An Overview of IPv6 Attack Tools

Sheharbano Khattak

September 18, 2012

Abstract

This document provides an informal overview of IPv6 attack tools. In particular, we are interested in classifying attacks based on whether or not an attack can be detected and perhaps mitigated at an edge monitoring device.

1 Introduction

IPv6 is no longer the research playground of network geeks. The Internet Society held World IPv6 Launch [**?**] day on June 6, 2012. The motivation was to to help participating products and services permanently deploy IPv6 on their networks. Current studies [**?**] suggest global IPv6 deployment of 23.6%. These figures are expected to increase in view of policy requirements, compatibility issues and/or simply the pressure to take the leap as more and more organizations and devices embrace IPv6.

The looming question for the security community is: *How safe are we in an Ipv6 Internet?* A number of IPv6 based attack tools have already emerged, with THC-IPv6 [**?**] and SI6 Networks IPv6 toolkit [**?**] taking the lead. Both these toolkits come with a number of tools that leverage vulnerabilities in IPv6 itself, or implementation disparities in operating systems and devices to compromise a victim machine. The attacks range from sniffing to outright denial of service.

In this document, we provide a high-level overview of the tools. In particular, we focus on two aspects: (i) Is the attack LAN based or of global scope? Put another way, can we detect the attack on an edge monitoring device? (ii) We use Bro IDS [**?**] to detect the attacks. We do not discuss detection approaches that incur serious processing penalties for Bro.

The .tex source file of this document is available online [**?**]. Also available online are some initial scripts [**?**]. The rest of this document has been organized as follows. Section **??** discusses THC-IPv6 toolkit. An overview of SI6 Networks IPv6 toolkit has been presented in **??** Note that in many cases, attack description has been quoted as it is from user manuals, terminal prompts or comments in code. The author does not claim ownership of such material.

2 THC-IPv6

General comments: Documentation is poor. Knowledge of IPv6 and related protocols is presumed. An understanding of the attacks is also presumed. There is a README file but that won't get you anywhere if you are an IPv6 newb.

3 Local Attacks

parasite6: ICMP neighbor solitication/advertisement spoofer, puts you as man-in-the-middle, same as ARP mitm (and parasite).

alive6: an effective alive scanng, which will detect all systems listening to this address.

Comments: Detect ping sweeps at the edge. Drop inbound packets having destination address ff02::1 (or any link-local address?) Don't reply to multicast ping requests?

fake_router6: Announce yourself as a router on the network, with the highest priority (default router). If a non-existant MAC/link-local address is provided, the result is DoS. Leverages router advertisements.

redir6: Redirect traffic to you intelligently (man-in-the-middle) with a clever ICMP6 redirect spoofer. Implants a route into victim IP which redirects all traffic to target IP to new IP (attacker's rogue router).

toobig6: MTU decreaser with the same intelligence as redir6. PMD: (Path MTU discovery) IPv6 figures out path MTU between source and destination by special probe packets. If the size of packet exceeds MTU at any point, 'toobig' msg is sent back to the source. PLPMTUD: (Packetization Layer Path MTU Discovery) Progresively increases packet size (narrows search range). When packet loss occurs, the sender knows that MTU has been exceeded (no toobig msg).

Comments: authors compare it to redir6 when they say "implants specified MTU on target". I am not completely sure what is the implication of this.

detect-new-ip6: Detects new IPv6 devices which join the local network Can run a script when a new IPv6 machine joins the network, e.g. a script to automatically scan these systems etc. It achieves this by intercepting/sniffing DAD(Duplicate Address Detection) (ICMPv6 neighbor solicitation).

dos-new-ip6: Detects new devices and tell them that their chosen IP collides on the network (DOS). Prevents new IPv6 interfaces (machines) to come up, by sending answers to duplicate IP checks (DAD). This results in DoS for new IPv6 devices. A fake MAC is used for each reply (by the attacker).

flood_router6: Floods a target with random router advertisements. Flood the local network with router advertisements. dst: FF02::1 (link-local all nodes)

flood_advertise6: Floods a target with random neighbor advertisements. dst: FF02::1

fake_advertiser6: Announce yourself on the network. Advertise IPv6 address on the network. Sent to all nodes multicase address of target not defined. Uses IPv6 neighbour advertisement.

sendpees6: A tool that generates neighbor solicitation requests with a lot of CGAs to keep the CPU busy. Send SEND (SEcure Neighbour Discovery) neighbour solicitation messages and makes the target verify many CGA and RSA signatures.

