Fragmentation Considered Poisonous, or: One-Domain-to-Rule-Them-All.ORG

Amir Herzberg
Department of Computer Science
Bar Ilan University
Ramat Gan, Israel
Email: amir.herzberg@gmail.com

Haya Shulman*
Fachbereich Informatik
Technische Universität Darmstadt/EC-SPRIDE
Darmstadt, Germany
Email: haya.shulman@gmail.com

***Abstract*—We present effective *off-path* DNS cache poisoning attacks, circumventing widely-deployed challenge-response defenses, e.g., transaction identifier randomisation, port and query randomisation.**

Our attacks depend on the use of UDP to retrieve long DNS responses, resulting in IP fragmentation. We show how attackers are often able to generate such fragmented responses, and then abuse them to inject spoofed, 'poisonous' records, into legitimate DNS responses.

We also studied how resolvers, name servers, domains and registrars, can defend against our attacks. The best defense is deployment and enforcement of DNSSEC validation. However, DNSSEC must be deployed correctly by both domain and resolver, which is challenging; we hope our results will catalyse this process, but it will surely take long time. In fact, recent study found less than 1% of resolvers reject responses upon DNSSEC validation failures. Note also that, ironically, adoption of DNSSEC by a domain, is the main reason for fragmented DNS responses (abused in our attacks). We therefore present several short-term countermeasures, which can complement DNSSEC, especially until DNSSEC deployment is complete.

We validated our attacks against popular resolvers (Bind and Unbound), and real domains in the Internet.

Keywords: DNS security, DNS cache poisoning, fragmentation attacks, off-path attacks.

I. Introduction

The correctness and availability of information in the Domain Name System (DNS) are crucial for the operation of the Internet. There is a long history of attacks on the DNS, most notably *DNS cache poisoning*, where the attacker provides incorrect records in a DNS response, which are then cached and served to clients by the DNS resolver. DNS cache poisoning may allow weak *off-path* attackers to redirect communication to incorrect, adversarial, servers, thereby enabling an off-path attacker to intercept and modify content; as a result off-path attackers can circumvent many defense mechanisms such as Same Origin Policy (SOP), domain blacklists and domain-policies (e.g., SPF), exposing users to a range of attacks, such as phishing, credentials-theft (e.g., XSS), and more.

To protect against DNS cache poisoning attacks, the IETF defined and standardised DNSSEC [RFC4033-4035]. DNSSEC authenticates DNS responses using digital signatures, providing security not only against off-path, but also against *Man-in-the-Middle (MitM)* attackers. However, DNSSEC deployment is challenging, mainly since it requires both, adoption by domains, and validation by resolvers. Although it was proposed more than 15 years ago, only about 2% of the domains are signed and about 1% of the resolvers perform validation of DNSSEC-enabled DNS responses, [1].

Currently, while DNSSEC deployment tarries, most resolvers rely on *challenge-response* mechanisms to prevent DNS cache poisoning, using existing fields in DNS queries, which are echoed in responses. DNS requests contain a short, 16 bit, *transaction identifier* (TXID) field, originally designed to match an incoming response with a pending request. Following Kaminsky's poisoning attack [2], additional sources of randomness were added, most notably source port and name server address randomisation; these defenses are widely deployed and standardised [RFC5452]. Such challenge-response mechanisms only require resolvers to properly randomise the fields in the requests, and validate that they were properly echoed in the responses; there is no dependency on support by the name servers, hence, deployment is easy. However, challenge-response mechanisms offer no protection against MitM attackers. Yet, the common belief (or hope) is that these fields, when chosen randomly, cannot be predicted by an off-path attacker and hence suffice to prevent off-path poisoning.

Indeed, it seems that many practitioners are pacified by these improved challenge-response mechanisms; the urge to deploy DNSSEC, especially following Kaminsky's attack, seemed to have reduced. We believe that this is a mistake, for two main reasons: **(1)** in practice attackers may often be able to gain MitM capabilities, e.g., wireless networks, insecure/malicious devices, or insecure routing, and **(2)** as we show in this work and our previous work [3]–[5], existing challenge-response mechanisms may not be secure, since the challenge values can often be guessed by *off-path attackers* in common network configurations.

In this work, we present an even more convincing argument: attacks which foil *all* deployed challenge-response defenses, *without* depending on the use of NAT or other special network configuration/device/vulnerability, as in [3], [5]. Furthermore, our attacks do not require guessing the challenge values. The

*This work was carried out while the second author was in the Department of Computer Science, Bar Ilan University.

attacks' requirements are modest: (1) we assume ability to trigger DNS requests, e.g., by redirecting a user, of a victim resolver, to a malicious website, (2) an off-path attacker, and (3) existence of sufficiently-long DNS responses. Long responses exist for some DNS queries, e.g., ironically, for domains using DNSSEC; we also show that in many domains allowing registration of sub-domains, e.g., `org`, an attacker can register a sub-domain which causes fragmented referrals from the parent (e.g., `org`), enabling our poisoning attacks.

The attacks are very efficient, and further enhanced with improved 'birthday optimisations', i.e., circumventing the 'birthday prevention' mechanism adopted by most resolvers following Kaminsky's attack.

The attacks are based on three observations: **(1)** although most DNS responses are short, some responses are long[1], and may get fragmented; **(2)** all challenge-response parameters, used to authenticate DNS responses (TXID, source port, and query), are contained in the *beginning* of the response, i.e., in the first fragment (if fragmented), and **(3)** it is possible, in certain situations, to replace the second fragment with a spoofed second fragment, tricking the resolvers into caching a poisoned record; see Figure 1. To launch the cache poisoning attack, the following challenges need to be overcome:

► FRAGMENTATION. typical DNS response are short, and hence are not fragmented. We found that some responses are fragmented. Furthermore, the attacker can often *cause* fragmentation, by registering a maliciously-crafted subdomain; see Section II).

