation mechanism at the IP layer. We extend more on this in Sections II-B4 and II-B3.

MATCHING THE IP-ID. If there is no restriction on the size of the recipient's fragment reassembly cache, then the attacker can simply send fragments covering all the IP-ID range and one will always match. Assuming communication is over IPv4, i.e., the IP-ID field is 16 bits, the distribution of all IP-ID values is of size $2^{16} = 65536$; namely, in the worst case the attacker has to craft 65536 fragments, and to trigger a single query at the resolver, to almost certainly match the IP-ID in the DNS response. We perform all our attacks in a restricted setting where the caching DNS resolver is running on a Linux OS, and $n = 64$ fragments, i.e., the *ipfrag_max_dist* is initiated with the default value of 64; a better success probability can be achieved if such a restriction is not imposed.

We next report on our findings of IP-ID allocation methods by the name servers of TLDs. We discuss the four cases of IP-ID assignment: *random*, *per-destination*, *global* and *mixed*, suggest strategies which the attackers can employ to predict the IP-ID values assigned by each of the methods and analyse the efficiency and the success probability.

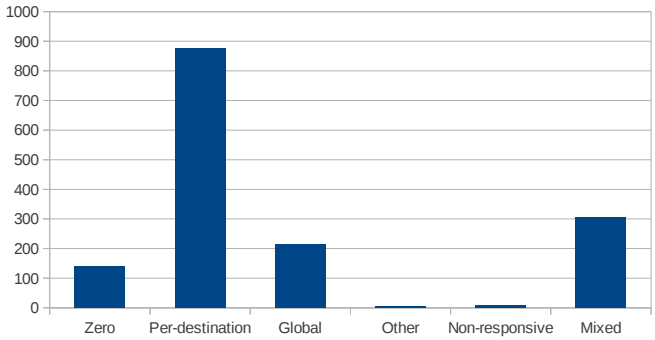

Figure 3. The IP-ID allocation methods among the DNS servers authoritative for (271) Top Level Domains (TLDs).

In Figure 4 we illustrate our results of IP-ID hitting rate for two domains: 404.gov implementing a per-destination IP-ID allocation and ssa.gov implementing a global IP-ID allocation (with average query rate of 100 per second). During the evaluation we sent a query to (each) name server every 5 milliseconds, and repeated the attacks 20 times.

2) Unpredictable IP-ID Allocation: Assuming that the attacker has no prior knowledge on the process according to which the IP-ID is incremented, and assuming that the IP-ID values in the authentic packet and in the spoofed fragment

[6]During this study we sent 50 requests to each name server with a 500 milliseconds delay between each request; we added the delay since some name servers would return *server failure* response when more than 10 packets were sent without any delay, and we found he 500 millisecond delay to be sufficient to receive the required responses.

[7]We also performed a similar study of IP-ID allocation algorithms according to 10^6 Alexa, [28], domains and obtained a similar distribution.

[8]It is important to emphasise that our study was conducted from within the network of our university, and findings may vary when performed from a different network, e.g., due to anycast routing supported by DNS servers.

[9]The zero IP-ID is assigned to DNS responses that do not get fragmented.

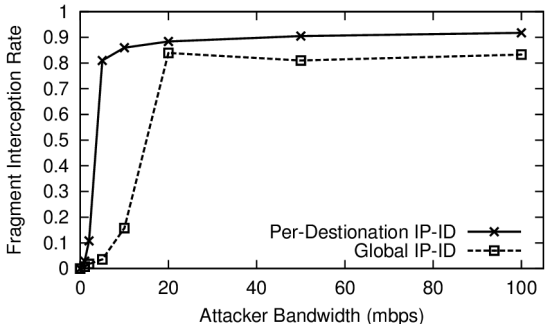

Figure 4. IP-ID hitting success rate for two domains each implementing a different IP-ID allocation method.

are selected independently, it suffices for the attacker to send 64 spoofed second fragments (assuming a restriction on the fragments reassembly buffer exists), to ensure success probability of roughly $1/1024$ of replacing the second fragment of a packet in a *single* attempt. After a 1024 attempts, i.e., repeated attacks, on average, the attacker can hit the correct IP-ID with a probability that is close to 1.

Under this category we classify four name servers, among the servers of TLDs, that support (what seems to be) an unpredictable IP-ID allocation.

However, most systems select the IP-ID sequentially. Of these, many use a single counter for all destinations (globally-sequential), as in Windows and by default in FreeBSD. Other systems, e.g., Linux, use *per-destination* IP-ID counters, where the first IP-ID to some destination is selected at random, and subsequent packets are assigned sequentially incrementing IP-ID values. In both of these cases, as we next show, the attacker can efficiently predict the IP-ID, achieving a significantly higher probability of success (compared to $1/1024$).

3) Per-Destination Incrementing IP-ID Allocation: In a per-destination IP-ID allocation the first IP-ID to some destination is selected at random and subsequent packets are allocated sequentially increasing IP-ID values. The DNS server maintains the counter mapping to the destination IP for some period of time, typically more than 30 seconds, e.g., see distribution of counter storage period that servers (authoritative for TLDs) maintain to their destinations, in Figure 5 (this is relevant only to servers supporting per-destination allocation).

In constrast to random IP-ID allocation, sequentially incrementing IP-ID assignment allows to implement a much more efficient attack. The idea is to narrow down the search space thus reducing the number of queries that the resolver is required to issue. We next describe the algorithm of the attacker. Assuming that the fragment reassembly buffer at the resolver is limited to n fragments, and $n = 64$ for the same source/destination and protocol tuple, the attacker should plant $n-2$ spoofed second fragments in the recipient's buffer, and should leave space for two authentic fragmens; this is required since otheriwse attacker's least recently arrived fragments will be evicted from the cache, when two authentic fragments enter the reassembly cache. Each i^{th} fragment, out of $n-2$ sent by the attacker, contains

$$\left\{i \cdot \frac{2^{|IP-ID|}}{(n-2)}\right\}_{i=0}^{(n-2)-1} = \left\{i \cdot \frac{2^{16}}{62}\right\}_{i=0}^{61} = \{i \cdot 1057\}_{i=0}^{61}$$

(where $|IP-ID|$ is the size of the distribution containing all possible IP-ID values). The puppet then issues 1057 DNS requests[10] for records that result in fragmented responses. With probability 1, one of the first fragments, sent in response to DNS requests, will be reconstructed with the second spoofed fragment of the attacker, that waits in the reassembly buffer; this is since the IP-ID values in the DNS responses sent to the resolver overlap the IP-ID values in one of the 1057 traps set by the attacker.

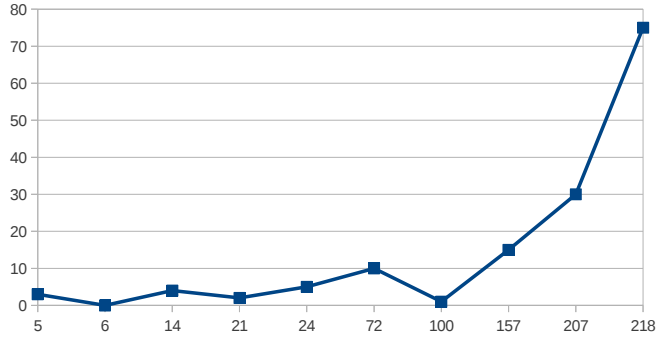

Figure 5. The time intervals that the name servers, implementing per-destination IP-ID allocation, maintain the mapping to some IP address; we mark the highest time with 75 minutes, however, large timeout intervals were observed. The x lane denotes the number of name servers, and the y denotes the period of time the counter was maintained by the server.