3.1 Global Scope Attacks

dnsdict6: Parallized dns ipv6 dictionary bruteforcer DNS zone transfers are usually prohibited. Performs reconnaissance on a dns zone by prefixing it with commonly used prefixes from a file. This is based on dnsmap, the new things are IPv6 support and multithreading for speed.

trace6: Very fast traceroute6 with supports ICMP6 echo request and TCP-SYN. If only target address is specified, then sends ICMPv6 ping requests. If target address + port specified, then sends TCP SYN.

Comments: On my test network, it does not work for global IPv6 address. However, if i use ping6 (the linux one), it works. If i do THC ping6 after Linux's ping6, it works. I suspect dependency on link-local addresses which are perhaps located and cached by Linux's ping6, that's why THC ping6 works later.

fuzz_ip6: Fuzzer for IPv6. Protocol fuzzer. User specifies a packet type and specifies a number of tests (e.g. do 100 pkts). Program fills each packet fields with random values. For each fuzzed packet, it pings the target to see if it's still alive (Echo reply received or not for a ping). It continues to do this until the target is no more alive or the specified number of packets have been tried.

Comments: Protocol sanity checks?

implementation6: Performs various implementation checks on ipv6. Tests various IPv6 specific options for their implementation. This can be used as a pen-tester e.g. to see what a firewall passes.

implementation6d: Listen daemon for implementation6 to check behind a FW. A sniffer that shows you what got through.

Comments: Are the authors suggesting that all the different things the tool tries should ideally be dropped by a firewall, and those that make through

indicate that we're vulnerable? I have looked at the things the tool checks and those are mostly different extension combinations, different extension sizes, check if source routing allowed, check if multiple destination option headers allowed and so forth.

fake_mld6: Announce yourself in a multicast group of your choice on the net. Advertise/Delete yourself or anyone of your choice in anu multicast group. You can also query on a N/W who is listening to multicast addresses. fake_mld26 does the same thing but works with MLDv2.

Comments: I am not entirely sure if this works outside LAN. Need to study MLD in more detail.

fake_mldrouter6: Fake MLD router messages

Comments: I am not entirely sure if this works outside LAN. Need to study MLD in more detail.

smurf6/rsmurf6: Local smurfer, known to work only against linux at the moment (same as smurf6 but with src=FF02::1). ICMP echo request with src=Victim IP and dst=Target IP (default: FF02::1). Victim will receive echo requests from Target on behalf of the attacker.

Comments: Tried this on my test nw with default option (target=FF02::1) and really clogs the victim. Kernel dropped many pkts. Should do egress filtering so that our machines cannot attack anyone (per packet inspection=very expensive)? Also filter inbound pkts with multicast dst.

exploit6: Known ipv6 vulnerabilities (from CVE) to test against a target. For exploitable overflows, 'AAA' strings are used. Vulnerable systems crash (can be check via ping).

Comments: I don't know details about what these vulnerabilities look like but the more important question is, are these transient or expected to stay around for a while? We want to target problems that will be relevant for a reasonable period of time.

denial6: A collection of denial-of-service tests againsts a target. test1: Large hop-by-hop header with router alert and filled with unknown options. This attack affects all routers on the N/W path to the target. test2: Large destination header with unknown options.

thcping6: Sends a hand crafted ping6 packet

comments: What's the motivation?

4 SI6 Networks IPv6 Toolkit

4.1 Local Attacks

ni6: This tool allows the assessment of IPv6 implementations with respect to a variety of attack vectors based on ICMPv6 Node Information messages.