► DEFRAGMENTATION CACHE POISONING. a spoofed second fragment, sent by the attacker, must have correct IP-ID, protocol, and IP addresses, to be reassembled with the authentic first fragment of a DNS response, and must be in the cache when the corresponding first fragment arrives. This is challenging, yet doable with good success probability; see Section III.

► ENSURING VALID DNS RESPONSE. the 'mixed' DNS response, consisting of a spoofed second fragment and an authentic first fragment, should be valid and cached; this requires addressing the following challenges, as addressed in Section IV:

- *UDP checksum:* the reassembled IP packet must have a correct UDP checksum, so that it is not rejected by the OS on the receiving host.
- *Valid DNS response:* the reassembled IP packet must have valid format and structure, and the challenge-response values should match those sent within the request, so that it is not rejected by the DNS resolver software.
- *Overriding cached records:* the injected spoofed records must comply with resolvers' caching policies so that they are *not only accepted, but also cached* by the resolver.

We present efficient attacks applicable to many popular domains and resolvers, overcoming these challenges. We have experimentally validated our attacks on two (popular) open-source resolvers that support DNSSEC: Bind 9.8.3 and Unbound 1.4.18. Some of our techniques may be useful for other purposes, e.g., to intercept traffic, improving the attack of [6], and for Denial-of-Service (DoS) attacks.

Hence, we believe that the DNS transactions should best be protected using cryptography, and DNSSEC seems to be the practical choice, with security based on experts-scrutiny as well as on analysis [7]. However, we highlight some issues with the deployment of DNSSEC, which require improvement. **First**, we show that, ironically, as long as resolvers do not 'strictly enforce' DNSSEC validation, adoption by domains may actually *facilitate the attack*, by providing long, fragmented responses. Adoption of multiple signing algorithms and multiple key lengths may further *exacerbate* the problem; the currently popular 1024-bit RSA is too weak. **Second**, when the resolvers cannot establish a chain of trust from the root zone to the target domain, they fall back to a non-validating mode, also facilitating the attack. **Third**, we show that DNSSEC does not prevent two variants of the attack: *fake subdomain injection*, when the (widely-used) DNSSEC NSEC3 OPT-OUT option is supported, and *name-server pinning* attack, which can be abused for DoS and traffic analysis.

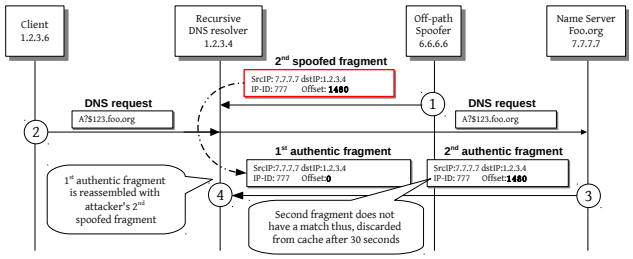

Fig. 1. Fragmentation based DNS cache poisoning attack. The off-path attacker, at IP 6.6.6.6, sends a spoofed second fragment to the resolver (step 1), and then triggers a DNS request, e.g., via a client visiting an attacker-controlled webpage (step 2). The response from name server is fragmented (step 3). The first fragment is reassembled with spoofed fragment (step 4), poisoning the cache of the resolver.

Contributions

(1) We show how to exploit fragmentation to perform efficient DNS cache poisoning attacks, circumventing challenge-response, and birthday protection, mechanisms, and motivate adoption of DNSSEC.

(2) We show efficient defragmentation-cache poisoning attacks (focusing on DNS responses); this includes study of IP-ID allocation among DNS name servers.

(3) We show an off-path *subdomain injection* attacks, against DNSSEC configurations using NSEC3 OPT-OUT. This improves over the result of [7], which required a MitM attacker, or vulnerable DNS implementations, e.g., not supporting source port randomisation.

(4) We evaluate an extensive range of short-term countermeasures against our attacks; a truly satisfying defense remains an important challenge, though.

[1]Originally, DNS packets were restricted to 512 bytes, but longer responses are possible using the EDNS record [RFC2671].

II. Using Fragmentation for DNS Cache Poisoning

IP fragmentation is rare in the Internet. Less than 1% of the traffic is fragmented. Hence the first challenge of our attacks, is to trigger a DNS request, whose response would get fragmented; we address this challenge in Subsection II-A. Then in Subsection II-B we present a high level overview of our cache poisoning attacks.

A. Fragmented Victim-Domain Responses

Most DNS responses are short and do not fragment. However, we found that there is a significant percentage of queries which result in long, fragmented, responses. This is especially true, once domains adopt DNSSEC. The most common long response are: (1) responses for `ANY` type requests (which return all the records in the target zone) and (2) DNSSEC enabled DNS responses, which contain cryptographic signatures and keys; see Figure 2 for a distribution of the length of responses for second-level domains (SLDs) of `gov` TLD, separating between SLDs which are DNSSEC-signed and those which are not. It is easy to see that a significant fraction of the responses, especially DNSSEC-enabled SLDs, are fragmented (assuming typical MTU of 1500 bytes or less). We note that as DNSSEC adoption proceeds, we expect to see more (and longer) such responses, due to the use of longer keys, multiple key lengths, and multiple algorithms.

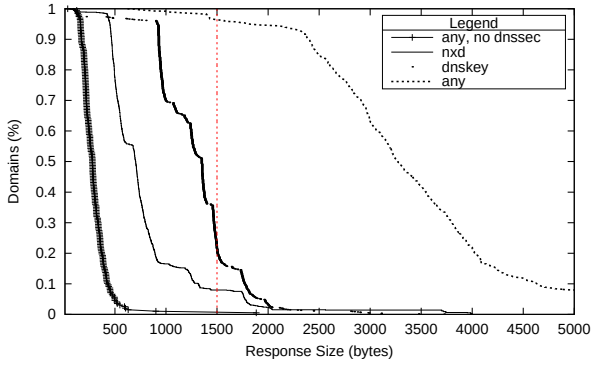

Fig. 2. Length of ANY, regular and NXDOMAIN responses of (DNSSEC-protected and 'unprotected') `gov` domains; domains taken from [8].

