School of Electronics and Computer Science
Faculty of Engineering, Science and Mathematics
University of Southampton

Severin Gassauer-Fleissner

05.04.2010

Protecting large campus style IPv6 networks from exploitation attempts

Project supervisor: Dr. Tim Chown
Second examiner: “Ed” Zaluska

A project report submitted for the award of
MEng. Computer Science

Abstract

With the expiration date on IPv4 growing closer more and more networks across the globe are starting to prepare for IPv6. Some networks take the so called "dual-stack" approach where they run both a IPv4 and IPv6 network in parallel and often while their IPv4 network is perfectly secure they don't realise the importance of securing the IPv6 network separately. The purpose of this report is to highlight and analyse some of these issues with IPv6, potentially discover new ones and to develop a solution that can be deployed to detect, log and in some cases counteract these attacks. More specifically part of this projects purpose is to investigate which, if any, threats are currently being exploited on the departments’ network infrastructure.

Table of Contents

Acknowledgements

The author would like to extend his thanks towards Dr. Tim Chown who came up with the project idea and provided excellent support and feedback throughout the course of the project. Furthermore thanks go out to the ECS systems team especially John Wynn who played a vital role in enabling the deployment of IPolice6 as well as providing comments and ideas how to improve the project from an administrators point of view. A special thank you goes out to Kathryn, both for being patient and understanding throughout the course of the project, and for her extensive help with proof-reading this report.

Statement of Originality

The research contained within this report is the author's own work, with sources referenced as appropriate.

The project software is wholly written by the author, with the exception of the core python libraries and the separately mentioned packet capture and manipulation library Scapy.

1 Introduction

It has been known for years that the IPv4 address space will be exhausted eventually - recent IANA research has estimated this to be by the 18th of September 2011 [1]. As this deadline looms more and more institutions and even governments [2] have started to prepare for the switchover to IPv4's successor the IPv6 protocol. IPv6 is thought to solve the exhaustion problem with its 2^{128} wide address-space. However, contrary to popular belief IPv6 not only provides a much larger address space, but also introduces a number of changes to the way it operates. For example IPv6 replaces the ARP protocol with the Neighbour Discovery Protocol (NDP). A lot of these features, while considered mostly useful and beneficial, pose a number of security threats that many administrators are unaware of. In addition to these new threats a lot of the "old" vulnerabilities of the IPv4 protocol remain.

In theory a lot of these problems are solved by implementing end-to-end security such as that provided by IPSec which is integrated into IPv6. This project, however, is concerned with analysing threats in the context of an enterprise sized campus-style network such as the one deployed by ECS. The trouble with this kind of network is that not all participants stand under the jurisdiction of the network administrators and therefore End-to-End security measures such as IPSec can't be deployed easily due to public key infrastructure problems and other administrative issues.

1.1 Terminology

The following definitions apply throughout this report:
Packet - A block of Data as the Network layer sees it i.e. Layer 3 Header(s) + Payload.
Node - Any device on a link.
Router - A node that is capable of forwarding packets on to other nodes.
Host - Any non-router node.
RFC - Request for comment: a document issued by the IETF defining a protocol used on the internet
(D)DoS – (Distributed) Denial of Service: A type of attack which aims to deny service to or from the target e.g. by flooding it with requests until it accepts no more connections.
IDS/IPS software – Intrusion detection / prevention software: A type of software that can be deployed on a network to detect malicious use, often signature based.
IPolice6 – The IDS/IPS framework created as part of this project, also referred to as "the software" or "the framework" during this report.

1.2 Background Reading

The investigation started by reviewing the IPv6 Specification (RFC 2460) [3] as well as reading Hagen. [4] This gave an overview of the new IPv6 Address format, scope and types as well as a basic structure of the main IP header and all extension headers. To summarise: IPv6 addresses are 128bit instead of their 32 bit IPv4 counterparts. This makes for greater address space and is one of the main reasons why it is likely, that IPv6 will eventually be implemented.

IPv6 specifies three types and scopes for addresses. The types are unicast, multicast and anycast. While unicast and multicast behave like in IPv4, anycasts are a new type of address. The idea is that certain services share an anycast address and if a node wishes to contact a service it sends a request to the anycast address and the "nearest" server providing the requested service will answer. Note also that the specification no longer features broadcasts, as they have been replaced by various multicasts. IPv6 also introduces new scopes defining the area in which an address is valid: global, site-local and link -local. [5] [4].

The previously mentioned extension headers are a useful new feature in IPv6, replacing the options field of IPv4. These headers are linked by a "next-header" field, indicating what type the next header is, making packet processing easier and faster as most extension headers don't need to be processed by all nodes en route. There are a few noteworthy exceptions to this rule:

There is the "Hop-by-hop Options" header, which as the name implies is processed by each hop along a packets' path. This could be abused by an attacker wishing to DoS a router by sending packets containing a lot of hop-by-hop options. [6] [7] In general routers will do packet processing purely in hardware using highly optimised applications-specific integrated circuits (ASIC), but once a packet contains hop-by-hop options it needs to be processed in the CPU, which is much slower. Researchers have recently found a way to exploit this weakness. [8] A further issue is the existence of the IP Router Alert option, which instructs a router to inspect the content of a packet more closely. [9] This possesses the same problematic as the hop-by-hop options. Furthermore the specification does not restrict the number of header occurrences in a packet, which means attackers are able to craft very long and "CPU expensive" packets.

Another exception is the Extended-routing-header (RH0). This one is also dangerous, as it provides a similar functionality as the Lose Source routing option in IPv4, and comes with all the associated security threats. [10] EADS, an IT security company, gave a talk on the topic at a security conference highlighting the potential dangers of RH0. They demonstrated how attackers can detect high value targets such as DNS servers by sending a request containing RH0 to an anycast address. Their experiments showed that by doing this the response will not be sent from the server nearest to the host sending the request, but by the one nearest to the node specified as the last node to visit in the routing-header. This essentially defeats the security of anycast, as attackers can work out the whereabouts of all servers for a particular kind of service. A further concern of EADS was the fact that RH0 can be used to bypass firewalls, accessing inaccessible hosts on the inside of a network via a host which is accessible from the outside e.g. a DMZed webserver. Furthermore RH0 can be used to "bounce" a packet between two nodes for extended periods of time causing congestion on a link or even DoSing a target by sending a packet to the target via two or more nodes which exchange the packet between them any number of times before forwarding it all on to the target, thus increasing the effective bandwidth of the attacker. [11] These findings also agree with other research done on the topic [12] .This is made worse by the fact, that most routers and operating systems do not have an option for "switching off" Routing Header support. EDS claim that RH0 is of no actual use to anyone apart from attackers, and it is advised to disallow their use on the network. [13]

The security research group “The Hackers Choice” (THC) gave a talk at a security conference in 2005, in which they showed further weaknesses in IPv6, most of which are related to the Neighbour Discovery protocol (NDP). NDP (RFC2461) is equivalent to IPv4's Address resolution protocol (ARP) but features built-in ICMP Router Discovery / Redirect. ARP was a popular target for attacks in IPv4, as it had no inbuilt security. NDP works similar to ARP in that when a node needs to resolve an address, it sends out a Neighbour Solicitation (NS) request on the "solicited-node" multicast address of the target. The target is then meant to responds with a unicast neighbour advertisement message containing its link-layer address. However, there exists no measure to stop any node from falsely answering such a request with its own layer two address, thus enabling Man-in-the-middle attacks. These findings agree with the other research that has been done on this topic. [14] [12] [15]

Another feature of NDP is Duplicate Address Detection (DAD), which is used by nodes who have just acquired an address to ensure that their new address is not already taken. This feature works by sending out a Neighbour Solicitation Request (NS) to the address being checked. If the node sending the request receives more answers than expected, the address is considered a duplicate and will not be used. Attackers can abuse this mechanic to prevent new nodes from joining the network by falsely responding to NS requests thus preventing a node from ever obtaining a valid IP address. [13] [5]

The NDP Router solicitation and advertisement method poses another threat. [7] [13] [15] Routers on multicast capable links periodically send out packets indicating their availability various options such as the MTU to be used on the link and most importantly their link layer address. These are known as Router Advertisements (RA). The intended use for this feature is to facilitate what is known as Stateless-address-auto-configuration, which allows nodes to automatically obtain a link-local IP address as well as the address of the default gateway, without the need for a dedicated (DHCP) Server. [16] There is no security in place preventing anyone from sending these advertisements though, thus attackers can fool other nodes into believing the attacker's machine is the default gateway on the link. [13] [15] [7]

It was mentioned earlier that NDP also contains the redirect function known from ICMPv4. The idea of a redirect is to enable routers that receive a request from a client to redirect it to a "better" next-hop router. For security reasons a redirect request needs to contain as much of the original packet as possible, up to the point where the entire redirect packet does not exceed 1280 octets in total. However, security researchers have found that by sending a fake ICMP Echo request claiming to originate from an off-link node, then following up with a redirect claiming to originate from the gateway containing the corresponding ICMP response, a host will be tricked into believing this to be a legitimate redirect. [13]

Unlike IPv4's ARP, IPv6 protects itself from non-link local NDP messages by specifying that NDP messages need to have a Hop count equal to 255, which is the maximum allowed. A router receiving a packet with a hop count of 255 is required to discard the packet. Thus NDP messages can never be forwarded off-link. [14]

IPv6 also specified the well-known and popular ICMP(v6) Echo message, also known as PING. Experiments have shown that ICMP Echo requests can actually be used to DDoS a target by sending a forged ICMP Echo request claiming to originate from the target to a local multicast address, which will result in all the nodes listening to the multicast to respond with ICMP replies. [13] [6]

It is also noteworthy that there exists a solution to almost all of the mentioned problems in the form of the "Security Architecture for the Internet Protocol" (IPSec), defined in RFC2401. IPSec is capable of encrypting the header and or the payload of a packet using Authentication Headers (AH) (RFC 2402) or Encapsulating Security Payload (ESP) (RFC 2406) respectively. Both provide data origin authentication which would prevent most of the previously mentioned attacks. [17] [18] [19] [7] However, this project is concerned with securing large scale campus-style IPv6 networks. In this type of network not all hosts are directly under the network administrator’s jurisdiction typically. Therefore implementing IPSec is not feasible due to key management and other administrative issues [7], which means other solutions need to be explored.

After reviewing all of the above literature, it is apparent that there are a lot of security related issues with deploying IPv6. THC has released a toolkit featuring "proof of concept" exploits for some of the discussed threats. This project involved running these tools, and analysing their signatures.

1.3 Previous work

There exist various tools that attempt to combat some of the aforementioned and other vulnerabilities in IPv6, such as NDPMon and RAMOND. [20] Unfortunately, these past works tend to focus mainly on problems associated with NDP and ignore non-ICMPv6 related issues, such as Hop by Hop options and router alerts. Furthermore these programs are written specifically for individual vulnerabilities, whereas the solution presented in this report takes a modular approach. This ensures high customisability and extensibility. The way NDPmon uses XML files, to store configurations has inspired its use in this project.

1.4 Project Goals

Based on background reading and the inspection of previous the following project goals were agreed on:

- Research and summarise the most significant threats and exploits for IPv6 drawing on both existing research and results of experiments carried out during this project.

- Analyse each threat, and work out efficient and accurate means of detecting exploitation attempts, by running exploits and seeing if they have certain signatures.
- Investigate live traffic on the university network to find out which, if any weaknesses are being exploited at present.
- Develop a tool that is capable of detecting, reporting and possibly preventing exploitation attempts.
- If there is time at the end of the project, issue an internet draft aimed at educating professionals about the security issues that exist within IPv6, and provides a set of recommendations on how-to combat these.

1.5 Specification

Following the above and further discussions with the project supervisor and member of the ECS systems staff the following specification was drawn up as a basis to develop an IDS/IPS framework that fulfils all the software-related project goals:

1. Although this tool is meant to be a solution for large scale enterprises implementing this would be beyond the scope of this project so it was decided to focus on defining a framework suitable for use in a large enterprise that can easily be extended later on. Furthermore a prototype that is capable of detecting all of the NDP weaknesses discussed earlier.

2. The tool should be platform independent as far as possible, therefore it seems beneficial to separate the packet capturing from the processing logic and define a common interface between the two. The processing logic will likely be written in a language that is platform independent such as Java or Python while the capturing mechanism will likely need to be custom made for each operating system. This prototype will be primarily concerned with providing a capturing module for the Linux operating systems.

3. The framework should be easily extensible, modular and flexible enough to theoretically detect any type of threat.

4. The framework should be easily and flexibly configurable to make the creation and sharing of configurations simple, which could be facilitated by the use of XML.

5. The framework should provide various useful library functions to the developers of other modules.

6. The software should be as extensible and flexible as necessary while being as user friendly and easy to use as possible.

7. This specification is based on the goal of developing modules for the framework capable of detecting most of the attacks facilitated by the THC toolkit. [13]

2 Threat Analysis

This section outlines how the security threats were analysed what the finding were, and how they affected the final design of IPolice6.

