Journal of Engineering and Applied Sciences 13 (Special Issue 9): 6967-6972, 2018
ISSN: 1816-949X

The Influence of Packet Manipulation on the IPv6 Firewall Effectivity

Josef Horalek and Vladimir Sobeslav
Faculty of Informatics and Management, University of Hradec Kralove, Hradec Kralove, Czech Republic

Abstract: The new version of IP protocol is slowly being adopted in computer networks. The transfer to IPv6 world will take some time and the IPv4 and IPv6 protocols need to coexist together. The aim of this study is pro present results from IPv6 packet manipulation efficiency analysis. This study discusses the measurements of various Linux/Unix IPv6 firewalls and their behaviour in specialized networking test which are based on RFC standard. Results were taken by scapy-advanced IP packet manipulator.

Key words: IPv6, RFC, firewall, testing, packet manipulation, scapy, specialized

INTRODUCTION

The new version of IP protocol-IPv6 is slowly but surely expanding across the internet infrastructure. Together with this development, a lot of new security challenges are arising due to the different network architecture of the previous protocol-IPv4. RFC 4864 (Van *et al*., 2007) discusses basic techniques of IPv6 network architecture protection. Firewall as a stateful packet filter can fully replace NAT (Hain, 2000) in protection against unwanted attempts for connection from outside the network and from hiding the network topology.

The transfer to IPv6 will not take place instantly and all at once, for some time the IPv4 and IPv6 protocols will coexist and the problems stemming from the duplicity will arise such as IPv4-mapped IPv6 addresses where the attacker is able to avoid the Ipv4 packet filters and break through the firewall by generating specifically formatted packets. Another problem is tunnelling IPv6 via. Ipv4 networks where for example, Ipv6-over-IPv4 (Steffann *et al*., 2013) using port 41 can be easily enabled or disabled. IPv6 encapsulated in UDP is a problem as passing through UDP must usually be enabled in firewall.

Therefore, a security gap appears in the firewall (Steffann *et al*., 2013). Regarding solely IPv6 problems, RFC 4942 (Davies *et al*., 2007) mentions the issue of the routing headers defined in RFC 246 and their possible abuse as a means for avoiding the firewall or for an amplification attack. Such an attack is a serious threat and RFC 5095 (Afilias *et al*., 2007) recommends all IPv6 nodes stop the support of the type 0 routing header and let this header type be filtered by their firewalls. However, it is not possible to filter all the routing headers. Forged packets in ICMPv6 (Deering and Hinden, 1998) error messages, filtering the traffic with anycast addresses, IPSec, procedures for processing the extended or unknown headers by the firewalls, firewall protection against the misuse of Pad1 and PadN options or link-local, another significant safety issue is the packets with overlapping fragments which can avoid the firewall. These very problems and whether a conflict with RFC would arise were the aim of the tests of firewall behaviour during the use of manipulated packets.

MATERIALS AND METHODS

Firewall tests with scapy: Host firewalls are an important part of multilayer network protection. The safety policies of an individual instance of the host firewall define what traffic is accepted from the internet or from the local network which is the essential advantage of such firewall as a firewall on the perimeter is not principally able to protect the host from the attacks from inside its own network or its segment. In Unix operating systems packets are filtered in the core and the packet filters can be either modules installed on request or they can be included directly in the core. Such modules are accessed with the corresponding userspace programs that allow the creation of filtering rules sets and other configurations. Windows firewall uses Windows Filtering Platform API (Nordmark and Bagnulo, 2009) with whose help it can interact with the packet processing within the framework of the operating system's network code. For the purpose of testing and checking the functionality, the following firewall software implementations will be used.

Corresponding Author: Josef Horalek, Faculty of Informatics and Management, University of Hradec Kralove, Hradec Kralove, Czech Republic

Windows firewall: Windows firewall with advanced security is a stateful firewall capable of inspecting and filtering IPv4 and IPv6 packets (Barre *et al*., 2011). It can block or allow the network traffic based on rules defined by an administrator. In its default settings it blocks the incoming packets unless they are a response to a previous request of the host or unless such traffic is explicitly permitted. Explicit permissions can be defined by a port number, application name, service name and other criteria. It is also, possible to specify rules for the host's outgoing traffic.