Comment: This looks LAN-specific. Need to check if this will work globally.

ra6: It allows the assessment of IPv6 implementations with respect to a variety of attacks based on ICMPv6 Router Advertisement messages. ra6 tool can be instructed to flood the victim with Router Advertisements from different sources (–flood-sources option), multiple packets may need to be generated.

rd6: It allows the assessment of IPv6 implementations with respect to a variety of attack vectors based on ICMPv6 Redirect messages. The tool can be instructed to flood the victim with Redirect messages from different sources (–flood-sources option), multiple packets may need to be generated.

rs6: It allows the assessment of IPv6 implementations with respect to a variety of attacks based on ICMPv6 Router Solicitation messages. The tool can be instructed to flood the victim with Router Solicitations from different sources (–flood-sources option), multiple packets may need to be generated.

na6: It allows the assessment of IPv6 implementations with respect to a variety of attack vectors based on ICMPv6 Neighbor Advertisement messages. The tool can be instructed to flood the victim with Neighbor Advertisements from different sources (–flood-sources option), multiple packets may need to be generated.

Comments: This looks similar to THC's flood_advertise6. The only difference i see is that this tool can use multiple fake src IPs for attack packets generated within the same attack session. THC seems to use the same fake src IP for packets in the same attack session.

ns6: It allows the assessment of IPv6 implementations with respect to a variety of attacks based on ICMPv6 Neighbor Solicitation messages. ns6 tool can be instructed to flood the victim with Neighbor Solicitations from different sources (–flood-sources option).

4.2 Global Scope Attacks

flow6: This tool performs a security assessment of the Flow Label generation policy of a target node. The tool sends a number of probe packets to the target node, and samples the Flow Label values of the corresponding response packets. Based on the sampled values, it tries to infer the Flow Label generation policy of the target. The tool will first send a number of probe packets from single

IPv6 address, such that the per- destination policy is determined. The tool will then send probe packets from random IPv6 addresses (from the same prefix as the first probes) such that the global Flow Label generation policy can be determined.

Comments: Why is it desirable for flow-label generation policy to be confidential?

frag6: frag6 is a security assessment tool for attack vectors based on IPv6 fragments e.g. provides option to specify IPv6 fragment payload size. Another option instructs the tool to determine the IPv6 fragment reassembly policy of the target. In order to determine the aforementioned policy, the tool performs a number of tests to determine how the target node processes overlapping fragments.

comments: Requires inspection of every packet (fragmented). Expensive– depends on prevalence of fragmented packets. Moreover, it is not an attack itself.

icmp6: It allows the assessment of IPv6 implementations with respect to a variety of attack vectors based on ICMPv6 error messages. It leverages the fact that unsough ICMPv6 error messages can be sent to hosts and malicious payloads can be embedded inside the error messages. For example, The ICMPv6 error message can embed an ICMPv6 Echo Request message with the Source Address set to 2001:db8:10::1 (i.e., Destination Address of the error message), and the Destination Address set to 2001:db8:11::2). This is useful for evading NIDS that drop echo requests at the border.

Comments: The manual mentions "The icmp6 tool supports IPv6 fragmentation, which might be of use to circumvent layer-2 filtering and/or Network Intrusion Detection Systems (NIDS)." How does IPv6 fragmentation helps in NIDS evasion? If they mean to send malicious exes over fragments so as to avoid VirusTotal kind of check, the upcoming file framework might be of some help in dealing with that. If the authors meant sending exploit payload over packet fragments, then Snort's signature detection will probably fail to detect it (i am not sure though if Snort does IPv6 packet reassembly). Bro, however, won't do anything any way as it does not rely on signatures.

jumbo6: This tool allows the assessment of IPv6 implementations with respect to attack vectors based on IPv6 jumbograms. Sends jumbograpm packet with payload size as speicified by user.

Comments: What's the attack?

scan6: It is an IPv6 host scanning tool. It implements a number of IPv6-specific host scanning techniques. If the tool is instructed to e.g. flood the victim with TCP segments from different sources (–flood-sources option), multiple packets may need to be generated.

