Testbed for Security Analysis of the DNS64 IPv6 Transition Technology in Virtual Environment

Gábor LENCSE †, ‡, and Youki KADOBAYASHI †

† Laboratory for Cyber Resilience, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630-0192, JAPAN

‡ Department of Telecommunications, Széchenyi István University
Egyetem tér 1, Győr, H-9026, HUNGARY

E-mail: † {gabor-l,youki-k}@is.naist.jp, ‡ lencse@sze.hu

Abstract In this paper, we demonstrate the feasibility of building a virtual network with several virtual Linux hosts for testing the security issues of the DNS64 IPv6 transition technology. This virtual testbed has so low hardware requirements that it can be operated under Windows 7 on an aging notebook having only 4GB of RAM and 2 CPU cores. We demonstrate the viability of the approach by testing different DNS64 implementations for the susceptibility to Transaction ID prediction attacks. The examined DNS64 implementations are BIND, TOTD, mtd64-ng and PowerDNS. A simple visual method is used for Transaction ID predictability testing. Besides the demonstration, further application possibilities of the testbed are also proposed.

Key words DNS64, IPv6 transition technologies, NAT64, Security, Testbed, Virtualization

1. Introduction

Building an appropriate testbed for the analysis of network protocols may involve significant costs (both regarding hardware purchase and human labor), which can be reduced by virtualization. Although we would rather recommend the application of real hardware in certain cases, e.g. for benchmarking measurements, virtual environments may be satisfactory in several cases, including security analysis.

Since the Internet became a part of our everyday life, network security became a crucial issue. The ongoing transition to IPv6, the new standard version of the Internet Protocol [1], requires the application of several IPv6 transition technologies [2], which involve further security vulnerabilities. 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]. We have also developed a methodology for the identification of potential security issues of different IPv6 transition technologies and demonstrated its viability on the example of DNS64 and NAT64 [3]. We have also pointed out the importance of DNS64 as well as the need for the individual vulnerability analysis of its most important implementations [3].

Our next step is to build a testbed for vulnerability testing of different implementations of the selected IPv6 transition technologies. We had different options, such as building a testbed from several single board computers [4], but we have finally chosen to use virtual environment. Our decision was driven by the cost and labor efficiency of the solution as well as our positive experience with the application of virtual testbeds in education.

In this paper, we build a virtual network with several Linux hosts for testing the security issues of the DNS64 IPv6 transition technology. We demonstrate its feasibility with a network of three virtual Linux boxes executed under Windows on an aging notebook having only 4GB of RAM and 2 CPU cores.

The remainder of this paper is organized as follows. In section 2, we discuss the need for a testbed for the security analysis of IPv6 transition technologies. In Section 3, we introduce the design and implementation of our virtual testbed. In Section 4, we demonstrate the usability of the virtual testbed by applying it for the testing of several DNS64 implementations whether they are susceptible to Transaction ID prediction attacks. Section 5 contains a short discussion and some of our future plans are also mentioned. Section 6 concludes our work.

2. Testbed for Security Analysis

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 different scenarios [2]. However, on the other hand, they also involve a high number of security issues [5]. Their security analysis may be done in various ways, e.g. Ref. [5] recommends the application of the STRIDE approach, which is a general

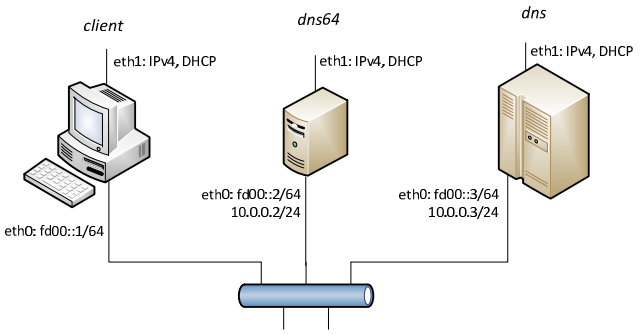


Fig. 1. Topology of the test network

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 [3], 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. For example, if we pointed out that DNS64 [6] is theoretically susceptible to cache poisoning [7], then the important practical question is, whether a given implementation, e.g. BIND [8] is actually susceptible to it or not. In our survey of IPv6 transition technologies [2], we have prioritized them to be able to analyze the security vulnerabilities of the most important ones first. In our before mentioned paper [3], we have developed a methodology for the identification of potential security issues of different IPv6 transition technologies and performed the theoretical analysis of DNS64 [6] and stateful NAT64 [9]. Our next step will be the individual vulnerability analysis of their most important implementations. For this analysis, we need a testbed, that is an isolated environment, where we can check whether the examined implementations indeed have the presumed vulnerabilities. The vulnerabilities can be proven by their successful exploitations. Of course, it is not permitted to perform these attack in live systems, but only in special isolated environments, which were specifically built for this purpose.

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). We propose the structure of a simple testbed for the security analysis of the DNS64 and the stateful NAT64 IPv6 transition technologies in section 3.2. Similar testbeds can be built for the security analysis of other IPv6 transition technologies.