4) Globally Incrementing IP-ID Allocation: A globally-incrementing IP-ID appears to be widely used, and in this case the attacker can simply learn the current value, and the propagation rate[11], by querying the name server directly. The method of exploiting the IP-ID advancement to measure the transmission rate of systems is not new, and was used in [31].

5) Mixed Incrementing IP-ID Allocation: As we mentioned earlier, some of the servers, authoritative for TLDs, use load balancing (305 out of 1139 to be precise) and more than one physical machine may be allocated the same IP address. For instance, the name server a0.pro.afilias-nst.info (IP 199.182.0.1) authoritative for pro TLD, appears to be using more than 4 physical machines, as we were able to identify four sequentially incrementing IP-ID allocations (i.e., four ranges which are incremented by units of 1), and one random. Figure 6 summarises the distribution of servers (authoritative for TLDs) which support load balancing, and each physical machine uses a different counter to the destination IP. Note that the number of instances of machines

[10]To cincumvent the caching of the resolver the puppet should issue requests for records that result in non-existing domain responses (RCODE=3) or NODATA responses.

[11]The query rate to name servers is known to be stable, [30].

behind a load balancer is dynamic, and some machines may go offline and new ones may emerge; furthermore, it may be the case that our requests did not reach some of the machine instances due to the way requests are rerouted to different machines. Therefore, the study in Figure 6 is meant to provide a general idea of the load balancing implemented by name servers.

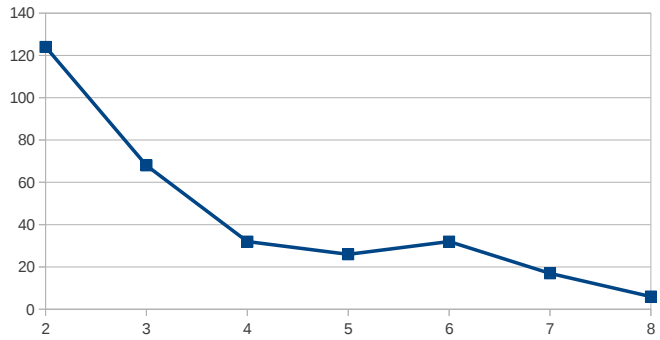

Figure 6. The distribution of the number of physical machines behind a single IP address; this holds for 305 name servers authoritative for TLDs, that support load balancing. The x corresponds to the number of name servers and the y to the number of counters detected behind an IP address.

We next show the impact of load balancing on the success probability of the attacker to hit the correct IP-ID. Let k be the number of different physical servers, such that each uses a different counter to the destination IP. In this case, the attacker can follow either of the following two strategies: (1) divide the fragment reassembly buffer into k logical buffers each of size $(n-2)/k$ fragments (where n is the actual size of the fragment reassembly buffer), or (2) issue additional $O(k)$ queries, in the worst case, in order to ensure that the load balancer relays the DNS request to the machine with the required IP-ID counter value; typically the name servers respond in a round-robin sequence, so on average, after k queries, the required name server will return the response.

In our attacks we focused on name servers that maintained a single counter to some destination. Specifically, by applying the technique in Section III-A, we forced, i.e., 'pinned', the resolver to query the name servers that did not implement load balancing.

C. Circumventing the Cache

To evade caching of the resolver the attacker can issue DNS requests for 'non-existing domain names' (NXD), i.e., responses containing an RCODE with name error, or responses indicating that the domain name exists but with a different type, i.e., with 'no data no error' responses. Such responses exist in *every* domain. We found that domains that use NSEC3 (which is currently the majority of the domains) to indicate NXD (and NODATA) responses, to be most suitable for our attacks as those responses get fragmented. This technique is similar to Kaminsky attack, as it allows the attacker to repeat the attack as frequently as required, by selecting a different random name (prepended to the real domain name) in each query.

D. Fragment Reassembly Spoofing: Applications

The IP-ID is the only 'unknown' which the attacker has to predict, and once succeeded, the attacker can impact the fragment reassembly process. In particular, fragment reassembly spoofing yields incorrect DNS responses, which can either result in a timeout at the resolver, and then subsequent retransmissions to the same resource record (which can also be ruined), or, when *correctly* formatted, those DNS responses can deliver spoofed resource records, which the resolver is bound to cache and return to benign clients. In Sections IV and III, we show how to apply and exploit fragment reassembly spoofing for disturbing attacks on availability and integrity of the DNS service; we also present an experimental evaluation of the attacks, that we performed, on BIND 9.8.3 and Unbound 1.4.17, running against real name servers authoritative for TLDs, e.g., org, mil. The vulnerabilities that allow our attacks are summarised in Table I.

III. Name Server Pinning

In this section we introduce a new attack: the *name server (NS) pinning*. NS pinning forces resolver to query a specific name server of attacker's choice. We propose two different techniques to perform NS pinning: (1) in Section III-A we apply name server blocking, and (2) in Section III-B we show how DNS poisoning can be used for NS pinning. The NS pinning is effective even if DNSSEC is correctly and fully deployed, since the glue records are not signed, [1].

In Section III-C we show that NS pinning can be applied as attack vector allowing a range of more complex attacks.

A. NS Pinning via NS Blocking

We showed that *off-path* attackers can intervene in the communication between the DNS resolvers and name servers by spoofing the fragment reassembly process. However, by merely injecting a spoofed second fragment, that gets reassembled with the authentic first fragment, the attacker does not gain much: the reassembled DNS response will most certainly be discarded by the resolver, e.g., due to incorrect UDP checksum. Also note that the reassembled DNS response is initially sent in response to a request which the attacker itself triggered. The attacker can apply this technique to inflict denial of service on the responses to legitimate clients. However, this requires a lot of effort on behalf of the attacker - in particular the attacker has to time its spoofed fragments with the queries of legitimate clients.

In this section we propose to utilise fragment reassembly spoofing to block DNS responses in a way that allows to dissuade DNS resolvers from querying particular name server(s), resulting in NS blocking. We then show how to apply *NS blocking* to perform NS pinning, i.e., force the resolvers to query name servers of attacker's choice. The NS pinning based on DNS blocking is applicable to DNS resolvers that follow a behaviour recommended in

RFC 4697 [12] and [13]. The recommendation suggests that resolvers should avoid querying non-responsive name servers, if there are two or more failed responses within some time interval (the time interval depends on the resolver implementation).

1) Response Blocking: To block responses to a query from a particular name server to the resolver, the off-path attacker needs to send an arbitrary fake second fragment y_2' with the anticipated IP-ID and other parameters, to match a legitimate response, as described above; for efficiency the attacker should generate small fragments, e.g., one byte long (plus the headers). The reassembly process using the legitimate first fragment and the fake second fragment usually fails, and both fragments are silently dropped due to incorrect checksum at the UDP layer or by the DNS resolver itself.

2) NS Blocking: The resolver will resend the query after a timeout, but the timeout periods are known to the attacker, who can easily send appropriate fake fragments to cause loss of each of the responses, until the resolver gives up. After few[12] successful *DNS response blocking* attempts the resolver marks the name server as non-responsive and does not send further DNS requests to it for the duration of the time interval; The time interval is set to 15 minutes in Unbound. Note that the attacker is not required to block consecutive responses, and blocking *any* two responses (from the same IP) within a time interval suffices for the resolver to block that name server. A similar behaviour of avoiding non-responsive name servers was observed by [13] in PowerDNS and WindowsDNS.

