IPv6 basic Security - PART I

Oussama Louhaïdia Supervisor: Jong Hyouk Lee

March 10, 2013

Contents

List of Figures

List of Tables

1 Introduction

IPv6 is taking over the internet world, it is now largely deployed within universities, companies, major internet services and government sites. It beeing the future of the internet, is due to the larger addressing space, its auto configuration mechanisms and mandatory security. Thanks to IPsec integrity, confidentiality and authentication are guaranteed at layer 3, but this comes after some configuration steps.

The Neighbor Discovery Protocol (NDP) is mandatory to establish those configurations, which make it a very important protocol, and any attacks over it will corrupt the network and its equipments, and makes IPSec useless.

In this study, we are focusing on IPv6 basic security issues mostly related to the Neighbor Discovery Protocol (NDP). Any IPv6 enabled node relies on the NDP for its essential IPv6 operations like auto-address configuration, default router configuration, movement detection in a mobile node case, etc. We are going to study the NDP in detail and then try to figure out possible attacks. This study is a fully engineering case, so it will end up to develop real running attack codes and perform test attacks. We'll design attacks over NDP vulnerabilities, and develop running codes for those attacks.

Note that this study is part of IPv6 Basic Security. Another part of IPv6 Basic Security is about protection of the NDP against various attacks.

Keywords: IPv6, NDP, ICMPv6, Threat Model, Attack Design, Security, RFC, Host ..

2 Protocols overview

2.1 Internet Protocol Version 6

IPv6 (Internet Protocol version 6) is a network connectionless protocol operating over layer 3 of the OSI model (Open Systems Interconnection). IPv6 is the culmination work done in the IETF in the 1990s to be the successor to IPv4. Its specifications were finalized in RFC 2460[7] in December 1998.

With 128-bit instead of 32-bit addresses, IPv6 has a much more important address space. This considerable amount of addresses allows greater flexibility in the assignment of addresses and a better aggregation of routes in the Internet routing table. The address translation (NAT), which was made popular by the

lack of IPv4 addresses is no more necessary. It is still used in some cases though.

IPv6 also has mechanisms for automatic assignment of addresses that facilitates also their renewal. The size of the subnets, variable in IPv4, was set at 64-bit in IPv6. Security through IPsec mechanisms are part of the basis of the Protocol specification. The header of the IPv6 packet has also been simplified.

The deployment of IPv6 on the Internet is complicated due to the incompatibility of IPv4 and IPv6 addresses. Automatic address translators have significant practical problems (RFC 4966[3]). During a transition phase where co-exist IPv6 and IPv4, the hosts have a double stack, meaning that they have both addresses, IPv6 and IPv4, and tunnels to cross the routers' groups which do not yet support IPv6.

In 2011, only a few companies have begun to deploy IPv6 technology on their internal network, Google and Wikipedia especially. In 2012, the deployment of IPv6 is still limited, the proportion of IPv6 Internet users is estimated at 0.5%, and this despite urgent calls to speed up the migration directed to the Internet access providers. In fact the exhaustion of available public IPv4 addresses is imminent.

Table 1: Some IPv6 addressing schemes

Unspecified	::/128
Loopback	::1/128
Multicast	FF00::/8
Link-local unicast	FE80::/10
Site-local unicast	FEC0::/10
Global unicast	Everything else

Table 2: Some multicast addresses scope

ffx1::/16	local
ffx2::/16	link local
ff02::1	All nodes on the local network segment
ff02::2	All routers on the local network segment
ff02::5	OSPFv3 All SPF routers
ff02::6	OSPFv3 All DR routers
ff02::8	IS-IS for IPv6 routers
ff02::9	RIP routers

Another point is the MAC mapping in IPv6, where the Ethernet MAC is derived by the four low-order octets OR'ed with the MAC 33:33:00:00:00:00, so for example the IPv6 address FF02:DEAD:BEEF::1:3 would map to the Ethernet

MAC address 33:33:00:01:00:03[1]. We may use this.

This was for the addressing plan, let us look into the packet format. It has a 320 bit (40 bytes) header, it has 8 fields among which the source and destination addresses, the `hop limit` (1 byte), wich is decremented by each crossed router, the next header (1 byte) indicated the header that encapsulates the IP one, the `payload field` (2 bytes) indicates the length of the data field transported by the packet (1460 maximum), the `flow label` (2,5 bytes) is used to label a particular data flow. Finally the `version` field should indicate 0x6.

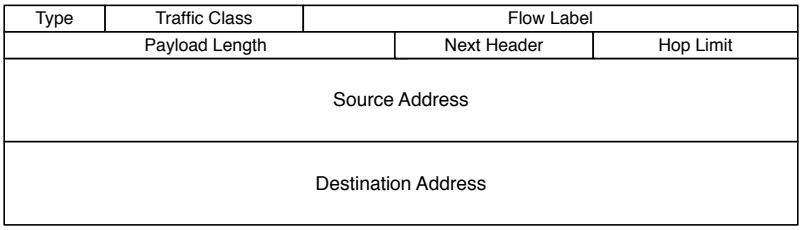


Figure 1: Syntax of an IPv6 header

2.2 Neighbor Discovery Protocol

Neighbor Discovery (ND) is an IPv6 protocol, based on ICMPv6 messages exchange between nodes. It is built upon five ICMPv6 packet types (Table-3): Router Solicitation (RS), Router Advertisement (RA), Neighbor Solicitation (NS), Neighbor Advertisement (NA), Redirect.