However, the attacker is not restricted to using existing domains, and can register domains, intentionally causing the referral from the parent to be very long (fragmenting), and use them to facilitate the attacks. We registered several such domains, under a number of TLDs, most notably `org`, `info` and `de`.

To generate long *referral* responses, we used the largest-possible number of name servers, and maximal-length names for each of them; we found the restrictions, supported by the registrars, to be lower than specified in the standard, but sufficient to cause fragmentation (and poisoning).

B. Cache Poisoning Attacks: Overview

In this section we describe two cache poisoning attacks exploiting fragmented DNS responses, type *answer* and *referral*. Depending on the response length and type, the second fragment may contain records that reside either in the `additional`, or `authority`, section (or both) of the DNS response. The attacker crafts a fake second fragment that replaces the authentic NS (name server) records in `authority` section, or A (IP address) records in `additional` section, with spoofed NS (or respectively A) records.

We show poisoning of an *answer* response in Figure 3; see poisoning of a *referral* response in technical report [9].

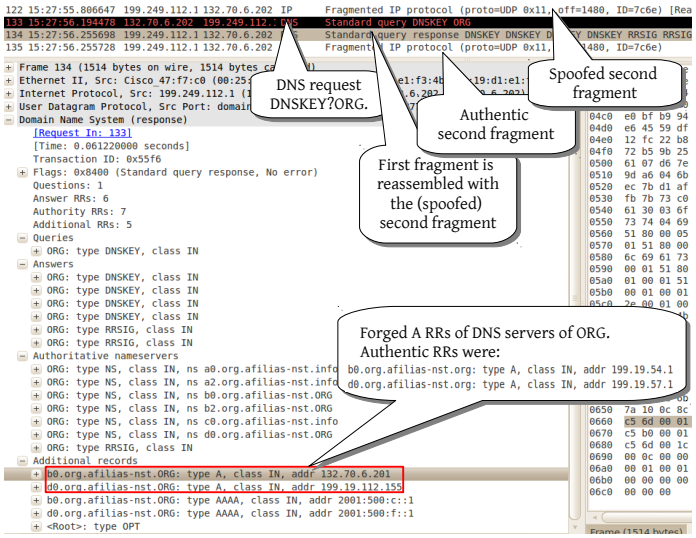

Fig. 3. A DNS cache poisoning attack exploiting fragmented DNS responses of `org` domain. The attacker replaces authentic IP addresses, of the name servers of `org`, with the address of a machine that it controls; the second address is also changed to adjust the checksum.

Poisoning Answer Response: A wireshark capture, in Figure 3, that was run on a network interface card (NIC) of a DNS resolver shows the packets that were received and sent by the resolver[2]; the resolver is at IP address 132.70.6.202 and the response arrives from 199.249.112.1 (one of the name servers of `org`). Line 122 shows a spoofed second fragment sent by the attacker (subsequently reassembled with a packet in line 134). Then a DNS request for a DNSKEY of `org` is sent (line 133); the response arrives in two fragments, such that part of the records in the *authority* section and all the records in *additional* section reside in second fragment[3]. Once the first fragment arrives and it is reassembled with the spoofed fragment waiting in the defragmentation cache. The response, illustrated in the main wireshark screen, is composed of a spoofed second fragment and a first authentic fragment, and contains spoofed IP addresses of name servers of `org`. The authentic second fragment, arriving in line 135, is discarded after a timeout, since there is no matching first fragment. The attacker replaces the IP addresses of the name servers of `org`

[2]The wireshark capture features only the packets relevant to the attack - this was accomplished by specifying an appropriate wireshark filter.

[3]Following disclosure of our attack, the name servers of `org` were patched to exclude the name servers records from responses containing DNSKEY records.

with spoofed IP addresses.

Poisoning Referral Response: The attack, described above, exploited fragmented DNS responses, however, most DNS responses are not sufficiently long to undergo fragmentation. For instance, the attack strategy above does not apply to name servers of `org` anymore, since following disclosure of our attacks they were configured to stop serving name server records in tandem with DNSKEY records in the same response. Yet, as we show, attacker is not restricted to using only legitimate DNS responses, and can facilitate maliciously created domains in its attacks. Specifically, an attacker can register a malicious subdomain and exploit *referral* responses, to it, in its attacks. The *referral* response is sent from the parent domain, and therefore can contain NS or A records of the parent domain or other children of the parent domain. In our experimental evaluation we registered a domain called `one-domain-to-rule-them-all.org`, which was specially-crafted so that the *referral* from a parent domain `org` is sufficiently long and is fragmented. The attack is similar to the procedure poisoning the *answer* response type; see details of the cache poisoning attack exploiting a *referral* response from `org` in a technical report [9].

III. Defragmentation-Cache Poisoning

In order to ensure that the first authentic fragment is reassembled with a spoofed second fragment, the attacker must perform a defragmentation cache poisoning. Defragmentation cache poisoning is *caching a spoofed second*[4] *fragment*, so that when the first fragment, of the response, arrives to the resolver, the IP protocol will merge the two fragments (and pass the resulting packet to UDP).

In order for IP to merge the spoofed second fragment with the legitimate first fragment, the two fragments must match in four parameters: source and destination IP addresses, transport protocol and the *IP identifier (IP-ID)* field. In IPv4, the IP-ID is a 16 bit field[5] selected by the sender of the IP packet.

In our setting, the attacker knows the IP addresses, and the transport protocol is UDP. Hence, the only parameter, which the attacker may not know, is the value of the IP-ID. A naive (brute-force) strategy is to try all possible IP-ID values, by sending multiple spoofed second fragments, each containing a different IP-ID value. However, the number of spoofed fragments that the defragmentation caches can store is limited and operating systems often restrict the number of cached fragments per each (source, destination, protocol) triple. Typical defragmentation cache size allows several thousands of fragments. Recent Linux kernel versions impose a default value of 64 fragments (enforced via *ipfrag_max_dist* parameter; see [10]).

