# Transparent Network Security Policy Enforcement

Angelos D. Keromytis
Distributed Systems Lab, University of Pennsylvania
angelos@openbsd.org

Jason L. Wright
Network Security Technologies, Inc. (NETSEC)
jason@openbsd.org

**Abstract**

Recent work in the area of network security, such as IPsec, provides mechanisms for securing the traffic between any two interconnected hosts. However, it is not always possible, economical, or even practical from an administration and operational point of view to upgrade the software and configuration of all the nodes in a network to support such security protocols.

One apparent solution to this problem is the use of security gateways that apply the relevant security protocols on behalf of the protected nodes, under the assumption that the "last hop" between the security gateway and the end node is safe without cryptography. Such a gateway can be set to enforce specific security policies for different types of traffic. While this solution is appealing in static scenarios (such as building so-called "intranets"), the use of Layer-3 (network) routers as security gateways presents some transparency and configuration problems with regards to peer authentication in the automated key management protocol.

This paper describes the architecture and implementation of a Layer-2 (link layer) bridge with extensions for offering Layer-3 security services. We extend the OpenBSD ethernet bridge to perform simple IP packet filtering and IPsec processing for incoming and outgoing packets on behalf of a protected node, completely transparently to both the protected and the remote communication endpoint. The same mechanism may be used to construct "virtual local area networks," by establishing IPsec tunnels between OpenBSD bridges connected geographically separated LANs. As our system operates in the link layer, there is no need for software or configuration changes in the protected nodes.

## 1 Introduction

Network bridges are simple devices that transparently connect two or more LAN segments by storing a frame received from one segment and forwarding it to the other segments. More intelligent bridges make use of a spanning tree algorithm to detect and avoid loops in the topology. We have implemented the basic form of an ethernet bridge in OpenBSD that also provides an IP filtering capability. Thus, the bridge can be used to provide a LAN-transparent firewall between hosts such that no configuration changes are needed on client machines, and only minor changes in network topology are necessary.

For this, we make use of *ipf,* the standard packet filtering mechanism available. As ethernet frames pass through the bridge, they are examined to see if they carry IP traffic. If not, the frame is just bridged. If the frame does contain IP traffic, the ethernet header is removed from the frame and copied. The resulting IP packet is passed on to *ipf*, which notifies the bridge whether the packet is to be forwarded or dropped. The ethernet header of the frame under examination is appropriately modified on the frame to be forwarded, and the resulting frame is then bridged as normal.

The bridge can also be used to enforce restrictions on which addresses can appear on each ethernet segment, which helps localize where ARP spoofing attacks can occur. Static MAC address cache entries are provided so hosts can be limited to a particular port and malicious users cannot force the bridge to send traffic to the wrong segment. The ability to learn MAC addresses dynamically is configurable on each port of the bridge, and broadcast discovery for machines unknown to the bridge can be toggled on a per port basis. Additionally, a mechanism is provided for filtering ethernet frames based on source and/or destination MAC address.

This functionality, useful on its own, can be coupled with the IPsec [9] support available in OpenBSD, to allow creation of virtual LANs. This is achieved by overlaying an IPsec-protected virtual network on the wide area network (or even the Internet itself). The changes necessary to the bridge and IPsec code for this were fairly minimal, due to compatibility of some design decisions made independently in the development of the two packages.

The enhanced bridge can also be used to provide transparent IPsec gateway capability for a host or even a network. In this mode, the bridge examines transient IP traffic and may, depending on security policy, establish IPsec security associations (SAs) with a remote host pretending to be the local communication endpoint for an IP session[1]. There are two main benefits from this. First, this allows protection of the communications of a host or network without changes to the protected hosts (which may not even be possible, for old, unsupported, or extremely lightweight systems). Second, the security gateway can act as a security policy enforcer, ensuring that incoming and outgoing packets are adequately protected, based on system or network policy.

### 1.1 Paper Organization

Section 2 briefly describes the bridge itself and the filtering of frames containing IP traffic. Section 3 describes the use of IPsec in conjunction with the bridge to build virtual LANs and transparent IPsec gateways. Section 4 discusses open ends and future work, and Section 5 concludes the paper.

[1]The term "IP session" is used here loosely to imply a packet flow between two hosts, one of which is on one of the local segments and is "protected" or "supervised".

