Report # I733-041R-2007
Date: 10/03/2007

Firewall Design Considerations for IPv6

Enterprise Applications Division
of the
Systems and Network Analysis Center (SNAC)

Information Assurance Directorate

Author
Casimir A. Potyraj, I733
(410) 854-5723
cas@thematrix.ncsc.mil

National Security Agency
Attn: I733
9800 Savage Rd. Suite 6704
Ft. Meade, MD 20755-6704
(410) 854-6191

This page intentionally blank

Table of Contents

Table of Figures

1 Introduction

This document provides opinion on IPv6 firewall design and security properties believed to be most beneficial in protecting IPv6 and IPv4/6 transition networks. The opinion intended as input into a larger process of determining what IPv6 Firewall Design Requirements should be. Firewall vendor expertise is needed to determine what is truly achievable and practical in modern design. *Firewall Design Considerations for IPv6* should not be interpreted as a DoD requirements document, though it may be referenced by such documents in the future. Design recommendations and approaches are occasionally offered, though these are purely informational and should not be interpreted as requirements for the DoD.

2 IPv4 and IPv6 Packet Header Comparison

Far too many comparison charts and "So What's New in IPv6?" summaries are already written on the IPv6 headers to justify another. Therefore, chapter 2 is brief and focused on specific changes in IPv6 that impact firewalls.

2.1 Improving Packet Processing Performance

The IPv6 header structure is improved in a number of ways. The hop-by-hop processing required by routers is minimized and endpoint processing of the packets is optimized. Although header bandwidth is a marginal performance factor, the amount of processing required on the header fields produces the biggest impact on performance. By minimizing header processing, the performance of IPv6 has the potential to surpass that of IPv4.

Product maturity is a *very* significant factor in the overall performance level. High performance IPv4 router designs implement a substantial amount of processing in hardware, whereas newer IPv6 devices are often largely implemented in software. This is logical from an economic standpoint, since companies don't want to commit designs to very expensive chip masks before the revelations of early testing and experience can be incorporated. Software is slower than hardware but easier to test and modify.

IPv6 removes some unnecessary fields. The IPv4 *header checksum* field is one such field that is not present in the IPv6 header. A checksum value at the IP layer must be recomputed at each hop and serves virtually no useful purpose. The link-layer is responsible for the delivery of bits across a wire, and therefore the link layer checksums are the ones that really matter. Transmission errors, for example, are detected by the link-layer checksums and can be corrected by re-transmission. The only additional errors that can be detected by an IP layer checksum are those that occur between the link and IP layers within a given node. This almost never happens and indicates a firmware/hardware failure when it does (i.e. not a recoverable transmission error). Furthermore, a portion of these very rare hardware failures will happen to be within the checksum creation or checking logic itself. The IP layer checksum, therefore, is more likely to harm performance than improve it.

The *identification* field, *flags* field, and *fragment offset* field are removed from the base IPv6 header and given to the Fragmentation extension header. The IPv4 header options are similarly removed and redefined as extension headers. The IPv4 *internet header length* field is removed because the IPv6 main header length is always fixed at 40 bytes. The IPv4 *protocol* field is renamed as the *next header* field in IPv6, which may contain the upper layer protocol identifier or may indicate the presence of extension headers.

Clearly, the primary goal of IPv6 header optimization was to improve intermediate router processing performance. The IPv6 main header is fixed at 40 bytes so intermediate routers only need to process the addresses and the *hop limit*, but do not have to check the header length, scan through options (if they exist), or check or re-make checksums. In IPv6, the value of *0x00* in the *next header* field of the main header indicates that hop-by-hop options exist that a router must process. If this next header value is non-zero, the routers can ignore everything beyond the main 40-byte header.

These optimizations will go a long way to improving IPv6 performance since the majority of packet "touches" are optimized. Firewalls, however, need to adapt to this new header format, which is the subject of the remainder of this document.

2.2 Impacts of IPv6 on the Firewall

The performance optimizations described above in section 2.1 do *not* extend to intermediary firewalls. In contrast to the goal of minimizing header processing in routers, firewalls always need to view the complete header and upper layer information in order to apply a robust security policy. Since firewalls need to consider all of the header information, these new distinctions between the IPv6 main header and the extension headers do not simplify the firewall's task but make it more difficult.

A firewall does not process all fields within a packet in the same manner. Some fields or header options (multiple fields) can be evaluated autonomously and instantly acted upon. This type of packet cleansing function can be applied to all packets across the board for conditions that are *always* bad or always good. Other fields must be evaluated as part of a set of conditions put up against the firewall's configured and prioritized filtering policy. These packets are *sometimes* allowed, and so the firewall cannot instantly determine pass/fail. It must sequentially apply an ordered set of policy conditions entry by entry. This is more of an access control function configured by a network's administrators.

An example of the former case is when a firewall is configured to drop IP options in IPv4 packets. Any packet with an IP option in the header is dropped outright without further processing. The latter case can be represented by configuring the firewall to allow Telnet packets for a particular user and to drop them for everyone else. To enforce this the firewall must collect a set of packet characteristics (IP addresses, protocol type, port numbers) and apply the set of configured policy rules to this information. Clearly some filtering is harder to implement than others, especially when the design is in hardware. The impact of the IPv6 header structure on firewalls must be evaluated in terms of how it affects the ability to implement the required filtering in hardware. In particular, the access control type of filtering presents a challenge in IPv6 over what was required in IPv4. The

essential fields of *address*, *protocol*, and *port* are fairly easy to locate in an IPv4 packet. The *protocol* value is always in the main header. The TCP/UDP headers can be found directly after the main header, which has a separate *header length* field in the event that options are present. In IPv6, however, the *protocol* may be contained in the main header or it may occur after a set of hop-by-hop options, or after several optional extension headers. The fact that it can be in any one of a number of places adds complexity to firewalls.

Optional extension headers (including some with a variable number of sub-options) can occur in different orderings or an unspecified number of times, all of which dramatically increases the complexity for the firewall. The firewall must step through the set of extensions to determine if any of the options are dangerous and also to extract the critical data fields needed for filtering that occur along the way. It must also be careful not to run out of resources as it traverses through the chain of headers. This is non-trivial since "any number" of extension headers and options can occur according to standards.

The IPv6 base specification[1] contains many occurrences of "should" and "may", as well as other exceptional language such as: "Each extension header should occur at most once except for ..."[1] and "IPv6 nodes must accept and attempt to process extension headers in any order and occurring any number of times in the same packet, except for the Hop-by-Hop Options ..."[2]. This latter quote is highly problematic since it seems to say that an implementation will be considered compliant even if it fails miserably, as long as it *tries* its best to process the packet headers. It's hard to find other instances of a good faith effort being specified as a network protocol requirement.

Although flexibility in the main IPv6 specification is convenient for future enhancements, it is also detrimental to network security because it allows adversaries more avenues of attack. The above requirement of "must accept and attempt to process" essentially guarantees hackers that no matter how strange and unreasonable their attack packets are, they can rest assured that implementations will try to execute them as best they can.

2.3 Striking Back with Firewall Design Consensus

Firewall designs need to clamp down on some of the unconstrained flexibility allowed by IPv6 specifications. A more rigorous set of constraints is needed to define what is *reasonable* in IPv6 headers so that firewall hardware architectures can be optimized in achieving their filtering task. Packets beyond these constraints of reasonability are potentially dangerous (i.e. most likely attack packets) and must be dropped without consuming an exorbitant amount of the firewall’s resources.

It is important to establish that a firewall has the right and obligation to operate outside of the standards that govern normal IP traffic. Any suggestion that firewalls "must accept and attempt to process extension headers in any order and occurring any number of times in the same packet", per the IPv6 specification, must be rejected. Whenever a firewall applies security filtering to any packet it is operating beyond the authority of specification. If a firewall drops a Telnet packet destined for an inside host, it has technically violated the IP standards which describe how that packet is delivered to the

destination address contained in the packet. Dropping the packet at the firewall is beyond the scope of the standards and this same rationale applies to the new IPv6 protocol as well.

So the question is not "Is a firewall allowed to reign in standards?" but rather "What is reasonable for the firewall do?" and "How much flexibility should a firewall tolerate?" In another example, suppose an IPv6 packet is created with five consecutive **Destination Options** extension headers, some containing real options and some with only padding. This is an unreasonable yet technically legal packet. If the firewall allows this packet it must process all five headers and have system resources available to deal with the results. If that is acceptable, then how about ten consecutive headers, or fifteen? If the design will not drop unreasonable packets the attacker need only determine the firewall's resource limitations. Once the firewall design *does* decide to draw a line and drop everything over the line, the task is simply to determine where the line should be drawn. The details in Chapter 3 aim to determine where to draw the line by arguing what is reasonable.

IPv6 header filtering will be most effective if a general consensus among firewall vendors is reached as to what is allowed and what should be dropped as unreasonable. In this way, a consistent front is established by which host and router designs can abide.

3 Firewall Filtering of IPv6 Headers

This chapter covers the IPv6 header and packet structure. Each field in the IPv6 main header is analyzed in section 3.1 and all currently defined extension headers are analyzed in sections 3.2 through 3.8 to determine their security implications and to propose a filtering method. Different varieties of the same extension header are analyzed separately (e.g. various **Destination Options** or **Routing Header** types).

Constraints on header ordering, header combinations, and duplication of headers are mentioned along the way, but readers should refer to Section 3.9 for a detailed and focused treatment of this important issue for IPv6 firewall design.

The proposed filtering method varies for different fields or headers as explained in section 2.2, from simple packet cleansing to a more complicated access control function. This document does not provide firewall configuration guidance, but rather identifies the kind of filtering that is expected to be most useful for the field, header, or option being analyzed. Terminology for three basic methods is defined here and used throughout the rest of the document.

- ***On-Presence*** - this method is only applied to optional extension headers and options and refers to filtering determined by their mere presence in a packet. It applies to all packets crossing a given interface and should be configurable (per interface) unless specified otherwise.
- ***On-Validity*** - this method applies to filtering actions relating to invalid or unwanted conditions in a main header or extension header. It to applies to all packets crossing a given interface unless stated otherwise. These filtering actions

generally do not need to be configurable since they correspond to conditions that are always bad.

- ***As-Condition*** - this method refers to access control filtering whereby a condition is available to the system administrator for use in defining its ordered list of filtering rules. Traditional IPv4 examples are: IP Address, protocol field, port numbers etc ... This method is used to identify the IPv6-specific conditions needed for firewall configuration.

3.1 IPv6 Main Header Fields

3.1.1 Version

Link layers identify IPv6 as a separate type from IPv4. For example, Ethernet type *0x0800* is now interpreted as IPv4 (formerly "IP") and IPv6 is assigned the type *0x86DD*. Therefore, firewalls should check that the *version* field is correctly matched with its received *link type* field and drop any malformed frames. Failure to do this could result in a packet of version x being evaluated by the firewall under the filtering rules of version y, with unpredictable results.

Version identification is particularly important in processing tunnel packets. If an outer tunnel layer contains an upper layer protocol value of *0x04* the firewall should verify that the *version* field of the inner layer indicates IPv4 and if the upper layer protocol value is *0x29*, the *version* field of the inner layer must indicate IPv6.

The *version* field should be filtered on-validity of these conditions.

3.1.2 Traffic Class

Firewalls should have a minimal capability of zeroing out the *traffic class* field or dropping packets with a non-zero *traffic class* field. This function is needed most for outbound traffic to eliminate unwanted data channels. The on-validity style of filtering is appropriate though this function should be configurable to enable/disable.

Firewalls may or may not choose to incorporate a more advanced role in Quality of Service (QoS) functionality, though the subject is beyond the scope of this report.

3.1.3 Flow Label

Firewalls should implement a similar function for the *flow label* field as described above for the *traffic class* field. Again, the goal is to block unwanted data channels.

3.1.4 Payload Length

The *payload length* field in IPv6 indicates the number of bytes remaining in the packet (or packet fragment) beyond the standard 40-byte IPv6 main header. The one exception to this rule requires the *payload length* to be set to *0x00* whenever the **Jumbo Payload** hop-by-hop option[2] is used.

As a minimum, the zero value in payload length must not cause a firewall to crash or bypass filtering of the packet. The zero value should also be used as a validity check for

the **Jumbo Payload** option to remove the invalid cases of two conflicting length indicators from potentially confusing internal nodes. The on-validity method of filtering applies in this case.

Other filtering on the *payload length* may be implemented by a firewall design or inherited from existing IPv4 designs, though none are recommended here because they do not detect very many potential attacks. For example, the firewall could check that the length of the IP payload plus 40 bytes is consistent with the size of the link layer frame that delivered the packet; however, this would only have relevance to the link directly connected to the firewall since that information is regenerated at every hop and/or detected by a router along the way.

As explained later in sections 3.9 and 3.6.1.1 of this document, firewalls must be able to extract the full set of header data from a packet, including upper layer protocol and port values. Once this data is obtained, the packet can be filtered properly. A truncated packet must be dropped if this required data is missing. If a packet is truncated such that the tail end of the application-layer data is missing, the firewall has no context or means of detecting this condition. Such a packet is still filtered properly, however and the application must deal with the garbled data.

3.1.5 Next Header

The *next header* field is a crucial field for the IPv6 firewall. It corresponds to the IPv4's *protocol* field which most often indicates one of three common upper layer protocols: TCP (*0x06*), UDP (*0x11*), or ICMP (*0x01* in IPv4, *0x3A* in IPv6). Of course there are many other values defined by the IANA[3] that are less common.

In IPv6, however, the *next header* field differs from the IPv4 *protocol* field in the fact that it doesn't always point to the upper layer protocol. It may point to an optional IPv6 extension header and that header may point to another optional header. Eventually, the last optional extension header points to the upper layer protocol. There is no specified limit on the number of extension headers that could occur although there is a limit of what is reasonable.