Comments: This tool caught my attention. Scanning an Ipv6 address space is known to be a tedious task given the vast address space. I would like to look at the implementation.

tcp6: This tool allows the assessment of IPv6 implementations with respect to a variety of attack vectors based on TCP/IPv6 segments. For example, a SYN-flood attack against port number P of the host H, TCP connection-reset attack against all active TCP connections in the local network. The tool can be instructed to e.g. flood the victim with TCP segments from different sources (–flood- sources option), multiple packets may need to be generated.

5 implementation6.c

All ping requests have data = (41)+ All the tests and script thc_detect.bro can be found here citeimpl6-git.

5.1 Analysis

Analyzing attacks generated by implmentation6 with Bro				
Attack Description	Result	Analysis w Bro	Signature	Remarks
Test1: Sends 1 ICMPv6 pkt with 1 hop-by-hop option (header len 8 bytes)	Passed	Event 1 captures the information	Data=(01)+	-
Test2: Sends 1 (fragmented) ICMPv6 pkt with 1 hop-by-hop header (header len 2048 bytes (2kb))	Passed	Event 1 captures the information	Data=(01)+ or (02)+, option type = 31	-
Test3: Sends 1 ICMPv6 pkt with 2 hop-by-hop header (header len 8 bytes)	Passed	Event 1 captures the information	Data=(03)+, option type 31, option type = 31	-
Test4: Sends 1 ICMPv6 pkt with 128 hop-by-hop header (header len 8 bytes)	Passed	Event 1 captures the information	Data=(04)+, option type 31	-
Test5: Sends 1 ICMPv6 pkt with 1 destination header (header len 8 bytes)	Passed	Event 1 captures the information	Data=(05)+, option type 31	-

Test6: Sends 1 (fragmented) ICMPv6 pkt with 1 destination option (header len (2kb) 2048 bytes)	Passed	Event 1 captures the information	Data=(06)+, option type 31	-
Test7: Sends 1 ICMPv6 pkt with 2 destination option (header len 8 bytes)	Passed	Event 1 captures the information	Data=(07)+, option type 31	-
Test8: Sends 1 ICMPv6 pkt with 128 destination option (header len 8 bytes)	Passed	Event 1 captures the information	Data=(08)+, option type 31	-
Test9: Sends 1 (fragmented) ICMPv6 pkt with 2000 destination option (header len 8 bytes)	Passed	Event 1 captures the information	Data=(09)+, option type 31	-
Test10: Sends 1 (fragmented) ICMPv6 pkt with 8172 destination option (header len 8 bytes)	Passed	Event 1 not generated	-	Kernel was dropping packets, and apparently the last fragment was missed. Bro could not reassemble the pkt and hence the event was not generated. Wireshark confirmed that the last fragment is indeed missing.
Test11: Sends 1 (fragmented) ICMPv6 pkt – Correct fragmentation	Passed	Event 1 not generated	Data=(0b)+	Expectedly, Bro did not generate event 1 because apart from fragmentation option, there were no other extension headers. So after reassembly, there is no extension header to alert about.