It is important to notice, that with most resolvers, NS pinnning is effective against a specific *name server IP*, and *not* limited to a specific domain. Namely, when a name server serves several domains, by blocking DNS responses to *one* domain that it serves we block its IP address in *general*, for all the domains that it serves. For instance, name server at IP $38.103.2.1$ serves paypal.com as well as isi-sns.info. By blocking responses to isi-sns.info the attacker blocks the name server and dissuades the resolver from querying it also for paypal.com. Since name servers often host multiple domains, DNS response blocking is very effective. According to the study that we performed, 71.53% of the name servers hosting $10,000$ top Alexa domains, and 26.28% of the name servers of TLDs, serve more than one domain.

3) NS Pinning: To force the DNS resolver to query a specific name server the attacker can repeat DNS response blocking for all the name servers, except one, which the attacker wishes the resolver to use.

The NS pinning via NS blocking is very useful and effective: (1) part of the zones, served by some name server, may support DNSSEC while others not. Support of DNSSEC typically implies fragmented DNS responses. Thus the attacker can exploit fragmented responses from one zone to block the name server for another zone whose responses are not fragmented; (2) attacker can create a new zone (with large responses) and choose to be served by same name servers as some victim domain whose responses are not fragmented (this is typically possible for second - and lower - level domains); (3) popular domains, e.g., paypal.com, may share name servers with much less frequently queried domains, e.g., info. This allows to block popular domains that are frequently queried, by launching DNS blocking against much less popular domains that are infrequently queried and match the IP-ID with little or no effort; (4) some name servers may be using load balancing, thus increasing the number of IP-ID counters. However, typically not all name servers of same domain use load balancing. Therefore, by blocking the name servers that use load balancing, the attacker improves the efficiency of the attack.

4) Experimental Evaluation: The NS pinning attack, against an Unbound 1.4.17 resolver using (fragmented) responses from 404.gov domain, is illustrated in Figure 7. In steps 1 and 2 the puppet coordinates with the attacker and issues a DNS request for $123.404.gov (where $123 is a random prefix). In steps 3 and 4, the off-path spoofing attacker (at IP 6.6.6.6) sends a forged second fragment, for all the possible name servers (i.e., a total of 2 spoofed fragments) except one which the attacker wants the resolver to use for its queries during the poisoning phase; the 404.gov domain has three name servers. This ensures that only one IP address will result in a valid response, and the other two result in a malformed DNS packets. The spoofed second fragment is incorrect, and contains a single arbitrary byte (in addition to headers). In step 5, the spoofed second fragment is reconstructed with the authentic first fragment resulting in a malformed DNS packet which leaves the fragments reassembly buffer. This malformed DNS response is then discarded by the resolver, and the IP of the name server is marked[13] as 'non-responsive'. When the authentic second fragment arrives, it does not have a match and is discarded after a timeout. As a result the resolver does not receive the response, and after a timeout it resends the DNS request to the next DNS server, step 6. The same procedure applies here, and the response is discarded. In step 9 a valid response arrives from IP 162.138.183.11. This way, by blocking the responses from all name servers except one, we forced the resolver to direct all its queries for 404.gov domain to 162.138.183.11.

The Wireshark capture, in Figure 8, that was run on the resolver, demonstrates the experimental evalutation, i.e., the DNS packets entering/leaving the network card of the

[12]The exact number of the attempts is resolver dependent, e.g., it is set to two attempts within a time interval for Unbound.

[13]In reality the resolver marks the server as 'non-responsive' after two unsuccessful responses; this is handled by sending two spoofed fragments with consecutive IP-ID in IP headers.

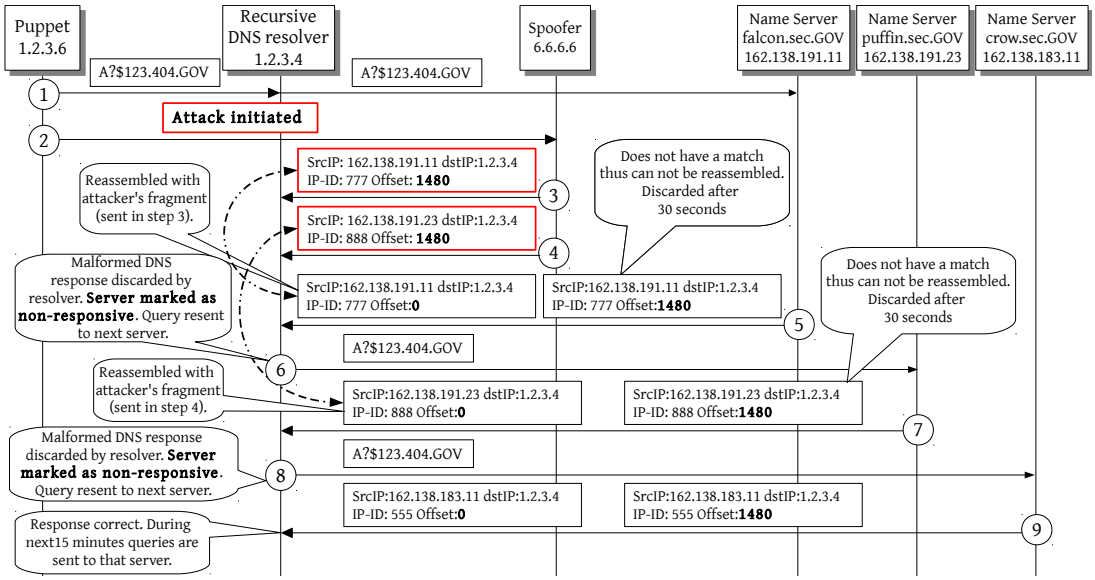

Figure 7. NS pinning via name server blocking attacks.

resolver. During the course of the experiment the puppet issued 6000 queries[14] to the resolver. The spoofer initiates the attack by sending three spoofed fragments to each IP address except 162.138.183.11. For simplicity, the capture presents only the packets exchanged between the resolver and the name server of 404.gov at 162.138.191.23 (by adjusting a corresponding filter in wireshark); the complete capture contains queries/responses from other name servers too. Packets numbered 18-20 are the forged fragments sent by the spoofer, with sequentially incrementing IP-IDs. Then puppet triggers a DNS request, packet 29. The response from the name server contains two fragments, packets 33 and 34. The first fragment is reassembled with spoofed fragment 18, resulting in a malformed packet which is discarded by the resolver.

The second fragment is discarded after a timeout. In packet 48 the resolver requests a public verification key of the 404.gov zone. The response contains three fragments 49-51; the first fragment is reconstructed with the spoofed fragment in packet 20, which also results in a malformed DNS response and is discarded. Note that this request, in packet 48, was sent at 19:28. Based on our tests it can be seen that when Unbound encounters a timeout twice for the same destination IP, it stops sending further packets to that destination for 15 minutes. Indeed, the next packet that is sent to that IP is packet number 6848, at time 19:43. The same scenario was observed with IP 162.138.191.11. The queries between 19:28 and 19:43 were sent only to 162.138.183.11, avoiding 162.138.191.11 and 162.183.191.23. Note that even if some of the responses (between packets 33 and 49) were valid and accepted by resolver, e.g., if they were not fragmented, it did not make a difference, and two timed-out responses in a 15 minute interval were sufficient for Unbound to stop querying those IP addresses; this fact shows that the success probability of the attack does not depend on the IP-ID selection mechanism.