2.1 Method

The purpose of the analysis was to run some attacks in a virtual test environment, to determine how they behave and what effect they have on the network. This was achieved by setting up the test environment as detailed in the next subsection. The attacks were run and observed on different machines, and their traces were analysed with the help of the network monitoring tool "Wireshark".

2.1.1 Test environment

The test environment was an IPv6 only internal network, comprised of four virtual machines. It was decided to use the Debian Linux distribution as operating system for all the machines because the author is most familiar with it. The machines used were:

- A "router"
  - This machine served as default gateway for all the other machines on the test network. This machine provided Router advertisements and IPv6 internet access via a 6to4 tunnel
- A attacker
  - This machine was used to compile and launch attacks from the THC toolkit
- A client machine
  - This served as "victim" to the exploitation attempts of the attacker
- A observer (IPolice6)
  - This machine is where all traffic was mirrored to, and where it was analysed. This machine would also eventually become home to the IPolice6 detection framework, which will be created later on in this project.

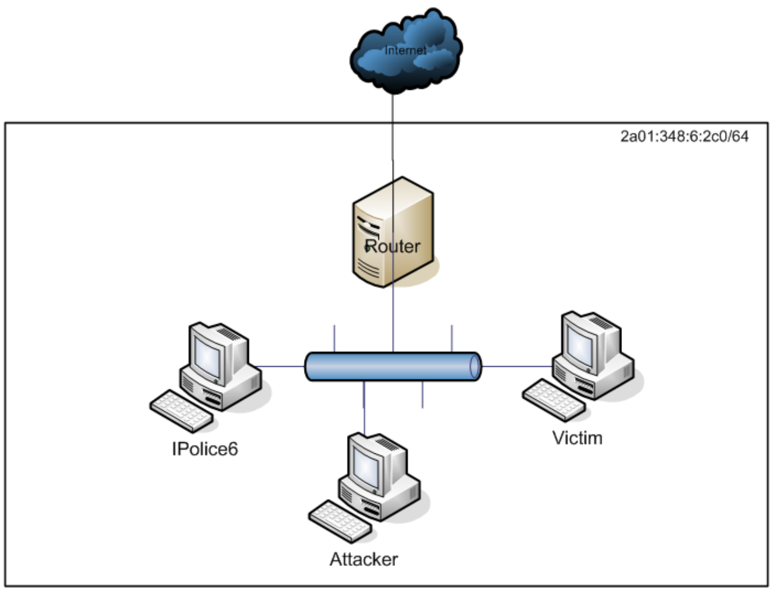
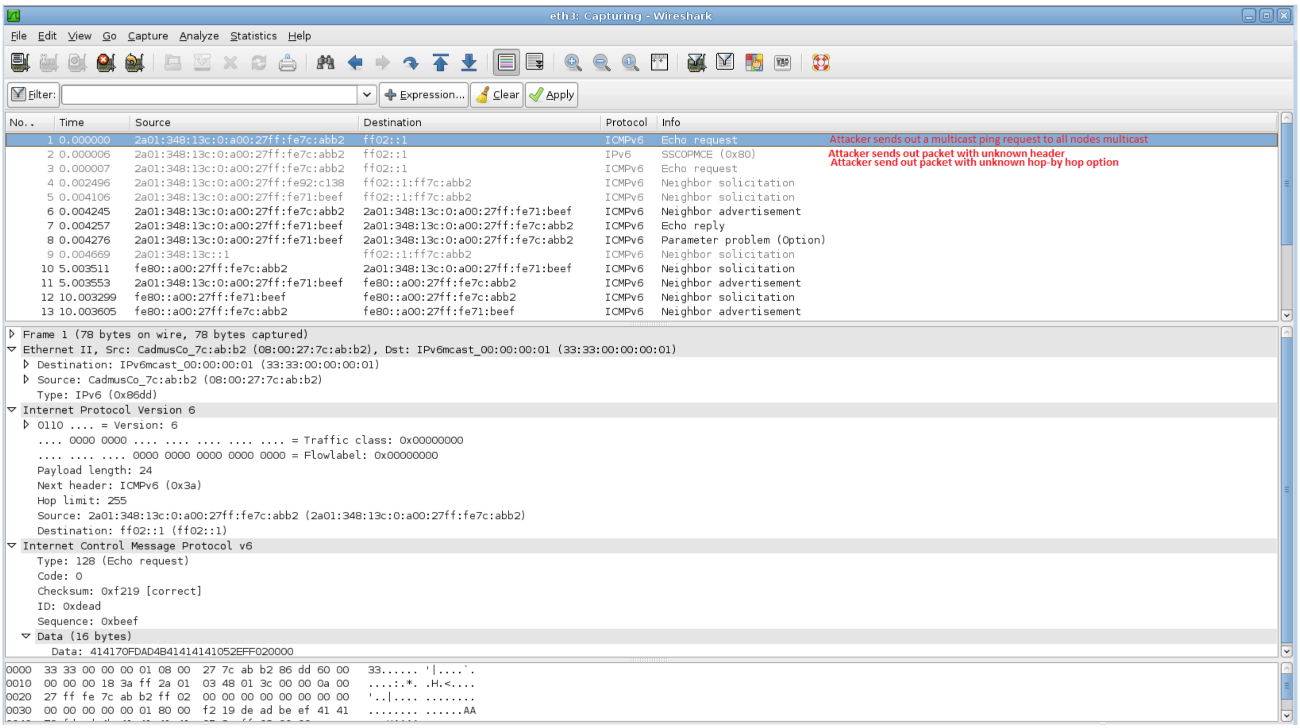


Figure 1 - Test Environment Topology

2.2 Analysis of attacks

The following section outlines some attacks, how they work and how they might be detected. This screenshot illustrates how Wireshark was used to track, trace and analyse attack signatures using the THC alive6 tool as an example.


eth3: Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Clear
Apply
Attacker sends out a multicast ping request to all nodes multicast
Attacker sends out packet with unknown header
Attacker send out packet with unknown hop-by hop option

```
No. .  Time        Source                              Destination                         Protocol  Info
    1 0.000000    2a01:348:13c:0:a00:27ff:fe7c:abb2   ff02::1                             ICMPv6    Echo request
    2 0.000006    2a01:348:13c:0:a00:27ff:fe7c:abb2   ff02::1                             IPv6      SSCOPMCE (0x80)
    3 0.000007    2a01:348:13c:0:a00:27ff:fe7c:abb2   ff02::1                             ICMPv6    Echo request
    4 0.002496    2a01:348:13c:0:a00:27ff:fe92:c138   ff02::1:ff7c:abb2                   ICMPv6    Neighbor solicitation
    5 0.004106    2a01:348:13c:0:a00:27ff:fe71:beef   ff02::1:ff7c:abb2                   ICMPv6    Neighbor solicitation
    6 0.004245    2a01:348:13c:0:a00:27ff:fe7c:abb2   2a01:348:13c:0:a00:27ff:fe71:beef   ICMPv6    Neighbor advertisement
    7 0.004257    2a01:348:13c:0:a00:27ff:fe71:beef   2a01:348:13c:0:a00:27ff:fe7c:abb2   ICMPv6    Echo reply
    8 0.004276    2a01:348:13c:0:a00:27ff:fe71:beef   2a01:348:13c:0:a00:27ff:fe7c:abb2   ICMPv6    Parameter problem (Option)
    9 0.004669    2a01:348:13c::1                     ff02::1:ff7c:abb2                   ICMPv6    Neighbor solicitation
   10 5.003511    fe80::a00:27ff:fe7c:abb2            2a01:348:13c:0:a00:27ff:fe71:beef   ICMPv6    Neighbor solicitation
   11 5.003553    2a01:348:13c:0:a00:27ff:fe71:beef   fe80::a00:27ff:fe7c:abb2            ICMPv6    Neighbor advertisement
   12 10.003299   fe80::a00:27ff:fe71:beef            fe80::a00:27ff:fe7c:abb2            ICMPv6    Neighbor solicitation
   13 10.003605   fe80::a00:27ff:fe7c:abb2            fe80::a00:27ff:fe71:beef            ICMPv6    Neighbor advertisement
```

```
▷ Frame 1 (78 bytes on wire, 78 bytes captured)
▽ Ethernet II, Src: CadmusCo_7c:ab:b2 (08:00:27:7c:ab:b2), Dst: IPv6mcast_00:00:00:01 (33:33:00:00:00:01)
  ▷ Destination: IPv6mcast_00:00:00:01 (33:33:00:00:00:01)
  ▷ Source: CadmusCo_7c:ab:b2 (08:00:27:7c:ab:b2)
    Type: IPv6 (0x86dd)
▽ Internet Protocol Version 6
  ▷ 0110 .... = Version: 6
    .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 24
    Next header: ICMPv6 (0x3a)
    Hop limit: 255
    Source: 2a01:348:13c:0:a00:27ff:fe7c:abb2 (2a01:348:13c:0:a00:27ff:fe7c:abb2)
    Destination: ff02::1 (ff02::1)
▽ Internet Control Message Protocol v6
    Type: 128 (Echo request)
    Code: 0
    Checksum: 0xf219 [correct]
    ID: 0xdead
    Sequence: 0xbeef
  ▽ Data (16 bytes)
      Data: 414170FDAD4B41414141052EFF020000
```

```
0000  33 33 00 00 00 01 08 00  27 7c ab b2 86 dd 60 00   33...... '|....`.
0010  00 00 00 18 3a ff 2a 01  03 48 01 3c 00 00 0a 00   ....:.*. .H.<....
0020  27 ff fe 7c ab b2 ff 02  00 00 00 00 00 00 00 00   '..|.... ........
0030  00 00 00 00 00 01 80 00  f2 19 de ad be ef 41 41   ........ ......AA
```

Figure 2- Wireshark trace for the alive6 attack

2.2.1 Checking for live hosts on a network (alive6)

This tool attempts to detect all IPv6 hosts on a network.

2.2.1.1 The attack

- Alive6 sends three packets to the all nodes local multicast address (ff02::1).
  - A valid ICMP echo request.
  - A "malformed packet" with an unknown header option.
  - An ICMP echo request containing an unknown hop by hop option.
- Every response received indicates a host.
- Also discover hosts that don't respond to ping requests, as all hosts should be sending ICMP Parameter problem messages in response to the "bad" packets.

2.2.1.2 Detection

- Hard to detect in general, though all-node multicast ping might be suspicious
- Alive6 sends two echo requests as well as a packet with an unknown header option.
- Alive 6 payloads are "4141801e744b414141410d4dff020000" and "8001801e744b414141410d4dff020000".

2.2.2 DoS via DaD (dos-new-ip6)

An attack used to prevent new nodes obtaining a valid IP Address by responding to any Neighbour solicitation requests sent.

2.2.2.1 The Attack

- Dos-new-ip6 sends spoofed Neighbour Advertisement in response to every neighbour solicitation request, thus causing the duplicate address detection test to fail

2.2.2.2 Detection

- It might be possible to trick the attacker into showing himself by sending a series of fake solicitation requests containing bogus IP addresses.
- If a Neighbour advertisement is received in response to all the requests there is a good chance that someone is exploiting this attack vector.
- If the attacker fakes MAC it’s hard to trace the source of the attack without switch table information.

2.2.3 Fake node (fake_advertise6)

An attacker announces himself with an IP address and MAC of his choosing, which allows him to “join the network” impersonating any device.

2.2.3.1 The attack

- The attacker sends out a ICMPv6 Neighbour advertisement containing an arbitrarily chosen IP and MAC to the all nodes multicast address

2.2.3.2 Detection

- Unsolicited advertisement can be detected by checking that any advertisement is preceded by a corresponding request. However attackers can fake the solicitation request, also unsolicited advertisements are permitted by the specification.
- If the attacker fakes MAC Address this attack is hard to trace.

2.2.4 Joining multicast groups (fake-mld6)

Nodes can arbitrarily join any multicast group on the network regardless of whether they are meant to or not. Attackers can gain further insight into the network internals, for example by joining the multicast group for a routing protocol.

2.2.4.1 The attack

- The attacker sends a multicast listener report to the link local multicast routers containing multicast address he wishes to join.

2.2.4.2 Detection

- Very hard to detect, as this works as intended.
- It might be possible to define a set of allowed multicast groups on a link and monitor for unauthorised joining attempts (a lot of administrative effort).
- Alternatively routers could be configured to refuse forwarding certain groups.

2.2.5 Fake router advertisements (fake-router6)

The attacker sends out an arbitrary routing advertisement in an attempt to define the default router for the network. This can lead to a Man-in-the-middle situation, as well as effectively DoSing the network by advertising a non-existent router.

2.2.6 The attack

- The attacker sends an ICMPv6 Router advertisement to the all link local all nodes multicast address.
- This RA contains an arbitrary prefix, router link local address as well as an infinite valid-lifetime in order to get precedence over all the other routers.

2.2.7 Detection

- Only allow certain prefixes.
- Statically define layer2 addresses of valid routers.
- Check for high (infinite) valid-lifetimes.

2.2.8 Counter

- Generate and send a spoofed RA with a lifetime of 0 which effectively deprecates it. [21]

2.3 NDP Man-In-The-Middle attack (parasite6)

This is the counterpart to IPv4 ARP spoofing. The attack puts an attacker as Man-in-the -middle by sending fake neighbour advertisements.