The attacker should cache almost this number of spoofed (second) fragments. We use B to denote the number of spoofed second fragments sent by the attacker, i.e., B is typically just a bit smaller than 64, e.g., $B = 62$.

Per-Dest	Global	Mixed	Zero	Random and other
57%	14%	20%	9%	1%

TABLE I
IP-ID allocation methods for TLD name servers (measured October 2012).

The question is then: what is the probability that the IP-ID, of one of the up to B spoofed second fragments (sent by the attacker), matches the IP-ID of the legitimate (fragmented) response.

The answer depends on the IP-ID assignment method of the name server. We found that servers can be classified according to five main allocation methods: 57% use *globally-sequential IP-ID* (incremented upon sending a packet, to any destination), 14% use *per-destination sequential IP-ID* (incremented upon sending a packet, to a particular destination), 20% use *mixed IP-ID* (two or more sequential sequences 'mixed up', probably due to multiple machines behind load balancer), 9% use *zero IP-ID* (assigned by some systems to all short packets), and 1% use *random/other IP-ID* (selected randomly or via an unidentified process); see Table I.

In our analysis of the match between the IP-ID values, we consider only[6] three IP-ID 'methods': *per-dest, global* and *random/other*. We show strategies which the attacker can employ to predict the IP-ID for each IP-ID assignment method, and analyse success probability and complexity of the attack.

All our attacks are initiated with the transmission of B spoofed second fragments, by the attacker, which are stored at the defragmentation cache of the resolver (for 30 seconds by default).

Consider first the case where the attacker has no information about the expected IP-ID value, i.e., the 'random/other' IP-ID category, and assume that the attacker triggers only a single request, to which a single response is sent. The IP-ID in the response should match one of the B fragments stored in the defragmentation cache; we subsequently show how to extend this to multiple requests, which circumvents the birthday protection and allows to significantly improve the efficiency of the attack. The probability of a match is $p = \frac{B}{2^{16}}$ and for $B = 62$ it is 0.0009. By repeating the attack, the success probability increases; the mean number of attempts to poison is $n = \frac{1}{p} \cong 1057$; for TTL of 5 minutes, this would be 3.5 days.

This average time to poison may be reasonable for many attacks. However, in the following subsections, we show that the success probability can be significantly improved, by: **(1)** taking advantage of the birthday paradox (Section III-A), **(2)** predicting the IP-ID for *sequential* (Section III-C) and *per-destination* (Section III-D) IP-ID allocation methods, and **(3)**

[4]We focus on the common case, where (legitimate) fragments arrive 'in order', i.e., the legitimate second fragment arrives after the first fragment has arrived; the adaption to the general case, where fragments arrive in reversed order is simple.

[5]In IPv6, the IP-ID is 32 bits; we focus on IPv4, since adoption of IPv6 is still limited.

[6]We ignore the *Zero* class, which contains name servers that assign a fixed IP-ID zero to all responses; we believe this is done by some systems when sending 'short' packets, which are unlikely to be fragmented. For simplicity, we classify 'mixed' under 'random/other' category.

circumventing TTL restriction (see technical report [9]).

A. Circumventing Birthday Protection

'Birthday attack' is a well-known optimisation for classical DNS poisoning attacks, where the attacker not only sends many responses (with different guesses for challenge values), but also generates multiple *requests* to the same query, to increase the probability of a match between some request and some response. Currently, most resolvers prevent the 'birthday attack', by deploying *birthday protection*: not sending multiple concurrent requests for the same query.

We show that 'birthday protection' can be circumvented via our fragmentation-based attacks. The idea is that the attacker can send multiple DNS requests, for *different* random subdomains of the victim domain; since these are different domains, birthday protection does not apply. All queries are of exactly the same length; hence, the second fragments in the responses to all of these requests are *exactly identical* to each other. Hence, a fake second fragment can match *any* of the (legitimate) first fragments. The impact is exactly that of the birthday paradox, if the attacker guesses the IP-ID (e.g., when the server uses random IP-ID assignment); the impact can be even higher, when the attacker can predict the IP-ID with some precision (see in following subsections).

For example, consider the maliciously-registered domain technique, instead of making the query to the malicious domain itself, e.g., `one-domain-to-rule-them-all.org`, we make requests to (many) subdomains thereof, e.g., `xyz.one-domain-to-rule-them-all.org`; all requests are of exactly the same length, hence, all responses have identical second fragment. The attacker can predict the value of this second fragment and replace it with a spoofed second fragment (with the same checksum). As a result, the spoofed response-fragments can match *any* of the authentic first fragments, i.e., we obtain the 'improved birthday paradox'.

B. Random IP-ID Allocation

We next analyse the probability for successful defragmentation cache poisoning, assuming random IP-ID allocation, i.e., the name server selects the IP-ID values in each response uniformly. Let n be a number of DNS requests triggered by the attacker. By simple analysis (in [9]), we find that the probability for successful poisoning is:

$$\Pr[\text{success}] = 1 - \left(1 - \frac{B}{2^{16}}\right)^n \quad (1)$$

See graph, based on Eq. (1), in Figure 4. However, very few servers support random IP-ID assignment method, and therefore, our focus is on per-destination incrementing and globally incrementing IP-ID allocations.