[14]Note that our goal was to test the behaviour of the resolver, and to check the frequency of the queries to non-responsive servers; in real attack, once the IP-ID is known it is sufficient to issue two queries to mark the server as non-responsive.

B. NS Pinning via Cache Poisoning

NS pinning can also be achieved via DNS cache poisoning. In this section we assume that the attacker can launch a DNS cache poisoning attack against a specific resolver. We report our findings on what we believe to be a DNS vulnerability, and show how to apply DNS cache poisoning to launch NS ponning attack.

When launching DNS cache poisoning against some domain a.tld, the attacker should set the time-to-live (TTL) on the NS (resp A) record, e.g., ns-malicious.a.tld, that it injects, to some high value. The resolver caches the record for a period that is a minimum between the TTL (on the record itself) and its own maximal caching time parameter. Note that it is reasonable to assume that there are other (legitimate) NS records for that victim domain that are already cached at the resolver; in fact since cache poisoning attacks are against popular domains, e.g., com, its name servers are typically cached at the resolver. Note though, that since the TTL on the spoofed record is higher than the TTL on the legitimate cached records, after the legitimate records expire from cache, the resolver will use only the remaining spoofed NS record.

During our experimental evaluation we noticed that for as long as the spoofed record ns-malicious.a.tld (authoritative for a.tld) is in the cache, the resolver will not request other NS records for a.tld and will only use that (spoofed) record. The problem is that even if that spoofed NS (resp A) record is non-existent or DNSSEC validation fails and as

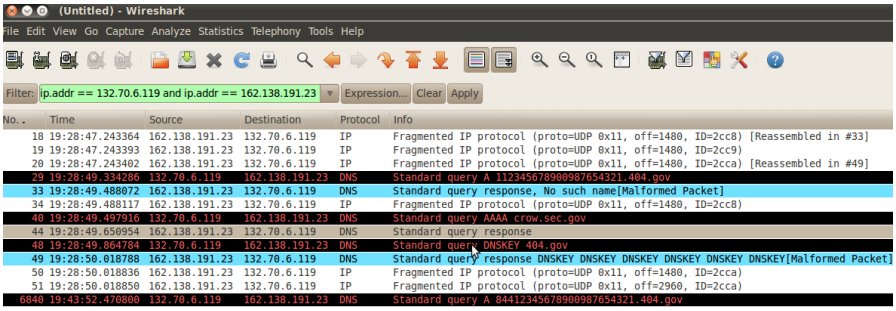

Figure 8. The wireshark capture of the attack, presenting only the packets exchanged between the name server 162.183.191.23 and the resolver. As can be observed, after two malformed responses the resolver refrains from sending further queries to that name server for 15 minutes. Fragmented packets are coloured in white, DNS requests in black, and reassembled DNS fragments in blue.

a result the requests of the resolver to that domain timeout, the resolver will not attempt to fetch the other records for that domain, but will contiue using that NS (resp A) record. Our NS pinning va cache poisoning relies on the fact that the delegation (glue) records, NS (name server) and A (IP address), located in `authority` and `additional` sections, are not signed, [1]. This allows the attacker to change the IP address (in the `additional` section) of the name server of some victim domain, to its own address, or to add a new name server (in `authority` section) for the victim domain. Such NS and A records are usually cached and used, for queries to the specified domain; see [36]. At this point, the attacker managed to cause queries for the victim subdomain to be sent to a machine controlled by the attacker.

After spoofing the glue records and injecting an NS record, the attacker will not be able to craft valid signatures. Strictly validating resolvers reject the responses with incorrect or missing signatures, resulting in timeouts for all requests to that domain. Note that the resolver does not attempt to check if there are other name servers for that domain. As a result the resolver is locked in a 'cache-or-crash' state, where it can either cache the spoofed records coming from the malicious name server, or constantly timeout, i.e., crash. Resolvers not performing strict validation, e.g., supporting permissive mode such as Unbound, cache the spoofed responses, while resolvers performing strict validation, e.g., Bind 9.8 timeouts upon subsequent requests. The result is a cache poisoning attack or an effective denial of service amplification attack.

This vulnerability, of redirecting DNSSEC enabled DNS requests to malicious server by a man-in-the-middle (since the delegation records are not signed), was pointed out by Bernstein [14], yet without a specific application for such an exploit. Bau and Mitchell, [4], refute Bernstein's claim of this being a vulnerability, by proving that it does not enable a man-in-the-middle attacker any additional capabilities, and conclude that it does not pose a significant threat. However, not signing the glue records exposes to NS pinning (leading to a range of other attacks), and to subdomain injection (Section IV).

C. NS-Pinning: Applications

NS-Pinning is rarely a goal by itself; more often, it can serve as a mechanism for other goals. We discuss three such goals: facilitation of DNS poisoning; degradation of service, and traffic analysis.

Facilitate DNS poisoning: In [11], Ramasubramanian and Sirer conducted a survey showing that a typical domain name depends on 46 servers on average, and names belonging to countries depend on a few hundred servers. They note that compromising a server can lead to domain hijacks and postulate that it is possible to hijack 30% of the domains in Yahoo and DMOZ.org directories; DNS servers are known to have vulnerabilities [37], [38]. However, [11] did not suggest a technique which can be used to force a resolver to query a specific name server. NS-pinning can provide exactly the necessary mechanism.

Also note that NS-pinning can assist in other DNS cache poisoning attacks, including these in the next section of this paper, as it allows the attacker to reduce the number of servers that the resolver can query, possibly to only one. As a result the attacker forces resolver to query name server of attacker's choice.

Off-path Degradation of Service: By blocking 'good' name servers, an attacker can cause resolvers to send their traffic to specific, 'bad' name servers. In particular, resolvers may resort to name servers with very high latency, causing unnecessary delays. Note that the zone administrators often deploy techniques to distribute the load between several physical servers sharing the same IP, e.g., using load balancing or Anycast [39]. Typically not all the name servers of a domain deploy such optimisations, e.g., 6 out of 13 root servers, [40]. Our technique allows the attacker to block those servers and to 'force' the resolver to query a name server which does not support such load balancing.

Off-path Cache or Crash Attacks: Configuration errors are common in DNS and DNSSEC, e.g., [41]. In particular, often there is a lack of coordination between the name servers serving some domain. The attacker can dussade the resolver from contacting correct name servers, and can instead force it to query bad name servers. For instance, consider a resolver that can establish a chain of trust to

the target zone (that supports DNSSEC). Assume that one name server in the target domain does not return updated signatures/keys, while all others do. If the resolver performs a strict validation of DNSSEC, e.g., Bind 9.8.3, and does not accept responses that do not validate, e.g., due to missing signatures, it will continuesly timeout, i.e., 'crash'. In contrast, consider a resolver that supports permissive mode, e.g., Unbound 1.4.17; in this case, the resolver will cache the wrong responses and return them to clients. We coin this the 'cach or crash attack'.

