Methodology for DNS Cache Poisoning Vulnerability Analysis of DNS64 Implementations

G. Lencse, and Y. Kadobayashi, *Member, IEEE*

***Abstract*—The trustworthy operation of the DNS service is a very important precondition for a secure Internet. As we point it out, DNS cache poisoning could be even more dangerous if it is performed against DNS64 servers. Based on RFC 5452, we give an introduction to the three main components of DNS cache poisoning vulnerability, namely Transaction ID prediction, source port number prediction, and a birthday paradox based attack, which is possible if a DNS or DNS64 server sends out multiple equivalent queries (with identical QNAME, QTYPE, and QCLASS fields) concurrently. We design and implement a methodology and a testbed, which can be used for the systematic testing of DNS or DNS64 implementations, whether they are susceptible to these three vulnerabilities. We perform the tests with the following DNS64 implementations: BIND, PowerDNS, Unbound, TOTD (two versions) and mtd64-ng. As for the testbed, we use three virtual Linux machines executed by a Windows 7 host. As for tools, we use VMware Workstation 12 Player for virtualization, Wireshark and tshark for monitoring, dns64perf for Transaction ID and source port predictability tests, and our currently developed "birthday-test" program for concurrently sent multiple equivalent queries testing. Our methodology can be used for DNS cache poisoning vulnerability analysis of further DNS or DNS64 implementations. A testbed with the same structure may be used for security vulnerability analysis of DNS or DNS64 servers and also NAT64 gateways concerning further threats.**

***Index Terms*—DNS cache poisoning, DNS64, IPv6 transition technologies, NAT64, security, testbed, virtualization.**

I. INTRODUCTION

SEVERAL *IPv6 transition technologies* [1] were developed to support the transition from IPv4 to IPv6, which we are currently faced with, and which is expected to last for several years or even decades. On the one hand, IPv6 transition technologies are important solutions for several different problems, which arise from the incompatibility of IPv4 and IPv6: they can enable communication in various scenarios [2]. However, on the other hand, they also involve a high number of security issues [3]. We have surveyed 26 IPv6 transition technologies, and prioritized them in order to be able to analyze the security vulnerabilities of the most important ones first [2]. DNS64 [4] and stateful NAT [5] were classified as having utmost importance, because they together provide the only solution for a communication scenario, which is very important now because of the exhaustion of the public IPv4 address pool, namely, they enable IPv6-only clients to communicate with IPv4-only servers.

We have also developed a methodology for the identification of potential security issues of different IPv6 transition technologies [6]. Ref. [3] follows the STRIDE approach, which is a general software security solution and it uses the DFD (Data Flow Diagram) model of the systems to facilitate the discovery of various threats. We have found this approach useful and amended the method in [6], where we have also shown that it is necessary to examine the most important implementations of the given IPv6 transition technologies, whether they are susceptible to the various threats that were discovered by using the STRIDE approach. We have pointed out that DNS64 is theoretically susceptible to *DNS cache poisoning* [7], and now the important practical question is, whether its different implementations are actually susceptible to DNS cache poisoning or not.

The purpose of this paper is to develop a simple and efficient methodology for DNS cache poisoning vulnerability analysis of DNS64 implementations. This paper is based on our workshop paper [8], in which we have presented our testbed and our method for Transaction ID prediction attack as well as our results for some specific DNS64 implementations. Now we give a more detailed introduction to cache poisoning including its further two components (source port number prediction, and the birthday paradox based attack), and also design and carry out their testing methods. Besides the DNS64 implementations included in our workshop paper, now we also include Unbound, because it showed much better performance than BIND [9].

The remainder of this paper is organized as follows. In section II, we examine, why DNS cache poisoning is so crucial

Submitted: December 28, 2017. This work was supported by the International Exchange Program of the National Institute of Information and Communications Technology (NICT), Japan.

G. Lencse was with the Laboratory of Cyber Resilience, Nara Institute of Science and Technology, 8916-5 Takayama, Ikoma, Nara, 630-0192 Japan. He is permanently with the Széchenyi István University, Győr, H-9026, Hungary. (e-mail: lencse@sze.hu)

Y. Kadobayashi, is with the Laboratory of Cyber Resilience, Nara Institute of Science and Technology, 8916-5 Takayama, Ikoma, Nara, 630-0192 Japan. (e-mail: youki-k@is.naist.jp).

concerning the DNS64 technology and we also elaborate the attack model of DNS cache poisoning. In section III, we survey the available test tools for DNS cache poisoning analysis and point out that they are not suitable for our purposes. In section IV, we design and implement a testbed for security analysis of DNS64 implementations. In section V, we select the DNS64 implementations to be tested and also present their setup. In sections VI, VII, and VIII, we design and carry out different tests for the possible components of the DNS cache poisoning vulnerability, namely, we test Transaction ID and source port predictability, as well as whether the DNS64 implementations send out multiple equivalent queries simultaneously, which would give an opportunity for an attack based on the birthday paradox. In section IX, we summarize and discuss our results, as well as we make suggestions for the elimination of the uncovered vulnerabilities. Section X concludes our paper.

II. Cache Poisoning Vulnerability of DNS64

The trustworthy operation of the DNS service is a very important precondition for a secure Internet. The ultimate mitigation for DNS cache poisoning, as well as for all other tampering type attacks against DNS, is DNSSEC [10]. However, concerning the cache poisoning vulnerability of DNS64 servers we cannot rely on DNSSEC for two reasons. First of all, its deployment rate is still very low. (As of 2016, it was 1.7% among the Alexa top 1 million web servers [11].) The other reason is DNS64 specific. The task of a DNS64 server is to synthesize an *IPv4-embedded IPv6 address* [12] for the domain names that do not have a AAAA record (IPv6 address). However, this a forged address from the DNSSEC point of view. Thus, a *security aware* and *validating* DNS client has to discard it. The best possible mode of operation is, when a security aware client asks the DNS64 server to perform the validation, see section 3 of [4]. In this case, the client has to trust in the DNS64 server. (And of course, tampering may happen while the packet travels from the DNS64 server to the client.)

Thus for protecting our DNS64 servers from DNS cache poisoning, we need to rely on the guidelines laid down in RFC 5452 [13]. Before addressing them, we need to clarify the attack model, that is, the conditions of a DNS cache poisoning attack. We always consider *blind spoofing*, which means that the attacker may not intercept the DNS requests from the attacked DNS server to the authoritative DNS server. The attacker may send DNS requests (for any domain name) and forged replies to the attacked DNS server.

Now, we first quote the most important conditions from RFC 5452, when a DNS server (called as "resolver" in the text) may accept information from a DNS reply packet, and then interpret them for our situation.

"DNS data is to be accepted by a resolver if and only if:

1. The question section of the reply packet is equivalent to that of a question packet currently waiting for a response.
2. The ID field of the reply packet matches that of the question packet.
3. The response comes from the same network address to which the question was sent.
4. The response comes in on the same network address, including port number, from which the question was sent.

In general, the first response matching these four conditions is accepted." (from section 3 of [13])

Condition 1 gives a very important protection against spoofed answers by setting up a time limit. This *time interval* is equal to the round trip time between the given DNS server and the authoritative DNS server plus the response time of the authoritative DNS server. (The latter may be increased by the attacker by a DoS attack against the authoritative DNS server.) In its calculations, the RFC uses 100ms as a typical value for the length of this time interval. Of course, an attacker may attempt to initiate the opening of this time window at any time by sending a request for an arbitrarily chosen domain name. However, if a domain name is already cached, it is usually protected, until its TTL expires.

Condition 2 significantly hardens the task of the attacker: the attacker has to guess the *Transaction ID* for a successful attack. To support guessing, the attacker may send DNS resolution requests to the DNS server for any domain names, including domain names, the authoritative DNS servers of which is under the control of the attacker, thus the attacker may observe an arbitrarily long sequence of the Transaction IDs generated by the attacked DNS server. Therefore, DNS servers must use hard to predict (cryptographic) random number generators to prevent the attacker from being able to predict the Transaction IDs. Thus, on average, a number of 2^{15} trials are necessary for a successful guess for the 16 bit long Transaction ID (within the given time period of about 100ms).

Condition 3 further hardens the task of the attacker, but not very significantly. There may be a few authoritative DNS servers for a domain, the IP address of which are known for the attacker, and the DNS server may use them in a round robin manner. The attacker needs to spoof exactly the right one. As their number is usually small, this condition contributes only with a small multiplication factor. As for the spoofing itself, there are some countermeasures against source IP address spoofing, such as reverse path checking by routers or firewalls. However, we may not rely on this optional protection: we suppose that it is not switched on, or the attacker is able to send the forged replies from the "right" direction.

Condition 4 has two contributions. The attacked DNS server may have more than one network interfaces (or more than one IP addresses may be assigned to the same interface), but this number is limited, thus it may be only a small factor. The *source port number* can be another significant factor, if the DNS server uses different, hard to predict source port numbers for sending out its every single request. As port numbers from 0 to 1023 cannot be used, the entropy is somewhat less than 16 bits.