Test12: Sends 1 (fragmented) ICMPv6 pkt – One-shot fragmentation which basically means to send a packet as a single fragment	Passed	Event 1 not generated	Data=(0c)+	Expectedly, Bro did not generate event 1 because apart from fragmentation option, there were no other extension headers. So after reassembly, there is no extension header to alert about.
Test13: Sends 1 (fragmented) ICMPv6 pkt – overlap-first-zero fragmentation	Failed - error reply	Event 1 captures the correct info (hop-by-hop extension)	Data=(0d)+ option type = 31	As per Wireshark, the checksum of the ICMPv6 packet is bad but the ICMPv6 error reply says bad parameter (unknown next header type). Frag header is followed by hop-by-hop extension
Test14: Sends 1 (fragmented) ICMPv6 pkt – overlap-last-zero fragmentation	Failed - error reply	Event 1 captures the correct info (hop-by-hop extension)	Data=(0e)+ option type = 31	As per Wireshark, the checksum of the ICMPv6 packet is bad but the ICMPv6 error reply says bad parameter (unknown next header type). Frag header is followed by hop-by-hop extension
Test15: Sends 1 (fragmented) ICMPv6 pkt – overlap-first-dst fragmentation	Failed - error reply	Event 1 captures the correct info (dest extension)	Data=(0f)+ option type = 31	As per Wireshark, the checksum of the ICMPv6 packet is bad. Frag header is followed by destination extension
Test16: Sends 1 (fragmented) ICMPv6 pkt – overlap-last-dst fragmentation	Failed - error reply	Event 1 captures the correct info (dest extension)	Data=(10)+ option type = 31	As per Wireshark, the checksum of the ICMPv6 packet is bad. Frag header is followed by destination extension

Test17: Sends 1 ICMPv6 pkt – source-routing (done)	Passed	Event 1 captures the correct info (routing header)	Data=(11)+	-
Test18: Sends 1 ICMPv6 pkt – source-routing (done)	Passed	Event 1 captures the correct info (routing header)	Data=(12)+	-
Test19: Sends 1 ICMPv6 pkt – unauth mobile source-route	Failed - no reply	Event 1 captures the correct info (routing header)	Data=(13)+	-
Test20: Sends 1 ICMPv6 pkt – mobile+source-routing (done)	Failed - no reply	Event 1 captures the correct info (routing header)	Data=(14)+	As per Wireshark, the checksum of the ICMPv6 packet is bad.
Test21: Sends 1 ICMPv6 pkt – fragmentation source-route (done)	Passed	Event 1 captures the correct info (routing header)	Data=(15)+	As per Wireshark, the checksum of the ICMPv6 packet is bad.
Test22: Sends 1 ICMPv6 pkt – fragmentation source-route (todo)	Failed error reply	Event 1 captures the correct info (routing header)	Data=(16)+	As per Wireshark, the checksum of the ICMPv6 packet is bad.
Test23: Sends 1 ICMPv6 pkt – hop-by-hop fragmentation source-route	Passed	Event 1 captures the correct info (routing+hop-by-hop header)	Data=(17)+, option type 31	-
Test24: Sends 1 ICMPv6 pkt – destination fragmentation source-route	Passed	Event 1 captures the correct info (dest+routing headers)	Data=(18)+, option type 31	-
Test25: Sends 1 ICMPv6 pkt – fragmentation hop-by-hop source-route	Failed error reply	Event 1 captures the correct info (hop-by-hop+routing headers)	Data=(19)+, option type 31	As per Wireshark, the checksum of the ICMPv6 packet is bad.
Test26: Sends 1 ICMPv6 pkt – fragmentation destination source-route	-	-	-	It looks exactly like test25. Looks like some mistake in THC code.
Test27: Sends 1 node information query	Failed no reply	No relevant event	Nonce: 1b 00 00 00 1b 00 00 00	Bro does not support rfc4620
Test28: Sends 1 inverse neighbor solicitation	Failed no reply	No relevant event	-	Bro does not support rfc3122

Test29: Sends 1 mobile prefix solicitation	Failed no reply	Event 2 (event mobile_ipv6_message(p: pkt_hdr)) not generated. Event 1 captures the correct info (dest header)	-	Bro does not support rfc3775
Test30: Sends 1 certificate path solicitation	Failed no reply	No relevant event	-	Wireshark says that the packet is malformed. Bro does not support rfc3971
Test32: Sends 1 ping6 with a zero ESP extension header	Failed error reply	Event 1 not generated	(1f)+	Bro does not generate event 1. Given that the pkt was malformed, Bro did not generate event 1. As per Wireshark, the victim generates an error reply saying: parameter problem–unrecognized next header type
Test33: Sends 1 ping from multicast (local!)	Passed	It depends on policy	(21)+	Alert for ping request from local multicast addrs
Test34: Sends 1 ICMPv6 pkt–frag+source-route to link local	Failed error reply	Event 1 captures correct info (routing header)	(22)+	As per Wireshark, the victim generates an error reply saying: parameter problem–erroneous header encountered
Test35: Sends 1 ICMPv6 pkt–frag+source-route to multicast	Failed error reply	Event 1 captures correct info (routing header)	(23)+	The routing header shouldn't specify multicast link-local address (ff02::1). As per Wireshark, the victim generates an error reply saying: parameter problem–erroneous header encountered