Off-path Traffic Analysis and Covert Channel: Many names servers provide side-channels allowing an attacker to learn or estimate the rate of requests handled by the server. In particular, one simple and effective side-channel is the IP-ID used by the name server. This mechanism also allows *off-path covert channel*, between an agent, say a bot b, which can use the resolver r, and an off-path attacker o, which can make queries to the name server ns.foo.bar. The bot can, e.g., encode information by signaling via the queries to ns.foo.bar (or possibly, signaling using distinct queries to several domains, each mapped to a specific, distinct name server). The attacker can communicate to the bot by signaling via loss of DNS responses.

IV. DNS Cache Poisoning

In this section we show how to employ fragment reassembly spoofing for modification of DNS responses by *off-path* attackers, in a way that pollutes the cache of DNS resolvers with spoofed A and/or NS records. This results in efficient and practical cache poisoning attacks on recursive DNS resolvers; our attacks allow *domain hijacking* (by injecting spoofed NS or/and A records for real domain), and *subdomain injection* (by injecting spoofed NS record for a non-existing domain). In Table III we report on the vulnerabilities, which we tested on two DNS resolver software that support DNSSEC: Bind 9.8.3 and Unbound 1.4.17; these vulnerabilities are related to: (1) the caching policies which the resolvers apply on records in DNS responses, (2) the DNSSEC validation which the resolver performs on DNSSE-enabled DNS responses, and (3) server selection algorithms. The attack begins in the same way as other cache poisoning attacks, whereby the attacker triggers a DNS request to some victim domain, e.g., using a puppet (malicious script confined in a browser). The off-path attacker also sends a spoofed second fragment (or a set thereof if the IP-ID is not known), as described in Section II. Assuming that the validation parameters (source/destination IP, protocol, IP-ID, offset) in the spoofed second fragment are correct, when the first authentic fragment arrives at the fragment reassembly buffer it is reassembled with the spoofed second fragment (that is already waiting in the reassembly cache). The complete packet is then passed to the upper TCP/IP layers.

In Section IV-A we describe the validation that the DNS resolvers apply on the DNS responses, and explain the conditions that must hold so that records in a DNS response get accepted and cached by the DNS resolver. The DNS cache poisoning on plain DNS responses is easy when the response is fragmented; this requires that the resolver supports EDNS (which allows responses over 512 bytes). Since DNSSEC is supposed to foil cache poisoning attacks we review DNSSEC, Section IV-B. We report on our findings pertaining to DNSSEC deployment vulnerabilities and outline few exploits thereof leading to effective and efficient cache poisoning attacks. We then present the cache poisoning attacks, Section IV-D, that we carried out against Bind 9.8.3 and Undound 1.4.17 resolvers poisoning entries for several popular DNSSE-enabled domains. We also validate the subdomain injection attack (Section IV-D2) against DNSSEC NSEC3 opt/out zones of [4], yet we carry out our attack by *off-path* attacker, as opposed ot a man-in-the-middle attacker used by [4].

A. Crafting a Valid DNS Response

When the DNS response arrives at the application layer of the DNS resolver the resolver validates its UDP checksum and then applies caching policies on the response to decide if to cache the records that it contains. We next elaborate on each of these checks.

1) Validating UDP Checksum of DNS Response: Matching the checksum is straightforward. The UDP checksum is simply a one's complement sum is performed on all the 16-bit values (of the payload) then the one's complement is taken of that value to populate the checksum field. Therefore, to match the checksum the attacker can query the name server in advance for the same resource record and learn the contents of the corresponding DNS response, including all contents in the second fragment. Then the attacker has to ensure that the checksum of the forged fragment y_2' is identical to the checksum of the authentic second fragment y_2, e.g., by concatenating two bytes after the EDNS record that 'fix' the checksum, or by changing two resource records: one to create the forged record, and another to fix the checksum.

2) DNS Records Caching Policies: To decide whether to cache the response the resolver applies its caching policies on the records inside the DNS responses. When constructing a spoofed second fragment and injecting spoofed DNS records, we essentially follow the known rules for injection of spoofed records into the DNS cache that were investigated and studied (most notably) by Kaminsky [42] and Son and Shmatikov [36], and by Bau and Mitchell [4] for DNSSEC enabled DNS responses.

We identified vulnerabilities pertaining to caching policies applied to records in DNS responses of type: non-existing domain (NXD, RCODE 3) and no data (NODATA, no error),

Resolver Software	Resolver Vulnerability				
	Response w/SOA	Permissive Mode	DNSSEC Stripped	Domain Insecure	NS Pinning
Bind (9.8.3)	×	×	×	×	×
Unbound (1.4.17)	√	√	√	√	√

Table III
DNS RESOLVER VULNERABILITIES, PERTAINING TO CACHING POLICIES, DNSSEC PROCESSING POLICIES, AND NAME SERVER SELECTION.

[43], which contain an SOA records in the `authority` section. We found that Unbound 1.4.17 caches the NS records that appear in NXD and NODATA type responses while Bind 9.8.3 does not cache the NS records if it detects an SOA record; we show attacks exploiting this in Section IV-D. This allows to launch effective and efficient cache poisoning attacks again Unbound resolver spoofing responses of NXD type; note that our attacks exploit DNSSE-enabled DNS responses, since plain NXD or NODATA responses are not fragmented. This vulnerability is summarised in the first column in Table III.

If the resolver supports DNSSEC it also applies DNSSEC validation on DNSSEC-enabled domains. We extend on this in Section IV-B.

B. Vulnerabilities of Incremental DNSSEC Deployment

RFC 3833 [44] requires DNSSEC to support incremental deployment, i.e., not to expose to new attacks and not to harm the Internet functionality. In this work we put this requirement to test. However, as our results indicate, the incremental deployment of DNSSEC may further exacerbate the DNS cache poisoning vulnerabilities[15]. The incremental deployment has impact on both the resolvers and the zones. With respect to resolvers the incremental deployment is related to running in some 'test non-validating' mode for resolvers, e.g., the permissive mode supported by resolvers, e.g., Unbound. The incremental deployment for zones means adopting DNSSEC, yet the resolvers cannot establish a chain-of-trust to them, since, e.g., the parent zone does not delegate the signed public-key of the child; these zones are called 'islands of security'.

We next extend on the implications of incremental DNSSEC deployment by zones and DNS resolvers.

1) Islands of Security: Although 30.1% of TLDs, and 2% out of 300,000 Alexa domains, already support DNSSEC, the resolvers cannot validate the public-keys of many of them, e.g., the parent zone does not delegate the signed public-key of the child while the child supports DNSSEC, e.g., most children of gov. Specifically an 'Island of Security' means that not *all* the zones from the root to the target zone deploy DNSSEC correctly. In this case the DNSSEC does not offer any protection, since in this case the DNS resolvers fall back to non-validating mode. But, worse, it can even expose to efficient and practical cache poisoning attacks.

2) Permissive Resolvers: A permissive resolver is one that supports DNSSEC, however, ignores DNSSEC validation failures, e.g., if the signatures are missing or invalid. We found that Unbound 1.4.17 DNS resolver has several explicit parameters allowing to turn different 'permissive' mode features on[16]: (1) `val-permissive-mode` (when set, the resolver does not return SERVFAIL when DNSSEC does not validate, e.g., due to incorrect or missing signatures); (2) `domain-insecure` (ignores the chain-of-trust and the domain is treated as insecure); (3) `harden-dnssec-stripped` (failing to validate DNSKEY data for a trust anchor will trigger an insecure mode for that zone). We generally call of these parameters: the permissive mode.