The n*ext header* field processing creates a significant increase in complexity for firewalls, because they could previously extract the IPv4 protocol value in a fixed location, but with IPv6 that same information must be tracked down by progressing through a variable number of headers. Furthermore, instead of one value there are now *n* values to extract because the policy decision may be contingent on the presence of certain extension headers. The extraction of the (very important) upper layer protocol value and the processing of optional headers are intertwined in IPv6, so firewall designers do not have the option of simply ignoring extension headers. For an IPv6 firewall to reliably enforce filtering upon protocol/port values of a packet, it must be able to either find the upper layer protocol value or recognize when it is unable to locate the value due to lack of resources or some other failing.

This document discusses the handling of each extension header separately in the sections to follow. Though technically it is a part of each header's processing, the extraction of the protocol value from the *next header* field will not be discussed repeatedly in each section. Refer to Section 3.9 for a detailed analysis of header ordering and a recommended algorithm.

Another important point regarding the *next header* field in IPv6 headers is that the same number space (255 possible values) is shared between extension headers and upper layer protocols with no scheme other than doctrinal assignment. If an unknown value is encountered, there's no way to know if it's an extension header or an upper layer protocol. A firewall should treat this as both protocol unknown and header extensions unknown, and must have the capability to drop such packets. The on-validity filtering method applies here. Although firewalls may be designed with a configurable option to allow unknown values of protocol/extension header to pass, the DoD will require the ability to drop them.

3.1.6 Hop Limit

The *hop limit* field in IPv6 is used as a security check in the processing of Neighbor Discovery and Address Auto-configuration packets. Therefore the hop limit value of 255 is a useful filtering condition.

Filtering the *hop limit* field is particularly valuable in processing tunneled IPv6 packets. The specifications [3] [4] [5] require that the hop limit be decremented prior to IP tunnel encapsulation; therefore, any tunnel packet with an inner layer hop limit of 255 must be considered invalid and dropped. If the firewall is able to drop any tunneled packets where the inside IP header has a 255 hop limit, it will remove a whole class of possible Neighbor Discovery attack packets which require a value of 255 in order to be valid at the final destination. Note that this should apply to all inner IP layers if there are more than one. The *hop limit* field is filtered on-validity.

3.1.7 Source and Destination Address

The IP address fields are critical elements in the IPv6 main header and should be filtered (as-condition) by IPv6 firewalls just as they are in IPv4. Individual firewall policy entries will typically use source and destination IP address as primary filtering conditions.

Filtering source addresses of incoming packets has a double security benefit. See Figure 3-1 below. A potential attacker has two obstacles to overcome. First, since his address is blocked (1), he must guess a valid source address (2). Secondly, he must subvert the basic routing structure in order to get a response packet returned, otherwise, the response goes to the real user (3) where it is ignored as a stray and possibly logged as a security event. A legitimate user can send a packet (4) and receive the response (5) with no obstacles to overcome.

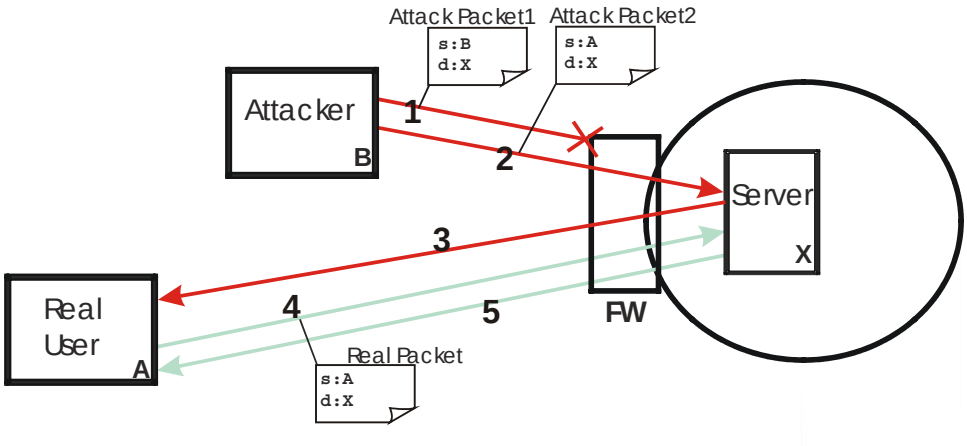


Figure 3-1: Double Benefit of Filtering Source Addresses

3.2 Option Types

IPv6 Options are individually defined in separate RFC documents and new options may continue to be added as needed. They are components (in Type-Length-Value format) that can appear in either of the two IPv6 extension headers designed to handle options: The **Hop-by-Hop Options** extension header or the **Destination Options** extension header.

The IPv6 standards provide one octet (256 values) to identify all option types without designating them as hop-by-hop or destination options. It is up to the RFC specifying the individual options as to whether they are valid in one or both of the extension headers. Firewalls need to know which options are allowed in which header if they are to enforce validity checking on the option type.

The standards also use the first two bits of the 8-bit type field to divide the space into four parts, each corresponding to a different action that a node should take if the option is unknown[4]. A firewall should **NOT** adhere to this practice. Instead firewalls should have a definite and configurable action for every option.

The set of currently undefined values for *option type* should be filtered as a group (on-presence) with a configurable option to ignore or drop them all. This group will be referred to as the **Unknown Options** in this document and currently consists of all values not specifically addressed in subsections of 3.2, 3.3, or 3.4 below.

To account for new options a firewall may anticipate the next few option values that would be assigned and separately decode them or have a programmable means to extract values out of the group of **Unknown Options**. This would allow new options to be distinguished from the **Unknown Options** and separately filtered on-presence. Advanced filtering (i.e. on-validity) can not be anticipated on a hereto-undefined option, but the individual ignore/drop capability may be useful.

IPv6 options are defined below if they are permissible in both the **Hop-by-Hop Options** and the **Destination Options** extension headers. If they are limited by specification to one extension header or the other, they are described in sections 3.3 or 3.4 as appropriate.

3.2.1 Pad 1 and Pad N Options

The **Pad1** and **PadN** options (*option type: 0x00* and *0x01*, respectively) are valid in both the **Hop by Hop Options** and the **Destination Options** headers. These options may appear more than once in either or both of these extension headers.

The pad "options" are only used to align other options to convenient byte boundaries but don't have any functionality themselves. The pad options should be accommodated as part of the firewall's packet processing but do not require security policy configurations.

A firewall should derive a filtering scheme for the pad options (on-validity) to abort processing and drop the packet if highly unreasonable padding is encountered. Although these packets may be benign, it is also possible that an attacker could use bizarre padding to confuse a firewall or expend a firewall's resources. By clamping down on unreasonable padding, this avenue of attack is removed.

The exact nature of the filtering scheme is left to firewall vendors, though some examples of unreasonable padding are as follows:

- More than one padding option back-to-back should never occur.
- **PadN** options with data length greater than 5 should never be needed (i.e. overall option length of 7). This assumes the RFC 2460 Appendix B guidelines of a maximum of an 8-byte boundary being defined.
- Any **PadN** option with data bytes that are not zeros should be dropped. This is suspicious behavior that may indicate a data channel.

3.2.2 Endpoint Identification Option

The **Endpoint Identification** option (*option type*: *0x8A*) is listed by the IANA as being related to the Nimrod routing system, circa 1996. There is no known RFC document that defines the option and only RFC 1992 (informational) to explain Nimrod routing.

It is not known if this option is/was intended as a **Destination Option** or a **Hop-by-Hop** option, or if there is any expectation of its implementation at all.

The on-presence filtering method is appropriate. Firewalls should detect this option and be configurable to ignore it or drop the packet.

3.3 Hop by Hop Options Extension Header

The **Hop-by-Hop Options** extension header, if present, must be the first header to follow the IPv6 main header. This is indicated by a value of *0x00* in the *next header* field in the main header. If a next header value of zero is contained at any place further down in the chain (i.e. in any extension header) the packet is invalid and should be dropped.

3.3.1 Jumbo Payload Option

The **Jumbo Payload** option has the *option type* value *0xC2* and is only valid as a **Hop-by-Hop** option[2]. This option allows the creation of very large IP packets and will likely only be appropriate on certain specialized networks. Network and link layer components that are not anticipating such packet lengths may experience performance degradation or failure. Furthermore, users with jumbo payloads may be able to achieve an unfair usage of bandwidth over other users. For these reasons, security policymakers should have the option of detecting and blocking this IPv6 option.

Firewalls should apply on-presence style filtering to this option and be configurable to allow it or drop the packet. If allowed, the firewall should apply additional on-validity style filtering on this packet. The following restrictions are specified and may be used as firewall validly checks:

- The IP payload length must be *0x00* when the **Jumbo Payload** option is present
- The **Jumbo Payload** option can only be used when the length is greater than 65,535 (i.e. the two most significant bytes of the jumbo length can not be *0x00*)
- The Jumbo Payload option cannot be used in conjunction with a **Fragmentation** extension header[5]

Note that any length checks performed on IP packets (Section 3.1.4) may be severely complicated by this option.

3.3.2 Router Alert Option

The **Router Alert** option has the value *0x05* for *option type* and is only valid as a hop-by-hop option[6].

The purpose of this option is to signal to routers that a closer inspection of the packet is warranted. The only security concern is regarding denial of service (DOS) attacks that could result if an attacker sends large numbers of packets with this option.

Firewalls should use on-presence style filtering on this option and be configurable to ignore it or drop the packet. The specification prohibits more than one **Router Alert** option to be present in a single packet even if the identifier within is different.

The IPv4 equivalent of this option is defined by RFC 2113[7]. Designers should be careful not to confuse the two versions which are very similar except that the option identifier (first byte) of the IPv4 header option is *0x94*. The value *0x94* is currently undefined for the IPv6 *option type* field.

3.3.3 Invalid Options for the Hop-by-Hop Options Extension Header

The following values are invalid in a **Hop-by-Hop** extension header *option type* field. Packets with these options in a hop-by-hop header should be dropped on-validity.

- Value *0x04*, Tunnel Encapsulation Limit
- Value *0xC9*, Home Address Destination option
- Value *0xC3*, NSAP Address option

3.4 Destination Options Header

The **Destination Options** extension header may appear in several locations and may appear more than once within an IPv6 header chain. This is reflected in the recommended header ordering guidance provided in Section 3.9.

3.4.1 Tunnel Encapsulation Limit Option

The **Tunnel Encapsulation Limit** (*option type*: *0x04*) is a destination option defined in RFC 2473[4]. The option is used to limit the number of IP-in-IP encapsulations that may be imposed on an original packet by informing tunnel entry points to reject packets that would otherwise be encapsulated and fragmented. The IPv6 Path MTU Discovery process already provides a means of signaling the optimal packet size back to an originating node, though there are still situations where an encapsulating node may have no other option than to fragment. The use of the **Tunnel Encapsulation Limit** option gives network administrators a better ability to detect and correct unwanted fragmentation scenarios when multiple tunnels are in use.

From a security standpoint, the **Tunnel Encapsulation Limit** option is non-threatening. A filtering strategy must be able to deal with all encapsulated IP layers of an arriving packet regardless of whether this option is present. One or more firewalls can be used to filter the IP layers, or the packet must be dropped if the number of layers exceeds the capability of the filtering strategy. A configuration to drop protocol type *0x04* and type *0x29* in the last analyzable layer achieves this requirement. Security issues with tunneled traffic are discussed in Section 5.2 of this document.

Firewalls are recommended to detect and ignore this option.

3.4.2 Home Address Option

The **Home Address** option is assigned the *option type* value *0xC9* and is part of Mobile IP processing defined in RFC 3775[8]. This option is only valid as a **Destination Option**.

The Mobile IP specification contains the following restrictions on placement:[6]

- It is illegal to have more than one **Home Address** option in a single IP header chain
- The **Home Address** option must appear after a **Routing Header** if one is present
- The **Home Address** option must appear before a **Fragmentation Header** if one is present
- The **Home Address** option must appear before the **Authentication Header (AH)** or **Encapsulating Security Payload (ESP)** headers if present

For a complete analysis of the Mobile IP scenario and a recommended firewall filtering strategy, refer to *A Filtering Strategy for Mobile IPv6[9]*. Conclusions from that analysis are stated below.

The analysis reveals three cases where filtering of the **Home Address** option is needed. First, the ability to drop all packets containing the option at a network boundary is needed. Separate configuration of this filtering for inbound and outbound traffic is required to support the various Mobile IP filtering scenarios (i.e. Home Network, Foreign Network, and Correspondent Network).

Secondly, the **Home Address** option must be filtered in conjunction with an ESP header and particular destination addresses in order to restrict home agent functionality to only the legitimate home agents. This is one of the rare occasions where the firewall **does** need to filter an ESP header in a packet. Inbound traffic at Home, Foreign, and Correspondent networks, is filtered to either allow or prevent home agents as site policy dictates.

Finally, it is highly desirable to have a firewall that can filter on the home address contained in the **Home Address** option. More specifically, to replace the IP source address with the home address from the **Home Address** option for the purposes of filtering the packet only (i.e. forwarded packets would be unchanged). Correspondent Network firewalls would use this source address swap to apply their normal filtering policies to all packets containing a **Home Address** option. This is an on-presence style *function* (rather than filtering action) since it applies to all occurrences of the **Home Address** option independently and prior to all other filtering actions.

If the address swapping function above is implemented, a firewall should also allow filtering on the true source address of packets with a **Home Address** option, to allow the rejection of unwanted Mobile IP Foreign Networks. A firewall would need the ability to distinguish these non-swappable source address filters from the normal ones that get swapped.

The analysis in [9] provides some alternate approaches to filtering Correspondent Networks if the home address swapping capability described above is not available. These approaches, require the as-condition style of filtering for the Home Address Destination Option. Specifically, a firewall would need to be configured to: allow "any source address" with a Home Address Destination Option and allowed protocol/port values for each allowed internal destination.