Compared to IPv4, this protocol is a combination of a set of protocol from the TCP/IPv4 family, including ARP and ICMP. Though IPv4 protocol suite doesn't define a process for unreachability detection.

It consists of a mechanism with which a node that has just been added to a network, see the presence of other nodes on the same link, in addition to viewing their IP addresses. This Protocol also deals with keeping clean the caches where is stored the information relating to the context of the network to which a node is connected. It uses ICMPv6 messages, and is the basis to allow IPv6 au toconfiguration mechanism [2]. It is used for the following purposes in Table 3.

2.3 Packets' syntax

In this subsection, we'll see the nominal form of a packet and the requirement for it to be validated by a receiving node. RFC 4861[5] details those requirements.

Table 3: Some NDP purposes

Router discovery	hosts can locate routers residing on attached links.
Prefix discovery	hosts can discover address prefixes that are on-link for attached links.
Parameter discovery	hosts can find link parameters (e.g., MTU).
Address auto configuration	stateless configuration of addresses of network interfaces.
Address resolution	mapping between IP addresses and link-layer addresses.
Next-hop determination	hosts can find next-hop routers for a destination.
Neighbor unreachability detection (NUD)	determine that a neighbor is no longer reachable on the link.
Duplicate address detection (DAD)	nodes can check whether an address is already in use.
Redirect	router can inform a node about better first-hop routers.

So for the packets we tend to forge in our experiments, those requirement should be fulfilled, in particular Checksums and packets' lengths.

Router Solicitation message Sent from Hosts soliciting a quick router advertisement. Must have more than 8 bytes length. Router solicitation messages with a multicast address (ff00::/8) as source are automatically dropped by the receivers. Figure 2 gives the general structure of those messages. Destination address of the IPv6 header that encapsulates a RS, should be a router multicast address (ff02::2) as seen in 2.1.

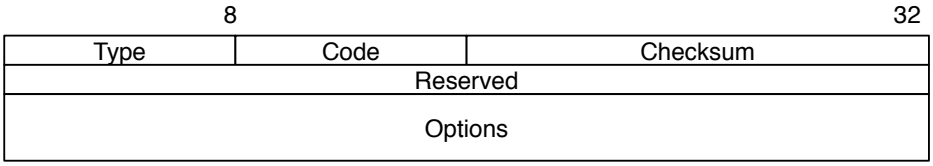


Figure 2: Router Solicitation message syntax

Router Advertisement message Sent periodically or in response to router solicitations, through an advertising interface. Destination is set to the soliciting

node or to nodes multicast address, if not a unicast address it'll be dropped by the receivers.

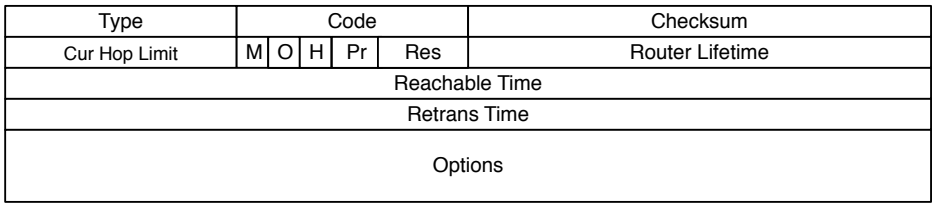


Figure 3: Router Advertisement message syntax

The M flag when activated means the addresses are available threw dynamic host configuration process (DHCP6). The O flag designates dynamic host configuration related Options (DNS). MO set to 0b00 means that no DHCP6 information is available.

Neighbor Solicitation message Helps a node resolves a target address while giving him his own link-layer address. Do NUD by unicast. Router Solicitation messages will be discarded if they doesn't respond to the following requirements: The hop-limit field isn't 255 (went through some router), invalid id checksum, ICMP Code field is not 0, ICMP inferior to 8 bits.

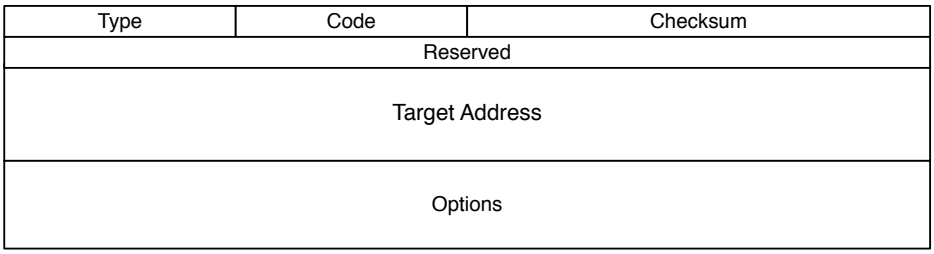


Figure 4: Neighbor Solicitation message syntax

Neighbor Advertisement message respond to NS, or propagate other informations quickly (unreliable).

Redirect message sent from a router (could be spoofed by an attacker) to hosts informing of a better first-hop, or that destination host is just a neighbor (target = destination address). Requirements: Sender is a link local address.

