Testbed for the Security Analysis of the 464XLAT IPv6 Transition Technology in a Virtual Environment

Ameen Al-Azzawi, Gábor Lencse
Department of Networked Systems and Services
Faculty of Electrical Engineering and Informatics
Budapest University of Technology and Economics
Budapest, Hungary
Email: alazzawi@hit.bme.hu, lencse@hit.bme.hu

***Abstract*—This paper focuses on one of the most prominent IPv6 transition technologies named 464XLAT. The paper aims at building a testbed of this technology and reviews its security analysis. Several virtual machines were used to implement the testbed. The design of the testbed is fully disclosed and its operation is typically demonstrated by an example DoS (Denial of Service) attack scenario which is implemented using the hping3 command. The testbed is suitable to point out the weak spots of the 464XLAT technology and it may also be used for the security analysis of further IPv4aaS (IPv4-as-a-Service) technologies like DS-Lite, MAP-T, MAP-E, and Lw4o6.**

***Keywords*—464XLAT; IPv4aaS; IPv6 transition; security; STRIDE; translation.**

I. INTRODUCTION

As the public IPv4 address pool was actually exhausted at the beginning of the last decade and the deployment of the native IPv6 has been slow, many IPv6 transition technologies have been invented to solve the issue. Several research papers have proposed different kinds of solutions to the problem starting with DNS64 [1] and NAT64 [2], which proved to be not fully successful due to IPv4 literals supporting issues and several IPv4 only applications [3]. As a result, 464XLAT [4] comes to overcome this problem by using its double translation mechanism. Despite 464XLAT being so efficient, this technology has its own security vulnerabilities.

In [5], we have analyzed the architecture of the 464XLAT technology on a theoretical level using its DFD (Data Flow Diagram) according to the STRIDE methodology [6] for its detailed security analysis, and we concluded that 464XLAT showed a lot of attacks possibilities all over its infrastructure. We have found that both sides of 464XLAT (CLAT: customer-side translator & PLAT: provider-side translator) have potential security vulnerabilities. One of the main possible attacks is the DoS (Denial of Service) attack aiming to exhaust the internal connection tracking table of the PLAT. The connection tracking stores tuples of source IP address, destination IP address, and source and destination port numbers in a unique hashing table in order to keep track of the new incoming packets and support Linux kernel with matching, dropping or forwarding any packet[7].

Some other test-beds have focused on performance analysis in terms of CPU, memory utilization, throughput, end-end delay etc. E.g. in [8], a test-bed was built using 4 workstations, and the conducted experiment was an evaluation for Linux operating systems in terms of IPv4-v6 configured Tunnel and 6to4 Tunnel. In [9], a test-bed was built using OPNET simulator to analyze 3 sorts of networks: IPv4 only, IPv6 only and 6to4 tunneling approach. In [10], another test-bed was presented based on OPNET simulator, it covered the analysis of transition mechanism over MPLS (Multi Protocol Label Switching).

Another testbed was developed by Marius Georgescu [11], in which the author used his testbed to measure the latency, throughput, and packet loss using 464XLAT transition technology and other ones such as MAP-E, MAP-T, DS-Lite, etc. Moreover, it showed that 464XLAT had better performance over others in terms of latency.

In this paper, we have taken the task to a more practical level and we built a working 464XLAT system using Debian Linux-based virtual machines to serve as a testbed for the security analysis of the 464XLAT IPv6 transition technology in a virtual environment. It also has a wide range of usage possibilities, because it has the flexibility of having multiple separate environments for each IP version, and the versions themselves can be altered later on to fit the need. The topology is simple and yet it can be very effective for other researchers to perform their own experiments. It is important to mention that the presented solution can be used for testing a high number of vulnerabilities. We demonstrate the usability of the testbed on the example of 464XLAT. However, this testbed can be used in so many other infrastructures, especially, when it comes to IPv4aaS (IPv4-as-a-Service) IPv6 transition technologies such as DS-Lite, MAP-T, MAP-E, and Lw4o6 [12].