## 2 Bridge

Bridges are devices that operate at the data link layer, tying together different ethernet (or other LAN) segments. In OpenBSD, the bridge is implemented as a pseudo-network interface inside the kernel. Real ethernet interfaces are added to the bridge interface as "bridge members," and for the purpose of using IPsec with the bridge, *enc* interfaces can be added on as members. The *enc* interfaces contain the security association (SA) for communication to remote LANs. In all ethernet drivers under BSD, received frames are assembled into mbufs [11], a data structure that provides for easy insertion and deletion of data with little or no data copying. The ethernet header is removed and passed along with a reference to the receiving interface and the mbuf containing the frame data to *ether_input()*. The bridge intercepts the frame early in this function, after a small amount of bookkeeping is performed.

On entry to the bridge code, the frame is checked to see if it is a broadcast, multicast, or unicast frame (Figure 1). Broadcast and multicast frames are copied and queued on the bridge (so they can be forwarded in all member interfaces), and the original frame is returned to *ether_input(),* so that it can be processed by the bridge machine itself. Unicast frames are checked to see if the destination matches any of the MAC addresses of ports on the bridge; if so, the frame is returned to *ether_input()* for local processing. If the frame is unicast and addressed to the bridge machine, the frame is queued and not passed back to *ether_input()*. When a packet is queued, a software interrupt is scheduled so that bridge processing will occur outside of the interrupt context of the ethernet card.

The bulk of the frame processing occurs in the software interrupt handler, *bridgeintr()* (see Figure 2). This routine loops through each bridge interface, pulling frames from their input queues. The source ethernet address and source interface are recorded into the bridge's address cache for each frame (after some address spoof-checking). The destination ethernet address is looked up in the cache; if the interface returned by the lookup is the same as the interface where the frame originated, no further processing is done. If the destination interface differs from the source, the frame must be forwarded (bridged). If the frame is for a multicast or broadcast destination, the frame must be forwarded to all member interfaces of the bridge. To avoid overloading *enc*

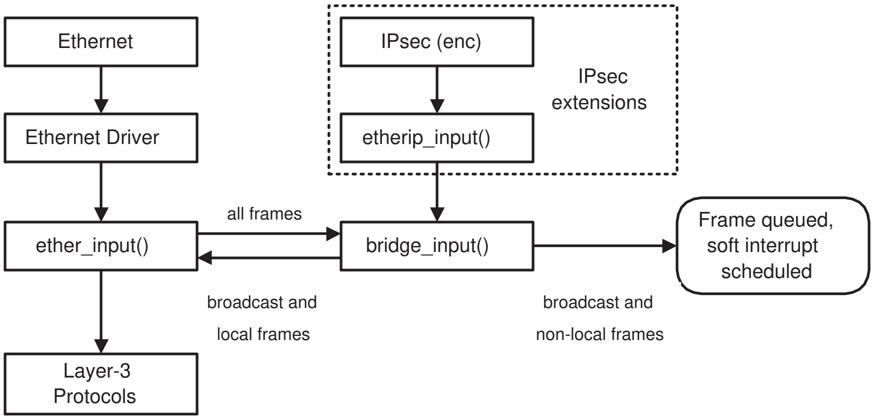


Figure 1: Frame flow from driver to bridge and layer-3 protocols.

interfaces with multicast traffic from fast ethernet interfaces, it is possible to disallow multicast packet and/or frame forwarding over the bridge. Currently, this is specified for the whole bridge. In the future, we would like to be able to specify this on a per-member interface basis.

### 2.1 Layer-3 Filtering

Before frames are forwarded, they are filtered by calling one of the *ipf* routines with the frame to be processed. This allows for standard filtering rules to be applied to bridge member interfaces as they would be for normal routed firewall. Rules are applied to all incoming frames that contain IP traffic and are bound to each member interface.

The *ipf* routines expect an IP packet to be passed to them, but the bridge operates in terms of ethernet frames. The ethernet header is examined to determine whether the frame contains an IP packet. Since there are two possible encapsulation methods for IP over ethernet, both must be examined and the appropriate amount of header information must be copied and removed from the frame, leaving the IP data intact. The resulting packet is passed to *ipf*, which either drops the packet or returns it. Packets that are not filtered have their ethernet headers reattached and are finally forwarded as determined by the bridge. Using this approach, we avoided having to modify *ipf* code at all.