2.3.1 The attack

- The attacker answers any neighbour solicitation requests with a fake neighbour advertisement.

2.3.2 Detection

- It is difficult to detect without having access to switch data, due to fact that attacker can use fake MAC addresses.
- Administrators could limit amount of IP’s a single node (MAC address) may claim to own. This would prevent the MITM component of the attack, but not the DoS, as attacker can use a different MAC every time.

2.4 Forged Router Redirects (redir6)

This attack abuses ICMPv6’s router redirect feature, to implant a next-hop router for a certain destination into the victim’s NDP cache.

2.4.1 The attack

- The attacker sends a forged ICMP Echo request to the victim claiming to be sent from the target.
- The victim will attempt to respond with an ICMP Echo Reply

- The attacker sends an ICMP Redirect message claiming to be from the “real” next hop router advising the victim of a “better” next-hop router to use
- This redirect will appear valid as the attacker knows that the content of the packet sent by the victim must have been the ICMP Echo reply.

2.4.2 Detection

- This is difficult to detect due to MAC spoofing.
- Administrators may flag all redirection attempts as suspicious.
- Might be possible to catch out novice attackers by testing if advertised Router is part of the all router multicast group.

2.4.2.1 Flooding (smurf6)

This is an attack that generates a lot of traffic on the local network.

2.4.2.2 The attack

- The attacker sends a forged ICMP echo request to the all nodes multicast address claiming to originate from the victim.
- This results in all nodes on the network sending ICMP echo responses to the victim

2.4.2.3 Detection

- A continuous stream of ICMP echo requests from one source to the all nodes multicast address may appear suspicious.

2.4.3 Causing heavy CPU load on target (sendpees6)

Attempts to use the SEND protocol to generate a lot of difficult (CPU expensive) packets to the victim as a form of DoS.

2.4.3.1 The attack

- The attacker sends packet with a long RSA key and CGA options to the target
- The target is under pressure to verify signatures and CGAs, which takes up a lot of CPU cycles.
- This attack is ideally aimed at a router that is already under high pressure.

2.4.3.2 Detection

- Flag up a flood of identical SEND packets from the same source featuring non-standard key lengths.

2.5 Abusing Routing Header 0

Attacker can use Routing header 0 which enables loose source routing functionality. This has already shown to be a major security flaw in IPv4, [10] and its use is widely discouraged. As described earlier loose source routing essentially defeats the purpose of the IPv6 any-cast security mechanism. Furthermore it allows the attacker to bypass firewall systems by “bouncing” their traffic via trusted DMZ nodes (e.g. webservers). It is for these reasons that it is generally discouraged and a lot of security experts are of the opinion that it should be disabled. [11]

2.5.1 The attack

- There isn't one attack as such but there exist a variety of ways in which this facility may be exploited.

2.5.2 Detection

- The detection is very simple, all that needs to be done is check for presence of Extension Header.

2.6 DoS using excessive Hop-by-hop options

As discussed earlier IPv6 features Hop-by-hop options, which are meant to be examined by every node on a packets path from source to destination. These options are often complex and cause the packet to be processed in software rather than hardware, which is a lot slower. Attackers can abuse this method to DoS all the routers between them and an arbitrary source by sending packets with a lot of complicated HBH extension headers. This is potentially very dangerous in conjunction with the aforementioned Loose source routing, as this enables attackers to specify the systems which should receive these complex packets. These threats are part of the reason why security experts are calling for the removal of HBH options from the IPv6 specification. [8]

2.6.1 The attack

- The attacker crafts long complex packets and sends them.
- Might be done in conjunction with source routing.

2.6.2 Detection

- The detection of malicious use is not easy as Hop-by-hop options are part of the protocol specification.
- It might be possible to check for sanity of the options and make an educated guess if options are legitimate.

2.7 Conclusion and effect on initial design

Having run these tests, and analysed the traces it becomes apparent that most of these attacks are relatively straight forward to detect and can be picked up by passively scanning the network. However, there are attacks such as the DaD-DoS which can only be detected by proactively probing the network. This meant, that the initial design had to be changed from an event driven, single threaded approach to a multi-threaded one. Furthermore modules will have to have a concept of “state”, to track flows enabling the detection of multi-phase attacks.

3 Design

The application has undergone several changes to its design during the course of the project. This sections purpose is to outline and explain the design process and all the decisions that were made along the way.

3.1 Initial

The initial design was the result of a brief analysis, and gave a rough idea of what the finished prototype might look like. The focus was on developing something very extensible, modular and flexible in a short amount of time. Back then the software, IPolice6, was thought to be single threaded and event-driven. Furthermore the capturing logic and the processing logic were separated. This was decided so that capturing could happen at the very bottom level of the kernel, even before the packet hit the network stack. This would have the great advantage that malicious packets could be dropped at the networks perimeter, thus actively shielding the network from attacks. The following graphic illustrates how this might be deployed.

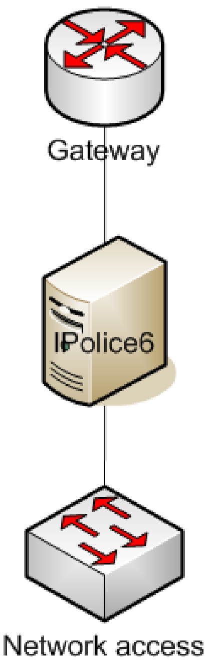


Figure 3 - Original Deployment topology

Extensive amounts of time and effort went into developing a kernel module capturing unit and eventually a working prototype was implemented. However, getting this kernel module to communicate with the user-land application proved to be too difficult to do in the short time available. This was due to the fact that material available on the topic was hard to understand and out of date [22] [23].

The following diagram shows the initial UML diagram. It is important to note that at this point design was still all very theoretical and programming language independent.

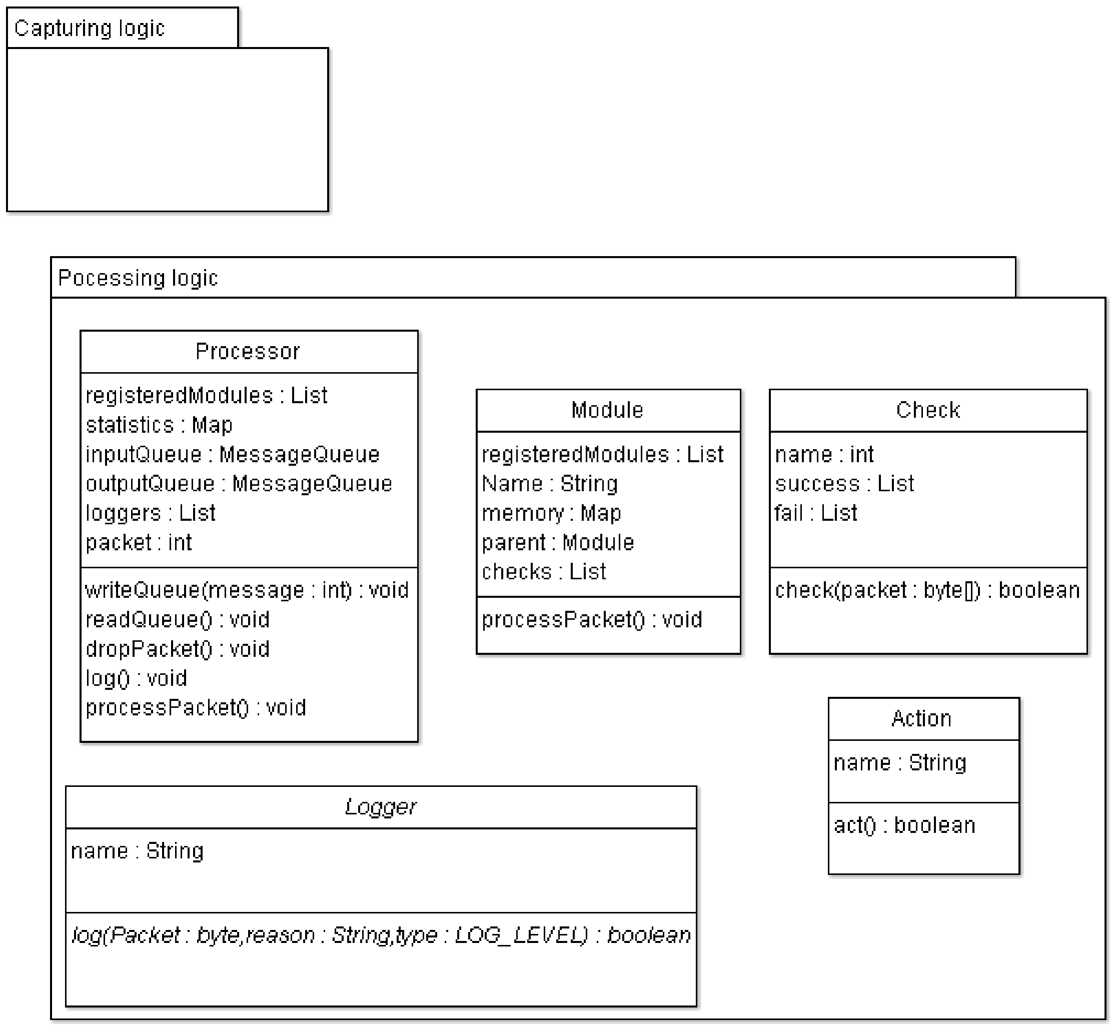


Figure 4 - Initial class design

3.2 Refined design

After it had become apparent that the initial design with its separated capturing and processing logic had failed the design was adapted to deal with this. This meant that it would no longer be possible to “drop” packets from the network stack, meaning that IPolice6 could be deployed passively. The solution was still thought to be event driven at this point as the idea of proactively probing the network had not been considered yet. At this point it was decided that python would be used as the primary programming language, for reasons detailed in the next section of this report.

The illustrations below show the abstract class diagram for the revised design, as well as the revised deployment scenario.

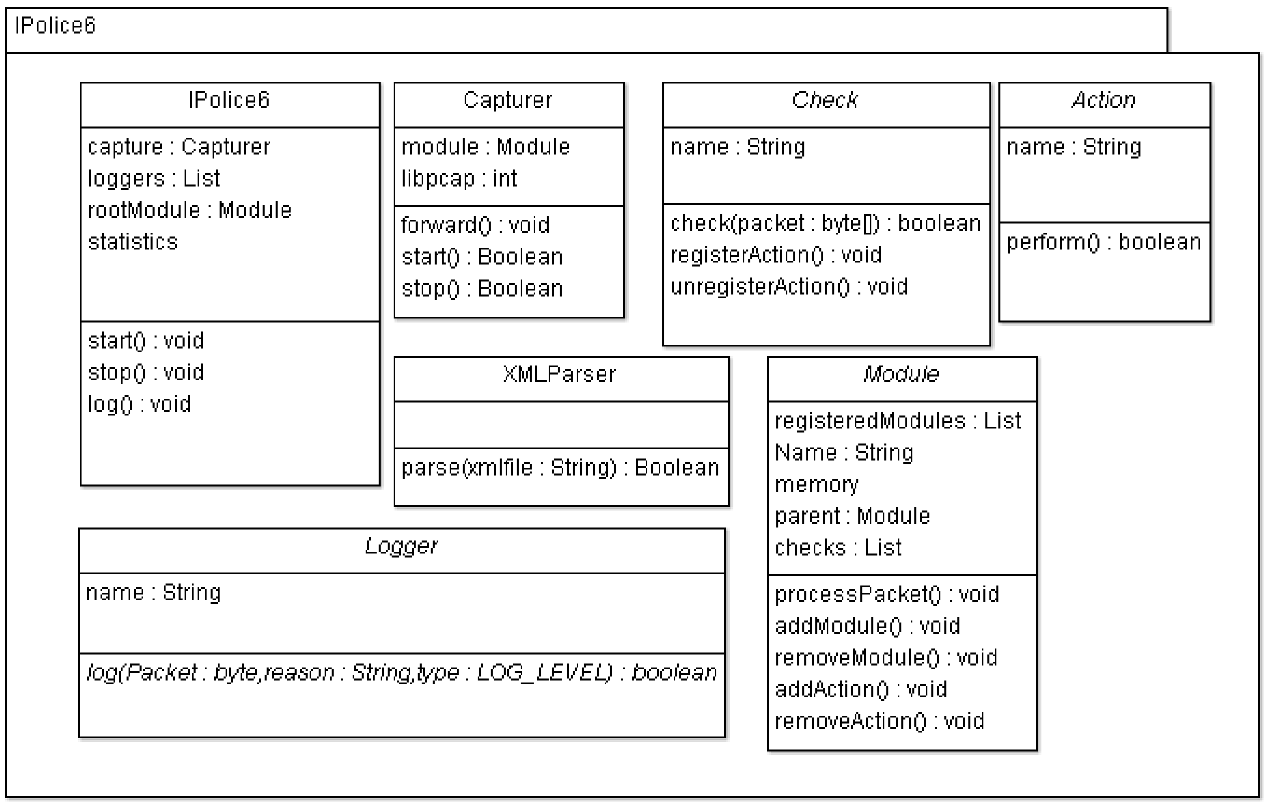


Figure 5 - Revised Class diagram

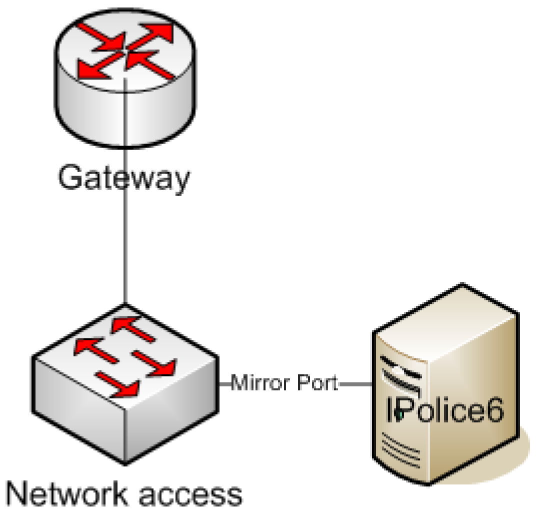


Figure 6 - Revised deployment topology

3.3 Final design

The final design features both a multi-threaded solution, where capture is independent of processing, and proactive scanning. The paradigm employed is that of the classical producer-consumer architecture, where the capturers act as producers and the processor as a consumer. Communication happens via a thread-safe queue python language-construct. Furthermore it was decided to remove the distinction between positive and negative actions, because experience had shown that when a packet passes a module and is not an exploitation attempt no action is required, or in fact desired. Refer to Appendix A for a detailed sequence diagram outlining the final design.