For simplicity and due to the low applied traffic and less sophisticated requirements, virtual machines will be sufficient in our case instead of actual Linux-based servers.

However, we will consider deploying the testbed later on (in upcoming research papers) using separate physical Linux-based machines for each topology element if it will be necessary.

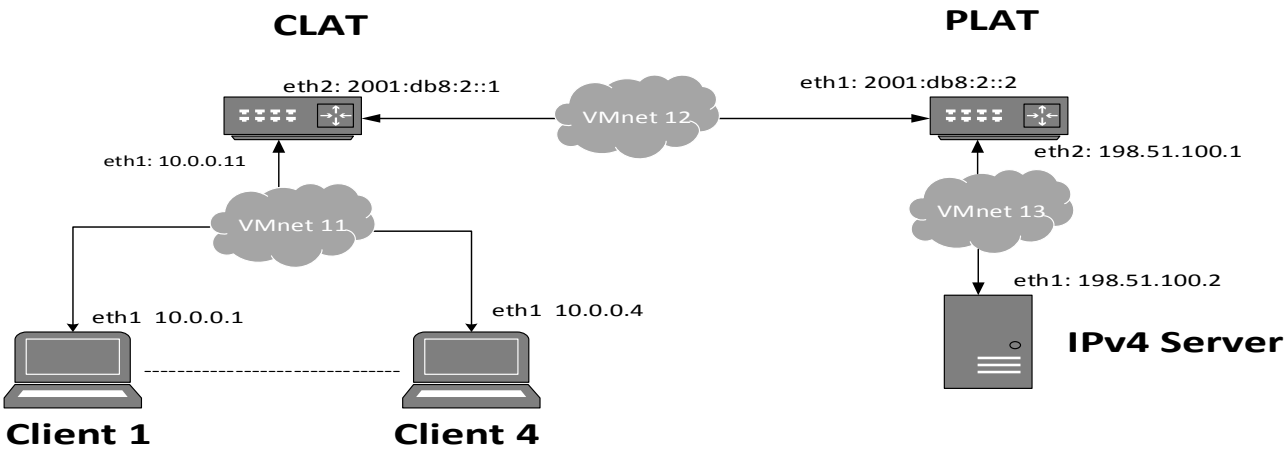

Fig. 1. 464XLAT testbed

This paper opens the door for a scientific challenge in the next upcoming papers, its main focus is the proposed test-bed and we have already taken it one step further by extending the range of the clients in another ongoing research work. We are also planning on using this testbed for testing purposes with other transition technologies.

In Section II, we lay out the structure of our testbed, its topology elements, and the operation of 464XLAT and its structure. Section III presents the most important vulnerabilities of 464XLAT.In section IV, we demonstrate a sample attack by 4 clients. Section V is where we show the results of our attack and analyze them. In section VI, we explain our future focus regarding this research topic. Finally, in section VII, we summarize and conclude the paper.

II. 464XLAT Testbed Design and Implementation

A) Overall Description

There was a high dependency on VMware-based machines in our testbed design. The reason behind that was due to its quick deployment and ease of configuration. The virtual machine images that we made quite good use out of them were based on a script called debian-vm written by Daniel Bakai [13] which is a good option when an engineer is looking for a machine with small size and low memory usage.

This machine has Debian 8.9 distribution installed, and all of those machines were run by VMware workstation 12 Player. (We are aware that the most current version of Debian is 10, but debian-vm supports only version 8.)

The main core of the 464XLAT infrastructure is composed of two translators (CLAT & PLAT).

- *CLAT (client-side translator)* algorithmically translates 1:1 private IPv4 addresses to global IPv6 addresses and vice versa [4].
- *PLAT (provider-side translator)* translates N:1 global IPv6 addresses with the previously set CLAT prefix to public IPv4 addresses and vice versa [4]. PLAT implements a stateful NAT64 gateway as described in RFC 6146 [2].

B) Testbed Topology

The topology of the 464XLAT testbed is shown in Fig. 1. It can be divided into two sides:

- On the left side, there are four clients (10.0.0.1/24 -- 10.0.0.4/24) and the CLAT.
- On the right side, there are the PLAT and the IPv4 server.