Obviously, for such resolvers, DNSSEC does not provide added security; yet, there appears to be a significant number of such resolvers [10], [45], apparently due to concerns about loss of connectivity due to interoperability and other problems upon enforcing DNSSEC. Such implementors deploy DNSSEC incorrectly or possibly via an 'incremental deployment', aiming to preserve the DNS functionality with intermediate Internet devices, e.g., firewalls, and legacy resolvers which may discard DNSSEC enabled DNS responses or strip DNSSEC signatures. This approach apparently assumes that permissive use of DNSSEC can provide evidence on whether the network can deploy DNSSEC fully without problems or not, while not *harming* their security; unfortunately, this is not the case and as we show such resolvers are open to poisoning.

C. DNSSEC NSEC3 opt/out

The NSEC3 DNSSEC record with opt-out option [46] allows attackers to create fake (non-existing) sub-domains. As [4] showed this facilitates XSS, phishing and cookie stealing attacks. Sub-domain injection attack by a *man-in-the-middle* attacker was proposed in [4]; [4], also suggested that the attack could be carried out by an off-path attacker, assuming that *only* the transaction ID in DNS packets is randomised. However, this assumption does not hold for most DNS resolvers, as they (at the very least) support source port randomisation. In Section IV-D2 we show that such an attack can be effectively carried out by an off-path attacker, that does not intercept and inspect packets, and against patched DNS resolvers, i.e., supporting source port randomisation, IP randomisation and DNS query randomisation.

In spite of the publication of this potential abuse by MitM [4], NSEC3 opt-out is still widely used, and often

[15]It is known that DNSSEC may also expose to denial of service attacks due to large DNS responses, but this is not the focus of our work.

[16]These parameters reside in the configuration file Unbound.conf.

even recommended, since it improves performance (esp. as long as DNSSEC is deployed only in small fraction of the domains).

D. DNS Cache Poisoning Attacks: Experimental Evaluation

We next report on our experimental evaluation of the attacks, against Bind 9.8.3 and Unbound 1.4.17 resolvers, using responses from real zones. Our attacks allow domain hijacking (by injecting an NS or an R record), or subdomain injection (by injecting an NS record for a new subdomain).

1) Domain Hijacking: Domain hijacking can be performed when either the resolver is supporting a permissive mode, or the target zone is an island of security. In this case, the attacker can replace the RR(s) in the authentic second fragment of a DNS response with (spoofed) NS or A RR(s) pointing at his name server; the TTL in those spoofed RRs has to match the TTL of the other RRs in the same RRset, in order for the resolver to accept and cache it.

We tested the domain hijacking attack in two scenarios by spoofing fragments from: (1) nxdomain/no-data responses and (2) DNSKEY responses. The 'nxdomain/no-data' (NSEC33) response is often fragmented in the `authority` section, with NS records (from the `authority` and A (and EDNS) records (from the `additional`) residing in a second fragment. This allows replacing the `authority` records in the second fragment with fake NS RRs (or `additional` records with spoofed A records); see Section IV-D2.

The 'existing domain' response, e.g., for DNSKEY or TXT request, is also often fragmented. Such responses typically contain records in the `additional` section too, and allow changing the IP of name server with IP of the attacker; see Section IV-D2.

2) Injecting NS RR to NSEC3 Response: Typically, responses of type 'non-existing domain (nxdomain)' or 'no data no error', in domains that support NSEC3, are of size between 1700 to 2000 bytes. This allows the attacker to replace the authentic NSEC3 or RRSIG RR(s) with a NS RR for a new name server; see Figure 9. If the response does not contain any other NS RRs then the attacker can set an arbitrary high TTL to ensure that his RR stays in cache when the authentic NS RRs for that domain expire. Note that in this attack, to save space in the paper, we injected two NS records: one for an existing domain and another for a new subdomain; we did this to validate subdomain injection and name server hijacking attacks in one response, see Figure 10. The record: www.sec.cs.biu.ac.il is a new (non-existing) subdomain under sec.cs.biu.ac.il and it points at the name server ams.sec.cs.biu.ac.il controlled by the attacker. The attacker triggers a DNS request (via a puppet) and synchronises (steps 1 and 2, Figure 9). Then (step 3) the attacker sends a spoofed second fragment containing an NS RR for domain sec.cs.biu.ac.il. This spoofed fragment is combined with the authentic first fragment (step 3) and

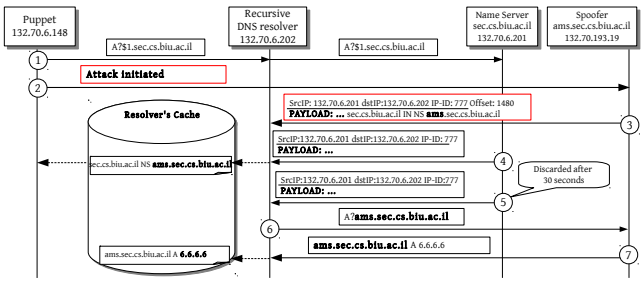

Figure 9. Poisoning an nxdomain (or no data) response, by replacing the NSEC3 RR with an NS RR.

enters the cache; the authentic second fragment is discarded after a timeout (step 5). Note that the attacker can provide any arbitrary NS RR, in particular, one that is not in the same domain as the victim; in this attack we spoofed the response with name for a new NS RR, i.e., **ams**.sec.cs.biu.ac.il, in our domain, i.e., sec.cs.biu.ac.il, for testing purposes to observe that the subsequent queries of the resolver to domain sec.cs.biu.ac.il are sent to ams.sec.cs.biu.ac.il and responses get cached.

The wireshark capture of the resulting poisoned DNS response is in Figure 10. The authentic fragment contains part of the RRSIG and two complete records, i.e., NSEC3 and a corresponding RRSIG. The spoofed fragment contained the authentic part of the RRSIG, spanning the first and second fragments, and two fake NS records which replaced the authentic NSEC3 and RRSIG.

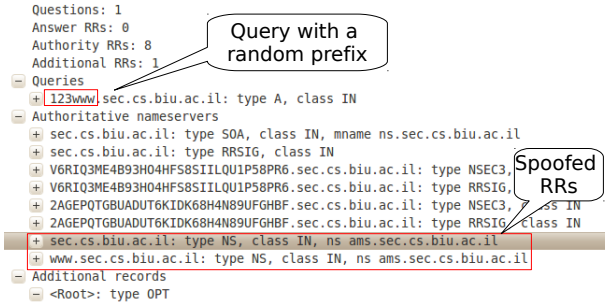

Figure 10. Poisoning an nxdomain (or no data) response for domain sec.cs.biu.ac.il, by replacing the NSEC3 RR with an NS RR.

Injecting A RR to DNSKEY Response: In this attack we spoof the IP for the name server of org domain, in a DNS response for a DNSKEY of org domain.

The resolver issues a DNS request for the DNSKEY of org; this is an indirect way to trigger a query, i.e., the resolver asks for the DNSKEY of some domain automatically, when the DNSKEY expires from cache, or when it needs to validate records for that domain, e.g., to be able to validate an A record or a non-existing domain (NSEC3) record; an attacker may also be able to cause a resolver, which does not support DNSSEC, to issue such a query, by sending an

appropriate request to the resolver. This query type is useful if the response to an nxdomain query is not fragmented.