4 Development and Implementation

The following section describes how the software was developed and went from the final design to the prototype presented later on.

4.1 Development style

It was originally intended to employ an evolutionary approach to design. This was adhered to until it became apparent that the kernel module approach had failed and had offset timing considerably. For this reason it was decided to switch to the Rapid Application Development (RAD) design methodology. RAD is a very flexible technique that works well for small projects with only one developer. [24]

RAD offered the flexibility and freedom needed to develop the application as research progressed and new ideas manifested themselves. RAD is often used to quickly create a working prototype of an application, which is what was required in this case.

One of the often cited disadvantages of RAD is that it relies on heavy end-user involvement and requires excellent communication skills between development team members. However, since the author was the only developer on this project and he stood in constant communication with the end-user via email and weekly meetings, these drawbacks were considered to be non-issues in this case. [25]

It was also deemed best to adopt a “bottom-up” approach to the development of the software. This meant initial efforts would focus on getting the core of IPolice6 in place and once that worked add-ons and modules would be developed on top of it. This approach also ensured that there would be a working system at any time, because even if an individual module broke the overall system would remain intact.

4.2 Source code management

In software projects it is important to have some form of source and version control along with a change-log. Though more important in team projects, it is still vital to have this functionality so changes and progress may be tracked, and reverted if need be.

The typical tool of choice is Svn. The author is familiar with Svn and has used it successfully in the past. However, there is currently a lot of hype about a new tool by the GNU developers by the name of Git which attracted the author’s curiosity. Git takes a novel approach to version management in that it hasn’t got a central repository like traditional solutions, but instead each copy of the source folder serves a repository in its own right. There is also no special server required. Git keeps track of changes in the project directory itself in a special .git folder which makes sharing and copying of repositories easy. [26]

4.3 Choice of language and tools

The choice of language was influenced by a few factors:
The first and foremost consideration was speed and ease of use. Time was a pressing matter and therefore the simpler the language the better. Traditionally tools like this tend to be written in the c programming language, as it gives good performance and is very near to the operating system. Nevertheless since the idea of an active network component had been abandoned performance wasn't that much of an issue any longer, as it wouldn't matter if intrusions were detected and reported a few seconds late as long as they are reported and net-flow remains unobstructed.

Of the numerous languages that fit these requirements the author only felt proficient enough with two: Java and Python. Each of these languages has its merits, but Python has the great advantage of being a scripting language, which allows runtime generation, evaluation and execution of program code. Python has one major drawback though; it's not very near to the operating system and natively does not support any low level network support. Fortunately there exist various third party libraries that provide wrappers to these low-level networking functions. One of these is the very powerful packet capture and modification library "Scapy".

Scapy is a powerful and flexible library developed by French security researchers. It takes a somewhat novel approach to packet capture and manipulation because it treats packets as "layers" rather than just data. When it captures a packet, it automatically analyses and dissects the packet into its individual layers. IPolice6 can make use of these features to check for the presence of certain headers (layers) and values therein. Scapy has included IPv6 support since *rev 916*, which makes it a suitable choice for this project.

Although the prototype presented in this project makes ample use of Scapys features, it is important to note that IPolice6 itself is in no way reliant on Scapy per se. Developers can write modules and plugins that do not require Scapy as long as they fulfil the required interface. Nevertheless it was decided to make use of Scapy for all the modules developed during this project because it saved IPolice6 from having to work with packets at byte level. This greatly aided the process of finding fields and headers within the chained header structure of an IPv6 Packet. [27]

As this software was developed in python it was deemed appropriate to adhere to the relevant programming style guide as define in PEP8 [28]. Adhering to these conventions should ensure that all code produced is easy to read, understand and reuse by other developers.

4.4 System Overview

As mentioned before the aim of this project was to produce a flexible and extensible IDS framework. This is ensured by providing a very loose specification as to which interfaces the modules need to fulfil to work with the system. The main components in IPolice6 are:

- IPolice6Module
    - Active

    - Passive
- IPolice6Check
- IPolice6Action
- IPolice6 (main)
- Capture
- Configuration parser

Users can register components by calling the appropriate register method on its “to be parent”. Users can write and add plugins as long as it fulfils the appropriate module interface. These interfaces are defined in the baseclasses python module. Alternatively developers may simply inherit from the baseclasses. If a module is added that does not fulfil the required interface it is not added, and an exception is raised.

4.4.1 IPolice6

Modules, capturers and loggers register with this. This application is responsible for processing the packet queue, which is filled by the capturing units. It does this by taking a packet from the queue and then in turn passing it to all the modules registered with it. The main application is also responsible for invoking all of the active modules.

4.4.2 Capturer

A capturer should be its own thread and contain a means of capturing a packet from the network and placing it in the IPolice6 processing queue. All capturers need to fulfil the SimpleCapturer interface in order to be considered valid.

4.4.3 IPolice6Module

Modules are designed to represent a unit that checks for a type of exploit, and they do so by invoking a series of checks. Modules have a set of actions registered with them which will be executed if the end result is positive. Modules can also register other modules to build a hierarchy if needed. Modules provide “memory”, which all its children have access to. This enables developers to keep track of the “state” of an attack in a module over time and is used to detect multi-stage attacks, such as the router redirect.

For a class to be a valid module it needs to provide at least all the modules specified in the Ipolice6Module class. Modules work by passing a packet to all the checks registered with it until there is either a positive or negative response. All modules need to fulfil the IPolice6Module interface.

Active modules are an extension of passive modules and in addition provide a private *_do_execute* method which contains the code that is run when the module is scheduled for execution. Active Modules need to contain at least all the methods specified within “IPolice6ModuleActive”

4.4.4 IPolice6Check

Single checks are represented by a Check object. An example for a check would be to see if a packet contains Hop-by-hop options. Checks are meant to be chained together in modules and can be configured to determine what a pass or fail of a check means. The default settings are that if a packet passes a check to just pass the packet on to the next check whereas a check failing constitutes an immediate negative response. All checks must fulfil the IPolice6Check in order to register successfully.

4.4.5 IPolice6Action

Actions are classes containing at least a "execute(packet)" method. This method contains the code to be executed when the action is invoked. This can be anything from; a simple call; to a logging function; up to creation and sending of a response packet. This is possible as the execute function gets passed the entire packet as a parameter. Actions need to fulfil the Ipolice6Action interface to be considered valid.

4.4.6 Configuration

XML has enjoyed an increasing popularity over recent years for the purpose of storing system configurations. XML is relatively easy to read for humans and well structured. Furthermore, there exist a plethora of excellent XML parsing libraries which makes its use both simple and convenient. The main reason why it was decided to use XML was a brief discussion the author had with John Wynn, a member of the ECS systems staff, who mentioned how annoying he found it that most tools feature their own syntax and configuration file structure and it would be nice to have something universally readable and understandable for once.

4.5 Feature Implementation

As discussed earlier it was decided to implement the system in a bottom-up fashion starting with the core and slowly working up towards the more advanced features.

4.5.1 Packet capture

The first thing to be done was to develop a packet capturer. This was initially done using Scapys "sniff()" facility. One challenge was dealing with continuous streams of traffic while still being able to notify the capturing unit to terminate from the main application. The issue was solved by supplying a stop_filter argument to the sniff method. This argument is an inline function, which is executed after each packet has been captured, and depending on whether the return value is True or False, causes capturing to continue or terminate. Furthermore a time-out of one second was added so that if no traffic was captured the system could still check the shutdown condition ensuring a safe and tidy shut-down every time.

The module was tested by sending pings at an interval of 0.l seconds from a test machine and verifying that the count of captured packets displayed by the capturer was correct. This worked as expected. However, there was some initial confusion because the display always showed double the amount of packets expected. It was soon discovered to be correct behaviour as this count included packets sent as well as received.

4.5.2 Core

Once the capturer was in place and working as required, work on the main class began. The main class is the heart and brain of the system, and holds a list of active and passive modules which are invoked whenever a packet is processed. Furthermore it provides a central logger which can be used by any module. The main class provides a thread safe queue for capturers to write to, and it runs in an infinite loop processing the contents of this queue as well as running all the active modules.

4.5.3 Plugins

With the core functionality in place the creation of some modules began. The first was a proof-of-concept demonstration capable of detecting the presence of the THC signature “0xdead” “0xbeef” as sequence and id within an ICMPv6 Packet respectively. The model comprised of two checks and one action. The first check checked for the presence of an ICMPv6EchoRequest message within a packet. The second check assumed that an ICMPv6 Header existed and attempted to access the sequence number and id and compared them to the THC signature.

The action was a simple LogAction that printed a warning message to the screen whenever such a packet was detected. Once finished, it was tested manually by crafting a packet matching the THC signature and sending it. This packet was detected and the test was followed up by running the corresponding THC tool (alive6), which was detected too.

The next step was to create the GenericFieldCheck (GFC). This class can be configured to check for the presence of any field and value within a packet, and several of these may be linked together in a module to implement a classic simple static packet filter. GFC can be supplied with either a list of values or a single value to compare the field value with. Furthermore by setting the checks pass and fail actions appropriately it can be used to test for either match or mismatch.

The functionality of GFC was tested by making up some “bad packet” signatures and hand-crafting packets which would trigger them. This worked sometimes but not always, and closer investigation showed the problem to be the shorthand notation for IPv6 which could get confusing when doing a direct comparison between two addresses. This prompted the creation of the *get_ip6_padded* utility function, which takes a shorthand IP address and expand it out to be the full 128bit representation by utilising regular expressions. Once all addresses were run through this function prior to comparison all the tests succeeded.

GenericFieldCheck also proved to be a great aid later on when creating more complex and sophisticated modules later on in the project. This is an excellent demonstration of the systems reusable component model. Anyone can write a plugin for IPolice6 and others can then incorporate it into their own components or collections thereof.

Once GFC was implemented work on more complex modules could begin. These modules tested for threats based on the discoveries outlined in an earlier section of this report. All these modules were tested by exposing them to both the attack tool as well as the manually crafted packets. Furthermore, Wireshark continued to be used to verify system behaviour. A lot of threats could be easily detected by using GFC alone though some more complex attacks required the creation of specific modules and tests, an example of this is the redirect6 module, which needs to keep state information about what packets it has seen before the current one.

4.5.4 Configuration

The system is configured via XML files which are parsed by the ConfigParser class on start-up. This class makes use of the python XML minidom library to parse and verify the configuration before proceeding to recursively parse the document. Since the entire configuration is a tree structure where each Module has a reference to its parent the parser uses a stack to keep track of the current “parent”, to be supplied to a module on its creation.

The parser makes use of Pythons introspection features to implement a generic dispatcher method which will automatically detect and deal with an XML element as needed. All the node processing functions are named _parse_nameofnode, so if a new type of element needs to be added to the XML then all that needs to be done is to write the appropriate function and the dispatcher will automatically invoke the right method.

Another noteworthy method within the parser is the _do_component method. This method instantiates and configures a component. It does this by attempting to get and resolve the Object type by using Pythons eval facility to get the class. A nice feature here is that the _parse_type function will attempt to dynamically import the requested component if it isn’t imported already. This is achieved by making use of regular expressions in conjunction with pythons “exec” statement, which allows the creation and execution of program code at runtime.

Another feature is that thanks to pythons’ ability to generate and execute code at run-time, users can write code as part of the configuration between <eval> XML elements causing the code to be executed at runtime. This is useful for example to have a configuration dynamically bind itself to the right interface on a machine.