### 2.2 Layer-2 Filtering

In addition to providing IP (Layer-3) filtering, the bridge is capable of filtering packets based on source and destination ethernet MAC address. The filtering rules follow a syntax much like the *ipf* rules and are applied in the order in which they are added. Rules can be applied both as a frame is received by the bridge (on input) or before the frame is sent out from the bridge (output).

The bridge can also be used to block all non-IP traffic. A flag on each member interface specifies whether it should allow non-IP traffic to be passed in or out based on the protocol field in the ethernet header. This allows frames to be blocked when they cannot be filtered by the Layer-3 mechanisms provided so that tunnels through other protocols cannot be created. The only protocols allowed through an interface with this flag are the protocols necessary for IP to function: IPv4, IPv6, ARP, and RARP.

### 2.3 Bridge as Normal Host

A machine acting as a bridge need not have an IP address. All of the filtering provided by the bridge and *ipf* can be handled in the absence of an IP address, and this is actually an easier case to handle.

For the bridge machine to act as a normal host, in addition to its duties as a bridge, several changes were necessary to the path a frame takes through the kernel. As discussed above, unicast frames that

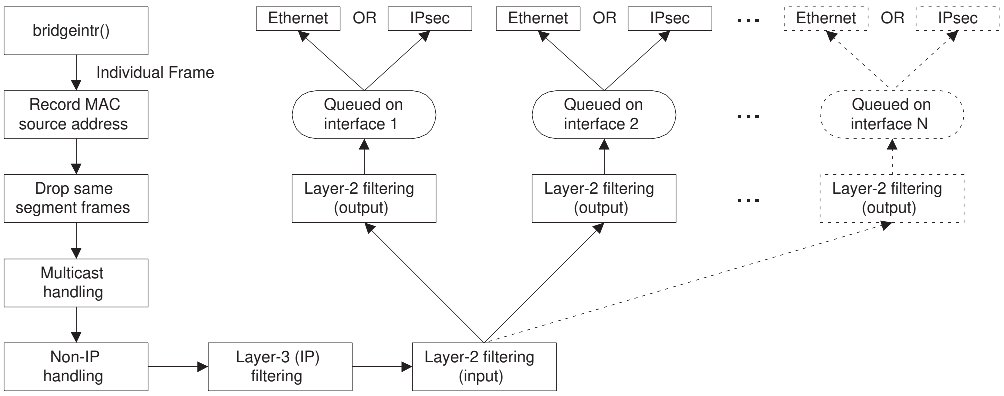


Figure 2: Frame flow from within *bridgeintr()* with Layer-2 and Layer-3 handling.

are addressed to any of the member interfaces of the bridge are simply returned to *ether_input()*. Broadcast and multicast frames must be copied. The original frame is returned to *ether_input()*, and the copy is queued on the bridge.

For frames sent by the bridge, *ether_output()* was modified to include a special case for interfaces that are bridge members and the frame to be sent is passed to *bridge_output()*. This function examines the ethernet destination address of the frame. For unicast destinations, the bridge address cache is used to locate the recipient. For multicast and broadcast destinations, as well as unicast destinations not found in the address cache, the frame is forwarded to all member interfaces of the bridge.

As a result of this design, a machine acting as a bridge can also participate on the LAN as a normal host. When, for example, it sends an ARP request for a host, it will be forwarded out of every member interface. When a reply is received on any interface, the source interface and address are added to the bridge address cache as well as its ARP cache, and the frame is processed as normal. From there, all unicast frames to the remote host will use the information from the address cache for sending frames only on the correct interface.

## 2.4 Bridge Security

As discussed previously, the bridge provides several methods for enforcing network security policy. One form of internal attack is MAC spoofing where one host forges packets using the ethernet MAC address of a victim host. The bridge provides two measures for preventing this attack from being completely successful: Layer-2 filters and static address entries.

For the Layer-2 filters, the ethernet MAC address of the potential victim is added to a set of rules. For the bridge interface on the segment where the host is supposed to be, rules are added to permit the address to be the source and destination of frames for input and output. On the other interfaces, the address is added to rules blocking it as a source address on input and destination address on output from each interface.

Additionally, adding a static address cache entry that binds the ethernet MAC address of the potential victim host to the bridge interface on the same segment as the host will prevent the bridge address cache from being polluted with invalid data. The bridge cannot prevent the attack from being successful on individual segments, but it can limit its scope in one segment only.