We note that NAT (more exactly: NAPT) devices may remove the entropy of the source port numbers, thus DNS servers should never be placed behind NAPT devices unless the NAPT devices are known to comply with RFC 6056 [14],

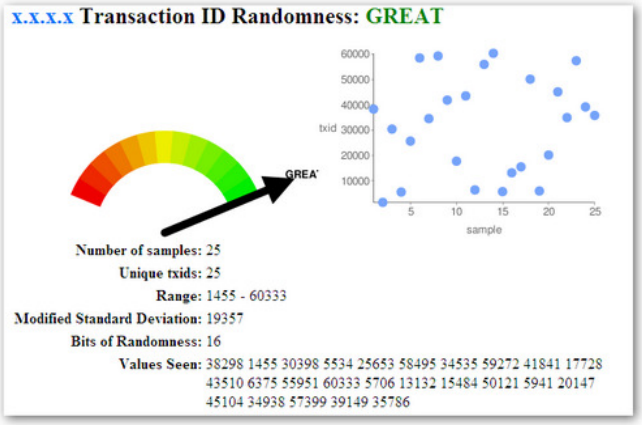

Fig. 1. Sample Transaction ID randomness testing results of the DNS-OARC DNS entropy testing tool. [20]

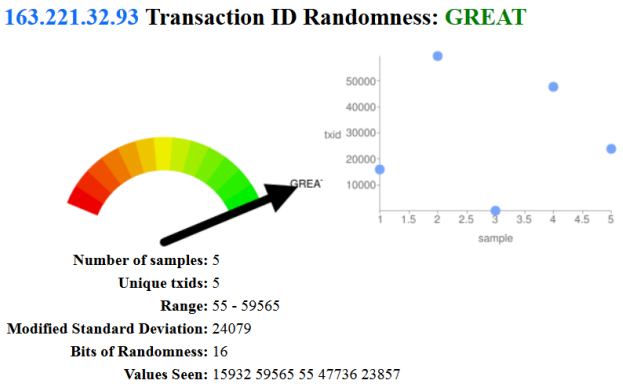

Fig. 2. Our Transaction ID randomness test result produced by the DNS-OARC DNS entropy testing tool.

which requires randomized source port number selection.

RFC 5452 [13] describes another form of attack, which is based on the birthday paradox. If the attacker may achieve that the DNS server sends out *multiple equivalent queries*, that is queries with identical QNAME, QTYPE, and QCLASS fields, *concurrently* (a new query is sent while another one still waits for an answer) then the forged replies of the attacker may match any of them, which significantly eases the attack. For further details, please refer to the CERT vulnerability note [15].

To sum up the essence of the above conditions, we need to check whether the analyzed DNS64 server implementations use hard to predict random numbers for both Transaction IDs and source port numbers and they do not send multiple equivalent queries concurrently.

III. Tools for Cache Poisoning Vulnerability Testing

Although Daniel J. Bernstein already disclosed the vulnerability of the DNS system as well as the possible solution in 1999 [16], and there was a CERT notification about the possibility of the birthday paradox based attacks in 2002 [15], some mainstream DNS servers implementations including BIND did not address the issue properly until the CERT notification in 2008 [17], which was triggered by Dan Kaminsky, who invented a more powerful cache poisoning method. His attack is built upon two ideas: it bypasses the protection of the TTL by using different random names from the attacked domain, and goes one hierarchy level higher: instead of trying to insert a forged "A" record into the cache of the attacked DNS server, it hijacks the whole attacked zone by including the IP address of a DNS server controlled by the attacker as an IP address of a DNS server for the attacked domain into an Authority record of a forged answer for a query for a random name from the attacked zone (to trick the bailiwick rule), see [18] for an in depth and well-illustrated description of the attack.

Then the alert was taken seriously, and patches were prepared for all those major DNS implementations that were still vulnerable. Also vulnerability testing tools were prepared and released.

A contemporary web based Transaction ID and source port randomness tester by DNS-OARC is still available [19]. It is documented and highly suggested by [20]. Although the demonstration screen at the documentation does not seem to be so bad, see Fig. 1, our experience was rather poor. When we tried it out, among others, we received the results shown in Fig. 2. We contend that it is not enough to test only five Transaction IDs. But we do not have an opportunity to tune the tests.

Another web-based testing tool is mentioned in the ICANN presentation of Kim Davies [21], but the tool is no more available at the URL mentioned on slide 33 of the presentation: http://recursive.iana.org/.

And there is another problem with these web-based tools: they require that the DNS server is configured in a live system.

We rather decided to build a *testbed*, that is, an isolated environment, where we can check whether the examined DNS64 implementations indeed have the presumed vulnerabilities by using any kind of tests with any parameters we consider necessary.

IV. Testbed Design and Implementation

A. General Considerations

Although we intended to design a testbed for the security analysis of DNS64 server implementations, we made our considerations with a broader mindset, so that the testbed may also be used for the security analysis of other IPv6 transition technologies, especially NAT64.

In general, the requirements for such a testbed usually include the following:

1. isolated environment, where attacks may be performed
2. ease of use
3. low cost.

A testbed for the security analysis of different IPv6 transition technologies should contain the fundamental basic blocks of the systems in which the given solutions are used. Practically it means that we need a few computers which are interconnected by IPv4 and/or IPv6 network(s). Such systems can be built in

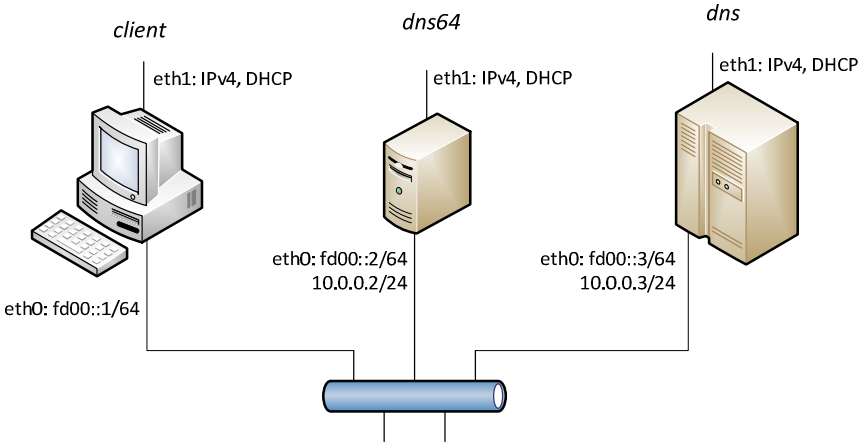

Fig. 3. Topology of the test network.

Table 1. Linux and VMware Network Settings for Virtual Machines.

Virtual machine name	**client**	**dns64**	**dns**
Role	IPv6-only client	DNS64 server	Authoritative DNS server
eth0 Linux settings	IPv6 static: fd00::1/64	IPv6 static: fd00::2/64 IPv4 static 10.0.0.2/24	IPv6 static: fd00::3/64 IPv4 static: 10.0.0.3/24
eth1 Linux settings	IPv4 DHCP	IPv4 DHCP	IPv4 DHCP
eth0 VMware settings	VMnet1	VMnet1	VMnet1
eth1 VMware settings	NAT	NAT	NAT

several ways, including the usage of:

1. server computers
2. desktop or laptop computers
3. single-board computers [22]
4. virtual machines.

We contend that the consecutive solutions result in less cost and higher comfort in use including easy mobility. Our decision was also influenced by the fact that we have been successfully using virtual Linux boxes (executed under Windows 7) for the practical education of DNS64 and NAT64 IPv6 transition technologies at the Budapest University of Technology and Economics since 2015.

As the existing virtual machine images were suitable for our current testing purposes, it was a convenient solution to reuse them. The virtual machine images were prepared by a script called **debian-vm**, written by Dániel Bakai [23]. (This script creates a small, low memory usage, user-defined Debian virtual machine disk image, which can be used in various hypervisors including VMware and KVM.) They contain Debian 8 distributions, which were now updated to Debian version 8.9. They were executed by VMware Workstation 12 Player.

B. Topology of the Test Network

We propose the structure of a simple testbed suitable for the security analysis of the DNS64 and the stateful NAT64 IPv6 transition technologies. Similar testbeds can be built for the security analysis of other IPv6 transition technologies.

The testing of DNS64 or NAT64 requires a network of three hosts. As for DNS64, they are: client, DNS64 server and authoritative DNS server, where the DNS64 server should be interconnected with both the client and the authoritative DNS server. As for NAT64, only the roles are different: client, NAT64 gateway and IPv4-only server; the topology is the same. Thus the same network can be used for the testing of the different implementations of both IPv6 transition technologies, only some software components need to be changed.

As for the attacker, two further hosts could have been added, one for tampering with each connections, but we eliminated them with a trick. First of all, we used a single shared medium to interconnect the three computers, see Fig. 3, thus only one extra device would have been enough. However, as in our current tests we used only wiretapping, it could be done at any of the three computers, thus no further computer was necessary.