Although the XML files created can get quite large, detailed and complex this is not a problem as it is always possible to abstract this complexity away, for example by creating a GUI application that allows users to easily build configuration files. This however was considered to be beyond the scope of this project.

5 Final Testing

Due to the nature of this project and the development style adopted tests were conducted on an on-going basis all the way throughout the development. This can be seen from the previous section on implementation, as every new module added was tested on completion as well as during development to ensure that progress was being made and the module was developing in the right way. Nevertheless it was still important to test the entire system more formally on completion.

Since there are two parts to this project testing will need to focus on both individually. For one there is the test of the framework and its configuration to check that it does what it is supposed to do, which was done in the beginning when the framework was first created. The other part is to test and assess the effectiveness of the modules developed in detecting known attack vectors especially how they work together.

The details of the most important/interesting tests can be found in Appendix B.

5.1 Test Environment

Development and testing were conducted using the same environment used for threat analysis. Furthermore, Wireshark was used to verify and debug the behaviour displayed by the system.

5.2 The test

To test the entire system a relatively complex sample configuration (Appendix C) was devised and loaded into the system. The first test was to verify that all the modules were loaded and configured as expected. This was done manually by starting the program with a debugger and setting a breakpoint right after parsing had finished. Each module and sub module was inspected manually to make sure all variables were set, and all modules had imported correctly. Once this test was passed the main loop of the program was allowed to run. The program was left in this state for a few minutes while monitoring memory and CPU usage to ensure it was behaving properly. This is where it became apparent that the performance was not acceptable, as CPU usage averaged at about 98% constantly. At first it was assumed the problem was due to inefficient processing but after several tests and testing the program with no traffic on the link at all, it turned out that the problem was caused by a badly implemented infinite loop in the main thread. To clarify: The main thread doesn't do anything once the program is running, all it does is initiate the other threads at start-up as well as running the configuration parser. Past that point it sits in a tight loop until the keyboard interrupt exception is raised, at which point it initiates a clean shutdown of the software, and the problem was this loop took up all of the CPU's resources without doing anything useful. The solution was to suspend execution for a second every iteration. Once this fix was implemented CPU usage returned to acceptable levels again (~3%). The next step was to launch exploits one by one to see if they were recognised. This was mostly done using the THC toolkit, although some testing was done manually by using Scapy in "interactive mode" to craft and send packets. As always Wireshark was also used to verify that program behaviour corresponded to the traffic on the network. The tools tested were alive6, dos-new-ip6, fake-router6, parasite6, redir6 and toobig6 from the

THC toolkit as well as some Scapy test scripts that would trigger RoutingHeader0 and excessive Hop-by-hop Options which were created extra for this test. All of these tools were detected, which was expected because all the individual modules had been tested individually during their creation and they were expected to behave in the same way once they were part of a more complex configuration.

N.B: The test scripts used can be found in the *“testscripts”* directory on the CD accompanying this project report.

6 Deployment

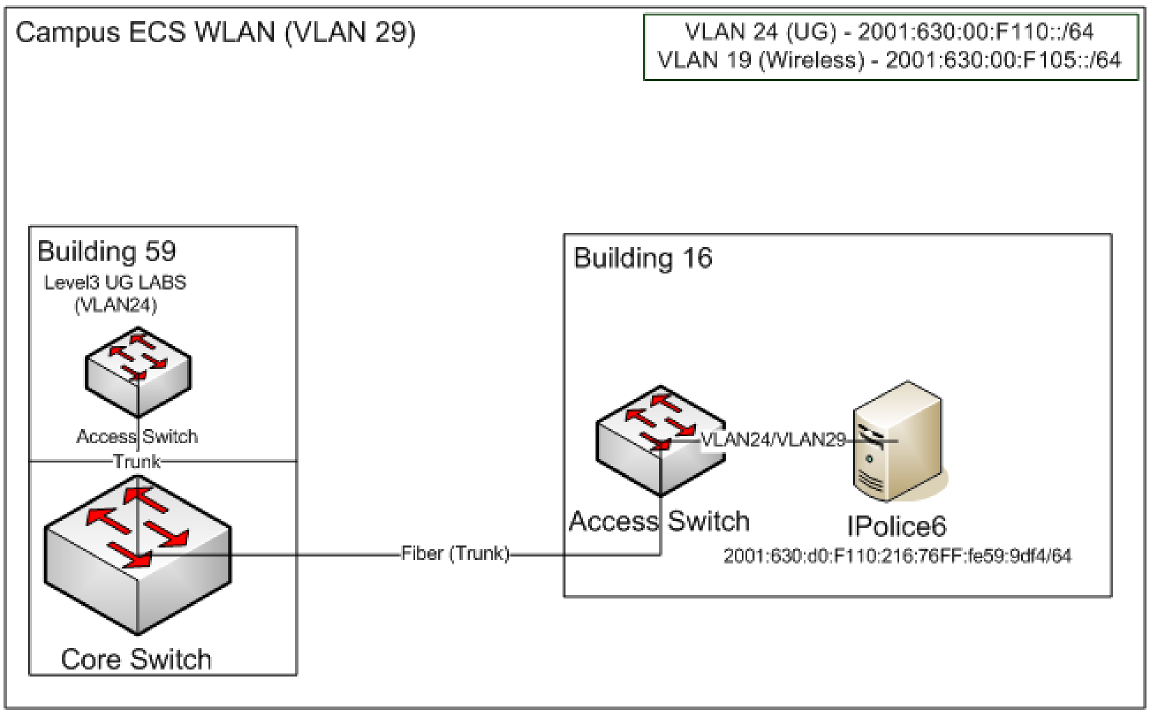
One of the focuses of this project was to research and analyse which, if any, vulnerabilities in the IPv6 protocol are currently being exploited on the departments' network. For this purpose the IPolice6 software was deployed and configured on a real physical test machine, which was then added to the ECS network with the help and permission of the system administrator. Initially the system was added to a monitoring port and VLAN on a switch handling all the traffic in the undergraduates' lab network. To great surprise this did not show any indication of exploitation attempts so to verify the system was still working the harmless "alive6" tool was launched. This was detected by the system as expected, which lends itself to the conclusion that the system was working and indeed no one was running any exploits on the network at present.

In hindsight this makes sense as most of the machines on the lab network are controlled and maintained by ECS, and it is unlikely an attacker would be staging attacks from them. It would make much more sense for an attacker to attempt to attack the network from a machine which he has full control over, for example a laptop connected via Wi-Fi. Having realised this, a request was made and granted to gain monitoring access to the traffic on ECS Wireless network in the hope of finding more interesting traffic there.

Although the tool was left to monitor traffic on both the UGLAN and the ECSWLAN over Easter vacation no suspicious traffic had been detected, apart from some that was accidentally caused by the system itself, due to a bug in the "counter rogue router advertisement" module. These findings lead to the conclusion that at present there really is no malicious IPv6 traffic flowing through the ECS network; at least none of the type the system currently tests for; it still remains entirely possible that someone is using a "zero day" exploit.

During deployment it became apparent that the system was not performing fast enough to handle all the packets on the link, and as a consequence some traffic escaped analysis. Investigation showed that the problem was that too much work was done in the processing unit. It was also determined that Scapy didn't perform well enough under high pressure and thus it was decided to replace it with a simpler packet capturing library called "Pcapy", which offers less features [29]. Pcapy is an example of a more traditional packet capturer that treats packets as a stream of bytes instead of objects, aiding performance greatly. After doing some tests with Pcapy, the capturer was rewritten to use Pcapy and to leave all the heavy work up to the processing module. This ensured that all packets got captured and queued properly. Further performance gains were achieved by implementing a byte level check to filter IPv6 from IPv4 traffic, thus ensuring only v6 traffic got queued and analysed.

The following graphic shows how the system has been deployed into the existing network topology.


Campus ECS WLAN (VLAN 29)
VLAN 24 (UG) - 2001:630:00:F110::/64
VLAN 19 (Wireless) - 2001:630:00:F105::/64
Building 59
Level3 UG LABS
(VLAN24)
Access Switch
Trunk
Core Switch
Fiber (Trunk)
Building 16
Access Switch
VLAN24/VLAN29
IPolice6
2001:630:d0:F110:216:76FF:fe59:9df4/64
Figure 7 - Deployment diagram

6.1 End user evaluation

The final phase of the deployment was marked by an informal meeting with John Wynn - a member of the ECS systems staff - to briefly demonstrate and discuss the system and its findings. John welcomed the way XML was used to configure the system and said it looked very clean and appeared well-structured. He added that it might be useful to add some comments to the configuration in order to further increase usability. Although the sample configuration is currently not commented, the system supports comments implicitly by the virtue of XML and the way the minidom library treats comments.

He praised the modularity and flexibility of the solution and mentioned that this was very rare among network appliances and usually only available in proprietary closed source enterprise solutions. John liked the reporting facilities, and said the approach of syslogging appeared reasonable. Nevertheless he did mention it would be useful to have a “timeline” feature that would track when a certain incident first occurred. This would greatly aid the task of tracing and tracking irregularities on the network. Due to the modular approach taken, this and other more detailed information could easily be added in form of a more complex logging module at a later point in time.

To the author’s surprise he didn’t seem concerned about the fact that the system hadn’t detected any malicious traffic, since he thought it quite unlikely that someone on the network was trying to hack the network using IPv6 exploits. He did add though, that as IPv6 becomes more popular, an associated rise in the number of IPv6 related exploitation attempts to be expected.

Overall, he seemed quite pleased with the system, and encouraged the idea of making it openly available to the public, saying “This software is certainly something we might consider using on our network in the future”.

7 Future development

The aim of this project was to develop a prototype of the system and some sample modules. Were this project to be developed further the author would like to turn it into a proper Python package and make it available to the public domain via Source Forge.

Furthermore it would be desirable to create a GUI configuration editor to simplify the task of creating and reading system configurations, thus lowering the barriers that prevent novice users from using the software and effectively broadening the potential user base. Among other things this editor would be capable of looking for all the components available on the system, inspecting their attributes and options, and presenting them to users in an easy to understand and configurable way, possibly featuring some kind of interactive wizard.

Another feature that may be useful would be to turn the system into a distributed solution, featuring a central management server that controls all the deployments within the network. For this to work the system would need to learn to act as part of a collaborative defence system [30] , that is capable of communicating with other systems on the network. This would have the added advantage of enabling the system to gather further information to aid in the classification of traffic. A problem at the moment is that the system is not capable of tracing attacks where the attacker uses a spoofed layer2 address. Were the system capable of communicating with the layer 2 devices on the network it would be possible to find out which port the attack originated from by inspecting the switch table. This could be realised by implementing support for the very popular SNMP protocol in IPolice6. [31]

This project leaves plenty of room for improvements and extensions because it has been designed with the bigger picture in mind, and one of the main goals was extensibility. For example, it might be possible to deploy this system as a proper firewall, or as a IDS/IPS solution by writing the appropriate modules. One interesting aspect of this software is that it incorporates the idea of memory and “state” which allows the detection and tracing of more sophisticated multi-stage attacks.

8 Evaluation and reflections

Before I conclude this report I would like to evaluate the project and reflect on some of the most important things I have learned about the area studied and software development during the course of this project.

8.1 Evaluation

In this section the project will be evaluated and checked one by one against the original project goals as stated at the beginning of this report.

"Research and summarise the most significant threats / exploits for IPv6 drawing on both existing research and results of experiments carried out during this project"

I fulfilled this part by reading an extensive amount and a broad range of material on the subject, which taught me a lot about the subject and its broader context. Not all of it was relevant for this specific project and has made it into this report, but it all helped me to gain a better understanding of the bigger picture when it comes to network security on a large scale.

"Analyse each threat and work out efficient and accurate mean of detecting exploitation attempts by running exploits and seeing if they have certain signatures."

This was done by starting to run tests and research early on in the project. I experimented with several tools and their options to get a feeling for the types of threat a campus style IPv6 enabled network might face. Ideally I would have liked to do some more research on this and to find some exploits and vulnerabilities of my own but unfortunately there was not enough time for this in the end.

"Investigate live traffic on the university network to find out which, if any, weaknesses are being exploited at present"

This is the part I am most unsatisfied with. I was really hoping to gain some insightful information from this, but unfortunately my tests turned out to be inconclusive, apart from the fact that I can say with high confidence though that none of the commonly known vulnerabilities were being exploited over the time period I was monitoring the network for. However at one point my tool was misconfigured and started sending fake routing advertisements itself, which were detected, showing the system worked in its deployed form.

"Develop a tool that is capable of detecting, reporting and possibly preventing exploitation attempts."

I think I have done more than this by developing a flexible and powerful framework capable of in theory detecting a wide variety of attacks and attack vectors that are not necessarily limited to the IPv6 protocol. However, this came at the price of increased complexity in configuration and use of the system. I built some proof of concept modules for this framework, targeting the THC attack toolkit detecting most attacks and

preventing them where possible. Unfortunately it wasn't possible to prevent many of these attacks without gaining access to network switch configurations. The reporting was done via writing to the console as well as logging to a Syslog server.
Also at present this tool is not performing well enough to be deployed actively along a networks forwarding path for networks as busy as the ECS WLAN; though it works fine for smaller and less busy networks.