Netfilter/IPtables: Netfilter is a framework for filtering embedded into Linux core from core Version 2.4 onwards. From Version 2.6 onwards, the core features full IPv6 stateful firewall. IPtables is a user-space CLI program for set configuration of filtering rules. For the IPv6 packet filter rules configuration, ip6tables is used. IPv4 and IPv6 filtering rules must be set up separately and they are independent of each other. Normally, three sets of rules exist (user can create their own):

- INPUT-for incoming packets
- OUTPUT-for outgoing packets
- FORWARD-for packets forwarded further

The rules in these chains are read in sequence and therefore their order matters. If a packet matches any of the defined rules, the rule forwards the packet towards its destination which decides what will happen next and searching through the rules table is stopped. Standard commands are as follows:

- ACCEPT-packet is allowed to pass
- DROP-packet is discarded
- QUEUE-packet is passed to the user space

Return: Packet stops passing through the chain in which it encountered RETURN. If it was passing through the main chain (e.g., INPUT) default policy of the chain is applied on it. If it was passing through a subordinate chain it returns to the main chain. User-defined chain-packet is forwarded to the user-defined set of rules.

If the corresponding rules are not found for the packet, default policy of the chain is applied. Rules can be created based on the source/destination address or port number, ICMPv6 message type, extension headers etc. As an example of the syntax, let us present an explicit ICMPv6 permission, assuming the traffic is not forwarded further:

- ip6tables-A INPUT-p icmpv6-j ACCEPT
- ip6tables-A OUTPUT-p icmpv6-j ACCEPT

IP firewall: Also, known as IPFW (McCann *et al*., 1996) it is a default (stateful) firewall in FreeBSD operating system. It supports both IPv4 and IPv6. Based on the rules chain definition, packets are processed by the core. The rules are numbered from 1-65535 and they are reviewed in ascending sequence. When a packet of a corresponding rule is found an action chosen by the rule is executed and the reviewing of the rules is terminated. If such a rule is not found, the default action of the rule 65535, i.e., silent packet drop is executed. For easier management, the rules can be assembled into so called rulesets. The basic actions that can be executed with the packets are as follows:

- Allow | accept | pass | permit-these keywords are functionally equivalent and authorize the given packet
- Check-state-verifies the packet against dynamic state table. An action connected with the given dynamic rule is executed if the match is found, otherwise, the process continues with following rule
- Count-increases counter for every packet that is corresponding to the rule. Then the process continues with the subsequent rule
- Deny | drop-silent packet dropping
- Ipfw add 100 allow ipv6-icmp from any to any in
- Ipfw add 110 allow ipv6-icmp from any to any out

Packet filter: Also, known as "pf," packet filter is a default stateful firewall in OpenBSD and OS X (from version 10.7 onwards known as "Lion"). It supports both IPv4 and IPv6. The rules are passed from the top to the bottom and the packet is evaluated against every rule unless "quick" keyword is used which differs from the firewalls mentioned above. Therefore, more valid rules can exist for the packet but only the last action defined adncorresponding rule is executed. It is for example possible to first define the rule of "block all packets" style and to enable specific exceptions only. The basic actions that can be performed with the packets are as follows:

- Pass-passes the packet further to be processed by the core
- Block-performs an action based on block-policy settings by default, silent packet drop is executed

An example of the syntax:

- Pass in inet6 proto ipv6-icmp from any to any
- Pass out inet6 proto ipv6-icmp from any to any

Table 1: Operating systems tested

Operating system	Firewall	Kernel version
MS Windows 7 SP1	Windows firewall	6.1.7001
MS Windows 8.1	Windows firewall	6.3.9600
MS Windows Server 2008 R2	Windows firewall	6.1.7601
MS Windows Server 2012 R2	Windows firewall	6.3.9600
Debian "Wheezy"	Netfilter/IPtables	3.2.0-4-amd64
Ubuntu 14.04 LTS	Netfilter/IPtables	3.13.0-24-generic
Fedora 21 workstation	Netfilter/IPtables-service iptables service replaced by firewall	3.17.4-301-fc21 .x86-64
CentOS	Netfilter/IPtables-service iptables service replaced by firewall	3.10.0-229 .el7.x86-64
FreeBSD	IP firewall	10.0-RELEASE
OpenBSD	Packet filter	5.6
NetBSD	IP filter	6.1.5
Oracle Solaris	IP filter	5.11

IP filter: IP filter is a default firewall distributed with NetBSD and Oracle Solaris. It is a stateful firewall and it supports both IPv4 and IPv6. The passing principle is the same as in packet filter, i.e., from top to bottom and the last corresponding rule is performed . Rules are set in the same way as in packet filter and "pass" and "block" are again basic actions. However , it is for example, also, possible to use "log" keyword to log occurrences of correspondence with the rule etc. An example of the syntax (note the similarity to packet filter):

- Pass in quick proto ipv6-icmp from any to any
- Pass out quick proto ipv6-icmp from any to any

IPv6 firewall functionality and testing efficiency: This chapter presents the results of the performed tests aimed at the efficiency and functionality of IPv6 firewalls and their behaviour during the processing of purpose fully modified packets.