Another form of internal attack, ARP spoofing, involves a host on the network using its own MAC address and forging ARP responses claiming to be another host. The bridge does not treat ARP packets different from other packets, so this attack is not directly preventable. The attacking host may be able to convince hosts on other segments that its ethernet MAC address is the one associated with the IP address victim host, but by using IP filters, actual IP packet communication through the bridge can be prevented.

## 3 Bridging and IPsec

The filtering capabilities offered by the bridge allow its use as a transparent packet filtering firewall. As was the case with traditional firewalls however, filtering by itself is not sufficient in fulfilling network security concerns. Network layer encryption, typically in the form of IPsec, is seeing increasing use in protecting traffic between networks, hosts, and users. Thus, we decided to augment the filtering bridge with IPsec capabilities.

This section starts with a brief overview of the IPsec implementation in OpenBSD, then describes the two configurations where bridging and IPsec may be used together.

The first of these configurations, "virtual LAN," is used to transparently and securely connect ethernet segments over a wide area network. This is achieved by encapsulating ethernet frames inside IPsec packets which are then transmitted to a remote bridge that removes the protection and forwards the frames to the local LAN.

The second configuration is what the standards call a "bump in the wire" (BITW) implementation [9], wherein a security gateway (the bridge) transparently implements IPsec on behalf of one or more "protected" hosts. This allows gradual introduction of IPsec in a network without changing the end host configuration or software. This configuration is also a common design feature of network security systems used by the military.

Perhaps more importantly, such a transparent IPsec gateway can be used to enforce security properties for communications between the protected (or supervised) hosts and the rest of the world. Packets traversing the gateway can be examined and, depending on system policy:

- They may be forwarded or dropped, similar to a packet filtering firewall.
- Outgoing packets may cause the security gateway to attempt to establish a security association (SA) with the destination host, pretending to be the originating host, if the packets are unencrypted. If the packets are already IPsec-protected, it could simply forward them (or, in our case, bridge them). Naturally, the security gateway may always opt to establish an SA with the destination, regardless of the existing security properties of the packet stream.
- Similarly, for incoming packets, the gateway could establish a security association with the originator if the packet was received unencrypted and/or unauthenticated, again pretending to be the destination host.
- Finally, the bridge can intercept incoming IKE [5] packets that request negotiation with one of the protected hosts, and perform the negotiation as that host.

Thus, in the last three cases, the security gateway acts as a transparent network policy enforcer. A routing firewall can perform the same functions, however it would have to establish tunnel SAs between itself and the remote host on behalf of the protected host. It would do so using its own IP address, an so would need to "prove" its right to proxy-IPsec for the end host. While this is trivial for static configurations, where the identities and network addresses of the two peers are known a priori, the situation becomes more complicated when trying to do opportunistic encryption.

The two primary envisioned methods for host authentication are DNSSEC [3] and X.509 [2]. In the former case, the domain name servers can securely provide the public key associated with a host name or address. That key may then be used to authenticate the IKE peer. In the X.509 case, a Certification Authority (CA) infrastructure is assumed to provide the public key of an end host or user (the protocols for doing so in a large-scale network such as the Internet are less well-defined than DNS). Here, the IP address of the host associated with a key is embedded in an X.509 certificate. In either case however, it is not immediately clear (and currently undefined) how to express the right of a firewall to establish SAs on behalf of a host. While work has recently started in the IETF IP Security Policy (IPSP) Working Group, development and deployment of a protocol that would allow security gateway discovery is some years away.

### 3.1 OpenBSD IPsec

IPsec in the OpenBSD kernel is implemented as a pair of transport protocols [7, 8]. Incoming IPsec packets are switched to the appropriate IPsec protocol for processing by *ipv4_input(),* following the usual packet processing path in the kernel (similar, for example, to TCP or UDP). Note that only packets destined for the local host are handled this way; IPsec packets that are passing through are simply forwarded if the host is configured to act as a router, otherwise they are dropped).