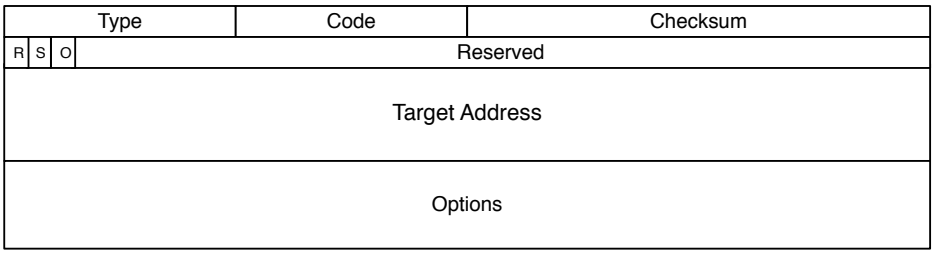
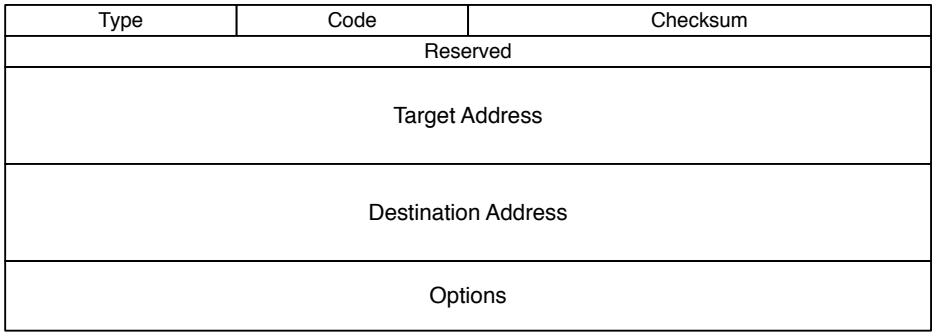


Figure 5: Neighbor Advertisement message syntax

Only routers can send redirects, authentication mechanisms exists but are rarely implemented.



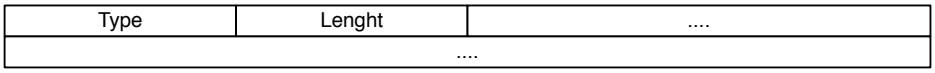

```
Reserved: These fields are unused
```

Figure 6: Redirect message syntax

Options stack Neighbor Discovery messages can include a number of options. Such options have the following general syntax (figure 7). A packet with option field length equal to zero, will be discarded automatically.

The draft [4] describes some well known option field issues. For example, NDP options being processed by the OS Kernel, an on-link host can send many NDPes with large options fields, which takes a lot of CPU power, and affect the host computing capabilities.

Option field may be extended, respecting some conditions. Section 3.6 of this [4] draft gives more details.




```
Type: type of option
  Source Link-Layer Address                    1
  Target Link-Layer Address                    2
  Prefix Information                           3
  Redirected Header                            4
  MTU                                          5
```

Figure 7: NDP Options field

2.4 Auto configuration process

In addition to the large addressing space, hosts auto-configuration is the main addition to the internet world since IPv4. Auto-configured addressed may be in one of those states in Figure 8: Tentative, Preferred, Deprecated, Valid, Invalid.

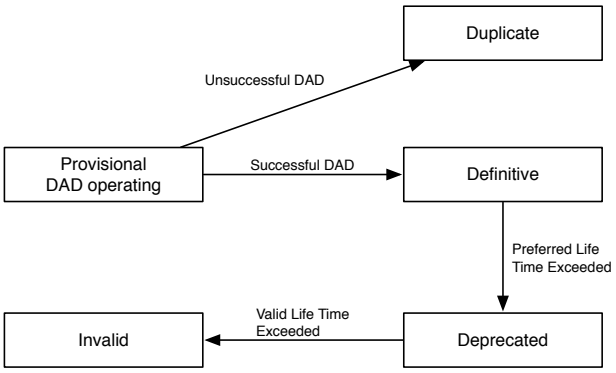


Figure 8: Life-cycle of an IPv6 address

A highly useful aspect of IPv6 is its ability to automatically configure itself without the use of a statefull configuration protocol, such as Dynamic Host Configuration Protocol for IPv6 (DHCPv6). By default, an IPv6 host can configure a link-local address for each interface. By using router discovery, a host can also determine the addresses of routers, additional addresses, and other configuration parameters. The Router Advertisement message indicates whether a statefull address configuration protocol should be used.

Address auto configuration can be performed only on multicast-capable interfaces. RFC 2462 describes address auto configuration.

2.5 Conceptual Model of a Host

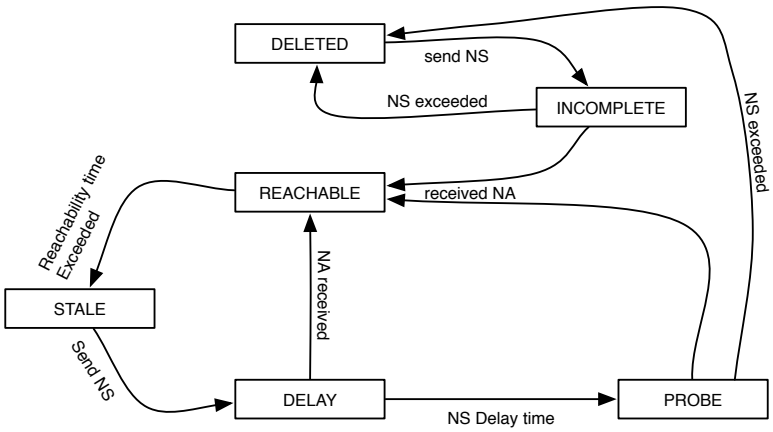


Figure 9: A node's cache states

As described in RFC 4861[5], this model describes the possible data structure that nodes will maintain for each of their IPv6 activated interfaces. Neighbor Discovery Process (NDP) is continually concerned by this task.

- Neighbor Cache: A set of entries about individual neighbors to which traffic has been sent recently (equivalent to ARP cache in IPv4)
- Destination Cache: A set of entries about destinations to which traffic has been sent recently
- Prefix List: A list of the prefixes that define a set of addresses that are on-link
- Default Router List: A list of routers to which packets may be sent. The algorithm for selecting a default router favors routers known to be reachable over those whose reachability is suspect.