In summary, firewalls should implement an on-presence address swapping function to apply to all packets across an interface if enabled. Regardless of whether address swapping is implemented, the firewall needs the as-condition style filtering for the **Home Address** option. All designs will need the as-condition filtering to enforce proper use of home agents. Designs without the swapping function will have additional need of the as-condition style filtering to protect Correspondent Networks.

3.4.3 Network Service Access Point (NSAP) Address Option

The **NSAP Address** option (*option type: 0xC3*) is assigned as a **Destination Option** by RFC 1888 and deprecated (reclassified as historic) by RFC 4048. At the time of this writing the value is still assigned though this is likely to change at some future time.

Firewalls should filter on-presence and be configurable to ignore the option or drop the packet.

3.4.4 Invalid Options for the Destination Options Extension Header

The following values are invalid in a **Destination Options** extension header *option type* field. Packets with these options in a **Destination Options** header should be dropped on-validity.

- Value *0xC2*, Jumbo Payload
- Value *0x05*, Router Alert

3.5 Routing Headers

The **Routing Header** extension headers are analyzed separately according to the routing type field.

3.5.1 Type 0 (Source Routing)

The **Type 0 Routing Header** is the functional equivalent to the IPv4 loose source routing option. This option is rarely used in IPv4 and is typically blocked by firewalls because of its security risks. These same risks exist for the IPv6 **Type 0 Routing Header**.

Back in Figure 3-1, filtering of source addresses was shown to have a double security benefit. If the same site accepted the **Type 0 Routing Header**, the double benefit is cut in half. An attacker would still need to know/guess a valid source address that is acceptable at the target site. However, with the **Type 0 Routing Header**, traffic would proceed along a deliberate path in both directions, allowing the attacker to insert himself in that path. See Figure 3-2 below.

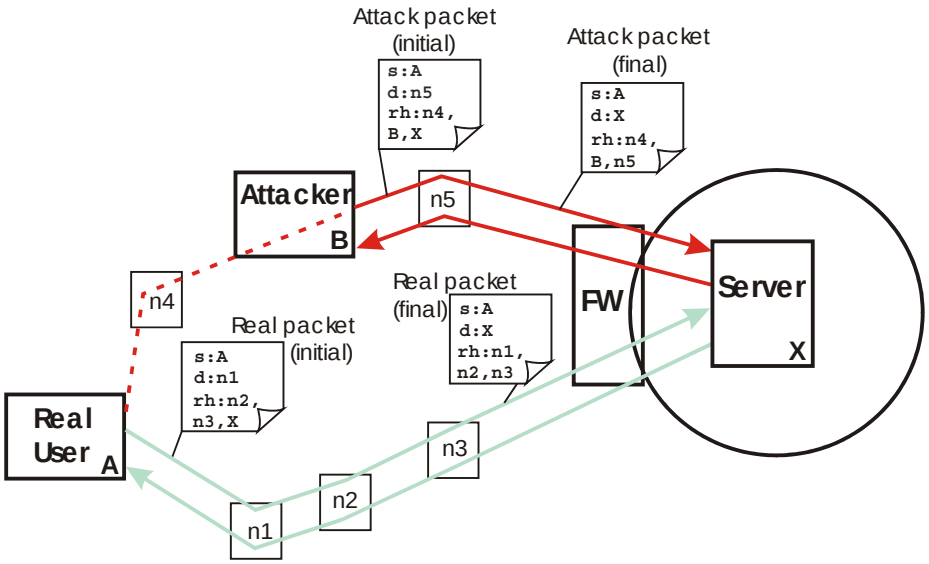


Figure 3-2: Type 0 Routing Header Threat

The **Type 0 Routing Header** is dangerous because it allows the attacker an easy means of receiving return traffic when pretending to be another user. The dashed line in the figure above indicates the portion of the path that the packet only appears to take. It actually originates from the attacker at node B.

Another threat from the **Type 0 Routing Header** is that it can hide the true packet destination from a firewall. If a firewall ignores the routing header and applies the filtering policy to the packet's destination address, it can be fooled into allowing traffic that should be blocked. Traffic is sent through the firewall to an allowed node on the inside, and normal processing of the routing header sends the packet to a different target node. Normally the firewall would block traffic sent to the target node.

Firewalls require on-presence filtering of the **Type 0 Routing Header** and should be configurable to ignore the header or drop the packet. Most networks will configure to drop this traffic as is done today with IPv4 source routing.

A second capability is also desirable for IPv6. It should be possible to allow the **Type 0 Routing Header** only for packets containing an IPsec header (either **AH** or **ESP**). With the security headers present, the attacker can no longer spoof the packet and so the above threats are removed. If this capability is provided, administrators should have three options in all:

- Drop all packets containing **Type 0 Routing Header**
- Ignore the **Type 0 Routing Header**
- Ignore the **Type 0 Routing Header** if a security header is also present, and drop all packets containing a **Type 0 Routing Header** without a security header[7]

The **Type 0 Routing Header** can actually strengthen an IPsec system in the sense that it can route encrypted packets through safer paths in an internet thereby reducing the vulnerability of the packets.

3.5.2 Type 1 (Nimrod routing)

Similar to the Endpoint Identification option (Section 3.2.2), the value *0x01* for the *routing type* field of a **Routing Header** is reserved by something called "Nimrod Routing" of which little is known and less is specified.

Firewalls should filter this extension header on-presence and be configurable to ignore it or drop the packet.

3.5.3 Type 2 (Mobile IP)

The **Type 2 Routing Header** is used exclusively in support of Mobile IPv6. Though structured similarly to the **Type 0 Routing Header**, the Type 2 header allows only one address swapping action to occur. This swap can only occur at the destination node indicated by the original destination address.

Type 2 Routing Headers are only processed by the final destination node and the packet cannot be forwarded again after the header is processed.[8] Therefore, the Type 2 header does not present the security threats that occur with the **Type 0 Routing Header** and should generally be allowed for use[9].

The specification recommends that a **Type 2 Routing Header** "SHOULD" occur after a **Type 0 Routing Header** if both are present. Since the Type 2 header effectively operates as a destination option, it makes no sense to ever have a Type 0 follow a Type 2 header. Firewalls should enforce this "SHOULD" as a "MUST".

The reader should refer to [9] for a complete analysis of the Mobile IP scenario. From that analysis, filtering of the **Type 2 Routing Header** is only needed when it is necessary to prevent a site from acting as a Foreign Network and this can be achieved with the on-presence style of filtering.

Firewalls should detect this option and be configurable to ignore it or drop the packet.

3.6 Fragmentation Header

Packet fragmentation at the IP layer poses a difficult challenge for a firewall trying to enforce a security policy. This section presents the security issues and will make some design recommendations for handling IPv6 fragmentation at the firewall; with a goal of being secure, practical to implement, and tolerant of all reasonable traffic.

3.6.1 Fragmentation Security Issues

3.6.1.1 Issue 1: Extracting Header Information

The first issue with fragmented packets is in the challenge of extracting critical header information needed to apply full security filtering. If there are **n** fragments of a TCP/IP packet, the TCP header and port information will only be in one of the fragments. The other fragments have insufficient information for independent judgment and would normally be dropped if it weren't taken into consideration that they are fragments of a larger packet.

If any of the fragments are dropped, the whole packet is discarded when the reassembly operation times out at the destination host. Therefore, if the firewall can apply full security filtering to any one fragment, it can consider remaining fragments to be harmless, not counting the loss of resources as the reassembly operations times out.

If critical header information is straddled across two fragments, the firewall's extraction task is complicated because it must maintain state across multiple packets. The missing fragment may be significantly delayed, fragments need to be queued, and the location of fields within a fragment must be synchronized with the previous fragment. Once the firewall begins down this path, it would be easier to perform full packet reassembly prior to filtering. Ideally, all of the required data needed to filter the packet can be extracted from a single fragment.

A firewall design must be able to determine if it has extracted the entire set of required data. For IPv6 this is significant because of the optional number of extension headers. It must be possible to drop a packet for which an incomplete set of data is extracted.

3.6.1.2 Issue 2: Fragment Overlaps

Filtering on a partially assembled packet can also be a security risk, which brings up the second issue with IP fragmentation: overlapping fragments. Fragment overlap attacks originated in IPv4 and unfortunately, the IPv6 specification does nothing to prevent them. The *fragment offset* field and fragment length are used to reconstruct the original packet and it should always be the case that the offset of fragment **n** plus the length of fragment **n** adds up to one byte short of the offset of fragment **n+1**. The offset value points to where the fragment is positioned in the overall reconstructed packet and there should never be any overlap in a normal fragmentation. The overlap attack creates packet anomalies whereby the fragments *do* overlap and it depends on individual implementations as to which data ends up being used in the final reassembled packet.

The fragment overlap attack works by sending fragments to a firewall which appear to be one thing but when reassembled at the inside host, are actually something else. For example, if the first fragment indicates that the packet is a TCP port 80 (http) packet, it might pass through the firewall. A subsequent overlapping fragment overwrites the port 80 portion with port 23 (Telnet) data, thereby sneaking past the firewall, which is

supposed to drop incoming Telnet packets. This attack works against firewalls that try to evaluate only the first fragment against a security policy.

3.6.1.3 Issue 3: Partial Set of Fragments

A third issue with packet fragments at a firewall is whether the security enforcement imposes any hardship on internal hosts by means of reassembly timeouts. Although evaluating/dropping a first fragment is secure, the remaining fragments get through creating an unwanted side effect. The partial set of fragments ties up resources on the destination host which waits for a missing fragment that never arrives. Fragmentation timeouts are specified to be 60 seconds, which is a long time to hold resources.

It is important to note that an attacker can always send fewer than all fragments as a deliberate denial of service (DOS) attack. Security filtering cannot cure this problem outright but can only move it from one place to another. For example, if a firewall performs packet reassembly it can shield the inside hosts from this attack, but now the firewall itself is subject to the same attack. If the firewall is shut down by a DOS attack, all inside hosts are disabled.

3.6.1.4 Issue 4: Nested Fragmentation

Nested fragmentation is a fourth issue. In IPv6, fragmentation is only allowed by the originating node, not by intermediate routers. The standard is ambiguous and possibly contradictory, however, as to whether an occurrence of nested fragments is technically compliant or not. It must be assumed that nested fragments are not intended behavior because RFC 2460 states that only initiating hosts can fragment and that “the lengths of fragments must be chosen such that the resulting fragment packets fit within the MTU of the path to the packets' destination(s)”[10]. Given these two requirements, we can conclude that there is never a justification for a second fragmentation header to appear in an IPv6 header chain[11]. However, the specification does not explicitly forbid it and it does have a statement saying that implementations should try to deal with any number of extension headers that occur in any order[12].

Unique to IPv6, nested fragmentation consists of a set of fragments which, after reassembly, contains yet another fragmentation header indicating that a fragment had been fragmented. This is different than what happens in IPv4 when a fragment is further fragmented by an intermediate router (which is allowed in IPv4). In that case, the existing fragmentation fields in the IPv4 header are modified as the fragments are broken into smaller pieces. Multiple fragmentation headers in an IPv4 packet is not possible because this information is part of the main header.

The security threats of nested fragmentation are in its complexity and obfuscation of the true nature of the packet. A firewall may be fooled or a host may be "crashed" if it is not able to deal with this unexpected case.

3.6.1.5 Issue 5: Fragmentation and Tunneling

Fragmentation and packet tunneling are interrelated because tunneling adds an outer IP header to an existing IP packet, creating a potential conflict with the MTU size. Fragmentation may be needed at a tunnel entry point if Path MTU Discovery can not adjust the packet size at the source.

Generally speaking, these fragmentation issues are more operational problems than security issues. RFC 4459 [10] explains a variety of cases that may be encountered and how the problems might be solved. Tunneling has its own security issues as described in section 5.2 of this document.

Fragments may be tunneled and tunneled data may be fragmented. A firewall design strategy for handling fragments should work in unison with its strategy for handling tunnels. It is also important that no design allow an attacker a means to escape filtering by using a particular combination of fragmentation and tunneling.

3.6.2 Two Firewall Design Approaches to Fragmentation Handling

A design decision must be made as to whether the firewall will reassemble fragments and filter the whole packet, or whether it will filter fragments and pass them through to the destination host on the secure side (inside). Both cases have advantages and disadvantages, so both will be viewed as acceptable design approaches by this document.

A firewall that has "deep packet inspection"[13] capabilities will likely choose to reassemble fragments since it needs to see deep into the application layer of packets. The more traditional firewalls may prefer to avoid the overhead of fragment reassembly to achieve higher performance (packet throughput). If a firewall is designed to filter tunneled traffic, the potential for fragmentation in multiple IP layers may be too labor intensive for a reassembly approach.

The lists of pros and cons below show that this is a non-trivial decision for a design team to make.

Advantages of fragment reassembly:

- security filtering is always applied to the full packet
- overlap attacks will fail because the firewall filters the fully reconstructed packet
- inside hosts are shielded from partial fragment DOS effects and deliberate DOS attacks via fragments
- deep packet filtering can be performed

Disadvantages of fragment reassembly:

- performance hit of having to do the reassembly on the firewall for all inside nodes (this could be huge)
- packets may need to be re-fragmented after filtering if they are too big for the inside MTU (even more of a performance hit)
- the partial fragment DOS attacks can be made against the firewall itself, which if successful, could effect all inside hosts
- tunneled traffic can be very complicated with fragments at inner layer, outer layer, or both

3.6.3 Recommendations for Firewalls that Reassemble

3.6.3.1 Issue 1 (data extraction) and Issue 2 (overlap attacks) for Firewalls that Reassemble

If reassembly is performed by the firewall, it is important that **full** reassembly be performed, and that the firewall filters and passes the reassembled packets (not the fragments). Issue 1 is eliminated since the full packet is filtered. The threat from issue 2 is also removed.