C. Implementation of the Test Network

We have implemented the test network shown in Fig. 3 by three virtual machines, each of which had a single CPU core, 128MB of RAM, and (theoretically) 40GB of hard disks, but the starting size of the images were under 1GB. (They were growing during the experiments, but remained under 3GB.) Table 1 shows the Linux and WMware settings used for the virtual machines.

We note that the IP version between the client, which is an IPv6-only client, and the DNS64 server must be 6. There is no restriction for the IP version between the DNS64 server and the DNS server, but when testing NAT64, IPv4 must be used

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fd00::1	fd00::2	DNS	97	Standard query 0x7c4a AAAA piglet.dns64.test
2	0.000707	10.0.0.2	10.0.0.3	DNS	88	Standard query 0xcad0 AAAA piglet.dns64.test OPT
3	0.001094	10.0.0.3	10.0.0.2	DNS	138	Standard query response 0xcad0 AAAA piglet.dns64.test SOA localhost OPT
4	0.001307	10.0.0.2	10.0.0.3	DNS	88	Standard query 0xee9d A piglet.dns64.test OPT
5	0.001423	10.0.0.3	10.0.0.2	DNS	171	Standard query response 0xee9d A piglet.dns64.test A 192.0.2.3 NS localhost A
6	0.001587	fd00::2	fd00::1	DNS	148	Standard query response 0x7c4a AAAA piglet.dns64.test AAAA 2001:db8::c000:203

Fig. 4. Wireshark capture taken during the functional checking of the DNS64 testbed.

between the NAT64 gateway and the IPv4-only server. Although we used IPv4 between the DNS64 server and the authoritative DNS server during our DNS64 vulnerability tests, we set also an IPv6 address at the authoritative DNS server to be able to reach it directly from the client for testing its operability.

We also note that the **eth1** interfaces were not necessary for the tests, we used them for providing the virtual machines with Internet access, which was sometimes necessary, e.g. for installing various packages under Debian Linux. We have separated this communication from the testing communication, which happened always through the **eth0** interfaces of the virtual computers.

D. Setup of a Basic DNS64 Testbed

The purpose of this setup was to check whether the testbed works properly. We have installed BIND9 [24] to both the **dns64** and the **dns** virtual machines.

1) 1. Setup of the DNS64 Server

The **/etc/bind/named.conf.options** file was used to set up the DNS64 function. The relevant settings were:

```
dns64 2001:db8:1::/96 { };
forwarders { 10.0.0.3;};
dnssec-validation no
```

2) 2. Setup of the Authoritative DNS Server

The **/etc/bind/named.conf.local** file was used to set up the authoritative DNS server. The relevant settings were:

```
zone "dns64.test" {
  type master;
  file "/etc/bind/db.dns64.test";
};
```

The content of the **db.dns64.test** file was:

```
$ORIGIN dns64.test.
$TTL 86400
@  IN  SOA  localhost.  root.localhost.  (
            2017090702    ; Serial
                 14400    ; Refresh
                  7200    ; Retry
                 72000    ; Expire
                  3600  ) ; Negative Cache TTL
;
@       IN      NS      localhost.

kanga   IN      A       192.0.2.1
owl     IN      A       192.0.2.2
piglet  IN      A       192.0.2.3
rabbit  IN      A       192.0.2.4
winnie  IN      A       192.0.2.5
```

E. Functional Checking of the Test Network

In this section, we demonstrate the correct operation of the test system, and also introduce the operation of DNS64 servers, which will be important later.

We tested the operation of the testbed by issuing the following command on the **client** computer:

```
host -t AAAA piglet.dns64.test dns64
```

The **host** Linux command was used to request a AAAA record for the **piglet.dns64.test** domain name from the DNS64 server executed by the host named **dns64**.

The DNS messages were captured by **Wireshark** on the **VMnet1** interface using the **port 53** capture filter. The six captured packets are shown in Fig. 4. Now we shall identify the six messages and observe their Transaction IDs, which are used to match the answer with the query. We will experiment with them later.

1. Request for a AAAA record from the client to the DNS64 server with Transaction ID 0x7c4a, generated by the **host** command.
2. Request for a AAAA record from the DNS64 server to the authoritative DNS server with a different Transaction ID, 0xcad0, generated by BIND.
3. An "empty" reply for the AAAA record request sent by the authoritative DNS server to the DNS64 server, and its Transaction ID is the same as that of the corresponding request.
4. Request for an A record from the DNS64 server to the authoritative DNS server with a different Transaction ID, 0xee9d, generated by BIND.
5. A valid reply with an A record sent by the authoritative DNS server to the DNS64 server, and its Transaction ID is the same as that of the corresponding request.
6. The reply of the DNS64 server to the client containing the synthesized *IPv4-embedded IPv6 address* [12] with the same Transaction ID as message 1.

Thus we have found that the testbed worked fine, and it was ready for testing.

V. DNS64 Implementation Selection and Setup

We have laid down our implementations selection guidelines in [2] as follows:

"As for the implementations, we only deal with those that are free software [25] (also called open source [26]) for multiple reasons:

- The licenses of certain vendors (e.g. [27] and [28]) do not allow reverse engineering and sometimes even the publication of benchmarking results is prohibited.
- Free software can be used by anyone for any purposes thus our results can be helpful for anyone.
- Free software is available free of charge for us, too.

Within the category of the free software implementations, we

give further priority to those, which are used widespread and/or are known to be stable and high performance (if such information is available)." [2]

Although several DNS implementations exist, only very few of them can do DNS64, thus finding such DNS64 implementations was not an easy task. We selected the following DNS64 implementations for testing:

1. BIND 9.9.5-9+deb8u12-Debian [24]
2. TOTD 1.5.2 (referred later as OLDTOTD) [29]
3. TOTD 1.5.3 (referred later as NEWTOTD) [30]
4. mtd64-ng 1.1.0 [31]
5. PowerDNS Recursor 3.6.2 [32]
6. Unbound 1.6.0 [33]

Remarks:

- Including BIND9 was a must as it is the de facto industry standard DNS server, therefore, it is very likely wide-spread used for DNS64 purposes, too.
- Some years before we have prepared a patch for TOTD, which resolved some security issues [30], and now we tested its both patched and unpatched versions.
- We also have a new tiny DNS64 proxy called mtd64-ng [31], which is currently developed in an ongoing university project. Although it is not yet ready for deployment, we have also included it.

We have already introduced the DNS64 configuration of BIND in section IV.D.1.

The configuration of both versions of TOTD was done in the **`/usr/local/etc/totd.conf`** file, the relevant settings were:

```
forwarder 10.0.0.3
prefix 2001:db8:404d::
```

The configuration of the mtd64-ng DNS proxy was done in the **`/etc/mtd64-ng.conf`** file, where the relevant settings were:

```
nameserver 10.0.0.3
prefix 2001:db8::/96
num-threads 1
```

The DNS64 configuration of PowerDNS was a bit more complex.

In the **`/etc/powerdns/recursor.conf`** file, we made the following relevant settings:

```
allow from=::/0
forward-zones=dns64perf.test=10.0.0.3
local-address=fd00::2
lua-dns-script=/etc/powerdns/dns64.lua
```

The content of the **`/etc/powerdns/dns64.lua`** file was:

```
function nodata ( remoteip, domain, qtype, records )
if qtype ~= pdns.AAAA then return pdns.PASS, {} end
setvariable()
return "getFakeAAAARecords", domain, "2001:db8::"
end
```

As for Unbound, its 1.4.22 version distributed in Debian 8.9 did not contain the DNS64 module, which was included from its next version, namely 1.5.0. Therefore we upgraded the **`dns64`** host to Debian 9.3 after the execution of all the experiments with the other DNS64 implementations.

As for its configuration, we have added the following lines to the **`/etc/unbound/unbound.conf`** file:

```
access-control: ::/0 allow
module-config: "dns64 iterator"
dns64-prefix: 2001:db8:bd::/96
forward-zone:
  name: dns64perf.test.
  forward-addr: 10.0.0.3
server:
  interface: fd00::2
```

VI. Transaction ID Prediction Vulnerability Testing

A. Details of the Measurements

We extended the configuration of our testbed to be able to examine the Transaction IDs of a high number of messages even if the examined DNS64 implementations use caching.

1) Name Space and Configuration for Testing

To be able to perform a high number of tests, we needed a name space which can be generated systematically. We have found that the name space used in our earlier DNS64 tests [34] would be appropriate. It was the following name space:

10-a-b-c.dns64perf.test, where a, b, c are integers from the [0, 255] interval.

We have used only the 10-0-{0..255}-{0..225} part of it. For generating the zone file, we used the modified version of the zone file generator script called **`gen-zonefile`**, which is shipped together with the **`dns64perf`** program (documented in [34] and available from [35]).