Neighbor cache is ordered in entries which can be in one of 5 states: INCOMPLETE, REACHABLE, STALE, DELAY, PROBE. Each entry should be REACHABLE so the node will be able to process IPv6 packets from another host.

3 Attack design

Two types of exploits could be developed over IPv6 enabled networks, local and remote ones. In this section, common and less common weaknesses are presented. Many types of attacks are possible[fig:10], among which Traffic hijacking or Fabrication, Denial of Service and Performance degrading.

Every mechanism, implicating IPv6, has his DoS type weaknesses defined within its RFC's security considerations section. In this work we'll try to define a more useful fishing attack.

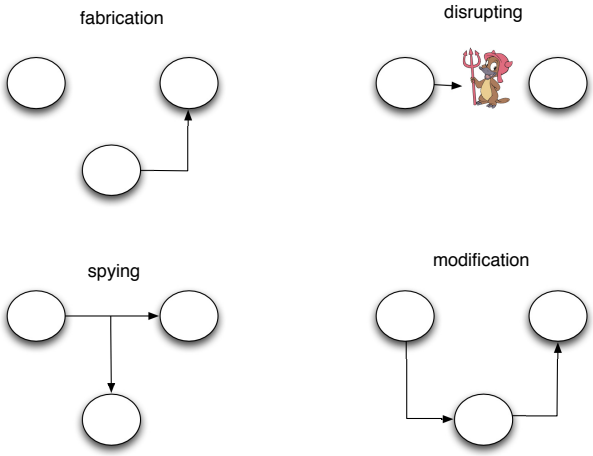


Figure 10: Four main attack types

3.1 DOS attack: DAD based

This is the simplest attack that could be implemented on a local link, yet its effects are major. To check the uniqueness of unicast and link-local addresses, machines must be running an algorithm called Duplicate address Detection (DAD) before using any new address. The algorithm uses the NS and a NA ICMPv6 control messages. If an address already in service is discovered, it cannot be assigned to the interface.

- A NA is received: the current address is used as a valid address by another machine. This address is not unique and cannot be accepted.
- A NS message is received (another DAD is currently running); also, the address is a temporary address to another machine, and that address cannot be used by any of the machines.

- Nothing is received by the end of 1 second (default value): the address is unique, it goes from a provisional to a valid state, and it is assigned to the interface.

An attacker who wants to prevent a victim from receiving an IPv6 address, could advertise to any host requesting a NS DAD message. The Figure 11 shows how this attack is performed.

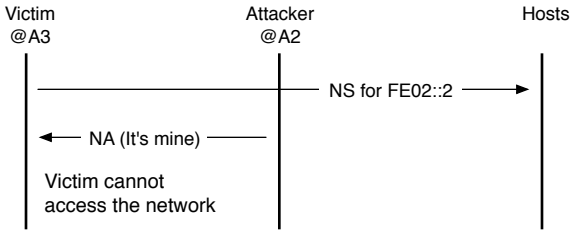


Figure 11: Attack over DAD mechanism

This attack is part of the `thc` toolkit under the name `./dos-new-ip6`[10].

3.2 MiTM attack: Traffic hijacking

In a first scenario, a victim attempts to discover routers on its local link via Router Discovery. It sends an ICMPv6 RS message requesting information from the routers on its local link. A legitimate router responds with an ICMPv6 RA for a lifetime x, and a certain priority, that lets the victim know that it is one of the link routers.

In return, host V installs a default route to its routing table that points to that router, which is erased after a period of x time.

If an attacker, manages to install himself on the link, he could attempt to advertise himself as a default router in the routing table of the victim. To do that, there is plenty of ways:

- Could be based on the lifetime of the configuration, which is one option of RA messages.
- Based on link local RA, an attacker could send a spoofed RA message with his address. This type of attack, was covered by Linux security updates.

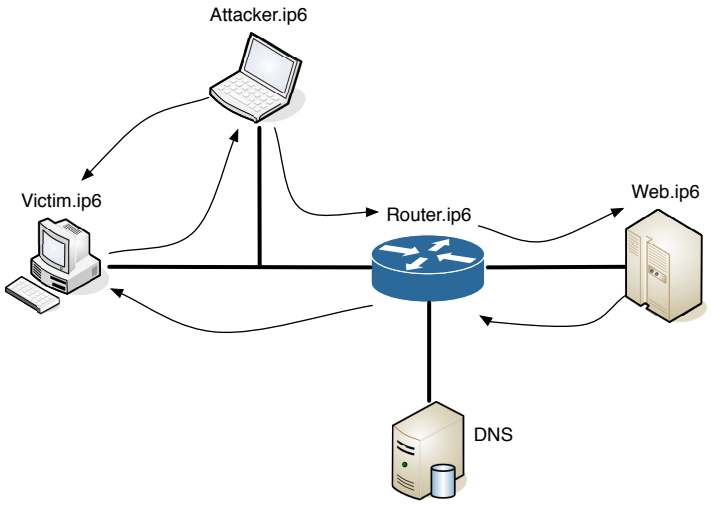


Figure 12: Topology of the attack

- Based on Redirect messages, an attacker could make a router consider him as a better router, and thus redirect traffic towards him. Another way is to spoof Redirect messages, if we know the default router of a node.