"If there is time at the end of the project, issue an internet draft aimed at educating professionals about the security issues that exist within IPv6, and provide a set of recommendations on how-to combat these"

This never happened as there wasn't enough time and the application deadline for internet drafts closed before I would have had enough information to warrant such a document.

Looking back at what I set out to do I believe I have fulfilled most of the project goals with the exception of writing the internet draft, which was optional.

8.2 Reflections

In this section I will discuss what I learned or found noteworthy during the course of this project.

The first thing I'd like to address is the fact I wasn't aware just how much time it can take and how hard it can be to get seemingly simple things such as IPC between a kernel module and user-land processes working properly, and how such "small things" can have a huge impact on a projects time plan as I spent nearly half a semester trying to get this working before having to give up.

The second thing is that I didn't realise how hard it can be to make something very extensible and modular without significantly affecting complexity and performance. I am confident I could have easily written a piece of "static" software that does what my current software does a lot faster and with a lot less complexity for the end-user. The design was highly experimental and despite the fact it didn't quite turn out as well as planned I am still quite pleased with the result.

9 Conclusion

Part of this project's aim was to answer the question of which, if any, threats IPv6 deployments in large-scale campus style networks face. This was answered fairly early on by reading the relevant literature and coming to the conclusion that these networks face essentially the same threats as their IPv4 counter parts plus additional and unique threats. It was shown that a lot of these threats are easily exploitable by simply downloading and running hacking-toolkits, and that there are currently no measures in place to prevent these types of attacks. Further analysis of the way in which these tools behave made it clear that most of these attacks have "signatures" to them, which make it possible to detect them. However, it also became apparent that in order to trace most of these attacks detailed switch-port data needs to exist and thus tracking and tracing these intrusions attempts would require a collaborative defence strategy [30].

Despite the fact that we can't track and trace these attacks easily this project showed it is easily possible to detect and in some cases counteract these threats. A software framework capable of doing this was discussed, implemented and deployed during the course of this project. Analysis of the threats have shown that some of the attacks that networks might face are quite sophisticated and may span several stages, so the system needed to be aware and capable of handling this. This led to the idea of creating IPolice6, an IDS/IPS framework capable of saving state information about the stage of an attack, which enabled the tracing of these more sophisticated attacks. This exceeds the capability of the most standard IDS/IPS software used today, as most of it is only capable of tracing and following through the state of a single connection. IPolice6's memory function is capable of holding any information or state users require. Furthermore IPolice6 features "active" scanning, which can probe and even attempt to "bait" malicious users to make themselves known, as the detect-dos-new-ip6 module proves.

The final and most interesting question this project attempted to answer was which, if any, of these exploits are currently seen on the department's network. To answer this question IPolice6 was configured and deployed on a test machine which monitored and analysed all the IPv6 traffic on both the undergraduate LAN and the Campus wide Wireless LAN. The system was deployed over the course of several weeks with temporary shutdowns for updates and changes to the configuration, yet failed to find evidence of any malicious activity. It is worth mentioning though, that when the system itself briefly started sending fake router advertisements these were detected and reported properly by the system. This leads to the conclusion that at present no IPv6 weaknesses are being exploited on the LAN; at least not any of the type the system is configured to probe for at the moment.

10 References

[1] IPv4 depletion. [Online]. (http://www.potaroo.net/ispcol/2008-10/v4depletion.html)

[2] Commission of the european communities, "Advancing the internet: Action Plan for the deployment of Internet Protocol version 6 (IPv6) in Europe," Brussels, Communication from the comission to the european parliament, the council, the european economic and social comittee and the comittee of the regions. 2008.

[3] R.Hinden S. Deering. (1998, Dec.) Internet Protocol, Version 6 (IPv6) Specification. Document.

[4] Hagen Silvia, *IPv6 Essentials*, 2nd ed.: O'Reilly Media, 2006.

[5] S. Deering R. Hinden. (2006, Feb.) IP Version 6 Addressing Architecture. RFC.

[6] Xinyu Yang, Ting Mia, Yi Shi, "Typical DoS/DDoS Threats under IPv6," in *International Multi-Conference on Computing in the Global Information Technology*, Guadeloupe City, 2007, p. 55.

[7] Joshi J.B.D, Tuladhar S.R Caicedo C.E, "IPv6 Security Challenges," *IEEE Internet Computing*, vol. 42, no. 2, pp. 36-42, February 2009.

[8] S. Krishnan. (2010, March) The case against Hop-by-Hop options (draft-krishnan-ipv6-Hop-by-hop-04).

[9] Ed. F. Le Faucheur. (2010, March) IP Router Alert Considerations and Usage (draft-ietf-intarea-router-alert-considerations-00).

[10] R. Hunt B. Harris, "TCP/IP security threats and attack methods ," *Computer Communications*, vol. 22, no. 10, pp. 885-897, June 1999.

[11] Arnaud Ebalard Philippe Biondi. (2007) IPv6 Routing Header Security. Conference Presentation.

[12] Cynthia E, Dunn Jeffrey J Martin, "Internet Protocol Version 6 (IPv6) Protocol Security Assessment," in *Military Communications Conference*, Orlando, FL USA, 2007, pp. 1-7.

[13] van Hauser. (2008) Attacking the IPv6 Protocol Suite. Conference Presentation.

[14] Kijoon Chae, HyoChan Bang, JungChan Na Hayoung Oh, "Comparisions analysis of Security Vulnerabilities for Security Enforcement in IPv4/IPv6," in *Advanced Communication Technology* , Phoenix Park, 2006, pp. 1583-1585.

[15] Grgic Kresimir Zagar Drago, "IPv6 Security Threats and Possible Solutions," in *World Automation Congress*, Budapest, 2006, pp. 1-7.

[16] E. Nordmark, W. Simpson, H. Soliman T. narten. (2007, Sep.) Neighbour Discovery for IP version 6 (IPv6). RFC.

[17] R. Atkinson S. Kent. (1998, Nov.) IP Encpasulating Security Payload (ESP). RFC.

[18] R. Atkinson S. Kent. (1998, Nov.) IP Authentication Header. RFC.

[19] R. Atkinson S. Kent. (1998, Nov.) Security Architecture for the Internet Protocol. RFC.

[20] Thibault Cholez, "Ndmon Technical Report," Madynes Research Group, 2006.

[21] J. Morse, "Trust Problems in Auto-Configured IPv6 Networks," Southampton, Individual Research Project 2007.

[22] N Saxena, B Mitchell BP Hehl. (2005) Applicability of RAD.

[23] Alan Howard, "Rapid Application Development: Rough and Dirty or Value-for-Money Engineering ?," *Communications of the ACM*, pp. Vol. 45 No 10 pp 27-29, 2002.

[24] JC Hamano L Torvalds. (2005) GIT-fast version control system.

[25] Biondi Philippe, EADS Corporate Research Center. (2005, September) Network packet manipulation with Scapy.

[26] Guido van Rossum. (2001, July) Style Guide for Python Code (PEP 8).

[27] Core Security Technologies. (2010) What is Pcapy ? [Online]. http://oss.coresecurity.com/projects/pcapy.html

[28] E. Wilhite D. Frincke, "Distirbuted Network Defense," in *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, United States Military Academy, West Point, NY, 2001, p. 236 ff.

[29] Simon Znaty and Jean-Pierre Hubaux Jean-Philippe Martin-Flatin, "A Survey of Distirbuted Enterprise Network and Systems Management Paradigms," *Journal of Networks and Systems Management*, pp. 9-26, 2004.

[30] S. Deering A. Conta. (1998, Dec.) Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6). RFC.

[31] M. Ajaykumar Venkatesulu Sameer Seth, *TCP/IP Architecture, Design and Implementation in Linux*.: Wiley, 2008.

[32] Kevin Kaichuan, "Kernel Korner - Why and How to Use Netlink Socket," *Linux Journal*, Jan. 2005.

[33] Takashi Arano, "IPv4 Address Report," Automaticaly generated report 2010.

[34] van Hauser. (2008) Attacking the IPv6 Protocol Suite. Conference Presentation.

11 Appendix A – Detailed UML Diagrams

11.1 Sequence Diagram of final design

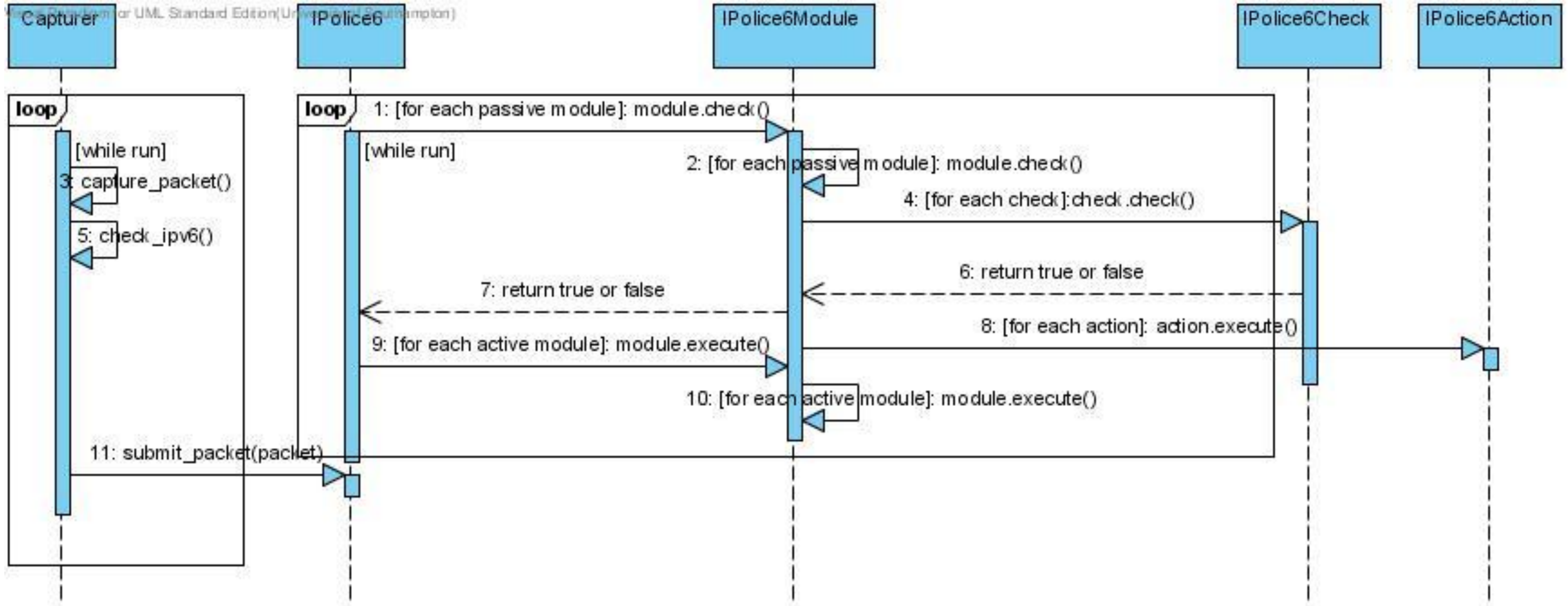

Visual Paradigm for UML Standard Edition(University of Southampton)
Capturer
IPolice6
IPolice6Module
IPolice6Check
IPolice6Action
loop
[while run]
3: capture_packet()
5: check_ipv6()
loop
1: [for each passive module]: module.check()
[while run]
2: [for each passive module]: module.check()
4: [for each check]:check.check()
6: return true or false
7: return true or false
8: [for each action]: action.execute()
9: [for each active module]: module.execute()
10: [for each active module]: module.execute()
11: submit_packet(packet)

Figure 8 - Sequence Diagram final system

12 Appendix B - Testing

12.1 Individual module tests

Capturer

Test	Expected result	Actual result	Comments / Fix
Test Capturer by sending sequence of pings to test machine and make sure it picks all of them up	Sequence number of ping identical to “number of packets” displayed on test machine identical	Pass	Tested using Linux ping command with an interval of 0.1 seconds
Test Capturer performance high load by sending pings from multiple machines	Packets captured equal to total amount of sequence numbers	Pass	Tested using 4 other machines on test network

12.2 IPolice6

Test	Expected result	Actual result	Comments / Fix
Start program and add modules	All modules registered	Pass	n/a
Ensure all packets from capturer reach capturing unit	All packets displayed by capturer also put in queue of processor	Pass	n/a
Attempt to load illegal module	NotAModuleException raised	Pass	n/a
Attempt to load proper modules	Module loaded	Pass	n/a

12.3 Configuration parser

Test	Expected result	Actual result	Comments / Fix
Ensure XML document is loaded	Loaded	Loaded	n/a

properly			
Test dispatcher and that every nodes parse method is called accordingly	-	Pass	n/a
Test stack and ensure parent module is correct	-	Pass	n/a
Test configuration is built properly	-	Pass	Used sample_config.conf and hand checked modules parameters are correct

12.4 GenericFieldCheck

Test	Expected result	Actual result	Comments / Fix
Test module loads	Module should load and appear in modules_passive list	Pass	n/a
Test single value check	If packet with set value is encountered should return True	Pass	n/a
Test multiple values	As above but any of	Pass	n/a
Test negations e.g. all of list vs. none of list present	-	Pass	n/a

12.5 Screenlogger

Test	Expected result	Actual result	Comments / Fix
Test module loads	Module should load and appear in actions_pass of parent module	Pass	n/a
Test logs properly	Should log properly formatted message to the screen	Pass. Minor bug: For some reason every entry appears twice	Unable to locate bug, attempted to use debugger to halt execution

			immediately after logging occurred to check when the second entry happens but happen simultaneously – possible bug in python logger module ?