The annotated screen caption of the attack is illustrated in Figure 11. The first line (122) contains the 'forged second fragment'; this fragment is kept in the defragmentation cache of the resolver, waiting for a matching first fragment (i.e., with the same set of `(source IP=199.249.112.1, dest IP=132.70.6.202, fragment ID=7c6e, protocol=UDP)`). In the next line (133), the resolver sends the DNS query to the name server.

Next line (134) is the first fragment of authentic response to the query, sent by the name server of org (at IP 199.249.112.1). This response matches the fake second fragment already in the defragmentation cache, hence it appears as a complete DNS response packet. The contents of this packet are described in the lower panes; in particular, see the two forged resource records in the additional section, which contain incorrect (adversarial) IP addresses for two of the name servers of the org domain.

Finally, notice that the authentic second fragment, received in line 135, has no matching first fragment (since the one received was already reassembled with the spoofed second fragment). Hence, it is entered into the defragmentation cache, where it is likely to remain until discarded (typically, after maximal lifetime of about 30 seconds).

V. Conclusions and Defenses

We showed how an off-path attacker can efficiently exploit fragmented DNS responses to poison DNS caches. Most DNS responses are short, and hence not fragmented; however, some DNS responses can be long and may get fragemented. Ironically, one reason for such long responses is the use of DNSSEC; however, we also show how attackers can cause such fragmentation, e.g., by intentionally registering domain names with long referral.

We also introduced the NS-pinning attack, whereby an attacker manipulated the choice of name server to be used to resolve a particular domain; and we showed NS-pinning attacks, in particular, exploiting fragmented responses.

The attacks are effective against valid implementation of the DNS and IP specifications; furthermore, we have confirmed effectiveness against several domains, using real network scenarios and common resolvers (Unbound 1.4.1 and Bind9).

We want to caution against drawing the conclusion that DNSSEC should not be used. In fact, the best defense against the poisoning attacks we described, is to apply DNSSEC correctly in *all* resolvers and domains (without using NSEC3 opt-out and allowing only strict validation); this will certainly prevent many of our poisoning attacks, and even defend against more powerful Man-in-the-Middle adversaries. Note, however, that current DNSSEC specification, would not prevent the NS-pinning attacks.

The vulnerability which allowed us to launch the poisoning attacks against recursive resolvers, is due to the fact that the resolver advertises a large EDNS buffer, which is usually larger than the MTU, e.g., 1500 bytes. Although support of large DNS responses is critical to facilitate DNSSEC enabled DNS responses, or public-key certificates [47], such long responses can be (temporarily) sent over TCP, using path MTU discovery and avoiding fragmentation. Resolver or even at intermediate gateway (firewall) can achieve similar result by setting a maximal EDNS buffer value to at most 1500, or even less, to avoid fragmentation (possibly using a variant of path MTU discovery).

Another short-term defense, which administrators of resolvers can apply, is to reduce the maximal number of fragments cached; e.g., currently 64 by default in Linux (per (source IP , dest IP , protocol) triplet). Of course, reduction in this parameter may also increase packet loss.

Resolves should also be careful to avoid caching NS or A records received within a 'non-existing domain' response.

Yet another possible defense, for name servers, is to always add a *random RR* to any packet over certain size (i.e., which may be fragmented). A simple type A resource record, containing random IP address for some fictitious domain name, would suffice. The TTL of such an RR should be set to zero to prevent the resolver from caching that record. This would prevent the attacker from being able to predict and (correctly) adjust the checksum value.

Finally, we suggest caution when deploying the proposal in [12], [13] (for server selection) which recommends to avoid querying non-responsive servers. Resolvers that do not conform to that recommendation, e.g., Bind9, are not vulnerable to our server-pinning attacks.

References

[1] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "Protocol Modifications for the DNS Security Extensions," RFC 4035 (Proposed Standard), Internet Engineering Task Force, Mar. 2005, updated by RFCs 4470, 6014. [Online]. Available: http://www.ietf.org/rfc/rfc4035.txt

[2] ——, "Resource Records for the DNS Security Extensions," RFC 4034 (Proposed Standard), Internet Engineering Task Force, Mar. 2005, updated by RFCs 4470, 6014. [Online]. Available: http://www.ietf.org/rfc/rfc4034.txt

[3] ——, "DNS Security Introduction and Requirements," RFC 4033 (Proposed Standard), Internet Engineering Task Force, Mar. 2005, updated by RFC 6014. [Online]. Available: http://www.ietf.org/rfc/rfc4033.txt

[4] J. Bau and J. C. Mitchell, "A security evaluation of DNSSEC with NSEC3," in *Network and Distributed Systems Security (NDSS) Symposium*. The Internet Society, 2010. [Online]. Available: http://www.isoc.org/isoc/conferences/ndss/10/

[5] D. J. Bernstein, "DNSCurve: Usable security for DNS," Posted at: http://dnscurve.org/, 2010.

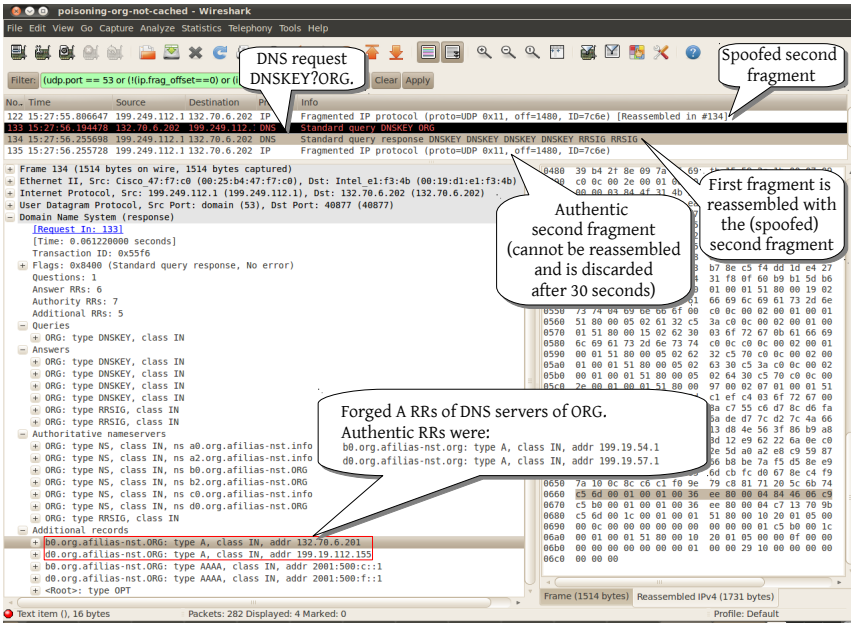

Figure 11. A wireshark capture presenting the packets in a DNS cache poisoning attack exploiting fragmented DNSKEY responses of org. The resolver issues a query for a DNSKEY, and the spoofer sends a poisoned second fragment containing the forged A records for NS of org.

[6] D. Kaminsky, "It's the End of the Cache As We Know It," Presentation at Blackhat Briefings, 2008.

[7] D. Dagon, M. Antonakakis, P. Vixie, T. Jinmei, and W. Lee, "Increased DNS forgery resistance through 0x20-bit encoding: security via leet queries," in *ACM Conference on Computer and Communications Security*, P. Ning, P. F. Syverson, and S. Jha, Eds. ACM, 2008, pp. 211–222. [Online]. Available: http://doi.acm.org/10.1145/1455770.1455798

[8] R. Perdisci, M. Antonakakis, X. Luo, and W. Lee, "WSEC DNS: Protecting recursive DNS resolvers from poisoning attacks," in *DSN*. IEEE, 2009, pp. 3–12.