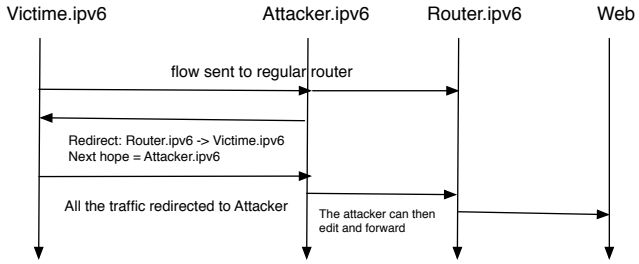


Figure 13: Redirects spoofing

- Or based on the priority option of the RA message, a fake router could place himself as a high priority router for some multicast group or unicast node, and then receives all the packets.

For the first option and according to RFC 4862[8], that says: "If the received Valid Lifetime is greater than 2 hours or greater than RemainingLifetime, set the valid lifetime of the corresponding address to the advertised Valid Lifetime" and "If RemainingLifetime is less than or equal to 2 hours, ignore the Prefix Information option with regards to the valid lifetime, unless the Router Advertisement from which this option was obtained has been authenticated".

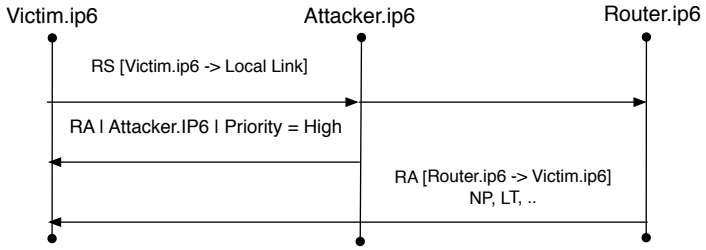


Figure 14: Prioritised router attack

Then an attacker can spoof a new router advertisement to the victim from the legitimate router and set its lifetime to 2 hours. This is a time consuming attack.

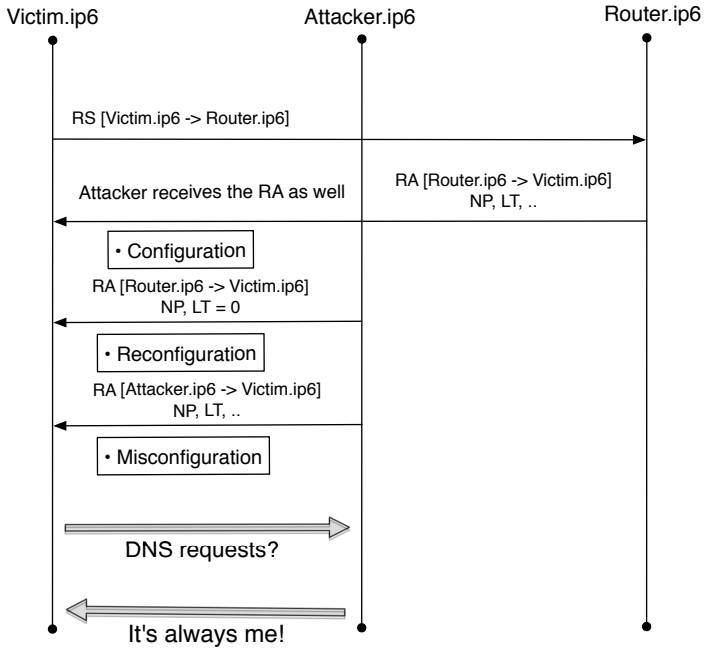


Figure 15: MITM Attack

The victim will remove the configured default route after 2 hours. The attacker will send then a RA to insert himself as the default router on the victim's routing table.

The option field is described in a previous section (see Packet syntax 2.3).

This is a half man in the middle attack, where an attacker receives only the

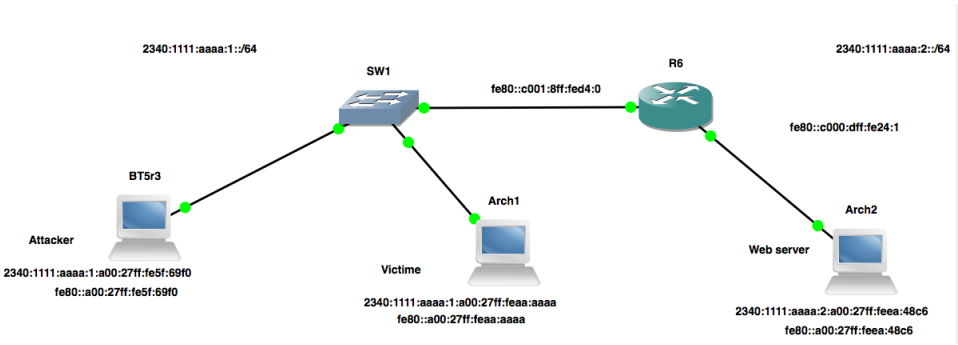


Figure 16: The test topology within GNS3

traffic that originates on the victim side, he doesn't receive the responses to those requests. An attacker first get this position then spoof DNS responses to redirect the victim to other web pages, in order to perform another type of attacks. He will forward non DNS traffic.

4 Implementation

To emulate the designed attacks, we'll be using GNS3 (Graphical Network Simulator) (www.gns3.net), an open source network simulator, that combines Dynamips and Virtualboxes to provide accurate simulations. It supports Cisco 1700, 2600, 3600, 3700, and 7200 hardware routers platforms and runs standard IOS images. In addition it is multiplatform.

The network interface configuration will be set to a Generic Driver adapter, controlled by the GNS3 environment.