Test36: Sends 1 ICMPv6 pkt–frag+srcroute from link local (local!)	Passed	Event 1 captures correct info (routing header)	(24)+	Alert if receive packet with a link-local address
Test37: Sends 1 ICMPv6 pkt–frag+srcroute from multicast (local!)	Passed	Event 1 captures correct info (routing header)	(25)+	Alert if receive packet with a multicast link-local address (ff02::1)
Test38: Sends 1 direct neighbor solicitation	Passed	Not generated: event icmp neighbor solicitation	-	Bro did not generate neighbor solicit msg
Test39: Sends 1 direct neighbor solicitation ttl¡255	Failed no reply	Not generated: event icmp neighbor solicitation	(41)+, ttl=63	Bro did not generate neighbor solicit event
Test40: Sends 1 ICMPv6 pkt–filled ignore hop-by-hop option	Passed	Event 1 generates correct info	(28)+	The attack basically fills up hop-by-hop with a bunch of options with option type 31. Alert for option type 31
Test41: Sends 1 ICMPv6 pkt–filled padding hop-by-hop option	Passed	Event 1 generates correct info	(29)+	The attack fills up hop-by-hop with a bunch of PadN option data.
Test42: Sends 1 ICMPv6 pkt–filled ignore destination option	Passed	Event 1 generates correct info	(2a)+	The attack basically fills up hop-by-hop with a bunch of options with option type 31. Alert for option type 31
Test43: Sends 1 ICMPv6 pkt–jumbo option size ¡ 64k	Passed	Event 1 generates correct info	(2b)+	The attack fills up dest header with a bunch of PadN option data.
Test44: Sends 1 ICMPv6 pkt–filled padding destination option	Failed error reply	Event 1 generates correct info. Event icmp parameter problem not generated.	(2c)+	As per Wireshark, the victim generates an error reply saying: parameter problem–erroneous header encountered

Test45: Sends 1 ICMPv6 pkt–jumbo option size ¡ 64k, length 0	Failed error reply	Event 1 generates correct info. Event icmp parameter problem not generated.	(2d)+	As per Wireshark, the attack packet is malformed (ethernet frame check sequence incorrect) and the victim generates an error reply saying: parameter problem–erroneous header encountered.
Test46: Sends 1 ICMPv6 pkt–error option in hop-by-hop	Failed error reply	Event 1 generates correct info.	(2e)+	As per Wireshark, the attack packet is malformed (ethernet frame check sequence incorrect) and the victim generates an error reply saying: parameter problem–unrecognized IPv6 option encountered.
Test47: Sends 1 ICMPv6 pkt–error option in dsthdr	Failed error reply	Event 1 generates correct info.	(2f)+	Alert for option type 195. According to Wireshark, the victim generates an error reply saying: parameter problem–unrecognized IPv6 option encountered.
Test48: Sends 1 ICMPv6 pkt–IPv6 Payload length 0	Failed no reply	-	-	Alert for payload length 0 As per Wireshark, the attack packet is malformed (ethernet frame check sequence incorrect)
Test49: Sends 1 ICMPv6 pkt–too large length field (IPv6 payload length 255)	Failed no reply	-	(31)+	-

Test50: Sends 1 ICMPv6 pkt–too small length field (IPv6 payload length 60)	Failed no reply	-	(32)+	As per Wireshark, the attack packet is malformed (ethernet frame check sequence incorrect). Also, the checksum is bad.
Test51: Sends 1 ICMPv6 pkt–(ping) bad checksum	Failed no reply	-	(33)+	As per Wireshark, packet checksum is bad.
Test52: Sends 1 ICMPv6 pkt–(ping) zero checksum	Failed no reply	-	(34)+	As per Wireshark, packet checksum is bad. Do we want to generate notice for zero checksum?
Test53: Sends 1 ICMPv6 pkt–fragment missing	Failed no reply	-	(35)+	In Wireshark, the attack looks like a single fragment with More Fragment set to No. So i don't see the point in *missing* fragment.