Outgoing packets are intercepted at *ip_output()*, where a check is made to see if IPsec processing is necessary. If so, the appropriate IPsec protocol output routines are called which encrypt/authenticate the packet, and then re-send it via *ip_output()* specifying that IPsec processing has already occurred (so as to avoid loops). Two methods are used to determine whether a packet needs to be IPsec-processed:

- If the packet originated from a local socket, it may have an attached Security Association (SA).
- If no such information is available, the source and destination addresses and ports from the packet are used to make a lookup in the kernel Security Policy Database (SPD). In OpenBSD, the SPD is implemented as a new protocol family in the radix tree, which is also used for routing entries. There are several benefits to using the radix tree:
  - Code reuse (and, thus, bug avoidance).
  - The radix tree internal representation is compact, allowing for large numbers of SPD entries without high memory requirements.
  - Lookup cost scales well with number of entries in the table.
  - Because a lookup returns the most specific result, it is easy to implement "backup" entries for packet classes (or, conversely, we can have special case handling of certain packet classes).

In both cases, the lookup provides enough information to locate the SA. Note that, on input, the packet itself contains enough information to locate the SA. The SA itself contains such information as the cryptographic algorithms and keys to be used, what optional features of the protocols are in use, various expiration timers, *etc.*

Both the SA and SPD tables may be populated either through manual keying utilities, or by some automated key management daemon (like IKE [5] or Photuris [6]). The interface to the kernel for either of these methods is via the PF_KEY socket [14], which is in many ways similar to the BSD routing socket.

Both in input or output, if the necessary cryptographic material has not been negotiated with the remote IPsec endpoint (for example, when doing on-demand or "opportunistic" IPsec), it is possible to notify a key management daemon which will then negotiate and install the proper SA and SPD entries in the kernel.

We have also implemented the *enc* pseudo-interface. Input packets that are IPsec-processed are made to appear as if they were received from the *enc* interface, by changing the interface pointer in the mbuf header. Thus, an administrator can easily filter non-IPsec protected packets using any packet filtering package. Furthermore, utilities like *tcpdump* [13] can be used to view the intermediate products of IPsec processing, for debugging purposes (this only works if IPsec processing takes place in the local host).

A more extensive (if somewhat dated) overview of the OpenBSD IPsec architecture is given in [10].

This is the standard IPsec processing that is more or less common across different implementations (and even operating systems). Use of IPsec in conjunction with the bridge, especially in the "bump in the wire" scenario, requires somewhat different processing. We shall describe these changes and requirements in the next two subsections.

### 3.2 Virtual LANs

Given the way the bridging code operates, in particular with respect to member interfaces being added to and removed from the bridge, it was a simple observation that we could extend the role of the *enc* interface so that it could be used by the bridge. Accordingly, we modified the *enc* interface such that an incoming and an outgoing SA can be associated with it, through the standard *ifconfig* command [2]. Currently, such SAs must be manually configured, via the *ipsecadm* utility.

[2]As an artifact of our implementation, more than one SAs can be associated with an *enc* interface.

The effect of these changes is that local area networks (LANs) may be bridged over a public network. All that is necessary is that each LAN contain an IPsec bridge connecting it to one or more other LANs. From the point of view of the bridge, the IPsec link is identical to an ethernet network, allowing for creation of arbitrary topologies. Layer-2 and Layer-3 filtering, spanning tree algorithms, *etc.* may all be used in the IPsec-bridged network with literally no changes.

The modifications needed to the *enc* and *bridge* interface code were minimal. For the bridge, the only changes necessary were to allow non-ethernet interfaces to be attached and initialized properly (for example, it is not necessary to switch an *enc* interface into promiscuous mode). In the *enc* code, the routine that handles transmission was augmented to pass all enqueued ethernet frames to IPsec for processing and further transmission, after encapsulating them in IP or IPv6. Note that no SPD lookup is necessary here, since the output SA to use is already known.

To support multiple bridge topologies on the same host, a configurable number of *enc* interfaces is created. This number may be set at kernel compile time. By convention, packets received on SAs that do not have an *enc* interface associated with them, are made to appear as if they arrived on the *enc0* interface. Furthermore, the *enc0* interface is not allowed to have any SAs associated with it, nor can it be attached to a bridge. Thus, packets on SAs that have an *enc* interface associated may be traced or filtered using that interface. For all other SAs, the *enc0* interface may be used.

Implementation of the Ethernet-over-IP protocol also proved straightforward, as the output side of the protocol and the first half its input processing are identical to IP-in-IP encapsulation. At the end of input processing, if its input interface is linked to a bridge, the packet is passed to the bridge input routine. If a frame is received over an IPsec SA, its input interface will be the *enc* interface associated with the SA (see Figures 1 and 2).