IPv4 will be turned off to delete any possible unwanted traffic, it can't be totally disabled though, as all the TCP/IP protocols depend on it, and as the system uses it (127.0.0.1). The kernel is built with its support. A kernel built without it will see neither of the two addressing protocols working.

The topology I'm using to test the attack, is compound of 3 hosts, a router and a switch. One of the hosts is the Vitim, the other is the Attacker and the third one is the web server the Vitim is trying to reach. All hosts are running a Linux 2.6, with full support of IPv6 through the "ipv6" kernel module.

The router is a Cisco 3720, with full support of IPv6 and RIPng protocols. I used c3620-j1s3-mz.123-19.image. The emulation with Dynamips requires this

IOS image of the emulated system, that could be found on the internet, or downloaded from cisco.com if any courses are attended at The Cisco Learning Network. Performing some manipulations on the router, pinging and traceroute went fine. The test lab is ready!

4.1 Libraries

UNIX was basically created to be a programming environment. Every peripheral is seen as a file, and interacting with it is reduced to writing/reading operations from this file. Sockets aren't an exception,

To code and test those algorithms, we'll be using the sockets API defined in the rfc2292[9]. The "sockets" API was developed in the early 80s for TCP/IP version 4, rfc3493[6] describes the changes and extensions in this API to adapt IPv6.

To make the transition from simple IPv4 socketing to IPv6, one should consider the length of the addresses, thus the structure that holds the IPv4 addresses: struct sockaddr_in should be replaced by struct sockaddr_in6 since the first one only supports 32 bits addresses. AF_INET6 address family was defined.

```
struct sockaddr_in6 {
       u_char       sin6_len;
       u_char       sin6_family;
       u_int16_t    sin6_port;
       u_int32_t    sin6_flowinfo;
       struct       in6_addr      sin6_addr;
};
```

```
socket(PF_INET6,SOCK_STREAM,0); /* TCP socket */
socket(PF_INET6,SOCK_DGRAM,0); /* UDP socket */
```

The IETF had standardized two sets of extensions for the IPv6 socketing, rfc3493 and rfc3542.

- RFC 3493 Basic Socket Interface Extensions for IPv6: Provides standard definitions for Core socket functions, Address data structures,Nameto-Addresstranslat ionfunctions,Address conversion functions
- RFC 3542 Advanced Sockets Application Program Interface (API) for IPv6: Defines interfaces for accessing special IPv6 packet information such as the IPv6 header and the extension headers.

We'll not focus a lot on those aspects, we'll be using THC-IPV6 framework to code our proof of concepts. It is an open source library and attack toolkit licensed under `GPLv3`, developped by the thc community `thc.org`, for UNIX like OS, because it uses the procfs[1].

Its documentation is available online on `thc.org/thc-ipv6/README` and is included in the current report Annex A.

thc-tools The thc-ipv6 toolkit have been around since 2004. The library contains a lot of useful functions we'll be using to develop our attack. The functions are detailed in the Annex I (official documentation) those functions include very useful short cuts to develop a complete solution.

Some functions are aimed to generate and send packets, add headers, get mac addresses from sniffing around, get own mac address, get own IPv6 address, send a packet as fragments, inverse packet to rapidly send a response by switching the sender and receiver's addresses, and many others.

It also include some IP-sec functionality concerning the generation and management of keys. Another functions gives us the simplest way to generate RA messages to advertise oneself over an interface, `thc_routeradv6.c`.

Current this toolkit is under a lot of limitations, it runs on Linux and Little Endian platforms only, it is also limited to 32-Bit and to the Ethernet. One needs to install `ssltools` and `pcaptools` to be able to compile the library.

4.2 The DoS attack

THC-IPv6 toolkit comes, in addition to the library, with a set of tools and exploits, ready to use, of the most commonly known ND vulnerabilities.

the DAD attack illustrated in the precedent section is actually illustrated by the `dos-new-ip6.c` program.

To test this program we'll disable the eth0 interface of `arch_1` host, launch the program on the `BT5r3` machine (attacker), and reactivate the IPv6 on the victime. The mechanism this DOS attack is based upon is decribed on the sections before. Pinging and tracerouting doesn't work anymore.

In fact the globale address isn't configured, and the local address is unresponsive

[1] `The /proc file system is a special filesystem in UNIX-like operating systems that presents information about processes and other system information`

as the host will drop any traffic on that interface. The address is only there to communicate DAD messages, it becomes definitive only if the DAD mechanism succeeds. (Figure 8)).

4.3 The MiTM attack

Our attack is a Half-MIT based, as described in a previous section, in order for an attacker to get a MIT priveleges, he can do it, as seen before, in many ways. The one we'll be using is the life time based one.

The attack is performed in two phases, first lifetime of the genuine router is set to zero, then the fake router is advertised with a high priority and a large lifetime. This results into making a rogue router attack.

`ip -6 route` command on the victim shows us the current routing table on the default interface, and how it changed to put the attacker as the default gateway.

5 Conclusion

In this project we have seen how does the IPv6 protocol manages its autoconfiguration futures, and how the processes implicated in this mechanisms are weak, and sensitive to protocol attacks. We build some attack scenarios over those weaknesses, and we tested them.

IPv6 still not largely deployed is a gold mine for security researchers, it is build to make internet more secure through its basic elements (IPsec, AAA, Firewalls, SEND, ..).

The small hosts density (up to 2^{127} host by subnet) makes scanning, worm propagation and other related threats harder. Though its addressing and configuration are complicated, and those complications result in many exploitable weaknesses.

THC-IPV6-ATTACK-TOOLKIT
(c) 2005-2012 vh@thc.org www.thc.org

Licensed under GPLv3 (see LICENSE le)

[..]