5.2 Detection

5.2.1 ICMPv6 Payload Signature

```
signature thc-ping-sig {
  ip-proto == icmp
  payload /([0-9a-f][0-9a-f]){2,}/
  event "THC signature found!"
}
```

Status Not working.

Remarks: For ICMPv6 packets, the event signature_match is not generated at all. If the signature is modified to trigger for any ICMP packet and run with an ICMPv4 trace, the parameter data in event signature_match turns out to be empty. It seems that we cannot inspect ICMP payloads even for version 4.

5.2.2 Option type in Hop by Hop and Destination headers

Detection parameter: If option type in hop-by-hop or destination extension headers is equal to 31, it indicates THC. The option type is an 8-bit field that can take on values defined by IANA [**?**]. The option type 31 is undefined and specific to THC.

Status
Working.
Remarks
We can generally check if an option type is equal to an undefined value (not defined by IANA).

5.2.3 Routing header should not contain link local addresses

Detection parameter: In IPv6, source routing can be performed by using the routing extension header. For packets from an external network, it doesn't make sense to have link local addresses inside the routing header. In Wireshark, i can see the Type-specific Data field of the routing header as addresses. In Bro, however, p$ip6$exts[idx]$routing$data yields the following, which is not useful.

```
\0\0\0\0\xff^B\0\0\0\0\0\0\0\0\0\0\0\0^A
```

Status
Not working.
Remarks
Bro should be able to parse the addresses in the type specific data field of the routing extension header.

5.2.4 RType=0 in routing extension header

Detection parameter: The field RType in routing extension header should not be 0 (deprecated and DoS risk [**?**])
Status
Working.
Remarks
None.

5.2.5 Connection end points having link local IP address

Detection parameter: None of the connection end points should have link-local IP address.
Status
Working.
Remarks
We are performing this check for every connection, which could be expensive.

5.2.6 Other

Notice for high number of extension headers
We may want to generate notice/event for when the number of extension headers in a packet exceed a threshold T. Within a single packet, extension headers can be chained on and on. However, we are limited by path MTU. In this case fragmentation comes to our rescue. So the number of extension headers that

can be stuffed inside the same packet is limited by the fragmentation offset which is a 13 bytes field in the fragment extension header. I am not sure if we should do this at the scripting layer because that entails counting the number of extension headers for every single IPv6 packet.

Bad checksum

We can generate notice/event for packets with bad checksum. I am not sure how useful that would be because packets can have bad checksum because of all sorts of reason, not necessarily related to THC.

6 Conclusion

This is my conclusion!

References

[1] *World IPv6 Day*. `http://www.worldipv6launch.org/`

[2] *Lars Eggert IPv6 Deployment Trends*. `http://eggert.org/meter/ipv6`

[3] *THC-IPv6 attack tool*. `http://thc.org/thc-ipv6/`

[4] *SI6 Networks IPv6 Toolkit*. `https://github.com/fgont/ipv6-toolkit`

[5] *Bro Intrusion Detection System*. `http://www.bro-ids.org/`

[6] *Sheharbano-IPv6 security*. `https://github.com/sheharbano/ipv6-security`

[7] *Bro scripts for DNS security*. `https://github.com/sheharbano/dns-security`

[8] *IANA IPv6 parameters*. `http://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xml#ipv6-parameters-2`

[9] *RFC 5095*. `http://tools.ietf.org/html/rfc5095`

[10] *impl6 git*. `https://github.com/sheharbano/ipv6-security/tree/master/implementation6`