C. Globally Incrementing IP-ID Allocation

A significant fraction of name servers use globally-incrementing allocation method. On first sight, predicting the IP-ID for globally-incrementing servers seems easy - attacker

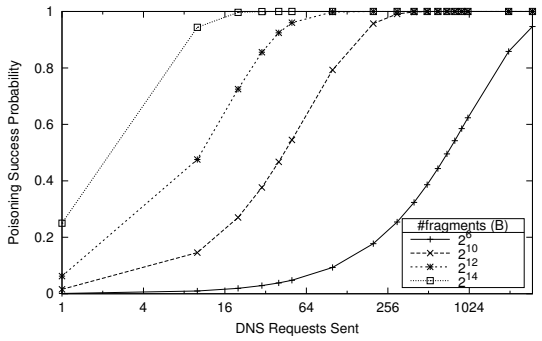

Fig. 4. Poisoning success probability per poisoning attempt, by analysis (Eq. (1)) and by experiments, for $B \in \{64, 1024, 4096, 16384\}$ (number of fake second fragments in cache) and different numbers of DNS requests, for random IP-ID assignment.

can query the name server directly and find out the current IP-ID value (since the same counter is used for sending packets to the attacker and to all other destinations, including the resolver). Indeed, such direct queries will be an important tool

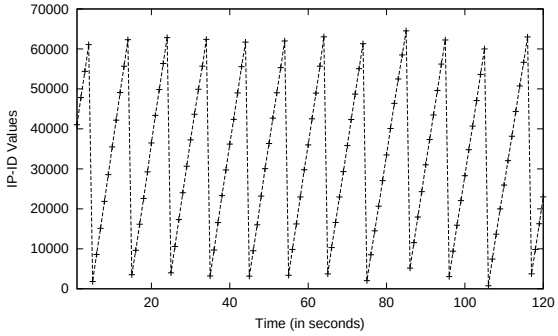

Fig. 5. Progress of IP-ID for the globally-incrementing name server a0.org.affilias-nst.info of org TLD.

in our extrapolation of the IP-ID value; however, the attack is more complex - the IP-ID may considerably change between the query by the attacker and the query by the resolver, since the name server receives queries from other sources too. Therefore, in busy nameservers, that receive queries at a high rate, the IP-ID may grow very rapidly. However, even if the DNS requests' rate to popular name servers is high, it is typically predictable. For example, in Figure 5, we show the measurements we ran on `a0.org.affilias-nst.info`, one of the name servers of `org`, that supports the globally-incrementing IP-ID allocation; notice how rapidly the IP-ID 'grows' across the cyclic 16-bit counter field, yet it can be seen that the increments are predictable.

Indeed, the IP-ID growth rate is rather predictable, by extrapolating from recent rates. The reason for this is that the query rate to name servers is stable, see [11]. We next present a strategy allowing an attacker to estimate the IP-ID value, that will be allocated by the name server to the response sent to the resolver. We then show an experimental evaluation of a

successful DNS cache poisoning attack using the extrapolation strategy above to perform defragmentation cache poisoning. For simplicity, in the analysis we assume a bound δ on the latency on all channels ($0 \leq latency \leq \delta$), instantaneous processing and negligible ('zero') transmission delay. The sampling procedure, illustrated in Figure 6, begins at time t_0^A, with the attacker sending a packet to the name server, to sample the current IP-ID value x. Once the name server receives the packet from the attacker, it sends a response, at time t_1^A, s.t., $t_A^0 \leq t_1^A \leq t_0^A + \delta$, with IP-ID value x. Upon receiving the response, at time $t_2^A \leq t_0^A + 2 \cdot \delta$, the attacker computes y, the estimate for the value of the IP-ID that will have been assigned by the name server to its response to resolver's query. The attacker then sends multiple responses, with IP-ID values in the range $[y - \frac{B}{2}, y + \frac{B}{2}]$; these are accepted at time $t_3^A \leq t_0^A + 3 \cdot \delta$. We denote the time when the resolver sends the request by t_0^R; to minimise the drift in the IP-ID value, between its sampling, at t_1^A, and the value used by the name server in sending the response, at t_1^R, we pick t_0^R as the minimal value such that $t_0^R \geq t_3^A$. Hence, it is best to use $t_0^R = t_0^A + 3 \cdot \delta$. The name server sends the response to the resolver at time $t_1^R \leq t_0^R + \delta = t_0^A + 4 \cdot \delta$.

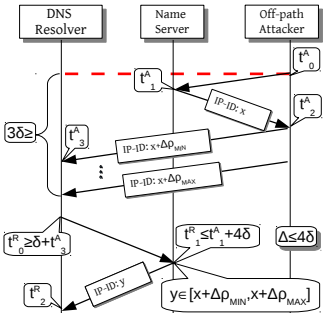

Fig. 6. Procedure of sampling IP-ID value and calculating value assigned to DNS response.

Assume that we have bounds ρ_{MIX} and ρ_{MAX} on the (respective) minimal and maximal rates at which the name server receives DNS requests. We can now bound the range of the IP-ID at time t_1^R, when the response is sent; this is the range of IP-ID values which should be in the fake second fragments which the attacker plants in the resolver's cache.

Claim 3.1: Let x be the IP-ID value sampled by the attacker at time t_1^A. Then the IP-ID used by the server at time t_R^1, to respond to the request from the resolver, is $y \in [x + 3\delta \cdot \rho_{MIN}, x + 4\delta \cdot \rho_{MAX}]$.

Once the attacker calculates the range of candidate IP-ID values that may be assigned to the response sent to resolver, it can send spoofed second fragments. The strategy depends on the relation between the defragmentation cache size B and the number of potential IP-ID values $N = \delta \cdot (4 \cdot \rho_{MAX} - 3 \cdot \rho_{MIN})$. If $B \geq N$, the attacker sends N fragments for all the potential IP-ID values: $y \in [x + 3\delta \cdot \rho_{MIN}, x + 4\delta \cdot \rho_{MAX}]$. Otherwise, if $B < N$, the attacker should send B fragments with the most probable IP-ID values, that are calculated based on typical rates, see Figure 7.

Notice that our analysis above is for a single DNS request, whose response the attacker attempts to match. The attacker can improve the success probability by circumventing the birthday protection and taking advantage of multiple DNS

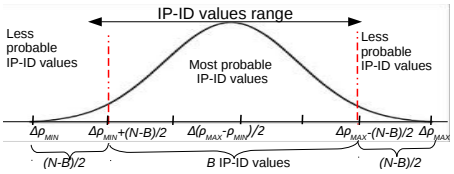

Fig. 7. The most probable IP-ID values are derived from the typical request rates arriving at the name server, i.e., removing too high or low rates from the range of potential IP-ID values.

requests; this requires to incorporate the transmission delay of the DNS requests and responses into the calculation of Δ (above). Experimental evaluation of DNS cache poisoning success for a single and multiple requests is plotted in Figure 8.