The **`/etc/bind/named.conf.local`** file of the authoritative DNS server was modified as follows:

```
zone "dns64perf.test" {
  type master;
  file "/etc/bind/db.dns64perf.test";
};
```

Thus, BIND used our newly generated zone file after its being restarted.

2) Execution of the Measurements

The measurements were performed by the **`dns64perf`** [34] program, which used sequential Transaction IDs from 0 to 65535. The command line of the test program was:

```
./dns64perf 0 1 1 dns64
```

The first argument specified the "a" parameter described above, the second argument meant that the test program needed to use only one thread, the third one specified the timeout of 1 second, and the last one was the host name of the DNS64 server to be tested.

The traffic was captured by the **`tshark`** program executed by the **`dns64`** host, the memory size of which was raised to 256MB, because 128MB was not enough and the **`tshark`** program exited during the measurement. All the packets from the **`eth0`** interface that matched the **`port 53`** capture filter were saved to a file. The following command line was used:

```
tshark -i eht0 -f "port 53" > imp-full
```

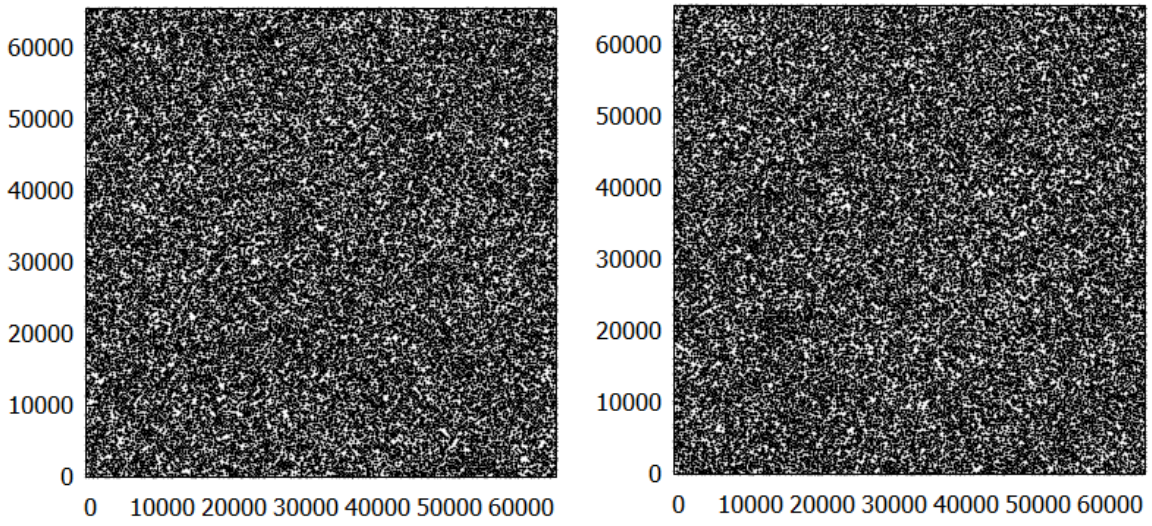

Fig. 5. BIND, Transaction ID input correlation (left) and autocorrelation (right)

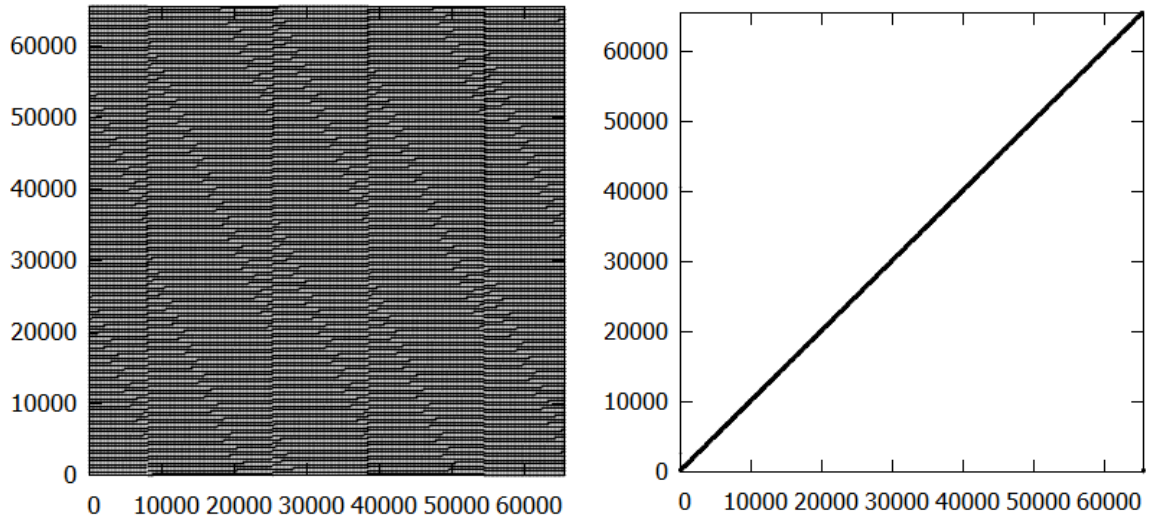

Fig. 6. OLDTOTD, Transaction ID input correlation (left) and autocorrelation (right)

where the *imp* string was replaced by the name of the tested DNS64 implementation.

B. Evaluation Method

Predictability of the Transaction IDs is a hard question. E.g. if pseudorandom numbers are used that were generated by a linear congruential generator (LCG), then they are predictable. There are a high number of methods described for testing randomness both in university lecture notes [36] and research papers [37].

Since our solution of using a testbed ensures us full control of the testing method, and gives us access to the raw results, we have the possibility to use multiple methods for evaluation if needed. We decided to use first a simple, graphical method, which is somewhat similar to that of the earlier mentioned entropy tester of DNS-OARC [19], but we contend that our method is more thorough than that.

We have checked two kinds of correlations using visualization. Before introducing them, let us define some notations first. Let i denote the ordinal number of a message in the message sequence introduced in section IV.E, where i is in [1, 6]. Let j denote the ordinal number of the AAAA record request sent by the **`dns64perf`** program, where j is in [0, 65535]. Let T_{ij} denote the Transaction ID of the i-th message from the six messages used to resolve the j-th query of the **`dns64perf`** program. As the test program uses sequential Transaction IDs from 0, it is sure that: $T_{1j} = T_{6j} = j$.

We use two graphs. An (x, y) plot of the (T_{1j}, T_{2j}) pairs may reveal correlation between the Transaction ID used by the **`dns64perf`** program and the first Transaction ID generated by the DNS64 program. An (x, y) plot of the (T_{2j}, T_{4j}) pairs may

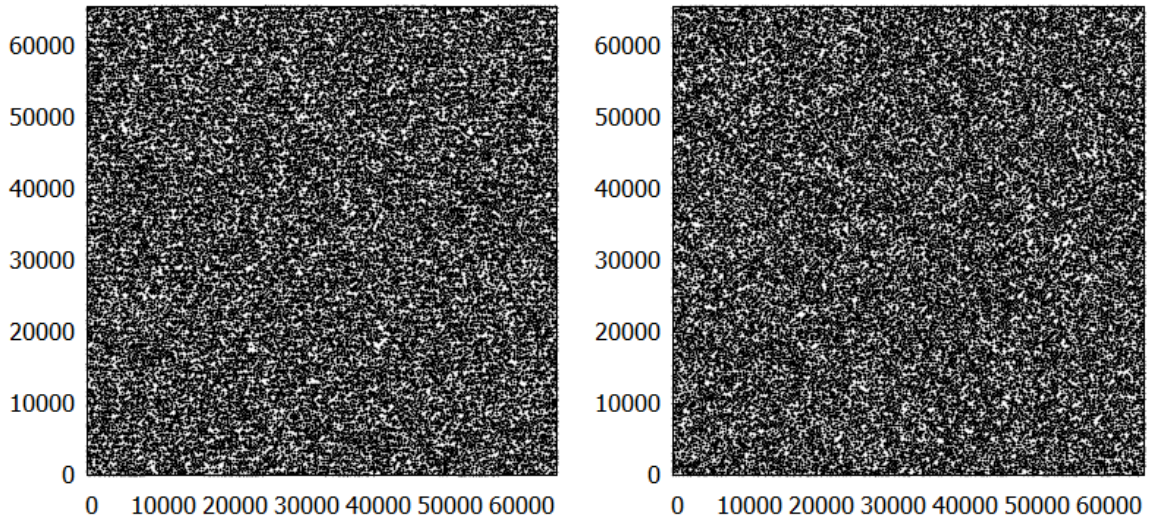

Fig. 7. NEWTOTD, Transaction ID input correlation (left) and autocorrelation (right)