Tested operating systems: All the operating systems (and firewall implementations they are distributed with) used for the purposes of testing on the hosts are summarized in Table 1. All the tested operating systems were fully updated before the testing. The 64 bit operating system variants were used during the testing.

RESULTS AND DISCUSSION

Hardware used during the testing: The tests have been performed using a physical personal computer with the following hardware configuration:

- Processor-AMD Phenom II X4 955 Deneb Quad-Core 3.2 GHz
- Memory-8 GB RAM
- Operating system-Xubuntu 14.04 LTS

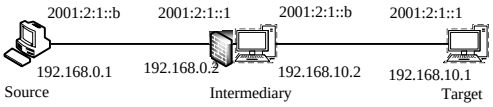

Fig. 1: Test topology for scapy

Virtual machines used as destinations or proxies were virtualized on the same PC using Oracle VM VirtualBox virtualization software of Version 4.3.24. Every virtual machine was assigned one virtual processor, 1024 MB RAM and hardware virtualization support (AMD-V). Both source and destination of the packets always had WireShark network traffic analysis software active.

Testing scenario and scapy utility results: For the testing of selected firewall's behaviour and communication with the modified packets, scapy utility was used. This packet manipulation utility allows the creation of almost any packet using pre-defined syntax.

For its performance, scapy uses so-called raw sockets because of which it was necessary to modify the testing topology to correspond to the test topology in Fig. 1. In the topology, the testing packets are trying to reach their target destination where they are blocked by the rules for the incoming packets. This, however, is not true for the packets created using raw sockets. Despite an explicit rejection the packet will not be discarded. Besides scapy, other utilities were used: randicmp6 and toobig6 from The Hacker's Choice IPv6 Toolkit. Rather that attempting to bypass the firewall in this set of tests we monitored whether the firewall forwarded certain packet types even if their format was for instance in conflict with RFC. The traffic using UDP protocol was being sent to port 69 and an explicit rule for its permission was always created. It was expected that the firewall should not give permission to the packets from the following scenarios:

- Sends all combinations of ICMPv6 type and code to the target. However, only some types will be of any interest to us
- Sends an ICMPv6 error message "TooBig" with too low MTU value
- Sends an ICMPv6 error message "TooBig" with too high MTU value
- Sends several encapsulated fragments (i.e., fragmented fragments). There is no reason for such packets to be created in the network python
- Unspecified address as the source
- Loopback address as the source
- Address "all nodes in the local network segment" as the source
- Packet with type 0 routing header. Passing it is a violation of RFC 5095

Table 2: The results for scapy

Debian	Ubuntu 14.04	Fedora 21	CentOS	FreeBSD	OpenBSD	NetBSD	Solaris
NO	NO	NO	NO	NO	NO	NO	NO
NO	NO	NO	NO	NO	NO	NO	NO
NO	NO	NO	NO	NO	NO	NO	NO
NO	Y	Y	NO	NO	Y	NO	Y
Y	Y	Y	Y	Y	Y	Y	Y
Y	Y	Y	Y	Y	Y	Y	Y
Y	Y	Y	Y	Y	Y	Y	Y
NO	NO	NO	NO	NO	Y	NO	NO
NO	NO	NO	NO	NO	NO	NO	NO
NO	NO	NO	NO	Y	NO	NO	NO
NO	NO	NO	NO	NO	NO	NO	NO
NO	NO	NO	NO	NO	NO	NO	NO
NO	NO	NO	NO	NO	NO	NO	NO
NO	NO	NO	NO	NO	NO	NO	NO
Y	Y	Y	Y	Y	Y	Y	Y
NO	NO	NO	NO	NO	NO	NO	NO
NO	NO	NO	NO	Y	Y	Y	Y
NO	NO	NO	NO	NO	NO	NO	NO
Y	Y	Y	Y	NO	Y	NO	Y
NO	NO	NO	NO	NO	NO	NO	NO
NO	NO	NO	NO	NO	NO	NO	NO
NO	NO	NO	NO	NO	NO	NO	NO
NO	NO	NO	NO	NO	NO	NO	NO