THE LIBRARY
==========
The library thc-ipv6-lib.c is the heart and soul of all tools - and those you may want to write.
Implementation is so simple, its usually just 2-4 lines to create a complete
ipv6/icmp6 packet with the content of your choice.

Your basic structure you use is
  (thc_ipv6_hdr *)
e.g.
  thc_ipv6_hdr *my_ipv6_packet;
  int my_ipv6_packet_len;
and you will never have to play with its options/ elds.

Whenever you want to build an ip6 packet, you just write:
  my_ipv6_packet = thc_create_ipv6(interface, prefer, &my_ipv6_packet_len,
                          src6, dst6, ttl, length, label, class, version);
if something fails, it returns NULL (only if my_ipv6_packet_len or dst6 do not exist or malloc fails).
The options to thc_create_ipv6 are:
 (char*) interface - the interface on which you want to send out the packet
 (int) prefer - either PREFER_LINK (to use the link local address) or
                PREFER_HOST to use a host ip6 address, and
                PREFER_GLOBAL to use a public (internet) IP6 address (default)
 (int *) &my_ipv6_packet_len - the size of the packet which will be created
 (unsigned char*) src6 - the source IP6 (OPTIONAL - will be selected if NULL)
 (unsigned char*) dst6 - the destination IP6 (in network format, 16 bytes long)
                         usually the result of thc_resolve6 ("ipv6.google.com");
 (int) ttl - the ttl of the packet (OPTIONAL - 0 will set this to 255)
 (int) length - the length which will be set in the header (OPTIONAL - 0 =
                real length)
 (int) label - the ow label (0 is ne)
 (int) class - the class of the packet (0 is ne)
 (int) version - the IP6 version (OPTIONAL - 0 will set this to version 6)
It returns NULL on errors or a malloc'ed structure on success.
free() it once you are done with it.