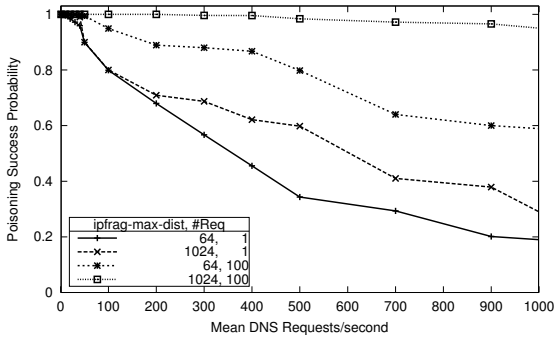

Fig. 8. DNS cache poisoning for globally incrementing IP-ID allocation, for name servers with different DNS request rates. In evaluations we use two defragmentation cache sizes, 64 and 1024 and test for 1 vs. 100 simultaneous DNS requests.

D. Per-Destination Incrementing IP-ID Allocation

In the popular per-destination incrementing IP-ID allocation method, the first IP-ID to some destination IP address is selected at random and subsequent packets sent to the same destination are allocated sequentially incrementing IP-ID values.

The attacker can efficiently hit the correct IP-ID by using a *meet-in-the-middle* strategy, see Figure 9. Let B be the defragmentation cache size. The attacker plants $\frac{2^{16}}{B}$ spoofed second fragments, each fragment i contains IP-ID value of $\{i \cdot \frac{2^{16}}{B}\}_{i=1}^{B}$. For $B = 64$, the sequence consists of IP-ID values multiples of $2^{10} = 1024$, and to reach 100% success the attacker should trigger 1024 simultaneous DNS requests, via the puppet, so that any of first 1024 fragments matches any of the 64 (spoofed) second fragments), see Figure 9; for $B = 1024$ the sequence consists of 1024 multiples of 64, and to attain 100% poisoning probability, the attacker should trigger 64 DNS requests.

Another approach is to wait for periods when only one (or two) name servers will respond. Indeed, the the number of machines behind a load balancer is usually very dynamic, and in periods of low load, many TLDs may use a single machine.

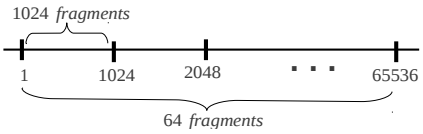

Fig. 9. DNS cache poisoning using a meet-in-the-middle strategy for IP-ID hitting for name servers with per-destination IP-ID allocation and *ipfrag_max_dist* of 64 fragments.

In our attacks, we adopted a third approach: we applied the technique in [3], to cause the resolver to avoid using the 'mixing' name servers, since each domain we attacked also had non-mixing name servers. In Section V-A, we present 'name server pinning' and its applications, including its use for DNS poisoning.

IV. Ensuring Valid Response

DNS responses are validated via: *UDP checksum, DNS response structure, DNS bailiwick* and *priority rules* and *challenge-response mechanisms*; invalid responses are discarded by the resolvers. Notice that since the challenge-response validators are all *always* in the first fragment, they are automatically satisfied. We next discuss each of these checks, and provide techniques allowing attackers to circumvent them.

A. UDP Checksum

The UDP checksum is one's complement sum over the payload. 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. The attacker then ensures that the checksum of the spoofed second fragment is identical to the checksum of the authentic second fragment: (1) by concatenating two bytes after the EDNS record, in case the spoofed record is shorter than the authentic record, which it replaces; or (2) by changing two bytes in another record, to fix the checksum.

B. Response Structure

Resolvers validate that the number of records in each section of the DNS response corresponds to the values specified in a DNS header. In our attack we inject new records *replacing* authentic records, e.g., replace the value of the `name` field of an NS record with a new value (of the name server that the attacker controls), and do not add new records. Therefore, the number of records in a spoofed second fragment is identical to the number of records in the authentic second fragment. Each injected spoofed record should be correctly formatted (according to DNS standard).

C. Bailiwick and Priority Rules

The ability of the attacker to spoof records, is limited in the same way that a regular name server is restricted. Specifically, resolves ignore records received in a response, if they do not satisfy the *bailiwick rules*, whose goal is to prevent a name server of one domain from defining (false) mappings for other domains. The resolvers also apply *priority rules* when deciding whether to cache records in responses, and when overwriting already cached records. For instance, during our experimental evaluation of the attacks we found that Unbound 1.4.18 and Bind 9.8.3 treat NS records, received in *non-existing domain* (NXDOMAIN) responses, differently: while Unbound caches such records, Bind ignores them; specifically, Bind does not cache NS records in responses that contain an SOA record; see full paper, [9], for more details.

V. DNSSEC is not a Panacea

In this section we show that some attacks still apply, even if DNSSEC is fully adopted and correctly validated; in particular, *name server pinning* (Section V-A) and the *non-existing subdomain injection* due to NSEC3 opt-out (Section V-B).

A. Name Server Pinning via Cache Poisoning

In DNS referral responses from the name server of a domain, say org, to a query `sub.org`, the name server provides details of the name servers of `sub.org`, specifically their domain-name (NS record, in authority section) and address (A record, in 'additional' section); these are referred to as the delegation or *glue* records. Even when DNSSEC is used, the glue records are sent *unsigned*, and used by resolvers to resolve future requests for domains within `sub.org`. Furthermore, the glue records are usually cached and used, for queries to that specified domain.

An attacker can use our poisoning attack, described in the previous sections, to poison the *glue records*. When DNSSEC is not (fully) deployed, this can be used to later poison responses to queries sent to the fake nameserver. In contrast, if DNSSEC is deployed, the records sent as answer to queries must be signed, so sending fake glue records does not seem to be useful, see [7]. However, fake glue records can be abused to cause *NS-pinning*, which, in turn, can facilitate DoS and traffic analysis attacks.