The firewall must NOT reassemble fragments, filter the result, and then pass the original fragments because there is no guarantee that inside hosts will reassemble overlapped fragments in the same manner as the firewall (i.e. it is outside of specification). The firewall may have to re-fragment the packet, which is not a security threat because the firewall is trusted not to create overlaps.

3.6.3.2 Issue 3 (partial set of fragments) for Firewalls that Reassemble

A reassembling firewall doesn't filter individual fragments and therefore does not *create* the unwanted side effect of a partial set of fragments. Although fragments may reach the inside hosts, they would be generated by the firewall due to a re-fragment operation after filtering. The inside hosts would therefore receive a full set of fragments very reliably.

A partial set of fragments sent by an attacker as a deliberate DOS attack is still a concern. Although the attack can not reach inside hosts, it can affect the reassembling firewall. A good design strategy to limit the effects of this attack is to incorporate a threshold on reassembly timeout occurrences over a period of time. For example, if X reassembly timeouts occur in Y minutes some action is initiated such as: temporarily block a source address, temporarily reduce the reassembly timeout, temporarily drop fragments, and/or generate an alarm. DOS attacks are difficult if not impossible to completely prevent. Quick detection of an ongoing attack is the most important element.

3.6.3.3 Issue 4 (nested fragmentation) for Firewalls that Reassemble

A firewall that reassembles should process the first **Fragmentation Header** encountered in a packet. If the reassembled packet contains another **Fragmentation Header** (i.e. nested fragmentation) then the firewall should drop the packet. Discouraging this behavior is better for all nodes in the long run.

3.6.3.4 Issue 5 (fragmentation and tunneling) for Firewalls that Reassemble

Firewalls must expect fragments to occur simultaneously in each IP layer of a tunneled packet. If a firewall is designed to filter multiple layers of IP tunneling, the worse case fragment reassembly is particularly painful. Both layers can be fragmented and the firewall can have multiple reassembly operations to perform before it is able to filter the packet.

Furthermore, reassembly of a fragmented inner IP layer is more complicated than that of untunneled traffic and could produce unwanted results. For example, assume that a packet is divided into ten fragments at the originating node ***A***, travels to a tunnel entry point ***B***, and then to its final destination ***C***. The result is that tunneled packets with the inner layer fragmented are sent from node ***B*** to node ***C***. If these fragments are reassembled by a firewall prior to reaching ***C***, it is not clear which of the ten outer IP headers is appropriate to assign to the reassembled packet and if any important data could potentially be lost in the other nine outer headers. The tunnel entry point is not expecting that nine of its next ten headers will be dropped and hence this potential for unexpected behavior. The standard states that the headers present in the first fragment are used to make the final reassembled packet[14]. However, this requirement is made with the assumption that fragments are only reassembled at the endpoint in which case the outer IP tunnel layers are gone (already processed). The case of a firewall reassembly midstream is unique and consequential.

Re-fragmenting a tunneled packet after filtering is also an issue. If the reassembled packet is too big for the firewall to forward, should it re-fragment in the same manner the data arrived? If yes, then it would need extensive resources to remember how the packet was originally split and could potentially need to perform multiple fragmentation operations if both layers were originally fragmented. If it fragments only the outer layer it could produce unwanted effects with the new arrangement such as burdening a tunnel exit point with fragment re-assembly operations. Network administrators may set up fragmentation and tunneling to operate in a particular way for maximum efficiency and the firewall could alter their intensions.

In conclusion, firewalls designed to filter multiple layers of IP traffic should avoid the fragmentation re-assembly approach due to the very high worse-case processing impact and the potential for creating unwanted side-effects.

3.6.4 Recommendations for Firewalls that do NOT Reassemble

3.6.4.1 Issue 1 (data extraction) and Issue 2 (overlap attacks) for Firewalls that do NOT Reassemble

Firewalls that do not reassemble fragments need another strategy for dealing with these security issues. In IPv6, the minimum specified MTU[15] is 1280 bytes, which is up from 576 bytes in IPv4. An implementation of IPv6, therefore, cannot be forced (or expected) to fragment any packet that is 1280 bytes or smaller. If the link layer media cannot handle 1280 bytes (e.g.; ATM) the fragmentation/reassembly must be performed by the link layer[16].

Even with IPv6's extension headers and increased address size, 1280 bytes is a lot of space and it is a very reasonable expectation that the entire IP header (including extensions) plus the upper layer protocol headers will fit within a first fragment. The 1280 byte space equals 320 four-byte words. A hypothetical worse-case packet using all of the known headers and options appears as follows (with the word size in parenthesis): IP main header (10), router alert option (1), routing header (50), home address option (5), fragmentation header (2), tunnel encapsulation option (1), mobile IP header (20), TCP (4)[17]. The routing header of 50 words assumes a very large number of defined hops (twelve) as a worse case. It is also very unlikely that any packet would use all of these headers simultaneously. Adding all of the headers up comes out to 93 words. Doubling everything except the TCP portion gives 89+89+4 =182 words, which can be used to represent a worse-case IP-in-IP tunneled packet.

This is still far below the 320 word boundary that corresponds to the 1280 byte MTU and thus it is reasonable for a firewall to expect that all extracted data will be found in the first fragment.

A closer look at the construction of a fragment is warranted. An IPv6 fragment consists of an "unfragmentable part", an 8-byte Fragmentation header, and some number (**n)** 8-byte blocks that make up the "fragmentable part". See Figure 3-3 below.

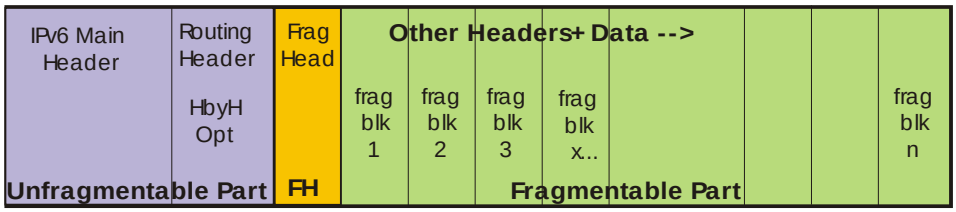


Figure 3-3: IPv6 Packet Fragment

The unfragmentable part consists of all data required for packet delivery, and therefore required to be present in each fragment as well. Typically, this will be the Hop-by-Hop Options and the Routing Headers, if they exist. All other headers in the original packet are placed in the fragmentable part. It is incorrect to deduce that these remaining headers were designed to be in the fragmentable part for fear of a header being larger than the minimum MTU. Instead, it is merely an efficiency measure. The unfragmentable part is minimized because it is repeated multiple times (once in every fragment).

The Fragmentation Header contains a *fragment offset* field which indicates the starting position (i.e. the x^{th} 8-byte block) of the data contained in this particular fragment. For example, the first fragment always has an offset of *0x00*. If that first fragment contains 120 8-byte blocks of data in the fragmentable part, the second fragment will contain a *fragment offset* value of *0x79* (decimal 121).
A secure firewall design strategy for handling fragments needs to be able to extract a complete set of critical data required for filtering. It is also preferable to extract all data from a single fragment to avoid the difficult task (in hardware) of maintaining state

across multiple packets. The expectation that all of the extracted data will appear in the first fragment has already been argued as reasonable. Furthermore, the integrity of the extracted data must be guaranteed by preventing subsequent fragments from overlapping back into the extracted data.

The 1280-byte minimum MTU boundary is not very useful as a filtering condition as shown in Figure 3-4. The second fragment will contain a *fragment offset* value that bears no relationship to the overall packet size of the first fragment.

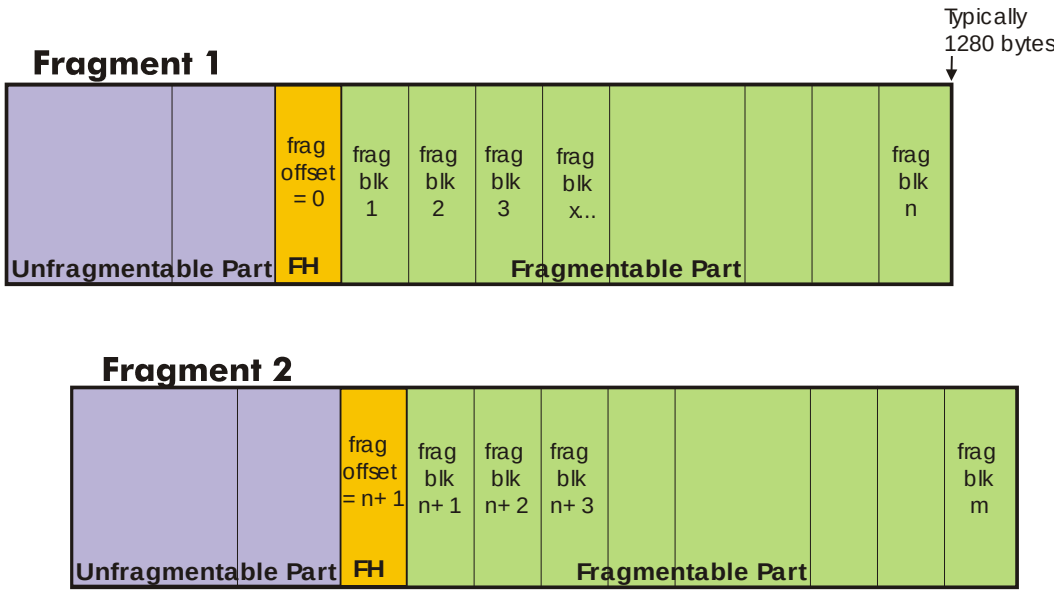


Figure 3-4: Fragment Offset Value

If the unfragmentable part of the first fragment expands, the Fragment Header moves to the right and there is less room for the fragmentable part. The value **n** goes down in this case as the overall fragment size stays the same. A check to prevent fragment overlap must verify that the offset of fragment 2 (normally n+1) is not erroneously set to a smaller value that would overlap with fragment 1. This check is not a function of the 1280 MTU size.

There is also no guarantee that the first fragment will be 1280 bytes though that should be the normal case. As an example of this, a 1500 byte original packet would probably be split into a 1280 byte first fragment with the remaining 220 bytes and unfragmentable part in fragment 2. It is not prohibited by the standard, however, to have an 800 byte/700 byte split since the fragments are required only to "fit within the path MTU" to the destination. Given that there is already a lot of complexity involving fragmentation-tunneling solutions as stated earlier (RFC 4459), it is highly undesirable to create any constraint here by assuming that the first fragment will always be at least 1280 bytes. For these reasons, there should be no required checks based upon the fixed value of 1280.

A better approach is to name a new boundary that will be referred here as the **EVAL_SIZE** as shown in Figure 3-5 below. The EVAL_SIZE is a number of 8-byte

fragment blocks from the start of the first fragment that is big enough to include all of the extracted data for filtering, and small enough to fit within the first fragment.

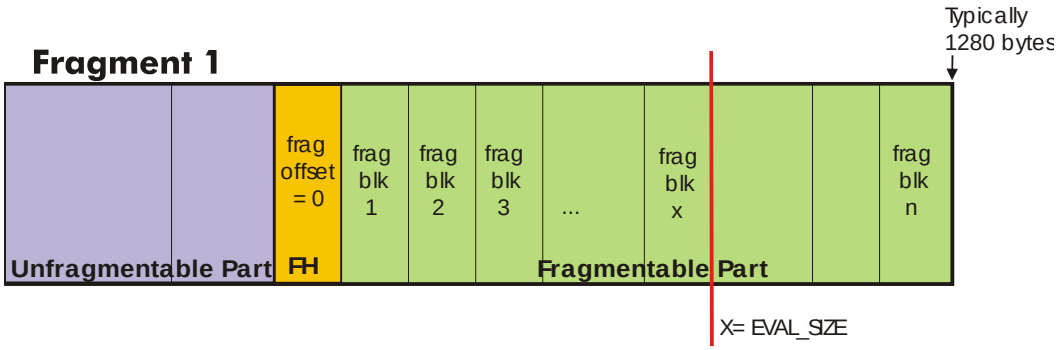


Figure 3-5: Design Recommendation, EVAL_SIZE

The firewall must perform three checks to securely handle fragments using this scheme. These checks should be relatively straightforward to implement, friendly to reasonable traffic, adaptable to extreme traffic, and, most importantly, secure. It must verify that:

1) There are at least EVAL_SIZE 8-byte blocks of "fragmentable part" data in the first fragment.
2) All of the data required to be extracted for filtering is contained within those first EVAL_SIZE blocks of the first fragment.
3) The *fragment offset* field of all non-first fragments is greater than EVAL_SIZE

A default value of 60 is recommended for EVAL_SIZE though designers are encouraged to make this configurable to values higher and lower than 60, allowing firewalls to adjust to unusual environments[18]. For example, packets with an extremely large amount of extension header data might require a larger EVAL_SIZE, whereas an environment that creates fragments much smaller than 1280 bytes may need to reduce the EVAL_SIZE. Once configured, the firewall would apply the chosen value to all fragment testing. The third check above is the *only* check performed on non-first fragments. If it passes, the fragment is allowed through without further processing.

The main advantage of this scheme is that a fixed test can be applied to each fragment based solely on its own contents. There are no coordinated checks required across multiple packets such as trying to determine if n+1 of fragment 2 overlaps with what is defined in fragment 1. Instead, the fixed value of EVAL_SIZE cordons off the filtered data in fragment 1 and the checks to other fragments prevent overlap back into this space. Variance in header construction and/or variance in packet length can cause the value of n to increase or decrease as long as it never crosses the worse case boundary of EVAL_SIZE.
The scheme addresses the first two security issues: data extraction and fragment overlaps[19]. Other approaches are possible but they must be secure with respect to these two issues. Regardless of the scheme, it should be considered “reasonable” in IPv6 to drop a fragmented packet if the full set of filtering data is not contained in the first fragment.