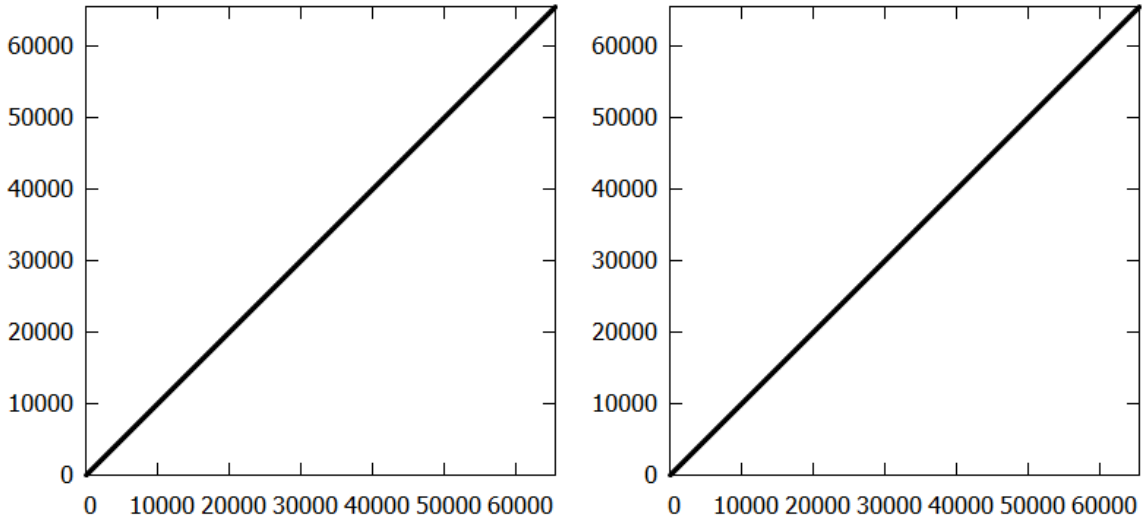

Fig. 8. mtd64-ng, Transaction ID initial correlation (left) and autocorrelation (right)

reveal correlation between the consecutive Transaction IDs generated by the DNS64 program. For simplicity, we will refer to the first one as *input correlation*, and the second one as *autocorrelation*.

We used **`awk`** scripts to extract the appropriate Transaction IDs from the text file output of the **`tshark`** program, and the graphs were prepared by **`gnuplot`**.

C. Measurement Results

Fig. 5 shows the input correlation and the autocorrelation of the Transaction IDs of BIND. They seem to be like noise, thus we can say that no predictability problems were revealed by our simple evaluation method.

The left graph of Fig. 6 shows the input correlation of the Transaction IDs of OLDTOTD. The regular patterns indicate that there is a problem with the predictability of the Transaction IDs. Before giving the explanation, let us have a look at the autocorrelation of the Transaction IDs of OLDTOTD on the right side of Fig. 6. Now, the predictability is even more deliberate. Let us look into the CSV file containing the (T_{1j}, T_{2j}) pairs for input correlation checking:

0, 55745
1, 56257
2, 56769
3, 57281
4, 57793

Whereas the T_{1j} Transaction IDs start from 0 and increase by 1, the T_{2j} Transaction IDs start from a different number and increase by 512. The CSV file containing the (T_{2j}, T_{4j}) pairs for autocorrelation checking can give us further help:

55745, 56001
56257, 56513

Table 2. Source Port Randomness Test Results

DNS64 Implementation	source ports observed in the experiments		
	minimum	maximum	std. dev.
BIND	1024	65535	18635
OLDTOTD	53	53	0
NEWTOTD	53	53	0
mtd64-ng	32768	61000	8136
PowerDNS	1025	65534	18655
Unbound	1024	65535	17467

56769, 57025
57281, 57537
57793, 58049

It is well visible that the consecutive Transaction IDs always increase by 256. And now we give the explanation. As we disclosed it in [30], the old version of TOTD generated sequential numbers as Transaction IDs. The increase of 256 is the result of the facts that the notebook used for testing has an Intel CPU, which uses LSB byte order (least significant byte first), whereas the network byte order is MSB (most significant byte first). The programmer could have been used the standard **`htons()`** function for the conversion, but omitting it is just a feature and not a bug, as Transaction IDs are just identifiers and they do not convey any special meaning. For more information about the bug, which randomly caused an unresponsiveness of the old version of TOTD, and for its correction, please refer to [30], where we have also described the elimination of its vulnerability for Transaction ID prediction attack.

Fig. 7 shows the input correlation and autocorrelation of the Transaction IDs of NEWTOTD. They seem to be like noise, which is exactly what we expected.

Fig. 8 shows the input correlation and autocorrelation of the Transaction IDs of mtd64-ng. They are two completely identical graphs, as the two CSV files were found also completely identical. It is visibly the graph of y=x function, because mtd64-ng reuses the Transaction ID of the received query and sends both of its own queries with the same Transaction ID, which is a serious vulnerability.

As we already mentioned, mtd64-ng is a result of an ongoing university project and it not yet ready to be used in production systems [31].

As for PowerDNS and Unbound, we have also performed the tests and evaluated the results. All their plots looked like the plots of BIND or NEWTOTD, thus we can state that we found no signs of Transaction ID predictability. (We omit the four plots, because we see no point in including further four "random art" images.)

VII. Source Port Number Randomness Testing

The results of the Transaction ID prediction tests could have been used also for port number randomness tests, but **`tshark`** did not include the port numbers in its output. (Its default output contains the same data as the Wireshark screen shown in Fig. 4.) Therefore, we had to make a new series of measurements using a different command line for **`tshark`** as follows:

```
tshark -i eth0 -f "src host 10.0.0.2 and
udp dst port 53" -T fields -e udp.srcport
> imp-srcports
```

The capture filter ensured that only IPv4 packets sent from the DNS64 server program at **`dns64`** (with source IP address 10.0.0.2) to the authoritative DNS server program (listening at port 53 of **`dns`**) be included. The output file contained only the source port numbers. As expected, the result files contained 131072 numbers, except for BIND, in the case of which there were 131073 numbers in the file. We have investigated the case and found that it was so because BIND also sent a query for the IP addresses of the root DNS servers. None of the other implementations did so.

We have summarized our results in Table 2. BIND, PowerDNS and Unbound follow the guidelines of RFC 5452 [13] and choose a source port number randomly from the largest available range of [1024, 65535]. Both versions of TOTD use source port 53 for all outgoing queries. This is trivially predictable. As for mtd64-ng, what can be seen from Table 2, is that the source port number range is [32768, 61000]. What cannot be seen from the table is that the same source ports are used for querying the AAAA record and the A record for the same domain name. This is deliberate from the raw measurement results, we show only the first 6 lines:

48926
48926
41556
41556
42713
42713

And it is also deliberate from the source code [38]. Although, this phenomenon does not mean predictability in the bind spoofing attack model, we recommend the usage of different source ports for the AAAA and A record queries.

It can also be seen from the source code, that mtd64-ng entrusts the source port selection to the operating system. It can be satisfactory, if the operating system complies with RFC 6056 [14], but we contend that is safer if source port randomization is done by the DNS or DNS64 implementation itself.

VIII. Multiple Equivalent Queries Vulnerability Testing

To be able to test, whether the examined DNS64 implementations send multiple equivalent queries concurrently, we had to modify the test program so that it can send multiple queries for the same domain name.

A. Test Program for Checking Birthday Attack Vulnerability

The **`dns64perf`** [35] test program was used as a starting point of our new **`birthday-test`** program. Its arguments are: **`b`**, **`n`**, **`timeout`**, **`IPv6Addr`** and **`port`**. Parameter **`b`** can

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fd00::1	fd00::2	DNS	103	Standard query 0x0300 AAAA 10-0-3-0.dns64perf.test
2	0.000191	fd00::1	fd00::2	DNS	103	Standard query 0x0301 AAAA 10-0-3-0.dns64perf.test
3	0.006545	10.0.0.2	10.0.0.3	DNS	94	Standard query 0xafb2 AAAA 10-0-3-0.dns64perf.test OPT
4	0.008880	10.0.0.3	10.0.0.2	DNS	144	Standard query response 0xafb2 AAAA 10-0-3-0.dns64perf.test SOA localhost OPT
5	0.009963	10.0.0.2	10.0.0.3	DNS	94	Standard query 0x2e30 A 10-0-3-0.dns64perf.test OPT
6	0.010194	10.0.0.3	10.0.0.2	DNS	177	Standard query response 0x2e30 A 10-0-3-0.dns64perf.test A 10.0.3.0 NS localhost A 127.0.0.1 AAAA ::1 OPT
7	0.010528	fd00::2	fd00::1	DNS	154	Standard query response 0x0300 AAAA 10-0-3-0.dns64perf.test AAAA 2001:db8::a00:300 NS localhost
8	0.010627	fd00::2	fd00::1	DNS	154	Standard query response 0x0301 AAAA 10-0-3-0.dns64perf.test AAAA 2001:db8::a00:300 NS localhost