The attacker sets very high time-to-live (TTL) in the poisoned glue records. As a result, these records will remain in cache, after the TTL, of any legitimate record for the same domain, expires. When all legitimate records expire from cache, the resolver will use the (only remaining) spoofed glue records, and for as long as the spoofed record is in cache, the resolver will not request other records for the same domain; this holds even if the poisoned glue record causes the resolver to send queries to a completely non-responsive address, providing an easy method for a DoS on resolver's clients.

B. Off-Path Subdomain Injection via NSEC3 Opt-Out

Bau and Mitchell, [7], have shown, that the use of NSEC3 opt-out allows a MitM attacker to create fake (non-existing) sub-domains. As [7] showed, this facilitates XSS, phishing and cookie stealing attacks. Our techniques allow off-path attackers to inject non-existing sub-domains. In spite of the publication of this potential abuse by MitM [7], NSEC3 opt-out is still widely used, and often even recommended, since it improves

performance (esp. as long as DNSSEC is deployed only in small fraction of the domains).

VI. Defenses

The best (long-term) defense against the poisoning attacks we described, is to apply DNSSEC correctly[7]; partial or incorrect use of DNSSEC does not prevent our attacks, e.g., see [12], [13]. DNSSEC foils our (off-path) poisoning attacks, and even defends against MitM adversaries. An exception is the NS-pinning attack; i.e., an attacker may still be able to deny resolution of a victim domain. Preventing NS-pinning seems to require either extensions to DNSSEC, or adoption of additional defense mechanisms.

In the following subsections, we describe shorter-term defense mechanisms and recommendations, to be deployed by resolvers, name-servers and registrars. Our recommended countermeasures could also help prevent the *NS-pinning* and the *sub-domain injection* [7] attacks; a better way to prevent sub-domain poisoning, is to avoid the DNSSEC NSEC3 OPT-OUT option. Since none of these defenses is a 'silver bullet', we present 'three lines of defense'. The *front line* comprises defenses that prevent fragmentation from occurring. The *second line* contains defenses that prevent the defragmentation cache poisoning. In the *last line* there are defenses that prevent DNS cache poisoning, even when the resolver receives a spoofed-defragmented response packet. We do not discuss 'classical' firewall-based defenses, such as detecting the attack and blocking suspect packets, and focus on attack-specific mechanisms.

A. Preventing Fragmentation

The poisoning attacks we presented are based on fragmentation, which normally is a very rare event in the current Internet. We therefore recommend to prevent or minimise the use of fragmentation, by refraining from sending long responses.

Zone operators and registrars should impose a maximal limit on the length of responses. Our most effective poisoning attack, exploits registration of domains with specially-crafted records, causing fragmentation of the referral response from the parent domain. Registrars and TLD name servers should consider placing careful restrictions on the registered records to prevent such fragmentation. Since registrars may not have any direct incentive in deploying this recommendation, guidance may be required.

Furthermore, it is difficult to impose a limit that is sufficiently low to prevent fragmentation. In particular, DNSSEC records are long, and we expect (and hope) they will become much more common in the future - and probably, with more and longer keys, resulting in much longer responses.

Resolvers should not ask for DNSSEC records, unless they strictly apply DNSSEC validation. Recently, [1] found that the *vast majority* of resolvers, that signal support of DNSSEC, do perform strict validation, i.e., *do not block unsigned or mal-signed responses*; this must change immediately. In fact, testing services should alert administrators and clients using such vulnerable setups. The resolvers can also prevent fragmentation by setting the buffer of the EDNS record to some low value, e.g., below 1500 bytes. As a result, when longer responses are returned, the name server sets the TC bit on, signaling to resolver to request the records over TCP.

Name servers can also prevent fragmentation, by using TCP to send long responses; indeed, few name servers already do this, apparently to avoid interoperability problems, e.g., with firewalls that block fragments. However, the use of TCP by name servers is known to be problematic, due to its significantly higher storage, communication and processing costs. Furthermore, the referral clogging attack, see [9], is much worse when name servers use TCP.

B. Preventing Defragmentation Cache Poisoning

Another line of defense is to prevent defragmentation cache poisoning. This may be accomplished by placing stronger restrictions on the defragmentation buffer, such as: (1) reducing the maximal number of cached fragments per each sender and protocol (typically around 64 to 100), (2) reducing the maximal time that a fragment is cached (typically 30 seconds), (3) when receiving fragments out of order, i.e., second and then first, delay the defragmentation process and in case another matching fragment arrives, and use the second fragment that arrives in order, i.e., after the first.

Adoption of such changes to the basic IP defragmentation process, requires careful validation and experimentation.

In this work, due to low support of IPv6 within the DNS servers, we focused on IPv4. Notice that the IP-ID field in IPv6 was modified from 16 bits to 32 bits. However, IPv6 specifications explicitly recommends the use of sequential IP-ID values; these may yet be predictable, using the techniques we presented.

C. Preventing DNS-poisoning via Spoofed 2nd Fragments

The 'last line of defense', to prevent DNS-poisoning includes recommendations both for resolvers and for name servers. These defenses are critical, especially to foil the *birthday protection circumvention*, see Section III-A.

Resolver - Random (length) prefix defense. Some DNS resolvers, e.g., Google Public DNS, were patched to attach a *random prefix* to queries, when the (expected) response is a referral, e.g., a query to the root or to a top-level domain. However, our fragmentation-based poisoning attacks circumvent this defense, since the random prefix is returned in the first fragment (together with all of the other challenges); the attacker can still change the second fragment. Note that while prepending the random prefix causes a change to the UDP checksum (of the entire packet), this makes no impact on the attack, since the attacker anyway makes sure to retain the checksum of the second fragment.