12.6 Syslogger

Test	Expected result	Actual result	Comments / Fix
Test module loads	Module should load and appear in actions_pass of parent module	Pass	n/a
Test logs properly	Should log properly formatted message to the syslog (var/log/messages/ipolice6)	Pass. Minor bug: For some reason every entry appears twice	Unable to locate bug, attempted to use debugger to halt execution immediately after logging occurred to check when the second entry happens but happen simultaneously – possible bug in python logger module ?

12.7 Final test

Test	Expected result	Actual result	Comments / Fix
Load configuration file	Return finished configured object which modules set-up as specified in configuration document	Pass	Verified by manually checking the structure of the modules and values of variables
Performance under no load	Fairly low around 2-3% CPU usage	Constantly ~95%	Make idle threads sleep between loop iterations
Test DetectAlive6	Should register and log signature of alive6	Pass	Tested using alive6 from THC toolkit
Test DetectDosNewIP6	Should register and log signature of dos-new-ip6	Pass	Tested using dos-new-ip6 from THC toolkit

Test DetectFakeRouter6	Should register and log signature of fake-router6	Pass. Problem where counter RA itself caused a trigger leading to an endless loop	Tested using fake-router6 from THC toolkit. Endless recursion fixed by configuring module to ignore advertisements with 0 lifetime
Test DetectParasite6	Should register and log signature of parasite6	Pass	Tested using parasite6 from THC toolkit
Test DetectRedir6	Should register and log signature of alive6	Pass	Tested using alive6 from THC toolkit
Test DetectTooBig6	Should register and log signature of toobig6	Pass	Tested using toobig6 from THC toolkit
Test DetectRH0	Should register and log packets containing a Routing Header 0 extension header	Pass	Tested using “triggerrh0.py” from “testcases”
Test DetectHbHDOS	Should log packets containing more than x HbHOptions	Pass	Tested using “triggerHop-by-hop.py" from “testcases”

13 Appendix C - Sample system configuration used for testing the modules

```
<config name="Detect IP6 exploits">
        <capturers>
                <capturer id="simple_capture">
                        <type>
                                <eval>capture.SimpleCapturer</eval>
                        </type>
                </capturer>
        </capturers>
        <modules>
                <module id="DetectHBHDos">
                        <type>
                                <eval>baseclasses.IPolice6Module</eval>
                        </type>
                        <init>
                                <name>DetectHBHDos</name>
                        </init>
                        <checks>
                                <check id="HBHoptcheck">
                                        <type>

        <eval>check_restrict_occurrence.CheckRestrictOccurrence</eval>
                                        </type>
                                        <init>

        <layer><eval>scapy.layers.inet6.IPv6ExtHdrHop-by-hop</eval></layer>

        <max_occurrence><eval>3</eval></max_occurrence>
                                        </init>
                                </check>
                        </checks>
                        <actions>
                                <action id="screenlogaction_HBH">
                                        <type>
                                                <eval>actions.logaction.LogAction</eval>
                                        </type>
                                        <init>

                                        </init>
                                </action>

                                <action id="syslogaction_HBH">
                                        <type>
        <eval>actions.syslogaction.SysLogAction</eval>
                                        </type>
                                        <init>

                                        </init>
                              </action>
                    </actions>
          </module>
          <module id="DetectRedir6">
                    <type>
                              <eval>thcdetect.DetectRedir6</eval>
                    </type>
          </module>
          <module id="DetectParasite6">
                    <type>
                              <eval>thcdetect.DetectParasite6</eval>
                    </type>
          </module>
          <module id="DetectDosNewIp6">
                    <type>
                              <eval>thcdetect.DetectDosNewIp6</eval>
                    </type>
                    <actions>
                              <action id="screenlogaction_DAD">
                                        <type>
                                                  <eval>actions.logaction.LogAction</eval>
                                        </type>
                              </action>
                              <action id="syslogaction_DAD">
                                        <type>

        <eval>actions.syslogaction.SysLogAction</eval>
                                        </type>
                                        <init>

                                        </init>
                              </action>
                    </actions>
          </module>
          <module id="DetectAlive6">
                    <type>
                              <eval>baseclasses.IPolice6Module</eval>
                    </type>
                    <init>
                              <name>alive6 detector</name>
                              <description>Checks for attempts to ping the all node
multicast address </description>
                    </init>
                    <checks>
```

```
				<check>
					<type>

	<eval>checks.generic_field_check.GenericFieldCheck</eval>
					</type>
					<init>
						<name>All node multicast check</name>
						<field>dst</field>
						<value>ff02::1</value>

	<layer><eval>scapy.layers.inet6.IPv6</eval></layer>
					</init>
				</check>
				<check>
					<type>

	<eval>checks.generic_field_check.GenericFieldCheck</eval>
					</type>
					<init>
						<name>All      node      multicast      ping
check</name>

	<layer><eval>scapy.layers.inet6.ICMPv6EchoRequest</eval></layer>
					</init>
				</check>

			</checks>
			<actions>
				<action id="syslogaction_alive">
					<type>

	<eval>actions.syslogaction.SysLogAction</eval>
					</type>
					<init>

					</init>
				</action>
				<action id="screenlogaction_alive">
					<type>
						<eval>actions.logaction.LogAction</eval>
					</type>
					<init>

					</init>
				</action>
			</actions>
		</module>
		<module id="DetectFakeRouter6">
			<type>
				<eval>baseclasses.IPolice6Module</eval>
```

```
            </type>
            <init>
                <name>Fake router detect</name>
            </init>
            <checks>
                <check>
                    <type>

    <eval>checks.generic_field_check.GenericFieldCheck</eval>
                    </type>
                    <init>
                        <name>Check prefix valid</name>
                        <field>prefix</field>

    <value><eval>util.get_ip6_prefix()</eval></value>

    <layer><eval>scapy.layers.inet6.ICMPv6NDOptPrefixInfo</eval></layer>

    <action_fail><eval>IMMEDIATE_POSITIVE</eval></action_fail>

    <action_pass><eval>CONTINUE</eval></action_pass>
                    </init>
                </check>
                <check>
                    <type>

    <eval>checks.generic_field_check.GenericFieldCheck</eval>
                    </type>
                    <init>
                        <name>Check valid mac</name>

    <layer><eval>scapy.layers.inet6.ICMPv6NDOptSrcLLAddr</eval></layer>
                        <field>lladdr</field>
i
    <testValues><eval>['08:00:27:79:92:83']</eval></testValues>

    <action_fail><eval>IMMEDIATE_POSITIVE</eval></action_fail>

    <action_pass><eval>IMMEDIATE_NEGATIVE</eval></action_pass>
                    </init>
                </check>
            </checks>
            <actions>
                <action id="screenlogaction_fakerouter">
                    <type>
                        <eval>actions.logaction.LogAction</eval>
                    </type>
                    <init>

                    </init>
				</action>
				<action id="syslogaction_fakerouter">
					<type>

		<eval>actions.syslogaction.SysLogAction</eval>
					</type>
					<init>

					</init>
				</action>
				<action id="counterraaction">
					<type>

		<eval>actions.counter_ra_action.CounterRAaction</eval>
					</type>
				</action>
			</actions>
		</module>
		<module id="DetectTHCSignature">
			<type>
				<eval>baseclasses.IPolice6Module</eval>
			</type>
			<init>
				<name>THC attack detect</name>
				<description>This checks for presence of dead
beef</description>
			</init>
			<checks>
				<check>
					<type>

		<eval>checks.generic_field_check.GenericFieldCheck</eval>
					</type>
					<init>
						<name>dead check</name>
						<field>id</field>

		<value><eval>int("0xdead",0)</eval></value>

		<layer><eval>scapy.layers.inet6.ICMPv6EchoRequest</eval></layer>
					</init>
				</check>
				<check>
					<type>

		<eval>checks.generic_field_check.GenericFieldCheck</eval>
					</type>
					<init>
						<name>beef check</name>
						<field>seq</field>
```

```
          <value><eval>int("0xbeef",0)</eval></value>

          <layer><eval>scapy.layers.inet6.ICMPv6EchoRequest</eval></layer>
                                        </init>
                              </check>
                    </checks>

                    <actions>
                              <action id="syslogaction_thc">
                                        <type>

          <eval>actions.syslogaction.SysLogAction</eval>
                                        </type>
                                        <init>

                                        </init>
                              </action>
                              <action id="screenlogaction_thc">
                                        <type>
                                                  <eval>actions.logaction.LogAction</eval>
                                        </type>
                                        <init>

                                        </init>
                              </action>
                    </actions>
                    <modules>
                    </modules>
          </module>
     </modules>
</config>
```

(startup_config.conf)