3.6.4.2 Issue 3 (partial set of fragments) for Firewalls that do NOT Reassemble

The process described above may drop a fragment while one or more remaining fragments still pass through. Although secure, the inside host wastes resources in this scenario, as it waits for reassembly timeout. A threshold mechanism should be incorporated to detect a certain number of dropped fragments from the same source address in a period of time. Upon activation of the threshold detector, the firewall should activate an alarm and/or temporarily block all fragments from the source. Administrators can quickly learn of the particular usage of headers or tunneling that is causing the firewall to drop fragments and take corrective action. The action of blocking a source address (rather than activating an alarm) should be optional to use since it could potentially become a means of a DOS attack.[20]

A firewall should not take the approach of dropping all non-first fragments following a first fragment that has already been dropped. This will not be reliable because there is no guarantee that the first fragment is transmitted first. In fact, some designs send fragments in reverse order with the last fragment sent first so the receiver can immediately calculate the size of memory needed to store all of the fragments.

A partial set of fragments can also be sent by an attacker as a deliberate DOS attack. A firewall that does not reassemble fragments can not easily detect or defeat this attack and it is the job of an administrator to correct. This should be tolerable since DOS attacks are relatively rare, highly detectable, and do not cause the permanent damage of other attacks (i.e. data loss or destruction). The reassembly timeout on inside hosts can be decreased from the standard default of 60 seconds to improve the resistance to these specific DOS attacks.

3.6.4.3 Issue 4 (nested fragmentation) for Firewalls that do NOT Reassemble

Nested fragments (issue 4) should be dropped regardless of whether the firewall reassembles fragments or not. For non-reassembling firewalls, the detection of two Fragmentation headers within the set of extracted/filtered data should cause the packet to be dropped. Refer to section 3.9.1 for a proposed Header Ordering Algorithm that implements this check.

3.6.4.4 Issue 5 (fragmentation and tunneling) for Firewalls that do NOT Reassemble

The scheme described above in section 3.6.4.1 is adaptable to extreme fragmentation-tunneling scenarios, if the implementation offers EVAL_SIZE values other than just the default value. Firewalls that receive first fragments much less than 1280 bytes may need to reduce the EVAL_SIZE value and those that receive packets with extremely large headers (i.e. larger than the worse-case estimated in this document) may need to increase the EVAL_SIZE. Administrators should be able to find a workable setting in the event that tunneling causes one of these unusual variants to occur.

Later in section 5.2, tunneling is discussed in more detail and with the objective of filtering both an inner and outer IP layer. Firewalls that filter multiple IP layers need a fragmentation-handling scheme that works in the worse case scenario where both IP layers contain fragments simultaneously. For the scheme described above, the default

EVAL_SIZE value of 60 8-byte blocks was chosen to protect enough bytes (from overlap attacks) to cover tunneled traffic in two layers of IP with all of the extension headers present. To fully adapt the scheme to two layers of filtering; however, a little more work is required to apply the three required checks to each layer. The value of EVAL_SIZE for an inner IP layer must be decreased since there is less remaining data to be extracted. The default value for the inner IP layer should be 16 8-byte blocks (128 bytes) assuming it is the last IP layer to be filtered.

Figure 3-6 below shows this worse-case scenario. An originating host transmits a fragmented packet. The fragment reaches a gateway where it enters a tunnel and must be fragmented again before being forwarded on to its destination[21]. The packet encounters a firewall before reaching its final destination or the tunnel exit point. The firewall, therefore, sees a tunneled packet with fragmentation in both layers.

The dashed lines in Figure 3-6 indicate that the IP_A layer fragments are too big for the IP_B tunnel layer and the remaining bytes are transmitted in a second IP_B fragment. All of the data that needs to be filtered occurs in line (1) of Figure 3-6, that is, the packet containing the first fragment of both the inner and outer layers.

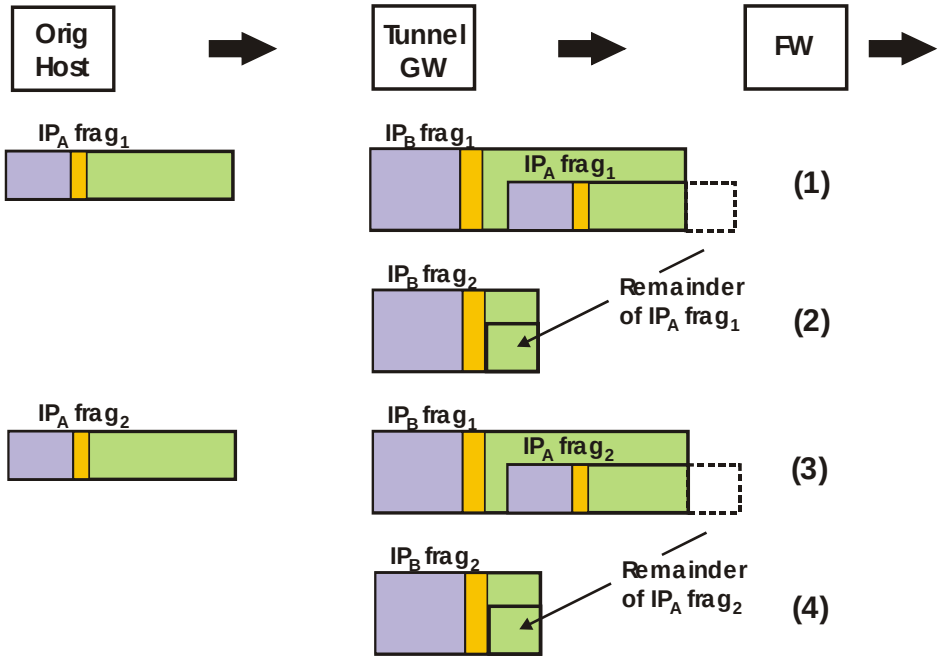


Figure 3-6: Worse Case Fragmentation-Tunneling Illustrated

The previously described scheme can be applied to both layers as follows. Refer to Figure 3-7 below for an illustration of the $EVAL\_SIZE_B$ boundary corresponding to the outer layer checks and $EVAL\_SIZE_A$ corresponding to the inner layer checks.

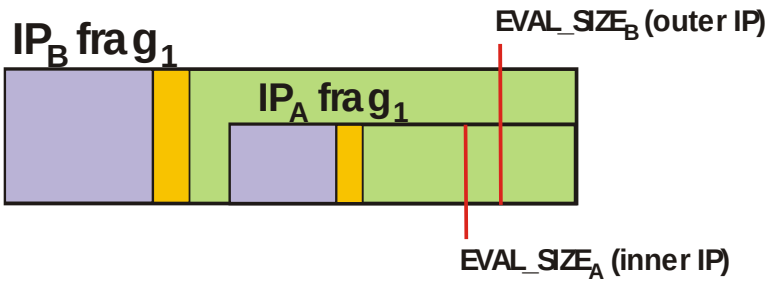


Figure 3-7: Fragmentation Checks for two layers of IP

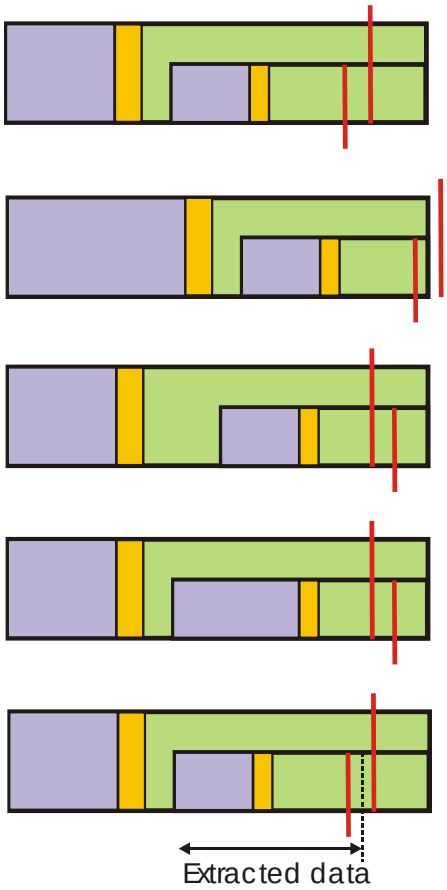
The outer layer (IP_B) requires non-first fragments to have offsets greater than $EVAL\_SIZE_B$ in which case the whole packet is passed without further analysis (no inner layer analysis required). Non-first fragments with offsets less than or equal to $EVAL\_SIZE_B$ are dropped. This corresponds to lines (2) and (4) of Figure 3-6: Worse Case Fragmentation-Tunneling Illustrated.

Similarly, non-first fragments of the inner layer (IP_A) must have inner offsets greater than $EVAL\_SIZE_A$ in which case the packet is passed without further analysis. Non-first fragments in IP_A with offsets less than or equal to $EVAL\_SIZE_A$ are dropped. This corresponds to line (3) Figure 3-6. Therefore, all but one packet in this scenario, can be judged solely on the values of *fragment offset* fields.

When the fragmentation offsets of both IP_A and IP_B are zero (i.e. first fragments in both layers), the processing continues and extracts the data needed for filtering. Additionally, the $EVAL\_SIZE_B$ boundary must occur prior to the end of the packet and the $EVAL\_SIZE_A$ boundary must occur prior or equal to the $EVAL\_SIZE_B$ boundary. Refer to Figure 3-7: Fragmentation Checks for two layers of IP. $EVAL\_SIZE_A$ can draw even with $EVAL\_SIZE_B$ but must not extend beyond it. Finally, the full set of data required for filtering must occur within the $EVAL\_SIZE_A$ boundary.

Figure 3-8, below shows the normal case and some failures cases for processing the essential packet (i.e. both fragments offsets are zero). The checks on all other packets (i.e. non-first fragments) are trivial and not shown below.

In summary, the fragmentation-tunneling scenarios are difficult to explain and illustrate as evidenced by the figures in this section. A scheme should be judged on how difficult it is to implement, not on how difficult it is to explain. The scheme presented here is believed to be easy to implement and does not require synchronization across multiple packets.



Good Case.

Failed Case: A very large unfragmentable part of the outer layer results in less than $EVAL\_SIZE_B$ blocks in the packet. This can also occur if the packet size is much less than 1280 (though that would be atypical).

Failed Case: A very large number of extension headers in the fragmentable part of the outer layer pushes $EVAL\_SIZE_A$ outside of the $EVAL\_SIZE_B$ limit.

Failed Case: A very large unfragmentable part of the inner layer pushes $EVAL\_SIZE_A$ outside of the $EVAL\_SIZE_B$ limit.

Failed Case: A very large number of extension headers in the fragmentable part of the inner layer pushes extracted data outside of the $EVAL\_SIZE_A$ limit.

Figure 3-8: Tunnel-Fragmentation, Exceptional cases

3.7 Mobility Header

The **Mobility Header** is the third and final header used in the Mobile IP scenario. Although the **Mobility Header** is defined as an extension header, it is also required that its *next header* field hold the value *0x3B* (no next header) so this header essentially acts as an upper layer protocol.

Again refer to [9] for a complete analysis of Mobile IP filtering. From that analysis several filtering capabilities are identified.

Firewalls must be able to resolve individual message types within the **Mobility Header**. Recognition of the Binding Update, Home Test Init, and Care-of Test Init in particular are needed, though a firewall should be able to filter any Mobility message type.

In addition to the message type, the firewall must be able to resolve the H flag within a Binding Update message. This distinguishes between home agent bindings and normal correspondent bindings and is significant to the filtering strategy. Depending on the particular scenario, a firewall may need to drop all Binding Update messages or drop all Binding Update messages with an H flag=1.

Firewalls should verify that the *next header* field in any **Mobility Header** is *0x3B* and drop any packet containing any other value.

The as-condition style of filtering is recommended for the Mobility Header as the best means of being prepared for any situation. The on-presence style of filtering may be good enough in some designs, though a very careful study of the scenarios in [9] should be made before choosing the on-presence style. The home agent element of the Mobile IP scenario will receive tunneled traffic containing a mixture of local traffic and relay traffic (reverse tunneling mode). Firewalls need the ability to distinguish these two paths for messages with a **Mobility Header**. Depending on the firewall's tunnel filtering capabilities and the physical nature of the home agent (i.e. one network interface or more than one), the filtering task may become complicated. The as-condition style will make this easier to manage.

3.8 ESP and AH

The ESP and AH security headers are IPv6 extension headers; however, they are best treated by a firewall as if they were an upper layer protocol value. The interaction of Firewalls and IPsec is handled as a separate topic in Section 6. Further details are deferred to that chapter.

3.9 Header Ordering and Duplicate Extension Headers

The firewall vendor community must reach a consensus on what is reasonable to accept in terms of IPv6 header ordering and occurrence. Throughout Section 3, the restrictions for individual headers and options have been identified according to their defining specifications (RFCs). Here, an overall header ordering algorithm is proposed that incorporates these specified restrictions and also imposes some additional reasonable restrictions on the unacceptably unconstrained nature of IPv6 headers.

3.9.1 IPv6 Header Ordering Algorithm

The algorithm illustrated below in Figure 3-9 represents a process for approving/rejecting IPv6 headers due to header ordering, duplicate headers, or certain prohibited combinations of headers. This algorithm is applicable to a single IPv6 header chain (i.e. nested IP headers would be analyzed individually). This algorithm is limited to header construction issues which is only a subset of the total packet filtering task.