The presented topology is simple in its structure, which can be used for multiple purposes, and it is based on several elements:

1) Clients: we could have used only one client, but in this paper, we used 4 of them to demonstrate the different behavior of network resources after running each one of those four clients.
2) Stateless NAT46 gateway (CLAT): this is where IPv4 packets are translated into IPv6 packets and sent over.
3) Stateful NAT64 gateway (PLAT): this is where IPv6 packets are translated back into IPv4 packets in a stateful way (besides their IP addresses, their source port numbers are also translated, when necessary) and finally, the packet is sent over.
4) IPv4 server: the machine where the original IPv4 packet was sent to and where it is received and replied for.

C) Testbed Implementation

In our testbed, each virtual machine has 1GB of RAM, 1 CPU core, and 20 GB of a hard disk.

We have separated the topology into three different virtual networks: VMnet11, VMnet12, and VMnet13.

- VMnet11: the network between the four clients and CLAT eth1. The network is IPv4 only.

TABLE I. LINUX AND VMWARE NETWORK SETTING FOR VIRTUAL MACHINES

VM name	Clients 1-4	CLAT	PLAT	IPv4 Server
eth0 Linux setting	DHCP	DHCP	DHCP	DHCP
eth1 Linux setting	Static IPv4: 10.0.0.1/24 - 10.0.0.4/24	Static IPv4: 10.0.0.11/24	Static IPv6: 2001:db8:2::2/64	Static IPv4: 198.51.100.2/24
eth2 Linux setting	N/A	Static IPv6: 2001:db8:2::1/64	Static IPv4: 198.51.100.1/24	N/A
eth0 VMware setting	NAT	NAT	NAT	NAT
eth1 VMware setting	VMnet11	VMnet11	VMnet12	VMnet13
eth2 VMware setting	N/A	VMnet12	VMnet13	N/A

- VMnet12: the network between the CLAT eth2and PLAT eth1. The network is IPv6 only.
- VMnet13: the network between PLAT eth2 and IPv4 server eth1. The network is IPv4 only.

Table I shows the Linux and VMware settings used for the virtual machines.

Because of its simplicity, the TAYGA [13] user space NAT64 implementation was used.

D) Stateless NAT46 gateway setup

The core configuration of this gateway is to set a specific prefix for the translation mechanism from IPv4 to IPv6 and other supporting configurations.

The "/etc/default/tayga" file was modified as follows:

```
RUN="yes"
CONFIGURE_NAT44="no"
```

Moreover,"/etc/tayga.conf" file is very essential to the translation process, as the below configurations were added:

```
tun-device nat64
ipv4-addr 10.0.0.9
ipv6-addr 2001:db8:2::9
prefix 2001:db8:a::/96
map 10.0.0.1 2001:db8:c::10.0.0.1
map 10.0.0.2 2001:db8:c::10.0.0.2
map 10.0.0.3 2001:db8:c::10.0.0.3
map 10.0.0.4 2001:db8:c::10.0.0.4
```

Finally, a bash script was added to configure special routes for the translated packets to be sent back and forth between NAT46 & NAT64 gateways and to enable IPv4 and IPv6 forwarding:

```
#!/bin/bash
ip route add 198.51.100.0/24 dev nat64
ip route add 2001:db8:c::/96 dev nat64
ip route add 2001:db8:a::/96 via 2001:db8:2::2
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
ip route del 2001:db8:a::/96 dev nat64
```

The last command was used to delete the automatically set route by TAYGA (based on TAYGA prefix).

E) Stateful NAT64 gateway setup

The same procedure with a similar configuration was repeated on NAT64 gateway (PLAT).

The /etc/default/tayga file with the same configuration:

```
RUN="yes"
CONFIGURE_NAT44="no"
```

We did not want TAYGA to configure stateful NAT44, as we set it by ourselves (see below).