Fig. 9. Wireshark capture taken during the birthday attack vulnerability test of BIND.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fd00::1	fd00::2	DNS	103	Standard query 0x0000 AAAA 10-0-0-0.dns64perf.test
2	0.000238	fd00::1	fd00::2	DNS	103	Standard query 0x0001 AAAA 10-0-0-0.dns64perf.test
3	0.000657	10.0.0.2	10.0.0.3	DNS	83	Standard query 0x7ca9 AAAA 10-0-0-0.dns64perf.test
4	0.000850	10.0.0.2	10.0.0.3	DNS	83	Standard query 0x7da9 AAAA 10-0-0-0.dns64perf.test
5	0.001938	10.0.0.3	10.0.0.2	DNS	133	Standard query response 0x7ca9 AAAA 10-0-0-0.dns64perf.test SOA localhost
6	0.002230	10.0.0.3	10.0.0.2	DNS	133	Standard query response 0x7da9 AAAA 10-0-0-0.dns64perf.test SOA localhost
7	0.002266	10.0.0.2	10.0.0.3	DNS	83	Standard query 0x7ea9 A 10-0-0-0.dns64perf.test
8	0.002475	10.0.0.2	10.0.0.3	DNS	83	Standard query 0x7fa9 A 10-0-0-0.dns64perf.test
9	0.002707	10.0.0.3	10.0.0.2	DNS	166	Standard query response 0x7ea9 A 10-0-0-0.dns64perf.test A 10.0.0.0 NS localhost A 127.0.0.1 AAAA ::1
10	0.002945	10.0.0.3	10.0.0.2	DNS	166	Standard query response 0x7fa9 A 10-0-0-0.dns64perf.test A 10.0.0.0 NS localhost A 127.0.0.1 AAAA ::1
11	0.003122	fd00::2	fd00::1	DNS	216	Standard query response 0x0000 AAAA 10-0-0-0.dns64perf.test AAAA 2001:db8:404d::a00:0 NS localhost A 127.0.0.1 …
12	0.003322	fd00::2	fd00::1	DNS	216	Standard query response 0x0001 AAAA 10-0-0-0.dns64perf.test AAAA 2001:db8:404d::a00:0 NS localhost A 127.0.0.1 …

Fig. 10. Wireshark capture taken during the birthday attack vulnerability test of OLDTOTD.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fd00::1	fd00::2	DNS	103	Standard query 0x0000 AAAA 10-0-0-0.dns64perf.test
2	0.000166	fd00::1	fd00::2	DNS	103	Standard query 0x0001 AAAA 10-0-0-0.dns64perf.test
3	0.001514	10.0.0.2	10.0.0.3	DNS	83	Standard query 0x2326 AAAA 10-0-0-0.dns64perf.test
4	0.001610	10.0.0.2	10.0.0.3	DNS	83	Standard query 0x916e AAAA 10-0-0-0.dns64perf.test
5	0.002527	10.0.0.3	10.0.0.2	DNS	133	Standard query response 0x2326 AAAA 10-0-0-0.dns64perf.test SOA localhost
6	0.002629	10.0.0.3	10.0.0.2	DNS	133	Standard query response 0x916e AAAA 10-0-0-0.dns64perf.test SOA localhost
7	0.002647	10.0.0.2	10.0.0.3	DNS	83	Standard query 0x3360 A 10-0-0-0.dns64perf.test
8	0.002723	10.0.0.2	10.0.0.3	DNS	83	Standard query 0xda62 A 10-0-0-0.dns64perf.test
9	0.002806	10.0.0.3	10.0.0.2	DNS	166	Standard query response 0x3360 A 10-0-0-0.dns64perf.test A 10.0.0.0 NS localhost A 127.0.0.1 AAAA ::1
10	0.002883	10.0.0.3	10.0.0.2	DNS	166	Standard query response 0xda62 A 10-0-0-0.dns64perf.test A 10.0.0.0 NS localhost A 127.0.0.1 AAAA ::1
11	0.003090	fd00::2	fd00::1	DNS	216	Standard query response 0x0000 AAAA 10-0-0-0.dns64perf.test AAAA 2001:db8:404d::a00:0 NS localhost A 127.0.0.1 …
12	0.003154	fd00::2	fd00::1	DNS	216	Standard query response 0x0001 AAAA 10-0-0-0.dns64perf.test AAAA 2001:db8:404d::a00:0 NS localhost A 127.0.0.1 …

Fig. 11. Wireshark capture taken during the birthday attack vulnerability test of NEWTOTD.

be used to perform multiple tests with a different domain name in each test. It is for convenience: when multiple tests are done, the DNS64 server may cache the previously used domain names and it is easier to use a different one for a new test, than restarting the DNS64 server. Parameter **`n`** specifies the number of queries to be sent. The rest of the parameters are to be interpreted as that of the original test program, that is, **`timeout`**, **`IPv6Addr`** and **`port`** specify the timeout value of the receive function, the IPv6 address (or host name) of the DNS64 server to be tested and the port number, where the DNS64 server listens, respectively. (The port number is optional, its default value is 53.)

The program sends **`n`** number of AAAA record requests for the 10-0-b-0.dns64perf.test domain name, where **`n`** and **`b`** should be in the [0, 255] interval. After sending all the queries, it also receives the replies, but it does not use them for any purposes. It receives them only to avoid the annoying "Destination Unreachable (Port Unreachable)" ICMP error messages.

The source code of the test program is available from [39].

B. Measurements and Results

The concurrently sent multiple equivalent queries vulnerability tests were performed in the same testbed as the previous two measurements. Wireshark (executed on the host computer under Windows) was used to monitor the behavior of the DNS64 implementations. We captured the packets on the VMnet1 interface using the **`port 53`** capture filter.

The usual command line was:

```
./birthday-test 0 2 1 dns64
```

(However, sometimes different values were used for **`b`**, e.g. 3 instead of 0 in the case shown in Fig. 9.)

The results produced by BIND can be seen in Fig. 9. Although we sent two queries for the AAAA record of the same domain name, BIND sent only one request to the authoritative DNS server for the AAAA record of the given domain name. (Its next query is for the A record.) Thus BIND is not vulnerable to the "birthday attack".

The results produced by OLDTOTD can be seen in Fig. 10. It sent two equivalent queries for the same resource records (first for AAAA records and then for A records). It can be also observed that the Transaction IDs were incremented by 0x100, as they took the values: 0x7ca9, 0x7da9, 0x7ea9, 0x7fa9.

We note that none of them is a serious problem, because TOTD does not use caching. Thus no cache poisoning attack against TOTD is possible. The attacker can at most achieve that a single client receives forged answer.

The results produced by NEWTOTD can be seen in Fig. 11. The only improvement over OLDTOTD is the proper Transaction ID randomization.

We performed two measurements with mtd64-ng because of the following reasons. As only one CPU core was assigned to the **`dns64`** virtual machine in the testbed, originally we set the

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fd00::1	fd00::2	DNS	103	Standard query 0x0000 AAAA 10-0-0-0.dns64perf.test
2	0.000440	fd00::1	fd00::2	DNS	103	Standard query 0x0001 AAAA 10-0-0-0.dns64perf.test
3	0.000986	10.0.0.2	10.0.0.3	DNS	83	Standard query 0x0000 AAAA 10-0-0-0.dns64perf.test
4	0.001942	10.0.0.3	10.0.0.2	DNS	133	Standard query response 0x0000 AAAA 10-0-0-0.dns64perf.test SOA localhost
5	0.002198	10.0.0.2	10.0.0.3	DNS	83	Standard query 0x0000 A 10-0-0-0.dns64perf.test
6	0.002564	10.0.0.3	10.0.0.2	DNS	166	Standard query response 0x0000 A 10-0-0-0.dns64perf.test A 10.0.0.0 NS localhost A 127.0.0.1 AAAA ::1
7	0.002801	fd00::2	fd00::1	DNS	198	Standard query response 0x0000 AAAA 10-0-0-0.dns64perf.test AAAA 2001:db8::a00:0 NS localhost A 127.0.0.1 AAAA …
8	0.003027	10.0.0.2	10.0.0.3	DNS	83	Standard query 0x0001 AAAA 10-0-0-0.dns64perf.test
9	0.003434	10.0.0.3	10.0.0.2	DNS	133	Standard query response 0x0001 AAAA 10-0-0-0.dns64perf.test SOA localhost
10	0.003641	10.0.0.2	10.0.0.3	DNS	83	Standard query 0x0001 A 10-0-0-0.dns64perf.test
11	0.003948	10.0.0.3	10.0.0.2	DNS	166	Standard query response 0x0001 A 10-0-0-0.dns64perf.test A 10.0.0.0 NS localhost A 127.0.0.1 AAAA ::1
12	0.004142	fd00::2	fd00::1	DNS	198	Standard query response 0x0001 AAAA 10-0-0-0.dns64perf.test AAAA 2001:db8::a00:0 NS localhost A 127.0.0.1 AAAA …