[9] A. Hubert and R. van Mook, "Measures for Making DNS More Resilient against Forged Answers," RFC 5452 (Proposed Standard), Internet Engineering Task Force, Jan. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5452.txt

[10] O. Gudmundsson and S. D. Crocker, "Observing DNSSEC Validation in the Wild," in *SATIN*, March 2011.

[11] V. Ramasubramanian and E. Sirer, "Perils of transitive trust in the domain name system," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. USENIX Association, 2005, pp. 35–35.

[12] M. Larson and P. Barber, "Observed DNS Resolution Misbehavior," RFC 4697 (Best Current Practice), Internet Engineering Task Force, Oct. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4697.txt

[13] Y. Yu, D. Wessels, M. Larson, and L. Zhang, "Authority server selection of dns caching resolvers," *ACM SIGCOMM Computer Communication Reviews*, April 2012.

[14] D. J. Bernstein, "Breaking DNSSEC," 3rd USENIX Workshop on Offensive Technologies, August 2009.

[15] C. A. Kent and J. C. Mogul, "Fragmentation Considered Harmful," Western Research Lab, Research Report 87/3, dec 1987, see also abbreviated version in proceedings of ACM SIGCOMM, 390–401, 1987. [Online]. Available: ftp://gatekeeper.research.compaq.com/pub/DEC/WRL/research-reports/WRL-TR-87.3.pdf

[16] J. Mogul and S. Deering, "Path MTU discovery," RFC 1191 (Draft Standard), Internet Engineering Task Force, Nov. 1990. [Online]. Available: http://www.ietf.org/rfc/rfc1191.txt

[17] M. Mathis and J. Heffner, "Packetization Layer Path MTU Discovery," RFC 4821 (Proposed Standard), Internet Engineering Task Force, Mar. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4821.txt

[18] M. Zalewski, "A New TCP/IP Blind Data Injection Technique?" BugTraq mailing list post, http://lcamtuf.coredump.cx/ipfrag.txt, 2003.

[19] F. Gont, "Security Implications of Predictable Fragment Identification Values," Internet-Draft of the IPv6 maintenance Working Group (6man), December 2011, expires June 17, 2012.

[20] ——, "Security Assessment of the Internet Protocol Version 4," RFC 6274 (Informational), Internet Engineering Task Force, Jul. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6274.txt

[21] Y. Gilad and A. Herzberg, "Fragmentation Considered Vulnerable: Blindly Intercepting and Discarding Fragments," in *Proc. USENIX Workshop on Offensive Technologies*,

Aug 2011. [Online]. Available: http://www.usenix.org/events/woot11/tech/final/files/Gilad.pdf

[22] S. Antonatos, P. Akritidis, V. T. Lam, and K. G. Anagnostakis, "Puppetnets: Misusing Web Browsers as a Distributed Attack Infrastructure," *ACM Transactions on Information and System Security*, vol. 12, no. 2, pp. 12:1–12:15, Dec. 2008.

[23] A. Herzberg and H. Shulman, "Security of patched DNS," in *ESORICS*, 2012.

[24] J. Postel, "Internet Protocol," RFC 791 (Standard), Internet Engineering Task Force, Sep. 1981, updated by RFC 1349. [Online]. Available: http://www.ietf.org/rfc/rfc791.txt

[25] J. Heffner, M. Mathis, and B. Chandler, "IPv4 Reassembly Errors at High Data Rates," RFC 4963 (Informational), Internet Engineering Task Force, Jul. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4963.txt

[26] P. Vixie, "Extension Mechanisms for DNS (EDNS0)," RFC 2671 (Proposed Standard), Internet Engineering Task Force, Aug. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2671.txt

[27] Kernel.org, "Linux Kernel Documentation," http://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt, 2011.

[28] Alexa, "The web information company," http://www.alexa.com/.

[29] T. Brisco, "DNS Support for Load Balancing," RFC 1794 (Informational), Internet Engineering Task Force, Apr. 1995. [Online]. Available: http://www.ietf.org/rfc/rfc1794.txt

[30] D. Wessels and M. Fomenkov, "Wow, that'sa lot of packets," in *Proceedings of Passive and Active Measurement Workshop (PAM)*, 2003.

[31] S. Sanfilippo, "About the IP Header ID," http://www.kyuzz.org/antirez/papers/ipid.html, Dec 1998.

[32] F. E. B. I. Domains, "Listing of Federal Agency Internet Domains," http://explore.data.gov/Federal-Government-Finances-and-Employment/Federal-Executive-Branch-Internet-Domains/k9h8-e98h, February 2012.

[33] S. S. (via bugtraq), "Icmp Fragmentation Needed Vulnerability Details: CVE-2001-0323," June 2001. [Online]. Available: http://www.cvedetails.com/cve/CVE-2001-0323/

[34] F. Gont, "Icmp attacks against tcp," October 2006. [Online]. Available: http://www.gont.com.ar/drafts/icmp-attacks/draft-ietf-tcpm-icmp-attacks-01.txt

[35] ——, "ICMP Attacks against TCP," RFC 5927 (Informational), Internet Engineering Task Force, Jul. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5927.txt

[36] S. Son and V. Shmatikov, "The hitchhikers guide to dns cache poisoning," *Security and Privacy in Communication Networks*, pp. 466–483, 2010.

[37] A. Khurshid, F. Kiyak, and M. Caesar, "Improving robustness of dns to software vulnerabilities," in *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM, 2011, pp. 177–186.

[38] S. Focus, "Bugtraq mailing list," http://www.securityfocus.com/vulnerabilities.

[39] T. Hardie, "Distributing Authoritative Name Servers via Shared Unicast Addresses," RFC 3258 (Informational), Internet Engineering Task Force, Apr. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3258.txt

[40] Z. Liu, B. Huffaker, M. Fomenkov, N. Brownlee, and K. Claffy, "Two days in the life of the dns anycast root servers," *Passive and Active Network Measurement*, pp. 125–134, 2007.

[41] V. Pappas, Z. Xu, S. Lu, D. Massey, A. Terzis, and L. Zhang, "Impact of configuration errors on dns robustness," in *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4. ACM, 2004, pp. 319–330.

[42] D. Kaminsky, "It's The End Of The Cache As We Know It," in *Black Hat conference*, August 2008, http://www.doxpara.com/DMK_BO2K8.ppt.

[43] M. Andrews, "Negative Caching of DNS Queries (DNS NCACHE)," RFC 2308 (Proposed Standard), Internet Engineering Task Force, Mar. 1998, updated by RFCs 4035, 4033, 4034. [Online]. Available: http://www.ietf.org/rfc/rfc2308.txt

[44] D. Atkins and R. Austein, "Threat Analysis of the Domain Name System (DNS)," RFC 3833 (Informational), Internet Engineering Task Force, Aug. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3833.txt

[45] S. Castro, M. Zhang, W. John, D. Wessels, and K. Claffy, "Understanding and preparing for dns evolution," *Traffic Monitoring and Analysis*, pp. 1–16, 2010.

[46] B. Laurie, G. Sisson, R. Arends, and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence," RFC 5155 (Proposed Standard), Internet Engineering Task Force, Mar. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5155.txt

[47] I. Violet, "Dnssec and tls," http://www.imperialviolet.org/2010/08/16/dnssectls.html, Aug 2010.