3. Testbed Design and Implementation

3.1. Choice of the Tools

We have been using virtual Linux boxes (executed under Windows 7) for the practical education of DNS64 [6] and NAT64 [9] 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 way to reuse them. The virtual machine images were prepared by a script called **`debian-vm`**[1], written by Dániel Bakai [10]. They contain Debian 8 distributions, which were now updated to Debian version 8.9. They were executed by VMware Workstation 12 Player.

3.2. Topology of the Test Network

The testing of DNS64 [6] or NAT64 [9] 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 server, IPv4-only server; the topology is the same. Thus the same network can be used for the testing of the different implementations of both 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. 1, 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.

3.3. Implementation of the Test Network

We have implemented the test network shown in Fig. 1 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 between the NAT64 server and the IPv4-only server. Although, we used IPv4 between the DNS64 server and the authoritative DNS server during our DNS64 tests, we set also an IPv6 address at the authoritative DNS server to be able to reach it directly from the client for testing purposes.

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

3.4. Setup of a Basic DNS64 Testbed

The purpose of this network was to check whether the testbed works properly. We have installed BIND9 [8] to both the **`dns64`** and the **`dns`** virtual machines.

[1] The script can create a small, low memory usage, user-defined Debian virtual machine disk image, which can be used in various hypervisors including VMware and KVM.

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

3.4.1. 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
```

3.4.2. 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
```

3.5. 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:

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

The **`host`** Linux command was used to ask for an IPv6 address (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. 2. 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 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* [11] with the same Transaction ID as message 1.

We can lay down that the testbed worked fine, and it seems to be usable for testing.

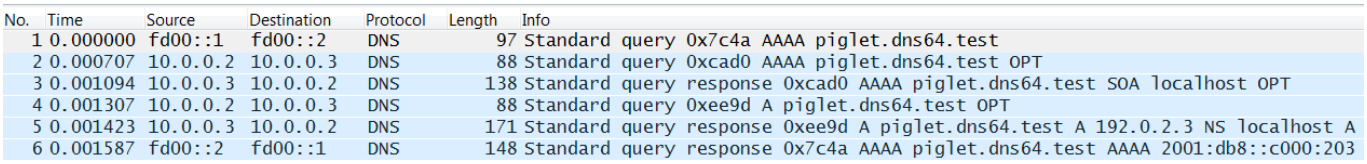
4. Testing Transaction ID Prediction Vulnerability

The usage of predictable Transaction IDs is a serious security issue, because it gives an easy opportunity for cache poisoning attacks [7]. We demonstrated the viability of our simple virtual testbed by examining several DNS64 implementations whether they are susceptible to this threat. To achieve this goal, 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.

4.1. The Details of the Measurements

4.1.1. 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



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. 2. Wireshark capture taken during the functional checkig of the DNS64 testbed.

the name space used in our earlier DNS64 tests [12] 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 [12] and available from [13]).

4.1.2. The Examined DNS64 Implementations and their Setups

We have tested the following DNS64 implementations:

1. BIND 9.9.5-9+deb8u12-Debian [8]
2. TOTD 1.5.2 (referred later as OLDTOTD) [14]
3. TOTD 1.5.3 (referred later as NEWTOTD) [15]
4. PowerDNS Recursor 3.6.2 [16]
5. mtd64-ng 1.1.0 [17]

We have already introduced the DNS64 configuration of BIND in Section 3.4.1.

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 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
```

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