Fig. 12. Wireshark capture taken during the birthday attack vulnerability test of mtd64-ng with 1 working thread.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fd00::1	fd00::2	DNS	103	Standard query 0x0000 AAAA 10-0-0-0.dns64perf.test
2	0.000136	fd00::1	fd00::2	DNS	103	Standard query 0x0001 AAAA 10-0-0-0.dns64perf.test
3	0.000658	10.0.0.2	10.0.0.3	DNS	83	Standard query 0x0000 AAAA 10-0-0-0.dns64perf.test
4	0.000818	10.0.0.2	10.0.0.3	DNS	83	Standard query 0x0001 AAAA 10-0-0-0.dns64perf.test
5	0.001763	10.0.0.3	10.0.0.2	DNS	133	Standard query response 0x0000 AAAA 10-0-0-0.dns64perf.test SOA localhost
6	0.001911	10.0.0.3	10.0.0.2	DNS	133	Standard query response 0x0001 AAAA 10-0-0-0.dns64perf.test SOA localhost
7	0.001918	10.0.0.2	10.0.0.3	DNS	83	Standard query 0x0000 A 10-0-0-0.dns64perf.test
8	0.002025	10.0.0.2	10.0.0.3	DNS	83	Standard query 0x0001 A 10-0-0-0.dns64perf.test
9	0.002171	10.0.0.3	10.0.0.2	DNS	166	Standard query response 0x0000 A 10-0-0-0.dns64perf.test A 10.0.0.0 NS localhost A 127.0.0.1 AAAA ::1
10	0.002327	10.0.0.3	10.0.0.2	DNS	166	Standard query response 0x0001 A 10-0-0-0.dns64perf.test A 10.0.0.0 NS localhost A 127.0.0.1 AAAA ::1
11	0.002544	fd00::2	fd00::1	DNS	198	Standard query response 0x0000 AAAA 10-0-0-0.dns64perf.test AAAA 2001:db8::a00:0 NS localhost A 127.0.0.1 AAAA …
12	0.002656	fd00::2	fd00::1	DNS	198	Standard query response 0x0001 AAAA 10-0-0-0.dns64perf.test AAAA 2001:db8::a00:0 NS localhost A 127.0.0.1 AAAA …

Fig. 13. Wireshark capture taken during the birthday attack vulnerability test of mtd64-ng with 2 working threads.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fd00::1	fd00::2	DNS	103	Standard query 0x0000 AAAA 10-0-0-0.dns64perf.test
2	0.000249	fd00::1	fd00::2	DNS	103	Standard query 0x0001 AAAA 10-0-0-0.dns64perf.test
3	0.001755	10.0.0.2	10.0.0.3	DNS	83	Standard query 0xf6cf AAAA 10-0-0-0.dns64perf.test
4	0.004063	10.0.0.3	10.0.0.2	DNS	133	Standard query response 0xf6cf AAAA 10-0-0-0.dns64perf.test SOA localhost
5	0.004902	10.0.0.2	10.0.0.3	DNS	83	Standard query 0xdf6c A 10-0-0-0.dns64perf.test
6	0.005326	10.0.0.3	10.0.0.2	DNS	166	Standard query response 0xdf6c A 10-0-0-0.dns64perf.test A 10.0.0.0 NS localhost A 127.0.0.1 AAAA ::1
7	0.006448	fd00::2	fd00::1	DNS	131	Standard query response 0x0000 AAAA 10-0-0-0.dns64perf.test AAAA 2001:db8::a00:0
8	0.006803	fd00::2	fd00::1	DNS	131	Standard query response 0x0001 AAAA 10-0-0-0.dns64perf.test AAAA 2001:db8::a00:0

Fig. 14. Wireshark capture taken during the birthday attack vulnerability test of PowerDNS.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fd00::1	fd00::2	DNS	103	Standard query 0x0000 AAAA 10-0-0-0.dns64perf.test
2	0.000399	fd00::1	fd00::2	DNS	103	Standard query 0x0001 AAAA 10-0-0-0.dns64perf.test
3	0.003776	10.0.0.2	10.0.0.3	DNS	94	Standard query 0xd58c AAAA 10-0-0-0.dns64perf.test OPT
4	0.006833	10.0.0.3	10.0.0.2	DNS	144	Standard query response 0xd58c AAAA 10-0-0-0.dns64perf.test SOA localhost OPT
5	0.008033	10.0.0.2	10.0.0.3	DNS	94	Standard query 0x5df0 A 10-0-0-0.dns64perf.test OPT
6	0.008523	10.0.0.3	10.0.0.2	DNS	177	Standard query response 0x5df0 A 10-0-0-0.dns64perf.test A 10.0.0.0 NS localhost A 127.0.0.1 AAAA ::1 …
7	0.009490	fd00::2	fd00::1	DNS	154	Standard query response 0x0001 AAAA 10-0-0-0.dns64perf.test AAAA 2001:db8:bd::a00:0 NS localhost
8	0.009634	fd00::2	fd00::1	DNS	154	Standard query response 0x0000 AAAA 10-0-0-0.dns64perf.test AAAA 2001:db8:bd::a00:0 NS localhost

Fig. 15. Wireshark capture taken during the birthday attack vulnerability test of Unbound.

number of working threads of mtd64-ng to 1. Due to this setting, mtd64-ng serialized the processing of the requests from our test program, as shown in Fig. 12. However, the DNS64 server of a large network with a high number of users should use multiple threads, therefore we executed the test also with two threads. The results in Fig. 13 reveal that mtd64-ng sends separate AAAA and A record requests for each client request. Although mtd64-ng currently does not support caching, thus it is not a serious vulnerability, the problem must be addressed later, because including caching is among the midterm development plans of mtd64-ng.

The results of PowerDNS and Unbound are shown in Fig. 14 and Fig. 15, respectively. None of them send out multiple equivalent queries, thus they are not vulnerable to birthday attacks.

IX. Summary, Recommendations and Discussion

We have summarized the results of the three kind of measurements in Table 3. As for BIND, PowerDNS, and Unbound, we have not found any vulnerabilities that could lead to cache poisoning. Although TOTD and mtd64-ng have several vulnerabilities that could lead to cache poisoning, they do not implement caching, thus cache poisoning is not possible in their cases.

As the implementation of caching is included in the midterm development plans of mtd64-ng, the protection against all three vulnerabilities must also be included. We recommend the usage of cryptographically secure random number generators [40] for generating Transaction IDs and source port numbers. The elimination of the vulnerability to birthday attacks seems to be a more difficult problem, as now the performance of mtd64-ng benefits from the solution that the requests from the clients are not stored in a central database, but they are distributed to the working threads. However, it will be necessary to centrally keep track of the queries sent by mtd64-ng to the authoritative DNS servers and are currently awaiting for an answer, in order to

Table 3. Summary of the Vulnerability Test Results

DNS64 Implementation	Attack Type			
	Transaction ID Prediction	Source Port Number Prediction	Multiple Equivalent Queries	DNS Cache Poisoning
BIND 9.9.5	no problem found	no problem found	protected	no problem found
TOTD 1.5.2	vulnerable	vulnerable	vulnerable	not applicable
TOTD 1.5.3	protected	vulnerable	vulnerable	not applicable
mtd64-ng 1.1.0	vulnerable	vulnerable	vulnerable	not applicable
PowerDNS 3.6.2	no problem found	no problem found	protected	no problem found
Unbound 1.6.0	no problem found	no problem found	protected	no problem found

eliminate the possibility of sending out multiple equivalent queries concurrently.

We note that all the examined DNS64 implementations are free software [25] (also called open source [26]), thus their source code may also be studied, as we did it in the case of TOTD [30]. The significance of our testing method is that it may also be used for closed source software, or in the cases when the subject of the study also includes the interaction with the random number generator of the operating system.

The very same framework could be used for the analysis of NAT64 gateways.

X. Conclusion

We have shown that DNS cache poisoning may be a crucial vulnerability of DNS64 servers and we have given an introduction to the three main components of DNS cache poisoning vulnerability, namely Transaction ID prediction, source port number prediction, and a birthday paradox based attack, which is possible if a DNS or DNS64 server sends out multiple equivalent queries concurrently.

After surveying the available test tools for DNS cache poisoning vulnerability analysis and pointing out that they are not suitable for our purposes, we have designed a methodology and implemented it in a testbed, which can be used for the systematic testing of DNS or DNS64 implementations, whether they are susceptible to the above mentioned three vulnerabilities.

We have selected BIND, PowerDNS, Unbound two versions of TOTD, and mtd64-ng for testing and also presented their setup. We have carried out their testing concerning the three possible components of the DNS cache poisoning vulnerability. We have pointed out several vulnerabilities in TOTD and mtd64-ng. As they do not currently support caching, thus, cache poisoning is not possible in their cases. As the implementation of caching is included in the midterm development plans of mtd64-ng, we have also given recommendations for the elimination of its uncovered vulnerabilities.

As for BIND, PowerDNS, and Unbound, we have not found any vulnerabilities that could lead to cache poisoning.

References

[1] E. Nordmark, R. Gilligan, “Basic transition mechanisms for IPv6 hosts and routers”, IETF RFC 4213, October 2005. DOI: 10.17487/rfc4213

[2] G. Lencse, Y. Kadobayashi, “Survey of IPv6 transition technologies for security analysis”, IEICE Technical Committee on Internet Architecture (IA) Workshop, Tokyo Japan, Aug. 28, 2017, *IEICE Tech. Rep.* vol. 117, no. 187, pp. 19–24.