In all, less than 300 lines of additional code in IPsec and the bridge were necessary to implemented the virtual LAN functionality.

When it comes to performance, the highest cost is, as might be expected, in the encryption. Figure 3 shows the cost of various algorithms (and combinations thereof). Note that AH only performs authentication (the packets are unencrypted).

Note however that it is usually the case that the wide-area network link over which the virtual LAN is established is much slower than the member LANs, and slower than the times shown above. Thus, realistically, the performance is limited mainly by the interconnecting infrastructure. The filtering capabilities of the bridge (blocking multicast/broadcast and non-IP packets) can be of some value here in managing the volume of traffic sent over the encrypted links.

Virtual LAN (vLAN) functionality is offered by a number of bridges, albeit it is used to mean something different from what we have described; more specifically, vLANs are used to compartmentalize a physical network into a number of “isolated” LANs. The main goal is to decrease the traffic “noise” as seen by machines that do not need to process it (*e.g.,* a printer does not need to receive NFS packets; likewise, normal hosts on the subnet probably do not need to see the AppleTalk packets the printserver uses to submit jobs to the printer). A side effect of vLAN employment is a limited form of security from casual packet sniffing. Such vLANs do not provide the same features our encrypting bridge offers (and vice versa).

### 3.3 Bump In The Wire

As mentioned in section 3, the bridge can also be used as a transparent IPsec box, sitting in front of a host or network and IPsec-processing packets traversing it. This configuration is called “bump in the wire” (BITW) in the IPsec architecture. The encrypting bridge as described in the previous section can be used almost as-is when the protected hosts or networks are configured to only talk to one remote host (or security gateway): an incoming and outgoing SA pair can be associated with an *enc* interface as before, and IPsec processing is done along the same lines. However, rather than encapsulating ethernet frames inside IP packets (and then IPsec-processing these), we extract the IP packets from the ethernet frames and do IP-in-IP encapsulation instead. The administrator can specify which of the two types of encapsulation should be used simply by setting the appropriate interface flag using the *ifconfig* command.

| Transform | Mbit/second |
|---|---|
| Software AH MD5 | 67.87 |
| Software AH SHA1 | 47.79 |
| Software ESP DES-MD5 | 19.56 |
| Software ESP Blowfish-SHA1 | 23.61 |
| Software ESP 3DES-SHA1 | 9.07 |
| Hardware ESP DES-MD5 | 62.12 |
| Hardware ESP 3DES-SHA1 | 63.12 |

Figure 3: Throughput of a TCP session over IPsec between two K6-3/550 boxes directly connected with 100Mbit/s ethernet. For the hardware numbers, we used a card with the Hi/Fn 7751 chip.

The SAs associated with the *enc* interface (which must be manually configured) can use the IP address of the bridge, or the IP address of the protected host. In the former case, the bridge exhibits the exact same characteristics as an encrypting gateway (packets sent to the remote host or gateway list the bridge's IP address as the source); in contrast to a gateway however, no configuration changes are necessary in the network or the protected host(s) when placing the bridge. Since SA configuration is manual, there are no issues with authentication during key establishment (as described in section 3).

When the SAs use the IP address of the protected host, the bridge is totally transparent to both the protected host and the destination host or gateway. There are two issues that need to be addressed in this configuration however:

- The IPsec standard specifies that IP fragments should not be IPsec processed in transport mode. That is, fragmentation must occur after IPsec processing has taken place, or tunnel mode (IP-in-IP encapsulation) must be used. Thus, the bridge must either use only tunnel mode IPsec, or reassemble all fragments received from the protected host, IPsec-process the re-constructed packet, then fragment the resulting packet. For performance and simplicity reasons, we decided to use the former approach. The disadvantage is that all IPsec-processed packets are 20 bytes larger (the size of the external IP header).

- Since the remote host is not aware of the encrypting bridge's existence, IPsec packets will be addressed to the protected host or network. Thus, we have to modify the bridge to recognize these addresses and process those IPsec packets. In fact, address recognition is unnecessary. The bridge only has to watch for IPsec packets (transport protocol ESP or AH), and use the information in the packet to perform an SA lookup. If an SA is found, the packet is processed. Otherwise, it may be dropped or allowed through depending on the filtering configuration.