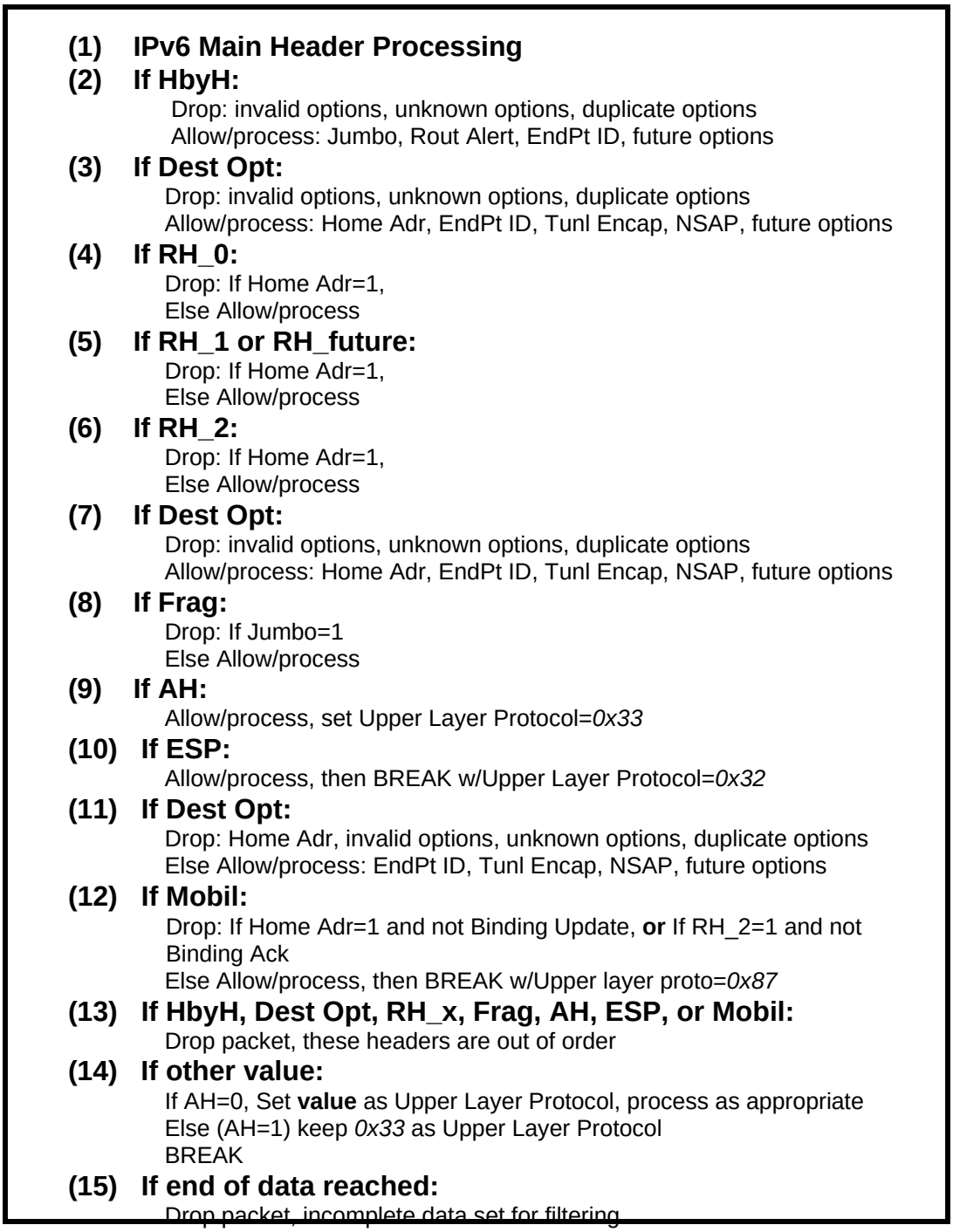


Figure 3-9: IPv6 Header Ordering Algorithm

Abbreviations used in Figure 3-9

AH - Authentication (extension) Header, see section 6
Dest Opt - Destination Options extension header, see section 3.4
EndPt ID - Endpoint Identification option, see section 3.2.2
ESP - Encapsulating Security Payload extension header, see section 6
Frag - Fragmentation extension header, see section 3.6
HbyH - Hop by Hop Options extension header, see section 3.3
Home Adr - Home Address Destination option, see section 3.4.2
Jumbo - Jumbo Payload option, see section 3.3.1
Mobil - Mobility extension header, see section 3.7
RH_0 - Routing (extension) Header, Type 0, see section 3.5.1
RH_1 - Routing (extension) Header, Type 1, see section 3.5.2
RH_2 - Routing (extension) Header, Type 2, see section 3.5.3
RH_x - Any Routing (extension) Header see section 3.5
Rout Alert - Router Alert option, see section 3.3.2
Tunl Encap - Tunnel Encapsulation Limit option, see section 3.4.1
NSAP - NSAP Address option, see section 3.4.3

The algorithm begins in step (1) with the processing of an IPv6 main header. As part of main header processing, a value is extracted from the *next header* field. This value is then evaluated by step (2) to determine if it is the **Hop-by-Hop Options** extension header. If the first header is the **Hop-by-Hop Options** extension header then it is evaluated as shown in step (2), else the header is compared against step (3). The correct usage of the algorithm is to continue to apply algorithm steps until a match is found and the action of that step applied. At that point, processing continues with the next header in the chain and with the next step in the algorithm. Once a step is reached, processing cannot go backwards or stay in place to evaluate the next header. In this manner, header ordering is enforced, particularly by the catchall step (13) that drops all packets matching that step.

As an example, assume an IPv6 header chain consisting of the main header followed by a **Type 2 Routing Header**, followed by a **Type 0 Routing Header**. The algorithm would proceed to step (6) and process the **Type 2 Routing Header**. At that point, the **Type 0 Routing Header** matches step (13) and the packet is dropped. The algorithm requires that a **Type 2 Routing Header** must occur after a **Type 0 Routing Header** if both are present.

The notation "=1" is used to indicate the occurrence of a particular header *thus far* in the algorithm processing. For example, assume an IPv6 header chain consisting of the main header followed by a **Destination Options** extension header containing the **Home Address** option, followed by **Type 0 Routing Header**. In this case, the **Home Address Destination Option** is matched by step (3) and allowed. The **Type 0 Routing Header**, however, matches in step (4) and the packet is dropped because the algorithm states to drop if "Home Adr=1", i.e. if the **Home Address** option has already occurred. This enforces the specified restriction that the **Home Address** option must appear after any routing headers. If ordering of the two extension headers in this example is reversed, the

Type 0 Routing Header matches in step (4) and the **Home Address Destination Option** matches in step (7) where it is allowed.

The following qualifications are also important in understanding the intent of the algorithm:

- The term "duplicate options" is meant to refer to any IPv6 options that occur more than once, whether it be in the same extension header or a subsequent one. This covers specified prohibitions of multiple occurrences of certain options such as the **Home Address** option and **Router Alert** option, and also applies the rule to all other options since there is no compelling reason to allow duplicates.
- **PAD1** and **PADN** are exceptions to the "duplicate options" rule as they are formatting constructs and may occur any number of times.
- The "duplicate options" rule applies only to the IPv6 options, not to the **Destination Options** extension header itself, which may occur more than once with different options contained within.
- The term "future options" and RH_future refers to any values chosen by the firewall design in anticipation of future expansion of IPv6 headers. Processing of these headers would need to be a trivial on-presence filtering style since details of the option are not yet known. This however, may be useful in bridging the gaps between new headers and design upgrades.
- The term "allow/process" means that the header is allowed at this point in the algorithm (not that the packet is declared to be allowed). It also means to set an "=1" flag to indicate the occurrence of the header and any options encountered. The firewall may apply other processing within the algorithm to extract essential data from the headers that will be needed for the full analysis against the firewall security policy. The ordering algorithm alone can be thought of as some necessary pre-processing but does not indicate the full processing of the packet.
- The word "BREAK" is used to indicate that this algorithm is terminated, though not necessarily in failure. For example, after the processing of an ESP header, the algorithm aborts because all data beyond is assumed to be encrypted.
- The algorithm BREAKs after **Mobility Header** processing because no more data is permitted according to specification. Processing a **Mobility Header** includes a check that the value of the *next header* field is *0x3B* (i.e. no next header).
- Step 12 processing stated here applies to the **Mobility Header** processing only, not to messages without a **Mobility Header**. It means that a **Home Address Destination Option** cannot occur with mobility messages other than the **Binding Update** and that the **Type 2 Routing Header** cannot occur with mobility messages other than the **Binding Acknowledgement**. This refers to the specified requirements in RFC 3775, section 6.1, paragraph 2.
- Step 14 BREAKs because an upper layer protocol value has been reached. If this value is of the set of unassigned values, the packet should be dropped since it is not known if it is an extension header or a protocol. The term "process as appropriate" in step 14 refers to any additional processing that may be needed for certain upper layer protocols such as TCP, UDP, and ICMP.

- The upper layer protocol value is recorded as *0x33* when an AH header is encountered (Step 9). This can be overridden if an ESP header is also present (Step 10). Firewall designers may instead opt to distinguish between ESP alone and AH+ESP though it should not be necessary.
- The upper layer protocol value of *0x33,* AH header, (Step 9) may also be overridden by a Mobility Header (Step 12), but not any other upper layer protocol (Step 14). Currently, the Mobile IP specifications allow but do not recommend protection using AH[22].
- "End of data reached" in step 15 means that the algorithm never obtained the upper layer protocol value before the end of the packet was reached. Though unlikely, this could occur with normal fragmented packets. More likely, it indicates a hostile packet and should be dropped.

3.9.2 Implications of the Header Ordering Algorithm

Some of the noteworthy restrictions enforced by the header ordering algorithm of Figure 3-9 are as follows:

- If present, the **Hop-by-Hop Options** header must be the first extension header and it cannot appear more than once. (per RFC 2460, section 4.1, para 5)
- **Destination Option** headers are allowed either prior to routing headers (RFC 2460, section 4.1, para 2), just before the upper-layer header (RFC 2460, section 4.1, para 2), or between routing headers and the **Fragmentation Header** (RFC 3775, section 6.3, para 5).
- Packets with multiple occurrences of a **Fragmentation Header** within an IPv6 header chain will be dropped per recommendation made in section 3.6.3.3. Also, multiple occurrences of **Hop-by-Hop**, **Routing Headers** of the same Type, **Mobility Headers**, and **AH** are dropped.
- A **Type 2 Routing Header** must occur after a **Type 0 Routing Header** if both are present, per a "SHOULD" recommendation in RFC 3775, section 6.4, para 4.
- A **Home Address** option can only occur in a **Destination Options** header located between **Routing Headers** and a **Fragmentation Header** if either of those exist. (per RFC 3775, section 6.3, para 5)
- A packet will be dropped if it contains both a **Fragmentation Header** and a **Jumbo Payload Hop-by-Hop Option** (per RFC 2675)
- If either the **AH** or **ESP** headers are detected AFTER the processing of a **Mobility Header**, the packet is dropped by Step 13, since the **Mobility Header** must occur after the security headers if they are present. In the case of the properly encrypted **Mobility Header**, the algorithm BREAKs in Step 10 without ever knowing about the **Mobility Header** within the encrypted portion of the packet. Messages with **Mobility Headers** may or may not need to be encrypted. Step 12 accounts for the unencrypted messages such as those sent to a Correspondent Node in the mobile IP scenario.
- All routing headers must occur prior to a **Fragmentation Header**. Though not stated explicitly as a requirement in the RFCs, if this restriction were not imposed, fragments could be reassembled only to be immediately re-fragmented when a routing header sends the packet to a different destination. This would be a

violation of the intent that no intermediate reassemble and/or fragmentation be performed.

It should be emphasized that the above ordering algorithm only applies legality checks on header usage and construction. It does not account for the filtering policy of those headers. For example, step 4 says to allow/process a **Type 0 Routing Header**. It may be the case that the firewall is configured to drop all packets with this header type. A drop *by policy* is not reflected in the algorithm. The header ordering algorithm would either be applied before policy filtering or in conjunction with it, per design decisions made for a particular firewall product.

Running packets through this algorithm also allows data to be collected to support policy enforcement. For example, it would be possible to detect packets that have both a **Home Address Destination Option** and an **ESP** header, or a **Type 0 Routing Header** and **ESP** header.

4 IPv6 Upper Layer Protocol Processing

The processing of upper layer protocols is largely the same in IPv6 as it was for IPv4.

Most of the protocol numbers used for IPv4 are carried over for IPv6. For example, TCP is still protocol *0x06* and UDP is still protocol *0x11* despite some minor changes such as requiring a UDP checksum. ICMPv6, however, was given a whole new protocol number *(0x3A)* whereas ICMPv4 is protocol *0x01*. Any v4-only or v6-only protocol that appears in the wrong version of IP should be considered illegal such as an IPv6 packet with a protocol value of *0x01*, which is undefined. The firewall should drop these illegal cases by default.

IPv6 also merges the extension headers with the upper layer protocol values in the same numbering space. For example, the Routing Header is protocol value *0x2B* and the Fragmentation Header is value *0x2C*. These and other IPv6 extension headers are undefined if they occur in IPv4.

4.1.1 UDP and TCP

UDP and TCP stay mainly the same. Stateful filtering and any other advanced filtering features that have evolved for these protocols in IPv4 over the years are still applicable and needed in IPv6.

4.1.2 ICMP

Firewalls in IPv6 should to be able to filter individual ICMP messages based on both the Type and Code values to allow the maximum granularity with respect to these messages. IPv6 firewall designs should steer away from the IPv4 practice of automatically dropping all ICMP messages or lumping them into “all or nothing” settings.

The ability to filter some of the new ICMP messages for IPv6 is essential (e.g. Neighbor Discovery, Router Renumbering). Also, it will no longer be tolerable to drop other ICMP messages due to new IPv6 features (e.g. Path MTU Discovery).