A variant on this technique can also help against fragmentation poisoning attacks. Specifically, the resolver can use

[7]By 'correct use of DNSSEC' we mean: **(1)** all domains signed properly from root, **(2)** resolvers ignoring responses not properly signed, and **(3)** avoiding use of NSEC3 opt-out.

prefixes of *random length*; this changes the contents of the second fragment, and hence the attack becomes harder, e.g., it is harder to preserve the checksum. Of course, the impact of the random-length prefix defense is limited, since the range is not that large (few hundred values at most). Still, the impact on the efficiency of the attack is significant. Of course, the random-length prefix defense, like the known random prefix defense, is limited to queries where the response is known to be a referral.

NAMESERVER - RANDOM SUFFIX DEFENSE. Yet another possible defense, for name servers, is to always add a *random record* to the end of any packet over a certain size (i.e., which may be fragmented). A simple type A resource record, containing a random IP address for some fictitious domain name, would suffice. This would prevent the attacker from being able to predict and (correctly) adjust the checksum value, and hence the vast majority of spoofed responses will be dropped (upon detection of incorrect checksum).

VII. CONCLUSIONS AND FUTURE WORK

We showed how an off-path attacker can efficiently exploit fragmented DNS responses to poison DNS caches. Our attacks are effective against standard implementations of the DNS and IP; we confirmed effectiveness against several domains in the Internet, and popular (standard) resolvers (Unbound 1.4.18 and Bind 9.8.3).

Most DNS responses are short, and hence not fragmented. Ironically, one reason for long responses is the use of DNSSEC; we also show how attackers can cause fragmentation by intentionally registering domain names with long referral responses.

This work follows three other DNS-poisoning attacks: Kaminsky's of 2008 [2], and our derandomisation attacks [3], [5]. Effective countermeasures, requiring 'only' (rather simple and efficient) changes in the resolvers, were proposed against all previously known attacks, see [RFC5452] and [3], [5], [14], however they do not protect against our fragmentation-based attacks. In Section VI, we also proposed defenses against the fragmentation-based attacks presented in this work. Notice that the proposed defenses are not trivial to implement; furthermore, they only reduce the efficiency of the attack, rather than completely prevent it. It therefore remains an important open challenge, to find better easily-deployable defenses against the fragmentation-based attacks.

We believe and hope that the combination of all our recent attacks on the challenge-response DNS defenses, will (finally) provide sufficient incentive to catalyse the adoption of DNSSEC (or other cryptographic mechanism providing strong security for DNS). In particular, notice that it is quite complex to *test* resolvers for defenses against all these attacks, hence, many resolvers may remain vulnerable for years. Indeed, based on the tests we ran on CAIDA traces, [15], we observed that many resolvers were not properly patched even against Kaminsky's attack, and still use predictable or even fixed ports.

On the other hand, we also present variants of the attack, that pose significant threats even to a fully-conforming, strict DNSSEC validation at both resolver and name server. Specifically, we show how an off-path attacker can **(1)** *create fake sub-domains*, demonstrating the off-path feasibility of the attack of [7], and **(2)** cause *name-server pinning*, i.e., cause resolvers to use incorrect name server, denying resolutions of domain names (in the victim zone), demonstrating an effective abuse of the observations of Bernstein [16]. We believe that it is important to defend against these two attacks, e.g., using the name-server-side defense against our attack (see Section VI).

Finally, some of the vulnerabilities and observations which we used in our attacks, may have additional implication and applications, e.g., our ideas of long subdomain generation.

ACKNOWLEDGEMENTS

This research was supported by grant 1354/11 from the Israeli Science Foundation (ISF), and by the Ministry of Science and Technology, Israel. We are also grateful for support for CAIDA's Internet Traces [15] that is provided by the National Science Foundation, the US Department of Homeland Security, and CAIDA Members.

REFERENCES

[1] O. Gudmundsson and S. D. Crocker, "Observing DNSSEC Validation in the Wild," in *SATIN*, March 2011.
[2] D. Kaminsky, "It's the End of the Cache As We Know It," Presentation at Blackhat Briefings, 2008.
[3] A. Herzberg and H. Shulman, "Security of Patched DNS," in *Computer Security - ESORICS 2012 - 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings*, ser. Lecture Notes in Computer Science, S. Foresti, M. Yung, and F. Martinelli, Eds., vol. 7459. Springer, 2012, pp. 271–288. [Online].
[4] ——, "Antidotes for DNS Poisoning by Off-Path Adversaries," in *International Conference on Availability, Reliability and Security (ARES)*, IEEE. IEEE Computer Society, 2012, pp. 262–267.
[5] ——, "Vulnerable Delegation of DNS Resolution," in *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, September, 2013. Proceedings*, ser. Lecture Notes in Computer Science. Springer, 2013.
[6] Y. Gilad and A. Herzberg, "Fragmentation Considered Vulnerable," *ACM Transactions on Information and System Security (TISSEC)*, vol. 15, no. 4, pp. 16:1–16:31, April 2013, a preliminary version appeared in WOOT 2011.
[7] 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].
[8] Federal Executive Branch Internet Domains, "Listing of Federal Agency Internet Domains," February 2012.
[9] A. Herzberg and H. Shulman, "Fragmentation Considered Poisonous," *CoRR*, vol. abs/1205.4011, 2012.
[10] Kernel.org, "Linux Kernel Documentation, 2011"
[11] D. Wessels and M. Fomenkov, "Wow, thats a lot of packets," in *Proceedings of Passive and Active Measurement Workshop (PAM)*, 2003.
[12] A. Herzberg and H. Shulman, "DNSSEC: Interoperability Challenges and Transition Mechanisms," in *International Conference on Availability, Reliability and Security (ARES)*. IEEE, 2013.
[13] ——, "Towards Adoption of DNSSEC: Availability and Security Challenges," Cryptology ePrint Archive, Report 2013/254, 2013.
[14] 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.
[15] CAIDA, "Anonymized Internet Traces 2012 Dataset."
[16] D. J. Bernstein, "Breaking DNSSEC," 3rd USENIX Workshop on Offensive Technologies, August 2009.