Now you can set extension headers on top of it:
  thc_add_hdr_route(my_ipv6_packet, &my_ipv6_packet_len, routers,

```
routerptr);
  thc_add_hdr_fragment(my_ipv6_packet, &my_ipv6_packet_len, o set,
more_frags,
                       id);
  thc_add_hdr_dst(my_ipv6_packet, &my_ipv6_packet_len, buf, bu en);
  thc_add_hdr_hopbyhop(my_ipv6_packet, &my_ipv6_packet_len, buf, bu en);
  thc_add_hdr_nonxt(my_ipv6_packet, &my_ipv6_packet_len, hdropt);
  thc_add_hdr_misc(my_ipv6_packet, &my_ipv6_packet_len, type, len, buf,
bu en);
The functions explained:
 _route: Add a Routing Forwarding Header (like IP Source Routing)
  (int) routers - the number of routers in routerptr
  (char**) routerptr - a *char[routers + 1] struct with router destinations
                       in network format. See alive6.c for an example.
 _fragment: Add a Fragment Header
  (int) o set - the o set on which to the data should be written (note:
                 put the o set location in bytes here, not in byte octets)
  (int) more_frags - set to 0 if it is the fragement, 1 on all others
  (int) id - an ID for the packet (same for all fragments)
 _dst: Add a Destination Options Header
  (char*) buf - a char bu er. you have to control this bu er yourself with
                but you want to write into it.
  (int) bu en - the length of buf
 _hopbyhop: Add a Hop-By-Hop Header
  (char*) buf - a char bu er. you have to control this bu er yourself with
                but you want to write into it.
  (int) bu en - the length of buf
 _nonxt: Specify that there will be no following headers whatsoever
  (int) hdropt - this options is currently ignored
 _misc: Specify a miscelleanous header. Use this if you want to design an
        invalid or non-existing extension header.
  (int) type - The type ID to specify the header as
  (int) len - The length to advertise the header as (OPTIONAL - -1 sets
this to
              the correct value)
  (char*) buf - a char bu er. you have to control this bu er yourself with
                but you want to write into it.
  (int) bu en - the length of buf
These functions return (int) 0 on success and -1 on error.

Finally you can add the stream or dgram headers.
  thc_add_icmp6(my_ipv6_packet, &my_ipv6_packet_len, type, code,  ags, buf,
                bu en, checksum);
  thc_add_tcp(my_ipv6_packet, &my_ipv6_packet_len, source_port,
              destination_port, sequence_number, ack_number,  ags,
window_size
              urgent_pointer, options, optione_length, data, data_length);
  thc_add_udp(my_ipv6_packet, &my_ipv6_packet_len, source_port,
              destination_port, checksum, data, data_length);
  thc_add_data6(my_ipv6_packet, &my_ipv6_packet_len, type, buf, bu en);
 _icmp6: Add an ICMP6 packet header
  (int) type: the ICMP6 type
  (int) code: the ICMP6 code
  (int)  ags: the ICMP6  ags
```

```
  (char*) buf - a char bu er. you have to control this bu er yourself with
                but you want to write into it.
  (int) bu en - the length of buf
 _tcp|_udp: Add an TCP or UDP header
  (ushort) source_port: source port
  (ushort) destination_port: destination port
  (uint) sequence_number: TCP sequence number
  (uint) ack_number: TCP acknowledgement number
  (ushort) checksum: UDP checksum, 0 = generate checksum (for TCP the
checksum is always calculated)
  (uchar)  ags: TCP  ags: TCP_SYN, TCP_ACK, TCP_FIN, TCP_RST, TCP_PSH, ...
  (uint) window_size: TCP window size
  (uint) urgent_pointer: TCP urgent pointer (usually 0)
  (char*) options: TCP options bu er, can be NULL
  (uint) options_length: the length of the TCP options bu er
  (char*) data: the data the protocol carries
  (uint) data_length: the length of the data bu er
 _data6: Add a miscellaneous header
  (int) type: the protocol ID
  (char*) buf - a char bu er. you have to control this bu er yourself with
                but you want to write into it.
  (int) bu en - the length of buf
These functions return (int) 0 on success and -1 on error.

Once you are done, you create and send the packet.
  thc_generate_pkt(interface, srcmac, dstmac, my_ipv6_packet,
                   &my_ipv6_packet_len);
  thc_send_pkt(interface, my_ipv6_packet, &my_ipv6_packet_len);
or combined into one function:
 thc_generate_and_send_pkt(interface, srcmac, dstmac, my_ipv6_packet,
                           &my_ipv6_packet_len);

 thc_generate_and_send_pkt: This generates the real and  nal IPv6 packet
and
                           then sends it.
  (char*) interface - the interface to send the packet on
  (unsigned char*) srcmac - the source mac to use (in network format)
                            (OPTIONAL, the real mac is used if NULL)
  (unsigned char*) dstmac - the destination mac to use (in network format)
                            (OPTIONAL, the real mac is looked up if NULL)
The thc_generate_pkt and  thc_send_pkt together provide the same
functionality.
You usually use these only if you do something like
  thc_generate_pkt(...);
  while(1) thc_send_pkt(...);
These functions return (int) 0 on success and -1 on error.

When you are done, free the memory with:
  thc_destroy_packet(my_ipv6_packet);

There are some important helper functions you will need:
  thc_resolve6(destinationstring);
    This resolves the IPv6 address or DNS name to an IPv6 network address.
    Use this for dst6 in thc_create_ipv6(). The result has to be free'd
```

```
when
    not needed anymore.
  thc_inverse_packet(my_ipv6_packet, &my_ipv6_packet_len);
    This clever functions switches source and destination address,
exchanges
    the ICMP header type (ECHO REQUEST -> ECHO REPLY etc.) and recalculates
    the checksum. If you dont have an idea what this might be useful for,
    go and play with your xbox :-)
  thc_ipv6_rawmode(mode)
    the integer value 0 disables sending packets in raw mode, a value of 1
    enables it. Be warned that all MAC stu  will result in a dummy mac
    once it is in mode == 1.

If you just want to do it very fast, there are some prede ned icmp6 creator
functions which sends impc6 packets in just one line of code:
  thc_ping6(interface, src, dst, size, count);
  thc_neighboradv6(interface, src, dst, srcmac, dstmac,  ags, target);
  thc_neighborsol6(interface, src, dst, target, srcmac, dstmac);
  thc_routeradv6(interface, src, dst, srcmac, default_ttl, managed, pre x,
                 pre xlen, mtu, lifetime);
  thc_routersol6(interface, src, dst, srcmac, dstmac);
  thc_toobig6(interface, src, srcmac, dstmac, mtu, my_ipv6_packet,
              my_ipv6_packet_len);
  thc_paramprob6(interface, src, srcmac, dstmac, code, pointer,
                 my_ipv6_packet, my_ipv6_packet_len);
  thc_unreach6(interface, src, srcmac, dstmac, icmpcode, my_ipv6_packet,
               my_ipv6_packet_len);
  thc_redir6(interface, src, srcmac, dstmac, newrouter, newroutermac,
             my_ipv6_packet, my_ipv6_packet_len);
  thc_send_as_fragment6(interface, src, dst, type, buf, bu en, frag_len);
These do what you expect them to do, so I am too lazy^H^H^H^H^Hbusy to
describe it in more details.

The following functions allocate memory for the result pointer, so remember
to free the result pointers from these functions once you do not need them
anymore:
  thc_ipv6_dummymac()
  thc_ipv62notation()
  thc_ipv62string()
  thc_string2ipv6()
  thc_string2notation()
  thc_resolve6()
  thc_get_own_ipv6()
  thc_get_own_mac()
  thc_get_multicast_mac()
  thc_get_mac()
  thc_lookup_ipv6_mac()
  thc_look_neighborcache()
  thc_generate_key()
  thc_generate_cga()
  thc_generate_rsa()

It helps a lot if you take a look at example usages. The best ones are
the tools from the thc-ipv6 package, especially implementation6.c and
```

References

[1] *RFC 2464*.

[2] December 2012.

[3] E. Davies C. Aoun, Energize Urnet. Reasons to move the network address translator - protocol translator (nat-pt) to historic status. *IETF*, July 2007.

[4] F. Gont. Security assessment of neighbor discovery (nd) for ipv6 draft-gont-opsec-ipv6-nd-security-00. *IETF*, December 2012.

[5] E. Nordmark T. Narten H. Soliman, W. Simpson. Neighbor discovery for ip version 6 (ipv6). *IETF*, September 2007.

[6] J. Bound J. McCann W. Stevens R. Gilligan, S. Thomson. Basic socket interface extensions for ipv6. *IETF*, February 2003.

[7] S. Deering R. Hinden. Internet protocol, version 6 (ipv6) specification. *IETF*, December 1998.

[8] S. Thomson T. Jinmei, T. Narten. Ipv6 stateless address autoconfiguration. *IETF*, September 2007.

[9] M. Thomas W. Stevens. Advanced sockets api for ipv6. *IETF*, February 1998.

[10] www.thc.org. *THC-IPV6-ATTACK-TOOLKIT*. www.thc.org, 2012.