A hybrid SA configuration may be used (where the bridge uses its address in one direction, and the protected host's address in the other). Such a configuration does not seem to offer any substantial benefit however (and may in fact result in confusing the administrator).

### 3.4 Transparent Policy Enforcement

While the mechanism described in the previous subsection is useful in its own right, its usefulness dramatically expands when the bridge is modified such that it can automatically establish SAs on behalf of the protected hosts.

Our IPsec implementation already supports dynamic SA acquisition by notifying a key management daemon like *isakmpd* [4] using the PF_KEY interface. SA acquisition occurs in the following two cases:

- An application requests some security service on a socket, by using the *setsockopt()* system call, and no SAs appropriate for the traffic pattern or security level exist.

- The kernel decides that a packet needs to be IPsec-processed, but no appropriate SAs exist. The kernel reaches this decision by consulting the SPD (as described in section 3.1).

We can use the same mechanism inside the bridge to dynamically establish SAs: when an IP packet (rather, an ethernet frame containing an IP packet) reaches the bridge and is about to be "transmitted" over an *enc* interface, an SPD lookup is made. If an SA appropriate for this packet already exists, it is used. Otherwise, the packet is dropped and a notification is sent to the key management daemon to establish such an SA. The granularity of the SA may be configured by the administrator (the same SA may be used for all traffic between the protected and the remote host, or just the specific TCP connection may be protected). Future packets with the same characteristics as the original packet will make use of the newly-established SA. Fortunately, no changes to the SPD are necessary.

This mechanism may be used to perform automatic re-keying for the virtual LAN or the simple "bump in the wire" configurations described in the previous two subsections.

However, left to its own devices, key management will establish an SA using the IP address of the bridge (and thus end up being functionally equivalent to an encrypting gateway). To really hide the bridge from the remote host, the source address of the protected host must be used. Thus the key management daemon, *isakmpd,* has to operate in the "bridge mode," wherein it asks the kernel to use a non-local IP address. This requires a minor change in the *bind()* system call code, to allow for socket binding to non-local addresses. To capture the responses, all UDP traffic to port 500 (the port used by the IKE protocol) is diverted to the bridge *isakmpd.* This is also necessary when remote hosts attempt to initiate an IKE exchange with a protected host. In both cases, *isakmpd* must be informed of and use the "local" address associated with the incoming packet. *isakmpd* also needs the "local" address so as to select the appropriate authentication information (*e.g.,* secret DSA [15] or RSA [12] key when doing X.509 or DNSSEC authentication). The changes to this effect are fairly minimal.

Incoming IPsec packets are processed as described in the previous subsection. Other incoming packets may cause an SA acquisition, depending on the security policy set by the administrator. Again, *isakmpd* needs to be informed of what IP address to use as the source address during the exchange.

The combination of packet filtering through *ipf* and SA-on-demand can be used effectively to enforce network security policy for the protected host(s). One particularly interesting configuration involves the bridge establishing SAs for unencrypted-only traffic; if end-hosts use IPsec or SSL for end-to-end packet security, the bridge simply forwards the packets. In another configuration, the bridge permits all packets through, but attempts to establish SAs for such communications and uses them if the remote hosts can do IPsec (also known as "opportunistic encryption").

## 4 Implementation Status and Future Work

Currently, the bridge lacks support for the spanning tree protocol which is part of the IEEE 802.1D standard[16], so care must be taken to ensure that loops are not created in the network. The Layer-2 filter rule system should be extended to provide a general mechanism for filtering specific ethernet protocols. We also intend to extend the bridge to allow for other types of LAN bridging (FDDI, PPP, *etc.*).

With regards to dynamic SA establishment, all traffic that traverses the bridge configured in the manner described in section 3.4 causes SA acquisitions. This is both undesirable and can have severe performance implications. A mechanism for the administrator to specify which packet flows should require IPsec protection (and thus cause an SA acquisition) is necessary. We are currently working on this issue.

More work needs to be done with regards to the performance implications of frequent IKE negotiations, as might be the case when the bridge is protecting a large network. Hardening against denial of service attacks (by exploiting too-aggressive SA acquisition rules) is also high in our to-do list.