5 Firewalls during IPv4-to-IPv6 Transition

When both IPv4 and IPv6 are present in a network, a consistent filtering policy is needed for the two protocols. It should not be possible for an adversary to gain an advantage from choosing one version of IP over the other. For *most* upper layer protocols and applications, the IP version upon which they run is inconsequential. Consistency in filtering is achieved in the very straightforward manner of configuring the firewall's access control policy equivalently for IPv4 and IPv6. A Telnet packet, for example, should not be blocked in IPv4 but allowed to the same node in IPv6.

The challenge to consistent IPv4/IPv6 filtering is in several areas:

1) The few upper layer protocols and applications that *are* new and unique to IPv6, or significantly changed in a manner that affects security
2) Differences in the basic IP header format and protocol
3) Peculiarities of transition traffic

New/improved protocols and applications (1) must be addressed by security analysis resulting in configuration guidance. New IPv6 header formats and protocol complexities (2) is the subject of Chapters 2 and 3 of this document. Transition traffic (3), is any unique traffic resulting from the coexistence of IPv4 and IPv6 packets, and is the subject of this chapter.

5.1 Types of IPv4-IPv6 Transition Traffic

Transition traffic follows three different approaches: Dual Stack, Tunneling, and Translation. Each of these general approaches is represented by numerous detailed schemes, called *transition mechanisms* by the Internet community, tailored for specific network scenarios or anticipated transition problems.

The Dual Stack methods consist of equipping nodes to handle both IPv4 and IPv6 packets. Other than the ability to handle both native IPv6 packets as discussed throughout this document and IPv4 packets, dual stacking does not present any additional challenges to the firewall.

Likewise, Translation schemes produce packets that are either IPv4 or IPv6, and again do not present any additional challenges to the firewall. Depending on the placement of the translation mechanism, the firewall sees either IPv4 or IPv6 packets only.

Tunneling schemes, however, do create unique traffic that are new to existing firewall designs. These transition mechanisms produce IPv6 tunneled within IPv4 or IPv4 tunneled within IPv6. Ideally, firewalls will be able to filter the inner IP layer of tunneled traffic with the same granularity that they filter normal IP. If such firewalls are not available, more awkward means of decapsulating and filtering (with additional firewalls) will be needed to achieve a consistent filtering policy for IPv4 and IPv6.

5.2 Filtering Tunneled Traffic

Transition mechanisms based on tunneling are the most problematic to existing firewall designs. Since these mechanisms along with dual stacking are likely to be widely used, there is an urgent need for improved firewall capabilities in dealing with tunneled traffic.

The discussion below is applicable to all combinations of IP-in-IP tunneling, not just v6 tunneled in v4. If improvements to a firewall design are made, it makes sense to extend the capability to all combinations.

5.2.1 IP-in-IP Tunnel Threats

There are several RFC specifications that redundantly cover the same basic tunneling concepts. RFC 2003 [3] describes IPv4-in-IPv4 tunneling, RFC 2473 [4] covers the IPv4-in-IPv6 and IPv6-in-IPv6 cases, and RFC 4213 [5] (formerly RFC 2893) covers the transition case of IPv6-in-IPv4.

The prescribed security measures differ between the three specifications with RFC 2003 and RFC 2473 containing no additional requirements to check or filter any condition. Security concerns are mentioned in these documents with recommendations to use IPsec for protection. RFC 4213 does require one additional security check for configured tunnels. It requires the tunnel source address (outer IP layer) of an incoming tunnel packet to match that of the configuration information in the decapsulator. The decapsulator automatically checks that the tunneled packets appear to come from the configured encapsulator. It also states that additional ingress filtering “MAY” be applied though there are no guarantees that this is implemented.

Refer back to Figure 3-1: Double Benefit of Filtering Source Addresses, where the filtering of source addresses in (un-tunneled) IP was illustrated. An attacker had to first guess a valid source address and then find a way to subvert the routing system in order to receive a response packet from his attack. Then in Figure 3-2, the Type 0 Routing Header (or IPv4 source routing) was shown to eliminate half of that benefit by only requiring the attacker know a valid source address. The general consensus has been that this is too dangerous, and that IPv4 source routing is almost always disabled by firewalls. Now compare these cases to the case of tunneled traffic.

Current firewall designs do not allow filtering of the inner IP layer of a tunneled packet. The outer source and destination IP addresses can be filtered along with the protocol type field, which is the value *0x04* for a tunneled packet with an IPv4 inner layer, and the value *0x29* when the inner layer is IPv6. This allows security administrators to enable/disable tunneling to particular endpoints (decapsulators) within their site by restricting type *0x04* and/or *0x29* to specific destination addresses. It also allows tunnel source address filtering exactly equivalent to the additional check mandated to decapsulators by RFC 4213 (for 6-in-4 configured tunnels only).

Given that only the source and destination addresses of the *outer* IP layer can be filtered, refer to Figure 5-1 below. This shows a network with a configured tunnel to a real user

A. Note that node **n** is a decapsulation point (likely a router) on the inside of the firewall such that the firewall sees tunneled packets.

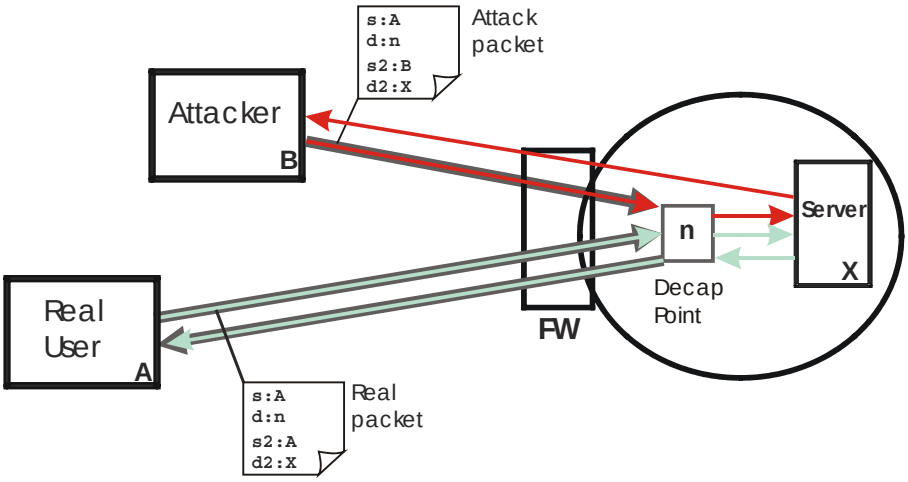


Figure 5-1: Filtering Source Address of Tunnel Traffic

The real user sends tunneled packets and the firewall verifies that the outer source and destination are allowed to pass tunneled traffic. Responses to this allowed traffic from the server return to node **n** and are tunneled back to the real user. There is nothing in tunnel packet processing that forces the return packet to be tunneled. It is merely routed as normal and it is shown tunneled here under the assumption that a properly configured tunnel would be set up bi-directionally. This is not a consequence of the incoming packet, but rather of the routing configurations of the inside network.

Next, we see that the attacker has spoofed the outer IP source address **(A)** and sent an inner packet with his own source address **(B)**. Since the firewall (and the decapsulator using the RFC 4213 check) both verify only the outer source IP address, this packet gets through. The return packet is not shown tunneled back to the attacker since the internal network is not configured to tunnel packets to address **(B)**. Instead, it escapes back out through default routing.

Tunneling can be exploited by an attacker, when there are no checks on the inner packet. The attacker need only know a valid outer source address of an existing tunnel. In conclusion, an IP-in-IP tunnel with filtering only on the outer source address produces the same level of threat as allowing IPv4 source routing.

5.2.2 IP-in-IP Tunnel Filtering Solutions

The above threats can be countered either by improving the firewall's capabilities or by using more than one firewall (or filtering device).

If, in the example above, the firewall allowed tunneled traffic only to internal destination **n**, and a second firewall filtered everything that came out of the tunnel endpoint at **n**, the full packet filtering would be applied to the inner layer. This works with the obvious disadvantage of doubling the numbers of firewalls required. If tunnel-filtering firewalls

are unavailable, this will be the only option short of disabling the use of tunnels altogether.

A tunnel-filtering firewall would have the capability of filtering both IP layers within the same device. The downside is a more complicated firewall design. To prevent attacks of the form shown in the previous section, a firewall would need the ability to filter the inside IP addresses against a security policy. In addition to the addresses though, it is the whole inner packet should be filtered. Otherwise a similar attack would be possible in which the real user sends packets with disallowed protocols/ports on the inner layer.

The recommendations here consist of a primary and secondary filtering task. The intent is that all firewalls should have the capability to perform at least the primary task, which is an improved version of the existing capabilities described above. Firewalls designed to filter both IP layers would need to incorporate both the primary and secondary tasks. See Figure 5-2 below.

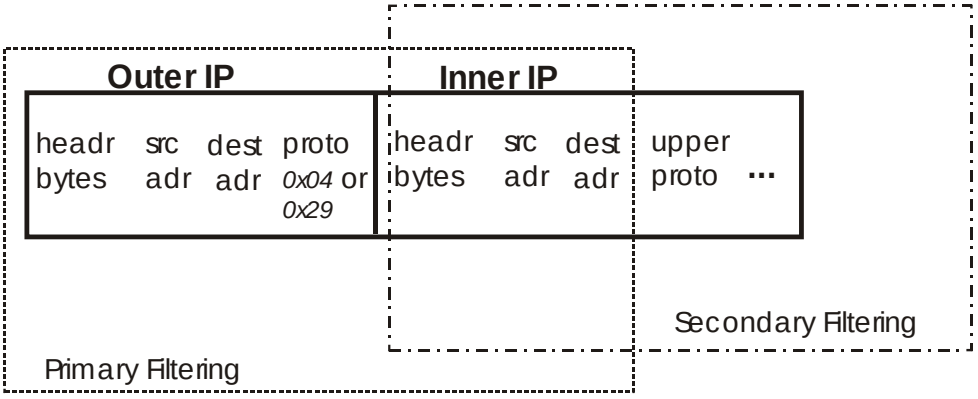


Figure 5-2: Primary and Secondary Tunnel Filtering

The primary filtering task includes the source and destination addresses of the inner IP layer, which is more filtering capability than is currently available. The primary tunnel filtering would be implemented in a manner similar to filtering the TCP protocol with configurable port values. In this case, protocol fields are *0x04* or *0x29* and the configurable values are the inside source and destination IP addresses. The capability allows a firewall to impose very basic source filtering on the inner layer of tunneled traffic and to regain the dual security benefit described back in Figure 3-1. This can be thought of as a restriction on tunnel usage (the WHO) though not the tunnel content (the WHAT). Such filtering is achieved without a huge increase in complexity to current design strategies.

The ability to wildcard any of the addresses should be allowed. Furthermore, filtering rules should allow the inner layer IP addresses to be version 4 or version 6 addresses regardless of the version of the outer addresses. Wildcards must be disambiguated with respect to version 4 and version 6.

Incorporation of the secondary filtering task *is* an increase in complexity because a packet essentially needs to be filtered twice. This will increase throughput delays for a packet and will impact performance. The extra cost to support this function however should be compared against the only other option; buying two firewalls.

The Secondary filtering applies only to the inner IP layer. The inner layer headers, IP addresses, protocols, ports, etc., are filtered against a policy in the same manner that a normal (untunneled) packet is filtered today.

The primary filtering task can be used to direct certain packets into the secondary filtering. For example, if a tunnel packet matches a primary filtering rule, the action could be pass, drop, or forward to secondary filtering. Some combinations of inner/outer addresses may allow the firewall to drop the packet without incurring the burden of secondary filtering.

The combination of primary and secondary rules can be used to allow certain types of traffic to be tunneled only from specific sources. The filtering overlap at the inner IP addresses, allows this capability. For example, a site may have tunnels inbound from several remote sites, but wants to allow Telnet to enter through only one of those tunnels. The Primary filtering is set to allow inner source address(es) **Y** to occur only with outer source address(es) **X**. The secondary filtering allows Telnet only for source address(es) **Y**.

Another possible scenario is provided by Mobile IP (See reference [9]). In this scenario, tunneled traffic into a Home Network consists of some traffic destined for internal nodes and some traffic being relayed in support of the reverse tunneling mode of Mobile IP operation. A firewall will want to use the primary filtering task to direct the internal (home network) traffic into secondary filtering and may want to send all other traffic out with no secondary filtering at all. Additionally, the Mobile IP tunneled traffic will continually arrive from new tunnel source addresses as the mobile node moves around. Therefore, the firewall would choose not to filter tunnel source addresses or inner source addresses. Instead, traffic is restricted to the home agent, which is tasked with making these checks.

There are a variety of different types of tunneling requiring different filtering, and a two-prong approach (primary and secondary filtering options) is more likely to provide the desired flexibility.

Another big advantage of the primary and secondary filtering approach is that the complexity manifests as increased hardware but not a radically new design approach. The primary filtering is largely the same as a basic IPv6-capable firewall except for the ability to direct packets into secondary filtering. The secondary filtering is essentially another instantiation of the same design. Some functions will likely still need to be coordinated across both layers of filtering, (e.g. refer to the fragmentation-tunneling recommendation in 3.6.4.4).

The inner IP layer may not require *exactly* the same configured filtering policy as a packet that arrived un-tunneled. For example, there may be some addresses (private addresses or site local addresses) that a site allows tunneled in that would have been dropped otherwise. A design should allow for separate filtering policies for the inside and outside layers even though they may end up being very similar.

5.2.3 Other Security Checks for Tunnels

Any restrictions on duplicate IPv6 options or extension headers described in section 3.9 should NOT be applied across IP layers. Each layer is filtered separately with regard to header legalities including the Fragmentation header. Fragmentation is not considered to be "nested" if it occurs in separate IP layers.

As stated in section 3.1.1, the firewall should check and drop any packet for which the *protocol* field (*0x04* or *0x29*) of the outer layer does not match the version of the inner IP layer (IPv4 or IPv6 respectively).