```
nameserver 10.0.0.3
prefix 2001:db8::/96
```

4.1.3. Execution of the Measurements

The measurements were performed by the **`dns64perf`** [12] program, which used sequential Transaction IDs from 0 to 65535.

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.

4.2. 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 [18] and research papers [19].

Since the focus of our investigation is on the testbed itself, and the predictability of the Transaction IDs is used only for demonstration, we used a very simple method, which can reveal only very trivial problems, but it is still suitable for demonstration purposes.

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 3.5, 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 6 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 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 transactions IDs from the text file output of the **`tshark`** program, and the graphs were prepared by **`gnuplot`**.

4.3. Measurement Results

Fig. 3 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. 4 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. 4. Now, the predictability is even more deliberate. Let

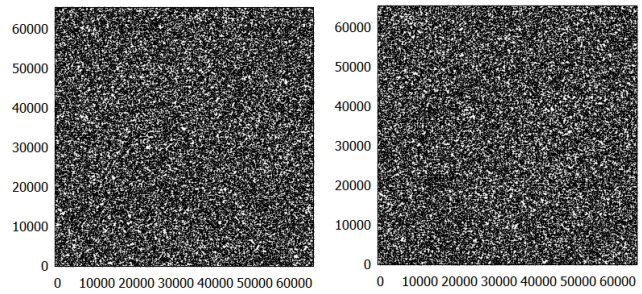


Fig. 3. BIND, Transaction ID input correlation (left) and autocorrelation (r.)

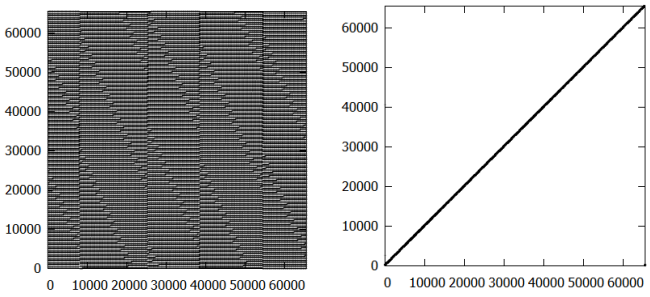


Fig. 4. OLDTOTD, Transaction ID input correlation and autocorrelation

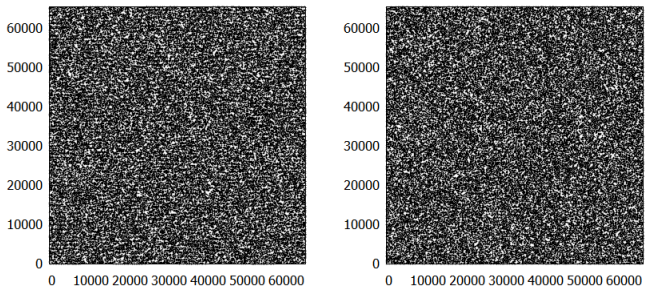


Fig. 5. NEWTOTD, Transaction ID input correlation and autocorrelation

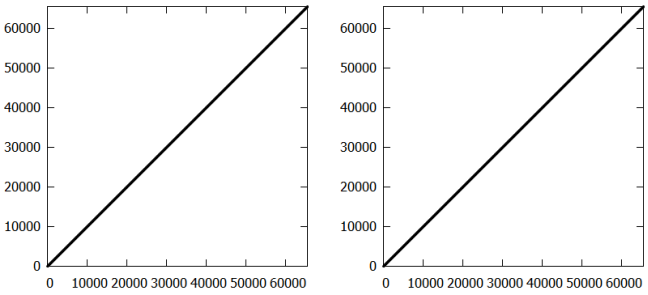


Fig. 6. mtd64-ng, Transaction ID initial correlation and autocorrelation

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
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 [15], the old version of TOTD generated sequential numbers as Transaction IDs. The increase of 256 is the result of the facts that our notebook 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 not a bug in itself. For more information about the bug, which randomly caused an unresponsiveness of the old version of TOTD, and for its correction, please see [15]. We have also described the elimination of the vulnerability for Transaction ID prediction attack in [15].

Fig. 5 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. 6 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 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.

We note that mtd64-ng is a result of an ongoing university project and it not yet ready to be used in production systems [17].

As for PowerDNS, we have also performed the tests and evaluated the results. Both of its 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 two plots, because we see no point in including further two “random art” images.)

5. Discussion and Future Work

Our visual method for detecting the predictability of the Transaction IDs is not more than a simple way to demonstrate the problem. The significance of our results is that our simple virtual network testbed is proved to be suitable for the security analysis of DNS64. In the given case, more advanced methods may be used to identify Transaction ID predictability on the basis of the extracted Transaction IDs.

We note that all the examined DNS64 implementations are free software [20] (also called open source [21]), thus their source code may also be studied, as we did it in the case of TOTD [15]. The significance of the testbed 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.

In our current tests, we used only eves dropping to observe the network traffic. Our further plans include the analysis of the currently tested and further DNS64 implementations, whether they are susceptible to cache poisoning [7]. To achieve this goal, we will need to send packets as well. There are various Linux tool available for that task.

6. Conclusion

Our simple virtual framework proved to be a usable tool in the vulnerability analysis of various DNS64 implementations. Thus, we conclude that our approach provides an easy and cost effective way for the vulnerability testing of the different implementations of several IPv6 transition technologies including DNS64 and NAT64.

Acknowledgment

This work was supported by International Exchange Program of National Institute of Information and Communications Technology (NICT).

References

[1] S. Deering and R. Hinden, “Internet Protocol, version 6 (IPv6) specification”, IETF RFC 8200, Jul. 2017.

[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] G. Lencse, Y. Kadobayashi, “Methodology for the identification of potential security issues of different IPv6 transition technologies”, unpublished, review version will be available from: http://www.hit.bme.hu/~lencse/publications/

[4] 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

[5] 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

[6] 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.

[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] Internet Systems Consortium, “BIND: Versatile, Classic, Complete Name Server Software”, [Online]. Available: https://www.isc.org/downloads/bind

[9] 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.

[10] D. Bakai, “Debian-VM”, https://git.sch.bme.hu/bakaid/debian-vm

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

[12] 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

[13] G. Lencse, “dns64perf source code”, http://ipv6.tilb.sze.hu/dns64perf/

[14] The 6NET Consortium, Ed. M. Dunmore, “An IPv6 Deployment Guide”, Sep. 2005. [Online]. Available: http://www.6net.org/book/deployment-guide.pdf

[15] 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/

[16] Powerdns.com BV, “PowerDNS”, [Online]. Available: http://www.powerdns.com

[17] 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

[18] 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

[19] 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

[20] Free Software Foundation, “The free software definition”, [Online]. Available: http://www.gnu.org/philosophy/free-sw.en.html

[21] Open Source Initiative, “The open source definition”, [Online]. Available: http://opensource.org/docs/osd