The filtering bridge can also provide a transition step for a "distributed firewall"-protected network, as described in [1]. It may also be used in conjunction with a distributed firewall to provide protection against low-level network attacks (those that a distributed firewall is not well-suited to counter), or to protect legacy systems that cannot be modified to support the required functionality. Very low-priced systems (motherboard, processor, small disk, two ethernet cards, moderate amount of memory) may be used in such a configuration; such systems may also be used as "personal firewalls," similar to various commercial products that have begun to make their appearance in the market recently.

## 5 Conclusions

We have given an overview of the OpenBSD bridge implementation, with our extensions for Layer-2 and Layer-3 filtering (at the ethernet and IP layer, respectively). For the latter, we used the existing kernel packet filter mechanism, *ipf*. We further presented our integration of bridging with IPsec to provide "virtual LAN" functionality, "bump-in-the-wire" support, and a transparent security policy enforcement box. This latter configuration is shown to offer significant flexibility to network administrators, as it can be used in various modes of operation to ensure traffic as well as host and network protection. Finally, we discussed the current implementation status and our plans for future work.

## 6 Acknowledgments

The bridge was originally developed as an undergraduate independent study at the University of North Carolina at Greensboro by Jason L. Wright with Dr. Suzanne Lea as an advisor. The code was contributed to the OpenBSD project and integrated into the source tree prior to the 2.5 release (May 1999).

The authors would also like to thank Theo de Raadt and Jonathan Smith for their suggestions and support during the course of this work. Theo de Raadt suggested many of the original concepts behind the filtering bridge and the virtual LAN. This work was partly sponsored by DARPA under grant F39502-99-1-0512-MOD P0001.

## 7 Availability

All the software described in the paper is available through the OpenBSD web page at:

`http://www.openbsd.org/`

## 8 Disclaimer

OpenBSD is based in Calgary, Canada. All individuals doing cryptography-related work do so outside countries that have limiting laws.

## References

[1] S. M. Bellovin. Distributed Firewalls. *;login: magazine, special issue on security*, November 1999.

[2] Consultation Committee. *X.509: The Directory Authentication Framework*. International Telephone and Telegraph, International Telecommunications Union, Geneva, 1989.

[3] D. Eastlake and C. Kaufman. Dynamic Name Service and Security. Internet RFC 2065, January 1997.

[4] Niklas Hallqvist and Angelos D. Keromytis. Implementing Internet Key Exchange (IKE). In *Proceedings of the Annual USENIX Technical Conference*, June 2000.

[5] D. Harkins and D. Carrel. The internet key exchange (IKE). Request for Comments (Proposed Standard) 2409, Internet Engineering Task Force, November 1998.

[6] P. Karn and W. Simpson. Photuris: Session-key management protocol. Request for Comments (Experimental) 2522, Internet Engineering Task Force, March 1999.

[7] S. Kent and R. Atkinson. IP authentication header. Request for Comments (Proposed Standard) 2402, Internet Engineering Task Force, November 1998.

[8] S. Kent and R. Atkinson. IP encapsulating security payload (ESP). Request for Comments (Proposed Standard) 2406, Internet Engineering Task Force, November 1998.

[9] S. Kent and R. Atkinson. Security architecture for the internet protocol. Request for Comments (Proposed Standard) 2401, Internet Engineering Task Force, November 1998.

[10] A. D. Keromytis, J. Ioannidis, and J. M. Smith. Implementing IPsec. In *Proceedings of Global Internet (GlobeCom) '97*, pages 1948 – 1952, November 1997.

[11] Kirk McKusick, et. al. *The Design and Implementation of the 4.4BSD Operation System*. Addison Wesley, 1996.

[12] RSA Laboratories. *PKCS #1: RSA Encryption Standard*, version 1.5 edition, 1993. November.

[13] Steven McCanne and Van Jacobson. A BSD packet filter: A new architecture for user-level packet capture. In *Proceedings of USENIX Winter Technical Conference*, pages 259–269, San Diego, California, January 1993. Usenix.

[14] D. McDonald, C. Metz, and B. Phan. PF_KEY Key Management API, Version 2. Request for Comments (Informational) 2367, Internet Engineering Task Force, July 1998.

[15] Digital Signature Standard, May 1994.

[16] Internetworking Task Group of IEEE 802.1. Information technology – telecommunication and information exchange between systems – local and metropolitan area networks – common specifications – part 3: Media access control (mac) bridges. Technical Report ISO/IEC Final DIS 15802-3, IEEE P802.1D/D17, LAN MAN Standards Committee of the IEEE Computer Society, May 1998.