Firewalls should be configurable to drop any packet that emerges from a tunnel with a value of 255 in the *hop limit* field of the main header. If this condition is allowed, it can be used in certain known attacks. Specifications require the encapsulators to decrement the hop limit of the inner layer packet so there are no cases where a value of 255 is legal. Firewalls should allow this check to be enabled/disabled.

5.2.4 Fragmented Traffic and Tunnels

Fragmentation reassembly by a firewall in conjunction with filtering multiple layers of IP is very complex and may produce unwanted side-effects (see section 3.6.3.4). Schemes for filtering fragments without performing packet reassembly should be applicable or adaptable to multiple IP layers (see section 3.6.4.4).

Firewalls must be able to extract all of the necessary header information from every packet in order to correctly apply the filtering policy. For unfragmented packets, the header ordering algorithm (section 3.9) includes a check at the end to enforce this requirement. The first fragment of a fragmented packet is more likely to terminate prematurely, hence the schemes offered in section 3.6.4 enforce this requirement as well. Fragmentation must not be allowed as a means to escape filtering.

5.2.5 Other Types of Tunnels

The feasibility of filtering other types of tunnels must be decided on a case-by-case basis. An existing IP-in-IP filtering design may be extended to handled other mechanisms. For example, Generic Routing Encapsulation (GRE) tunneling (RFC 2784) with IP as the inner (tunneled) data would be easily added to a design that already filters the IP-in-IP case.

6 IPv6 Firewalls and IPsec

6.1 The Filtering vs. Communications Security (COMSEC) Strategies

There are two primary strategies in use today for achieving network security: a filtering strategy and a COMSEC strategy. COMSEC is a DoD term used to refer to any system that employs an encryption mechanism as a complete security layer across all traffic. IPsec is an encryption mechanism that can be used to employ the COMSEC strategy. Firewalls are the typical means for employing the filtering strategy.

A filtering strategy or COMSEC strategy is used to strengthen a network by mitigating weaknesses that may exist in internal nodes and applications. The weaknesses (specification errors, design flaws, software bugs etc...) are still present, but the adversary cannot exploit them.

Filtering and COMSEC strategies are very different in *how* they mitigate network weaknesses. Firewalls inspect and scrutinize packet content (the WHAT) and drop any traffic that looks dangerous or unnecessary according to a policy. The challenge is to define the policy and create the filtering capability that can drop unwanted traffic without dropping necessary traffic.

With the COMSEC strategy, however, the packet sender (the WHO) is most important quality. The cryptographic mechanisms positively identify a sender and also bind the traffic to that sender. The packets are allowed or dropped based on whether they came from an authorized source. The WHAT is not as important and a COMSEC strategy usually has far less capability to scrutinize the actual packet contents; maybe none at all depending on where it is implemented in the stack. COMSEC also provides additional security properties such as confidentiality, integrity, and non-repudiation.

Filtering is considered a weaker strategy because it can't really know if traffic is legitimate, only that it looks good. COMSEC is considered strong security because it verifies (with very high probability) that traffic comes from a legitimate source. The choice of which to use is not as straightforward. COMSEC systems must have existing trust relationships for support. Since the WHO is used as the security linchpin, there must be a clear distinction between who is allowed and who is not. The COMSEC strategy works well in a closed network community between trusted parties, but cannot be applied effectively in an untrusted environment (e.g. the public Internet).

Both of these strategies are different and both are needed in today's varying network environments.

6.2 Handling IPsec at the Firewall

Given that the filtering and COMSEC strategies are fundamentally different, it stands to reason that their security policies are not interchangeable. The access control policy of the firewall, therefore, should not apply to traffic containing an AH or ESP header. Even though AH and ESP are extension headers they should be treated as if they are the upper layer protocol from the standpoint of the firewall. In fact, the ESP header prohibits any

further packet inspection because the data is encrypted. The AH header, although non-encrypting, should also override the filtering of the upper layer protocol that follows. This is not a hole in the firewall, but merely a deference from one security model to another. If a packet contains an IPsec header, the IPsec mechanism should handle it.

As an example, a firewall may be configured to drop all Telnet packets because Telnet is dangerous in the hands of an attacker. If the packet contains an AH header followed by Telnet, the firewall should defer judgment (not drop) to the IPsec implemented at the packet's destination. If there is no IPsec implemented or enabled at the destination, the packet is dropped there. The end host does not have the option of ignoring an AH header and accepting the packet anyway.

It is important to be able to filter the AH and ESP packets in conjunction with other extension headers. Examples are with the **Home Address Destination Option** per section 3.4.2 and the **Type 0 Routing Header** per section 3.5.1.

The header ordering algorithm in section 3.9 declares the AH or ESP security headers to be the upper layer protocol except when an AH is followed by a **Mobility Header**.

6.3 IPsec and Firewall Combinations

A likely and preferred scenario consists of a firewall protecting IPsec device(s) behind it as depicted in Figure 6-1. Case **(A)** in the figure shows a firewall shielding the IPsec device from unwanted traffic, potential DOS attacks, etc... Case **(B)** shows that part of the internal network may be IPsec-protected and the rest in need of robust filtering protection. A network employing a DMZ is an example of Case **(B)**.

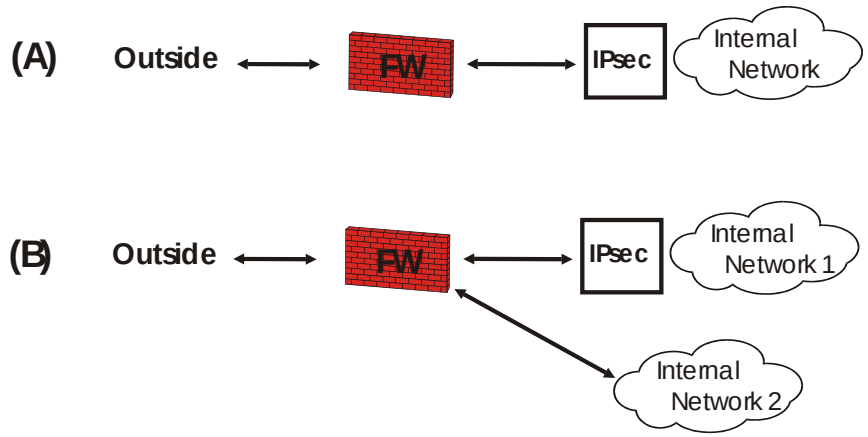


Figure 6-1: Preferred Firewall and IPsec Combinations

These are preferred scenarios because the IPsec is positioned to completely protect a network or segment thereof. The firewall is positioned to provide additional protection to the IPsec-protected network. In order to achieve a true COMSEC strategy, the IPsec in these examples must be configured to block all unencrypted traffic, which is not always achievable operationally. If the IPsec protects only *some* traffic and passes other traffic

in the clear, the overall strategy is a filtering strategy with encryption enhancements; a less than optimal situation.

Sections 6.2 and 3.9 above recommend that the ESP and AH headers be considered the upper layer protocols so that the scenarios of Figure 6-1 are easily supported. Firewalls can direct the protected traffic to internal IPsec destinations while still fully filtering unprotected traffic.

A firewall could also be used behind an IPsec device as shown in Figure 6-2. Although this scenario does not present any special demands on the firewall, it is also not likely to be needed. The IPsec model already provides basic access control over traffic types, down to the protocol and port level. IPsec does not have as many detailed filtering capabilities, but these are arguably not needed in the trusted IPsec system. Virus scanners and IDS devices are more likely to be used inside of an IPsec-protected network.

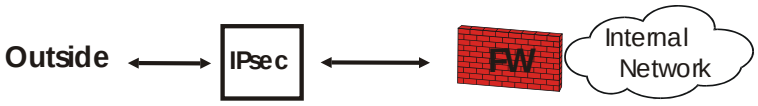


Figure 6-2: Unlikely Firewall and IPsec Combination

As shown in Figure 6-3 below, a firewall and IPsec implementation could be combined side-by-side into a single security device (e.g. an encrypting firewall) though this is not an optimal situation. This is exactly the same as the case described above for Figure 6-1 when IPsec is configured to allow some traffic to pass unprotected. The IPsec terminates at a point where unencrypted traffic is passed through the firewall. The weaknesses of the firewall model threaten the IPsec model in this case.

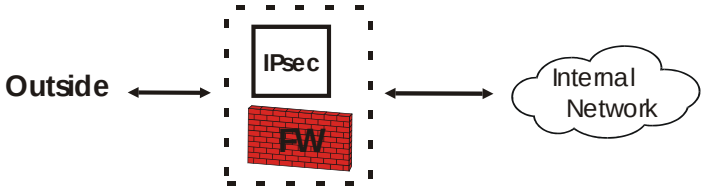


Figure 6-3: Sub-optimal Firewall and IPsec Combination

In conclusion, IPsec is most effective when it can be deployed as a true COMSEC strategy such that **all** traffic is protected at an IPsec boundary. Network designers should strive to partition their networks in this manner if at all possible. The IPsec implementation can be moved inward to protect only part of a network or even moved all the way to the end hosts. Of course, the IPsec can only be pushed back so far. Once there is a single node that must send both protected and unprotected traffic, the overall system becomes sub-optimal (i.e. the weaker filtering strategy).

The filtering strategy, though weaker than a COMSEC strategy is sometimes the only option. Operational requirements may demand communications between networks for which no trust relationships exists or no cryptographic keys can be established. The

standard Mobile IP scenario is also an example of the filtering strategy. Even though IPsec is employed, the Mobile IP home agent only uses it in a very limited manner to protect specific Mobile IP traffic.

7 In Summary

As promised at the outset, this document does not declare requirements but attempts to identify where the security issues are with IPv6 and firewall design. Designers are encouraged to implement as much of the functionality as possible and to provide feedback to the author regarding areas that are problematic for modern firewall design.

The areas deemed to be most important are:

- An ability to recognize all extension headers and locate the upper layer protocol information
- The header ordering algorithm or some modified version of it to account for illegal, duplicate, and unreasonable header constructs
- Some solution to the inspection of fragmented packets or the ability to drop them
- At least the minimal tunnel filtering capability described as the “Primary Tunnel Filtering”

8 Reference Documents

[1] Deering S., Hinden R., “Internet Protocol Version 6 (IPv6)”, **RFC 2460**, Dec 1998
[2] Borman D., Deering S., et al., “IPv6 Jumbograms”, **RFC 2675**, Aug 1999
[3] Perkins C., “IP Encapsulation within IP”, **RFC 2003**, Oct 1996
[4] Conta A., Deering S, “Generic Packet Tunneling in IPv6 Specification”, **RFC 2473**, Dec 1998
[5] Nordmark E., Gilligan R. “Basic Transition Mechanisms for IPv6 Hosts and Routers”, **RFC 4213**, Oct 2005
[6] Partridge C., Jackson A., “IPv6 Router Alert Option”, **RFC 2711**, Oct 1999
[7] Katz D., “IP Router Alert Option”, **RFC 2113**, Feb 1997
[8] Johnson D, Perkins C, et al., “Mobility Support in IPv6”, **RFC 3775**, Jun 2004
[9] Potyraj C., “A Filtering Strategy for Mobile IPv6”, 19 Sept 2007, http// [TBD]
[10] Savola P., “MTU and Fragmentation Issues with In-the-Network Tunneling”, **RFC 4459**, Apr 2006

Endnotes

[1] RFC 2460, section 4.1, paragraph 2
[2] RFC 2460, section 4.1, paragraph 5
[3] IANA is the Internet Assigned Numbers Authority
[4] RFC 2460, section 4.2, paragraph 3
[5] This restriction is also incorporated into the header ordering algorithm presented in section 3.9.
[6] RFC 3775, section 6.3
[7] This wording is to emphasize that Routing Header and security header combination should be allowed, but that other header checking is still performed.
[8] RFC 3775, section 6.4, paragraph 2
[9] Since this characteristic is being relied upon, security testing of routers and nodes should include tests to verify that the Type 2 Routing Header cannot be forwarded beyond the original destination.
[10] RFC 2460, section 4.5
[11] The wording “second fragmentation header to appear in an IPv6 header chain” was chosen instead of “second fragmentation operation to be applied to a packet” to make it clear that a tunneled IP packet with separate inner and outer IP headers may in fact be legally fragmented twice (once in each IP layer). Nested fragmentation only refers to two fragmentation headers occurring in the same IP header chain.
[12] RFC 2460, section 4.1 paragraph 5
[13] "deep packet inspection" is a marketing term referring to a more recent trend in firewalls whereby application data is filtered for suspicious content, virus signatures, etc....
[14] RFC2460, Section 4.5, last paragraph
[15] MTU: maximum transmission unit, is the number of bytes of the largest packet that a link layer is capable of forwarding
[16] Note that link layer fragmentation does not affect firewall filtering because a firewall is a link terminating device.
[17] The Jumbo Payload option is not included in this calculation since it is illegal in a fragmented packet
[18] The default value of 60 is chosen based on the size of the worse case header example provided earlier in the section for a tunneled packet (i.e. 182 4-byte words). Of the 182 words, 114 words are in the fragmentable part, which corresponds to 57 8-byte blocks. This is rounded up to 60 to allow for the presence of padding.
[19] The second and third checks ensure the completeness and integrity of the filtered data. The first check is a necessary consequence of the scheme since normal non-first fragments would fail the third test if the EVAL_SIZE boundary were allowed to drift into the 2nd fragment.
[20] The attacker could send packets to deliberately activate the threshold detector thereby disabling that source address at the firewall.
[21] The optional IPv6 Path MTU Discovery function is supposed to prevent these double fragmentation scenarios by communicating the “tunnel MTU” back to the originating host, but in this worse-case scenario it is assumed this is not operable for some reason.
[22] RFC 3775, Section 5.1, para 1