- Packet with type 2 routing header and "Segments Left" header field different from 1
- Packet with type 100 unallocated routing header. Passing it is a violation of RFC 2460
- Packet with invalid chain of headers: Hop-by-hop, Hop-by-hop. Violation of RFC 2460
- Packet with invalid chain of headers: Destination options, Hop-by-hop. Vio-lation of RFC 2460
- Packet with invalid chain of headers: Hop-by-hop, Destination options, type 2 routing header, Hop-by-hop. Violation of RFC 2460
- PadN options can legally contain zeros only (RFC 2460) otherwise it be-comes so-called hidden channel
- Packet with invalid chain of headers: Hop-by-hop (except for Pad1 and PadN they can occur only once) duplicate Jumbo Payload occurrence. See RFC 4942
- Packet with invalid chain of headers: Destination options, duplicate jumbo payload occurrence. See RFC 4942
- Packet with invalid chain of headers: Hop-by-hop, duplicate router alert occurrence. See RFC 4942
- Packet with invalid chain of headers: destination options, duplicate router
- Packet with invalid chain of headers: hop-by-hop, duplicate tunnel encapsulation limit occurrence. See RFC 4942
- Packet with invalid chain of headers: destination options, duplicate tunnel encapsulation limit occurrence. See RFC 4942
- PadN should not insert more than 5 octets

The third set of tests was executed using scapy as well. Once again, we use UDP protocol and port 69 which is now blocked by the firewall. We send IPv6 packets that we encapsulate in another IP protocol header.

- Sends IPv6 packet encapsulated in IPv6 header
- Sends IPv6 packet encapsulated in IPv4 header

Table 2 presents test results of individual operating systems and their firewalls. The testing scenarios proposed above were executed using scapy utility and randicmp6 and toobig6 from The Hacker's Choice IPv6 Toolkit. UDP traffic destined to port 69 was generated and explicitly allowed in firewall. Scenarios 22 and 23 are the exceptions where IPv6 was encapsulated in IPv6 and IPv4 respectively. In these two scenarios an explicit block of combination of UDP protocol and destination port 69 was created. These tests were aimed at whether the given packet was passed or blocked by the firewall. Except for already mentioned Scenarios 22 and 23 where the packets were expected to be blocked based on the explicit access rules, packets created in conflict with RFC should not be passed further. The results were marked Y when the firewall behaviour worked properly and NO if the firewall rules were bypassed. As it can be seen, neither of the firewalls does implicitly follow RFC protocols in terms of ICMPv6 filtering (Scenario No. 1). However, all of the tested firewalls allow the creation of granular rules for ICMPv6 filtering dependent on type and code. Therefore , it is the administrator's responsibility to assemble the access rules set to ensure proper firewall behaviour.

All the firewalls, also, allow ICMPv6 error messages "Packet Too Big" with too low or too high MTU values (Scenarios No. 2 and 3, respectively) to pass. The firewall administrator cannot reject such packets specifically based on MTU value. ICMPv6 messages "Packet Too Big" belong among those that are necessary for the Ipv6 communication and they generally must not be filtered.

In Scenario No. 4 fragmented fragments were created normally there is no real reason for such packets to be created. GNU/Linux distributions with older core ver-sions (Debian and CentOS in our test) pass such packets; Ubuntu 14.04 LTS and Fedora 21 block them. Therefore, development towards the correct firewall behaviour can be seen. NetBSD and Oracle Solaris with their IP Filter firewall, also had interesting results. While IP filter version distributed with Oracle Solaris blocked such traffic, NetBSD version did not.

All the firewalls coped well with Scenarios 5-7 where packets with source addresses that should not be passed further were generated.

In Scenarios 8-10 we focused on routing headers. By default, Netfilter/IPtables do not work with such headers at all it is necessary to make use of "rt" module and perform the filtering in specifically created rules chain. Once the specific rules have been created it is possible to achieve the behaviour completely compatible to RFC. IP Firewall and packet filter in default settings each managed to succeed once. FreeBSD system's IP Firewall correctly did not forward routing header with unallocated type, however it did not follow RFC 5095 in regard to outdated type 0 routing header. OpenBSD system's packet filter apparently performs inspection of routing header type but it ignores "Segments Left" field, otherwise it would discard the packets from Scenario No. 9. For IP filter it is necessary to create explicit prohibition or prohi-bitions (with use of "v6hdrs" keyword) otherwise it does not work with such headers.