14 Appendix D – Sample log excerpts generated by IPolice6 (deployment and test systems)

```
Apr 21 11:38:03 ipolice6 ipolice6: alive6 detector triggered offending packet: ###[
Ethernet ]### dst = 33:33:00:00:00:01 src = 00:16:76:59:9d:f4 type = 0x86dd ###[
IPv6 ]### version = 6L tc = 0L fl = 0L plen = 48 nh = Hop-by-Hop Option Header
hlim = 255 src = 2001:630:d0:f110:216:76ff:fe59:9df4 dst = ff02::1 ###[ IPv6
Extension Header - Hop-by-Hop Options Header ]### nh = ICMPv6 len = 2 autopad
= On \options \ |###[ Scapy6 Unknown Option ]### | otype = 128 [10:
discard+ICMP, 0: Don't change en-route] | optlen = 1 | optdata = '\x8a' |###[ Scapy6
Unknown Option ]### | otype = 213 [11: discard+ICMP not mcast, 0: Don't change
en-route] | optlen = 206 | optdata =
"KAAAA\x1f'\xff\x02\x00\x00\x00\x00\x00\x00\x00\x00" ###[ ICMPv6 Echo Request
]### type = Echo Request code = 0 cksum = 0x3bc6 id = 0xdead seq = 0xbeef data
= "\x80\x01\x8a\xd5\xceKAAAA\x1f'\xff\x02\x00\x00"
Apr 21 11:38:03 ipolice6 ipolice6: THC attack detect triggered offending packet: ###[
Ethernet ]### dst = 33:33:00:00:00:01 src = 00:16:76:59:9d:f4 type = 0x86dd ###[
IPv6 ]### version = 6L tc = 0L fl = 0L plen = 48 nh = Hop-by-Hop Option Header
hlim = 255 src = 2001:630:d0:f110:216:76ff:fe59:9df4 dst = ff02::1 ###[ IPv6
Extension Header - Hop-by-Hop Options Header ]### nh = ICMPv6 len = 2 autopad
= On \options \ |###[ Scapy6 Unknown Option ]### | otype = 128 [10:
discard+ICMP, 0: Don't change en-route] | optlen = 1 | optdata = '\x8a' |###[ Scapy6
Unknown Option ]### | otype = 213 [11: discard+ICMP not mcast, 0: Don't change
en-route] | optlen = 206 | optdata =
"KAAAA\x1f'\xff\x02\x00\x00\x00\x00\x00\x00\x00\x00" ###[ ICMPv6 Echo Request
]### type = Echo Request code = 0 cksum = 0x3bc6 id = 0xdead seq = 0xbeef data
= "\x80\x01\x8a\xd5\xceKAAAA\x1f'\xff\x02\x00\x00"

Apr 28 17:15:26 IPOLICE6 ipolice6: DetectHBHDos triggered offending packet: ###[
Ethernet ]### dst = 08:00:27:79:92:83 src = 00:00:00:00:00:00 type = 0x86dd ###[
IPv6 ]### version = 6L tc = 0L fl = 0L plen = 40 nh = Hop-by-Hop Option Header
hlim = 64 src = ::1 dst = ::1 ###[ IPv6 Extension Header - Hop-by-Hop Options
Header ]### nh = Hop-by-Hop Option Header len = 0 autopad = On \options \ |###[
PadN ]### | otype = PadN [00: skip, 0: Don't change en-route] | optlen = 4 | optdata
= '\x00\x00\x00\x00' ###[ IPv6 Extension Header - Hop-by-Hop Options Header ]###
nh = Hop-by-Hop Option Header len = 0 autopad = On \options \ |###[ PadN ]### |
otype = PadN [00: skip, 0: Don't change en-route] | optlen = 4 | optdata =
'\x00\x00\x00\x00' ###[ IPv6 Extension Header - Hop-by-Hop Options Header ]### nh
= Hop-by-Hop Option Header len = 0 autopad = On \options \ |###[ PadN ]### |
otype = PadN [00: skip, 0: Don't change en-route] | optlen = 4 | optdata =
'\x00\x00\x00\x00' ###[ IPv6 Extensio

Apr 28 17:05:24 ipolice6 ipolice6: THC attack detect triggered offending packet: ###[
Ethernet ]### dst = 00:16:76:59:9d:f4 src = 00:50:56:a0:04:da type = 0x86dd ###[
IPv6 ]### version = 6L tc = 0L fl = 0L plen = 96 nh = ICMPv6 hlim = 64 src =
2001:dead::250:56ff:fea0:4da dst = 2001:630:d0:f110:216:76ff:fe59:9df4 ###[
ICMPv6 Parameter Problem ]### type = Parameter problem code = unrecognized
```

```
IPv6 option encountered  cksum = 0x190e  ptr = 42 ###[ IPv6 in ICMPv6 ]###  version
= 6L  tc = 0L  fl = 0L  plen = 48  nh = Hop-by-Hop Option Header  hlim = 255  src =
2001:630:d0:f110:216:76ff:fe59:9df4  dst = ff02::1 ###[ IPv6 Extension Header - Hop-
by-Hop Options Header ]###  nh = ICMPv6  len = 2  autopad = On  \options \  |###[
Scapy6 Unknown Option ]###  | otype = 128 [10: discard+ICMP, 0: Don't change en-
route]  | optlen = 1  | optdata = '\xc3'  |###[ Scapy6 Unknown Option ]###  | otype = 92
[01: discard, 0: Don't change en-route]  | optlen = 216  | optdata = 'KAAAA!-
\xff\x02\x00\x00\x00\x00\x00\x00\x00\x
Apr 28 17:05:24 ipolice6 ipolice6: THC attack detect triggered offending packet: ###[
Ethernet ]###  dst = 00:16:76:59:9d:f4  src = 00:14:1b:3d:2c:00  type = 0x86dd ###[
IPv6 ]###  version = 6L  tc = 0L  fl = 0L  plen = 96  nh = ICMPv6  hlim = 64  src =
2001:630:d0:f110::2  dst = 2001:630:d0:f110:216:76ff:fe59:9df4  ###[ ICMPv6
Parameter Problem ]###  type = Parameter problem  code = unrecognized IPv6 option
encountered  cksum = 0x5c73  ptr = 42 ###[ IPv6 in ICMPv6 ]###  version = 6L  tc =
0L  fl = 0L  plen = 48  nh = Hop-by-Hop Option Header  hlim = 255  src =
2001:630:d0:f110:216:76ff:fe59:9df4  dst = ff02::1 ###[ IPv6 Extension Header - Hop-
by-Hop Options Header ]###  nh = ICMPv6  len = 2  autopad = On  \options \  |###[
Scapy6 Unknown Option ]###  | otype = 128 [10: discard+ICMP, 0: Don't change en-
route]  | optlen = 1  | optdata = '\xc3'  |###[ Scapy6 Unknown Option ]###  | otype = 92
[01: discard, 0: Don't change en-route]  | optlen = 216  | optdata = 'KAAAA!-
\xff\x02\x00\x00\x00\x00\x00\x00\x00\x00' ###[
```

```
Apr 23 15:41:33 IPOLICE6 ipolice6: Fake router detect triggered offending packet:
###[ Ethernet ]###  dst = 33:33:00:00:00:01  src = 08:00:27:7c:ab:b2  type = 0x86dd
###[ IPv6 ]###  version = 6L  tc = 0L  fl = 0L  plen = 64  nh = ICMPv6  hlim = 255
src = fe80::1  dst = ff02::1 ###[ ICMPv6 Neighbor Discovery - Router Advertisement
]###  type = Router Advertisement  code = 0  cksum = 0x77e4  chlim = 255  M = 0L  O
= 0L  H = 0L  prf = High  P = 0L  res = 0L  routerlifetime= 65535  reachabletime=
16384000  retranstimer= 1966080 ###[ ICMPv6 Neighbor Discovery Option - MTU
]###  type = 5  len = 1  res = 0x0  mtu = 20 ###[ ICMPv6 Neighbor Discovery Option -
Prefix Information ]###  type = 3  len = 4  prefixlen = 64  L = 1L  A = 1L  R = 0L  res1
= 0L  validlifetime= 0xffffffff  preferredlifetime= 0xffffffff  res2 = 0x0  prefix =
2001:beef:: ###[ ICMPv6 Neighbor Discovery Option - Source Link-Layer Address
]###  type = 1  len = 1  lladdr = 08:00:27:7c:ab:b2
```

```
Apr 23 15:04:40 IPOLICE6 ipolice6: DetectDoSNewIp6 triggered offending packet:
###[ Ethernet ]###  dst = 33:33:00:00:00:01  src = 08:00:18:59:4a:28  type = 0x86dd
###[ IPv6 ]###  version = 6L  tc = 0L  fl = 0L  plen = 32  nh = ICMPv6  hlim = 255
src = 2a01:348:13c:0:8ac8:eff:fe8e:38ca  dst = ff02::1 ###[ ICMPv6 Neighbor
Discovery - Neighbor Advertisement ]###  type = Neighbor Advertisement  code = 0
cksum = 0xecd2  R = 0L  S = 0L  O = 1L  res = 0x0L  tgt =
2a01:348:13c:0:8ac8:eff:fe8e:38ca  ###[ ICMPv6 Neighbor Discovery Option -
Destination Link-Layer Address ]###  type = 2  len = 1  lladdr = 08:00:18:59:4a:28
```

15 Appendix E – Initial project brief

15.1 The Problem:

There are problems with the current internet protocol version 4, some of which are related to the amount of available address space. The IETF has proposed a replacement protocol v6, which is being adopted more and more by companies and organisations across the globe. However since v4 and v6 are not compatible with each other this makes communication between hosts using different versions. To combat this problem, numerous transition methods exist, which enable V4 to V6 communication. One of the most popular ones being the "dual stack" approach where the entire network infrastructure is aware and capable of communicating using whichever protocol version required. This solution has various drawbacks and one major concern associated with it is network security. While most networks these days employ an adequate level of perimeter security to protect the network the great majority of these solutions focus merely on the IPv4 protocol. This however does not provide any protection from attacks focused on vulnerabilities in the IPv6 specification which leaves networks vulnerable to exploitation of these vulnerabilities.

15.2 A possible solution:

To address this problem I propose an IPv6 aware intrusion detection / prevention system. It will be deployed at the perimeter analysing incoming IPv6 packets assessing the threat the traffic might pose to the system. This system is not a replacement for a traditional firewall but serves as an additional security mechanism, which incoming traffic needs to pass prior to accessing the network.

15.3 Project scope:

The project I propose is split in two major parts:

The first part will focus on investigating the IPv6 protocol as specified in RFC2460 trying to establish the criteria which a packet must meet to be considered valid. During this investigation I will also attempt to find vulnerabilities / ambiguities in the RFC which a potential attacker might be able to exploit. Furthermore if there is time I plan to take a look at the implementation and behaviour of IPv6 on the Linux and Windows Operating systems to discover potential weaknesses in the individual implementations which could be exploited by attackers.

Armed with the knowledge of the research conducted in Part I, I will then attempt to design and implement a piece of software that can be deployed on a Unix based OS, which is capable of analysing and classifying the threat level of incoming IPv6 traffic . Administrators will be able to assign a set of actions to be taken on a per class basis. These actions will at the most basic level be as simple as accepting (forwarding) or rejecting (dropping) the packet. However one of the things I would like to add to this project if time allows it are advanced features such as forwarding packets to certain destinations and / or logging incidents to syslog or other logging servers.
A further outcome of the project might be a internet draft, containing guidelines and suggestions directed at those who wish to implement similar systems in the future

depending on whether the discoveries made during the course of this project warrant such a document.

16 Appendix F – Initial time management

As outlined in the main body of this report the initial time plan was discarded once the RAD method had been adopted. This appendix illustrates what the plan was initially and up to the point where it was dropped.

16.1 Initially Planned

The following chart summarises what was planned at the beginning of the project.

Initial	OCT	NOV	DEC	JAN	FEB	MAR	APR
Background							
Investigation of Behaviour		🟨	🟨				
Literature review	🟨						
Tools and techniques		🟨	🟨				
Design							
Requirements		🟨					
Specification			🟨				
Implementation							
Prototype		🟨	🟨				
Beta			🟨	🟨	🟨		
RC						🟨	🟨
Testing		🟨	🟨	🟨	🟨	🟨	🟨
Report Writing							
Interim	🟨	🟨	🟨	🟨			
Final					🟨	🟨	🟨

Table 1- Initial Project Schedule (Adaption of original Schedule found in Appendix A)

16.2 Work done

The following charts summarise all the work done so far

Done	OCT	NOV	DEC	JAN	FEB	MAR	APR
Background	🟩	🟩	🟩				
Investigation of Behaviour							
Literature review	🟩						
Tools and techniques		🟩					
Design		🟩	🟩				
Requirements		🟩					
Specification			🟩				
Implementation							
Prototype			🟩				
Beta							
RC							
Admin							
setup/configuration of T.E		🟩	🟩				
Document management	🟩	🟩	🟩				
Testing							
Report Writing							
Interim	🟩	🟩	🟩	🟩			
Final							

16.3 Work Remaining

The following chart summarise all the work remaining to be done

Remaining	OCT	NOV	DEC	JAN	FEB	MAR	APR
Background							
Investigation of Behaviour				🟨	🟨		
Literature review							
Tools and techniques							
Design							
Requirements							
Specification							
Implementation							
Packet capture				🟨			
Packet analysis					🟨	🟨	
Responses						🟨	
System configuration						🟨	🟨
Logging							🟨
Admin							
setup/configuration of T.E				🟨	🟨	🟨	🟨
Document management				🟨	🟨	🟨	🟨
Source management				🟨	🟨	🟨	🟨
Testing				🟨	🟨	🟨	🟨
Report Writing					🟨	🟨	🟨
Interim							
Final					🟨	🟨	🟨

17 Appendix G – Directory structure of submitted CD

This section contains an index of the content on the CD accompanying this submission. This is taken from the readme.txt file in the directory root.

```
.
= CD Directory Structure =
.
|-- IPolice6 # Holds source code and configuration files for the program.
|   |-- IPolice6.py # Dummy pointing to ipolice6.py
|   |-- actions # Holds all specific actions.
|   |   |-- __init__.py # Placeholder required by python to treat directory as package.
|   |   |-- counter_ra_action.py # Inpired by RAMOND will deprecate and send a rputer advertisements.
|   |   |-- logaction.py # Simple proof of concept logger that logs to the screen.
|   |   |-- syslogaction.py # More sophisticated logger capable of logging to a syslog server.
|   |   `-- thcaction.py # Very simple proof of concept action.
|   |-- baseclasses.py # Holds all the main building blocks of the application , specified interfaces for module components.
|   |-- capture.py # Simple multi-threaded capturer based on pacapy that filters out IPv6 traffic.
|   |-- checks # Holds all the checks.
|   |   |-- __init__.py # Placeholder required for module use.
|   |   |-- check_restrict_occurrence.py # Simple check that looks for a specific header, and counts the occurrence.
|   |   |-- generic_field_check.py # Bread and butter checker. Flexible check that can check one or more values in a given header, offers all functionality needed for a classic packet filter.
|   |   |-- generic_type_check.py # Cut down version of above used if only the type is needed but not the value.
|   |   `-- thc_checks.py # Set of checks containing all that is neeeded to detect some of the more advanced thc attacks (Redir, Parasite , DAD etc).
|   |-- conf.py # Global config file that defines constants for universal use.
|   |-- configparser.py # XML Parser that parses a configuration, build and returns an IPolice6 instance.
|   |-- configs # Folder that holds all the various sytem configurations.
|   |   |-- sampleconfig.conf # Initial toy config.
|   |   `-- startup_config.conf # More advanced (deployed) configuration. System always defaults to this file if no specific config is specified.
|   |-- errors.py # Defines Module specific errors.
|   |-- ip6modules # Contains all the active and passive modules.
|   |   |-- __init__.py # Needed by python.
|   |   `-- thcdetect.py # Contains all the specific modules needed for THC detection.
|   |-- ipolice6.py # The main class and executable.
|   |-- readme.txt # Brief explanation of how to deploy, run and configure IPolice6.
|   |-- run.py # alternative run script to start up application (deprecated).
|    `-- util.py # Module containing a set of useful library functions, mostly concerned
with handling IPv6 addresses and their various notations.
|-- logs # Contains all the logs collected during the deployment phase of the project.
|   |-- ipolice6.log # log file.
|   `-- ipolice6.log.0 # log file.
|-- readme.txt # This readme.
`-- testscripts # Contains the no thc automated test scripts used to trigger system
behaviour during testing.
    |-- triggerHop-by-hop.py # Designed to trigger the Hop-by-hop detection module
specified in the XML.
    |-- triggerparasite6.py # Designed to send a lot of NS queries for various IPs, making
it appear like the system has more than one global IP.
    `-- triggerrh0.py # Designed to trigger the RoutingHeader0 module.

7 directories, 30 files
```