[3] M. Georgescu, H. Hazeyama, T. Okuda, Y. Kadobayashi, and S. Yamaguchi, “The STRIDE towards IPv6: A comprehensive threat model for IPv6 transition technologies”, *Proc. 2nd International Conference on Information Systems Security and Privacy*, Rome, Feb. 2016. DOI: 10.13140/RG.2.1.2781.6085

[4] M. Bagnulo, A Sullivan, P. Matthews and I. Beijnum, “DNS64: DNS extensions for network address translation from IPv6 clients to IPv4 servers”, RFC 6147, Apr. 2011. DOI: 10.17487/rfc6147

[5] M. Bagnulo, P. Matthews and I. Beijnum, “Stateful NAT64: Network address and protocol translation from IPv6 clients to IPv4 servers”, IETF RFC 6146, Apr. 2011. DOI: 10.17487/rfc6146

[6] G. Lencse, Y. Kadobayashi, “Methodology for the identification of potential security issues of different IPv6 transition technologies: Threat analysis of DNS64 and stateful NAT64”, *Computers & Security*, vol. 77, no. 1, pp. 397-411, August 1, 2018, DOI: 10.1016/j.cose.2018.04.012

[7] S. Son and V. Shmatikov, “The hitchhiker’s guide to DNS cache poisoning”, in *Proc. Security and Privacy in Communication Networks - 6th International ICST Conference (SecureComm 2010)*, Singapore, Sep. 7–9, 2010, pp. 466–483, DOI: 10.1007/978-3-642-16161-2_27

[8] G. Lencse and Y. Kadobayashi, “Testbed for security analysis of the DNS64 IPv6 transition technology in virtual environment”, IEICE Communications Society Internet Architecture Workshop, Tokyo, Japan, Oct. 13, 2017, *IEICE Tech. Rep.*, vol. 117, no. 239, pp. 19-24.

[9] G. Lencse and Y. Kadobayashi, “Benchmarking DNS64 Implementations: Theory and Practice”, *Computer Communications*, vol. 127, no. 1, pp. 61-74, September 1, 2018, DOI: 10.1016/j.comcom.2018.05.005

[10] R. Arends, R. Austein, M. Larson, D. Massey, S. Rose, “DNS Security Introduction and Requirements”, IETF RFC 4033, Mar. 2005. DOI: 10.17487/rfc4033

[11] J. Linkova, “Let’s talk about IPv6 DNS64 & DNSSEC”, APNIC Blog, 2016, https://blog.apnic.net/2016/06/09/lets-talk-ipv6-dns64-dnssec/

[12] C. Bao, C. Huitema, M. Bagnulo, M Boucadair and X. Li, “IPv6 addressing of IPv4/IPv6 translators”, IETF RFC 6052, Oct. 2010. DOI: 10.17487/rfc6052

[13] A. Hubert, R. van Mook, “Measures for making DNS more resilient against forged answers”, IETF RFC 5452, Jan. 2009. DOI: 10.17487/rfc5452

[14] M. Larsen, F. Gont, “Recommendations for transport-protocol port randomization”, IETF RFC 6056, Jan. 2011. DOI: 10.17487/rfc6056

[15] CERT, "Various DNS service implementations generate multiple simultaneous queries for the same resource record", Vulnerability Note VU#457875, [Online]. Available: https://www.kb.cert.org/vuls/id/457875
[16] D. J. Bernstein, "DNS forgery", [Online]. Available: http://cr.yp.to/djbdns/forgery.html
[17] CERT, "Multiple DNS implementations vulnerable to cache poisoning", Vulnerability Note VU#800113 [Online]. Available: http://www.kb.cert.org/vuls/id/800113
[18] S. Friendl, "An Illustrated Guide to the Kaminsky DNS Vulnerability", *Unixwiz.net*, [Online]. Available: http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html
[19] DNS-OARC, "Test my DNS", web based Transaction ID and source port randomness tester, [Online]. Available: https://www.dns-oarc.net/oarc/services/dnsentropy
[20] InfosecEvents, "More DNS cache poisoning testing tools", [Online]. Available: http://infosecevents.net/2008/07/24/more-dns-cache-poisoning-testing-tools/
[21] Kim, Davies, "DNS cache poisoning vulnerability: Explanation and remedies", ICANN presentation, Viareggio, Italy, Oct. 2008, [Online]. Available: https://www.iana.org/about/presentations/davies-viareggio-entropyvuln-081002.pdf
[22] G. Lencse, S. Répás, "Benchmarking further single board computers for building a mini supercomputer for simulation of telecommunication systems", *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems*, vol. 5. no. 1, 2016, pp. 29–36, DOI: 10.11601/ijates.v5i1.138
[23] D. Bakai, "Debian-VM", [Online]. Available: https://git.sch.bme.hu/bakaid/debian-vm
[24] Internet Systems Consortium, "BIND: Versatile, Classic, Complete Name Server Software", [Online]. Available: https://www.isc.org/downloads/bind
[25] Free Software Foundation, "The free software definition", [Online]. Available: http://www.gnu.org/philosophy/free-sw.en.html
[26] Open Source Initiative, "The open source definition", [Online]. Available: http://opensource.org/docs/osd
[27] Cisco, "End user license agreement", [Online]. Available: http://www.cisco.com/c/en/us/products/end-user-license-agreement.html
[28] Juniper Networks, "End user license agreement", [Online]. Available: http://www.juniper.net/support/eula/
[29] The 6NET Consortium, Ed. M. Dunmore, "An IPv6 Deployment Guide", Sep. 2005. [Online]. Available: http://www.6net.org/book/deployment-guide.pdf
[30] G. Lencse and S. Répás, "Improving the performance and security of the TOTD DNS64 implementation", *Journal of Computer Science and Technology* (*JCS&T*, Argentina), vol. 14, no. 1, Apr. 2014, ISSN: 1666-6038, pp. 9–15. http://journal.info.unlp.edu.ar/journal/
[31] G. Lencse and D. Bakai, "Design, implementation and performance estimation of mtd64-ng a new tiny DNS64 proxy", *Journal of Computing and Information Technology*, vol. 25, no. 2, Jun. 2017, pp. 91–102, DOI:10.20532/cit.2017.1003419
[32] Powerdns.com BV, "PowerDNS", [Online]. Available: http://www.powerdns.com
[33] NLnet Labs, "Unbound", [Online]. Available: http://unbound.net
[34] G. Lencse, "Test program for the performance analysis of DNS64 servers", *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems*, vol. 4, no. 3, 2015, pp 60–65. DOI: 10.11601/ijates.v4i3.121
[35] G. Lencse, "dns64perf source code", http://ipv6.tilb.sze.hu/dns64perf/
[36] R. Jain, "Testing random number generators", Washington University, Saint Louis, lecture notes, 2008, [Online]. Available: https://www.cse.wustl.edu/~jain/cse567-08/ftp/k_27trg.pdf
[37] I. Petrila, V. Manta, F. Ungureanu, "Uniformity and correlation test parameters for random numbers generators", *Proc. 2014 18th International Conference on System Theory, Control and Computing (ICSTCC)*, Sinaia, Romania, Oct. 17–19, 2014, DOI: 10.1109/ICSTCC.2014.6982421
[38] D. Bakai, "mtd64-ng: A lightweight multithreaded C++11 DNS64 server", [Online]. Available: https://github.com/bakaid/mtd64-ng/
[39] G. Lencse, "birthday-test source code", http://ipv6.tilb.sze.hu/DNS-birthday-test/
[40] M. Welschenbach, "Large Random Numbers", In: *Cryptography in C and C++,* 2nd Ed, Apress, Berkeley, CA, 2013. DOI: 10.1007/978-1-4302-5099-9_12

Gábor Lencse received his MSc and PhD in computer science from the Budapest University of Technology and Economics, Budapest, Hungary in 1994 and 2001, respectively.

He has been working full time for the Department of Telecommunications, Széchenyi István University, Győr, Hungary since 1997. Now, he is an Associate Professor. He has been working part time for the Department of Networked Systems and Services, Budapest University of Technology and Economics, Budapest, Hungary as a Senior Research Fellow since 2005. At the time of writing this paper he was a Guest Researcher at the Laboratory for Cyber Resilience, Nara Institute of Science and Technology, Japan, where his research area was the security analysis of IPv6 transition technologies.

Dr. Lencse is a member of IEICE (Institute of Electronics, Information and Communication Engineers, Japan).

Youki Kadobayashi received his Ph.D. degree in computer science from Osaka University, Japan, in 1997.

He is currently a Professor in the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. Since 2013, he has also been working as the Rapporteur of ITU-T Q.4/17 for cybersecurity standardization. His research interests include cybersecurity, web security, and distributed systems.

Dr. Kadobayashi is a member of IEEE Communications society.