The /etc/tayga.conf file with similar configuration also:

```
tun-device nat64
ipv4-addr 198.51.100.9
ipv6-addr 2001:db8:2::9
prefix 2001:db8:a::/96
map 10.0.0.1 2001:db8:c::10.0.0.1
map 10.0.0.2 2001:db8:c::10.0.0.2
map 10.0.0.3 2001:db8:c::10.0.0.3
map 10.0.0.4 2001:db8:c::10.0.0.4
```

The bash script has almost similar configuration here:

```
#!/bin/bash
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
ip route add 10.0.0.0/24 dev nat64
ip route add 2001:db8:c::/96 via 2001:db8:2::1
```

Moreover, NAT64 has to have a route for all outgoing packets (from NAT64 towards NAT46), to dev nat64(the translation interface). In this case, there will be no need for the below command, because TAYGA will set it by default:

```
ip route add 2001:db8:a::/96 dev nat64
```

Another layer of filtering (firewall) was added on PLAT eth2, NAT44 was configured in order to apply the feature of MASQUERADING, which replaces the source IP address of every incoming packet with the public IPv4 address of the eth2 interface (198.51.100.1).

The feature is enabled by applying Netfilter well-known successful firewall as the below command:

```
iptables -t nat -A POSTROUTING -o eth2 -j MASQUERADE
```

From now on, the outgoing packets from PLAT eth2 to IPv4 server will have the source IP address of eth2 (198.51.100.1).

Moreover, on each attacking client, the default route for every single packet heading towards the IPv4 server on the other side of the topology, and of course, the same procedure of enabling IP forwarding applied here as well:

```
#!/bin/bash
ip route add 198.51.100.0/24 via 10.0.0.11
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
```

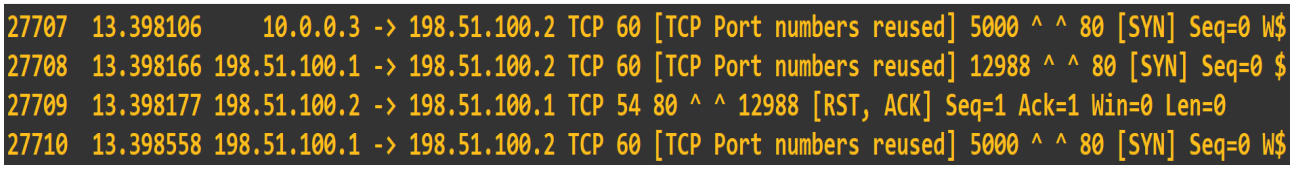

Fig. 2. IPv4 Server eth1 tshark capture

III. 464XLAT VULNERABILITIES

Several vulnerabilities of 464XLAT were uncovered in our previous paper [5], such as spoofing of NAT64 gateway, DoS, FoS, buffer overflow attack, etc. However, we now focus on specific ones against the stateful NAT64 operation of the PLAT device:

- Denial of service attack, when the CPU usage is so high that it does not allow the PLAT to receive and process packets anymore.
- The exhaustion of the source port number pool of the stateful translator.
- Potential attacks against the connection tracking table including its capacity, hashing table size, and applied connection timeout.

IV. SAMPLE ATTACK

The attack is started from the client-side targeting the IPv4 server. The focus of the process will be on the PLAT performance in terms of CPU usage and source port exhaustion.

Client one (10.0.0.1s) was used for the central control. SSH authentications were established between client one and the rest of the network elements. The attack was through hping3 command with specific arguments:

```
hping3 -S -p80 -s5000 -k 198.51.100.2 –iu500
```

-S: TCP Syn attack.
-p: destination port number.
-s: source port number.
-k: to maintain the same port number and avoid its increment.
-iu: to control the number of sent packets per second.

The attack process was as below:

a. Starting the measurements, then waiting for 5 seconds to monitor the performance of the attack.
b. Starting attack with client 1, then waiting for 5 seconds.
c. Starting attack with client 2, then waiting for 5 seconds.
d. Starting attack with client 3, then waiting for 5 seconds.
e. Starting attack with client 4, then waiting for 5 seconds.
f. Ending the measurements, then stopping the four attacks.