In Scenarios 11-13 packets with invalid header chains were generated. Neither of the firewalls succeeded in its default settings here. Netfilter/IPtables must make use of ipv6header or ip6headerorder modules and create special rules chain. On top of that the administrator must manually define all header combinations that should be rejected by the firewall. Similar situation occurs in other firewalls as well. IP firewall and IP neither of the firewalls checks the content of PadN (Scenario No. 14) and it is not possible to solve this situation with explicit firewall rules definition either. Creation of the packets with invalid extension header settings chains was the main aim of Scenarios No. 15-20. Netfilter/IPtables packets from Scenarios No. 15 and 19 were discarded automatically for explicit work with these settings "hbh" and "dst" modules were necessary and so was stating all the combinations to be discarded. IP firewall, packet filter, nor IP filter mention how to define the rules for these scenarios in their respective reference guides. However, they always implicitly block some of the packets in these scenarios. The exact cause of this behaviour remains unknown. Neither of the firewalls checks the length of PadN (Scenario No. 21) and again it is not possible to solve this situation with explicit firewall rules definition. Scenarios No. 22 and 23 where IPv6 packets were encapsulated in another IPv6 and IPv4 header, respectively, resulted in a failure as well. By encapsulating the packet its UDP header gets changed and the permission rules are avoided based on the protocol and port in all firewalls. The administrator must be ready for such traffic and filter it based on protocol 41 (an example of such a rule in ip6tables:-A FORWARD-p 41-j DROP). The best of the tested firewalls is probably Netfilter/IPtables, especially, thanks to the system of modules that allow the creation of granular rules. From the results it is apparent that real IPv6 firewall environments are not generally in accordance with RFC documents. Moreover, creating it can require a great amount of work (necessity to implicitly name combinations of extension headers to be discarded) and therefore it is susceptible to errors. It is also, related to high knowledge and time requirements on the administrator. This implies that speaking of firewalls IPv6 protocol does not make life any easier as many new possibilities have arisen (extension headers) and therefore, there can be higher variety of attacks. In some firewalls, extension headers are not even adequately supported which is also, true for ICMPv6 message content (TooBig).

CONCLUSION

The aim of this study was to analyse the issues of firewalls in the environment of IPv6 protocol. First, the analysis of the current state has been made with researchers relying mainly on corresponding RFC documents. Relevant parts of IPv6 protocol have been presented along with the definitions of the essential terms. IPv6 firewalls have then been practically tested in virtualized lab environment. Several host firewalls distributed with selected operating systems have been tested. These firewalls are likely to become of even greater importance in IPv6 as an idea of true end-to-end connectivity is emerging. Higher parts of the responsibility for safety will go from the centralized defence of the network perimeter to every host. Every firewall has been the subject to the series of tests using

selected software utilities being aimed at known weak points in IPv6 specification. The behaviour of respective firewalls has been recorded and discussed. In conclusion, it can be stated that firewalls work inconsistently not only with RFC but also with each other. To achieve more correct performance that would be closer to RFC, a quality access rules setmust be created.

ACKNOWLEDGEMENTS

This research and the contribution were also, supported by project "Smart = Solutions for Ubiquitous Computing Environments" FIM, University of Hradec Kralove, Czech Republic (under ID: UHK-FIM-SP-2016- 2102).

REFERENCES

Afilias, J.A., P. Savola and G.N. Neil, 2007. Deprecation of type 0 routing headers in ipv6. Stand. Track, 1: 1-7.

Barre, S., J. Ronan and O. Bonaventure, 2011. Implementation and evaluation of the Shim6 protocol in the Linux kernel. Comput. Commun., 34: 1685-1695.

Davies, E.C., S.K. Ericsson and P. Savola, 2007. IPv6 transition-coexistence security considerations. Inf., 1: 1-41.

Deering, S.C. and R.N. Hinden, 1998. Internet protocol, version 6 (Ipv6) specification. Stand. Track, 6: 1-39.

Hain, T., 2000. Architectural implications of NAT. Inf., 1: 1-29.

McCann, J., S. Deering and J. Mogul, 1996. Path MTU discovery for IP version 6. Stand. Track, 6: 1-15.

Nordmark, E. and M. Bagnulo, 2009. Shim6: Level 3 multihoming shim protocol for ipv6. Stand. Track, 3: 1-124.

Steffann, S., S.J.MP.C. Steffann and I.V. Beijnum, 2013. A comparison of IPv6-over-IPv4 tunnel mechanisms. Inf., 1: 1-41.

Van, G.D.V., T. Hain and R. Droms, 2007. Local network protection for ipv6. Inf., 1: 1-36.