After running the script from client1, we monitored the performance of PLAT and IPv4 server simultaneously. As anticipated, the traffic on the PLAT was increasing after adding the first client attack till it reached its maximum traffic, when the fourth client was added.

During the attack, the original destination was the IPv4 server (198.51.100.2). The packets were translated smoothly as planned (i.e. IPv6 packets arrived at PLAT eth1, then got translated back to IPv4 packets and finally their source IP was replaced with PLAT eth2 IP address (198.51.100.1) due to the enabled masquerading feature). Fig.2 shows a snapshot of the tshark result on IPv4 server eth1, where packets arriving from the PLAT are displayed with their masqueraded source IP address.

It is worth mentioning that we have experienced some faulty results as it is obvious with the first row of Fig.2, wherethe source IP address 10.0.0.3 is appearing instead of the public IP address of the eth2 interface. Iptables seem to work perfectly fine except for the masquerading feature of some packets. We attribute this issue to the low resources and capability of the VM itself (PLAT) and not a bug in iptables.

V. RESULTS

Our attack revealed the followings:

- The double translation mechanism worked well.
- At the PLAT, the source port number was forced to be changed.
- PLAT CPU utilization has increased after adding each client.
- Some faulty un-masqueraded packets were noticed due to Netfilter iptables unanticipated behavior.
- The number of faulty translations increased after increasing the packet rate of the attack.

Fig. 3 illustrates the CPU utilization and un-masqueraded (that is, faulty) packets distribution.

It is fairly obvious that percentage of CPU idle is in decreasing pattern. It started with 99%, which means that the CPU has amost zero task to execute, then it gradually falls down as the first client attack comes in place till it ends up with 29% where have four clients deployed. This number could be also decreased again by increasing the packet rate ratio and adding more clients as well.

As for the faulty (un-masqueraded) packets, it kept increasing with 3 clients and it reached around 40 packets per second at the end of experiment.

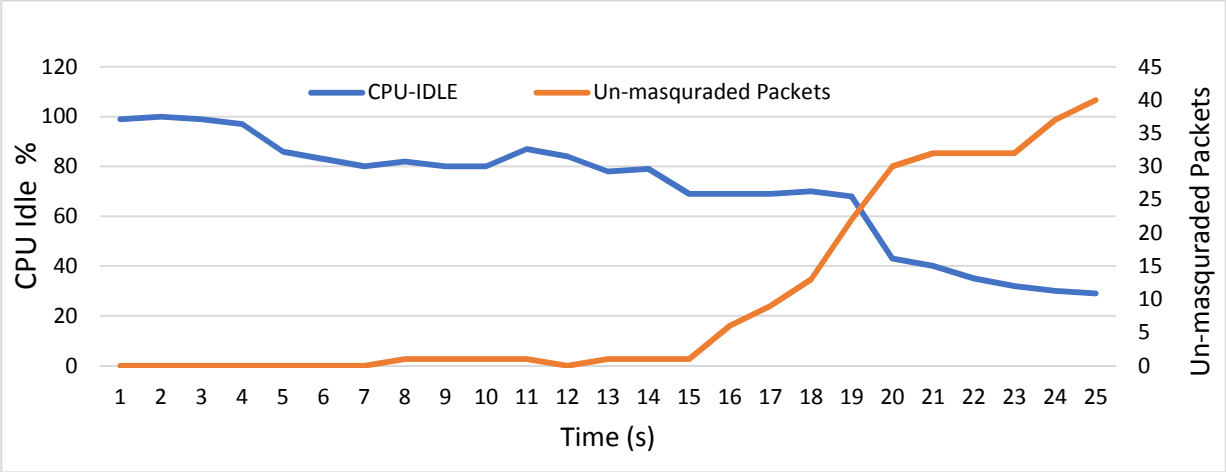

Fig. 3. PLAT CPU utilization & IP un-masqueraded Packets

VI. Plans for Future Research

Our future focus will be on building the following to-attack scenarios:

- Exhausting the pool of PLAT eth2 source ports.
- Exhausting CPU capacity of the PLAT, thus making it unable to receive and process incoming packets.
- Exahusting PLAT connection tracking table.
- Finnaly, find methods to mitigate each attacking scenario.

We also have an ongoing research work that takes this test-bed one step further, by adding more clients and performing DoS attack, etc.

It would require more sophisticated tools and a higher rate of flowing packets to the PLAT. We would also consider other IPv6 transition technologies such as DS-Lite and put them under test using our current testbed (with little possible tweaking).

VII. Conclusion

The proposed testbed proved to be an effective and convenient tool for the security analysis of the 464XLAT IPv6 transition technology and has shown very promising performance in terms of simulating an actual journey of packets through the double translation mechanism. Therefore, the most important takeaway from this testbed is that it could be used with other technologies(not just 464XLAT), too.

Acknowledgment

The authors thank Ahmed Al-hamadani for reviewing and commenting the manuscript.

References

[1] M. Bagnulo, A Sullivan, P. Matthews and I. Beijnum, “DNS64: DNS extensions for network address translation from IPv6 clients to IPv4 servers”, IETF RFC 6147, 2011, doi:10.17487/RFC6147.

[2] M. Bagnulo, P. Matthews and I. Beijnum, “Stateful NAT64: Network address and protocol translation from IPv6 clients to IPv4 servers”, IETF RFC 6146, 2011, doi:10.17487/RFC6146.

[3] S. Répás, T. Hajas, G. Lencse, “Application compatibility of the NAT64 IPv6 transition technology,” in Proc. 37th International Conference on Telecommunications and Signal Processing, Berlin, 2014, pp. 49–55, doi:10.1109/TSP.2015.7296383.

[4] M. Mawatari, M. Kawashima, C. Byrne, “464XLAT: Combination of stateful and stateless translation, IETF RFC 6877 (2013).

[5] A. Al-Azzawi and G. Lencse, "Towards the identification of the possible security issues of the 464XLAT IPv6 transition technology," Proc. 2020 43rd International Conference on Telecommunications and Signal Processing (TSP 2020), Milan, Italy, 2020, pp. 439-444, doi: 10.1109/TSP49548.2020.9163487.

[6] A. Shostack, Threat modeling: Designing for security, 1st Edition, Wiley, Indiana, 2014.

[7] M. Boye, “Netfilter connection tracking and NAT implementation”, in Proc. Seminar on Network Protocols in Operating Systems, Aalto University publication series, 2013, pp. 34-39. https://wiki.aalto.fi/download/attachments/69901948/netfilter-paper.pdf

[8] S. Narayan, P. Shang, and N. Fan, “Network performance evaluation of internet protocols ipv4 and ipv6 on operating systems”, in Proc. Sixth international conference on Wireless and Optical Communications Networks, WOCN’09, pages 242–246, Piscataway, NJ, USA, 2009. IEEE Press

[9] S. Sasanus and K. Kaemarungsi. “Differences in bandwidth requirements of various applications due to ipv6 migration”. in Proc. The International Conference on Information Network 2012, ICOIN ’12, pp. 462–467, Washington, DC, USA, 2012. IEEE Computer Society.

[10] P. Grayeli, S. Sarkani, and T. Mazzuchi. Performance analysis of ipv6 transition mechanisms over mpls. International Journal of Communication Networks and Information Security, vol. 4, no. 2, 2012.

[11] Georgescu, M., Hazeyama, H., Kadobayashi, Y. and Yamaguchi, S., “Empirical analysis of IPv6 transition technologies using the IPv6 Network Evaluation Testbed”, in Proc. Internat. Conf. on Testbeds and Research Infrastructures, 2014, pp. 216-228. Springer.

[12] G. Lencse, J. Palet Martinez, L. Howard, R. Patterson, I. Farrer, “Pros and cons of IPv6 transition technologies for IPv4aaS”, active Internet Draft, 2021. [cited 2021-06-24] [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-v6ops-transition-comparison

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

[14] TAYGA: Simple, no-fuss NAT64 for Linux” http://www.litech.org/tayga