RC23642 (W0506-109) June 28, 2005
Computer Science

IBM Research Report

Leveraging IPSec for Mandatory Access Control of Linux Network Communications

Trent R. Jaeger
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Serge Hallyn, Joy Latten
IBM Systems and Technology Group
11501 Burnet Road
Austin, TX 78758

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Leveraging IPSec for Mandatory Access Control of Linux Network Communications

Abstract

We present an implementation of mandatory access control for Linux network communications that restricts socket access to labelled IPSec security associations. The Linux Security Modules (LSM) framework defines a reference monitor interface that enables security modules (e.g., SELinux) to enforce comprehensive mandatory access control (MAC) for Linux version 2.6. The current LSM control over network communication is limited, however. The LSM interface enables control of process access to sockets, but socket communications can only be restricted by network interfaces and IP addresses. We cannot use LSMs to control access to particular applications on remote machines or reliably associate request processing with the appropriate remote principals. The original proposal based on IP Security Options (IPSO) was found to be too expensive for unlabelled communications, so an alternative mechanism is necessary. Prior work on the Flask security architecture showed that IPSec can be used to enable MAC control on network communication. In this paper, we translate this approach into the Linux system, version 2.6.12. We describe our design for enforcement, which is based on the Linux 2.6 IPSec implementation called the XFRM subsystem (pronounced "transform"). We detail the modifications necessary to the kernel and user-level ipsec-tools to support IPSec policy specification and negotiation. Finally, we show how security function can be enabled using these LSM hooks with the SELinux LSM.

1 Introduction

We present an extension of the Linux Security Modules (LSM) framework available in Linux version 2.6 that leverages IPSec to enable mandatory access control (MAC) on network packets. The LSM framework defines a comprehensive reference monitor interface for Linux that enable loadable kernel modules to enforce MAC policies. A variety of modules (called LSMs) have been written to this interface that implement different forms of MAC [9, 10, 16]. Further, significant effort to enable LSMs for security is being undertaken by Linux vendors. For example, RedHat includes the SELinux LSM enabled in its Fedora Core 3 distribution.

Unfortunately, the MAC controls offered by the LSM framework are limited at present. The LSM framework enables a module to control which processes may use which sockets to send and receive messages, but little control over how the sockets are used. Network control is currently implemented by filtering inbound and outbound packets in protocol processing and via Netfilter, respectively. The power of the filters depends on the information available regarding the packet at this time. The original LSM proposal used the IP Security Options [18] to encode labelling information regarding a packet. The overhead of extracting this information and maintaining labels as packets are fragmented and defragmented added significant overhead to packet processing, even when no security information was specified. This part of the LSM proposal was rejected by the Linux networking subsystem maintainers, so LSMs currently control socket access by restricting the network interface and IP addresses that sockets may use. As a result, MAC control of networking is coarse-grained and not based on strongly authenticated remote parties.

Because of these limitations, it is not possible to build a LSM-based MAC system across multiple machines. Suppose we have two LSM systems one running a process labelled `master` that manages a distributed computation and one running a process labelled `helper` that performs some offloaded computations. When a packet is received

by the LSM Linux kernel destined for the `master` process, it is not possible for the kernel to determine if the packet should really be delivered. The IP address may be spoofed or the packet may not be from the `helper` process, so the integrity of the application may be compromised. Historically, such protections are implemented at user-level, but then the LSM system cannot reason about the overall integrity of the system.

To enable MAC on network communications, we leverage the kernel's mechanism for secure communication for the IP protocol, IPSec [12, 13, 14, 17]. IPSec enables the kernel to automatically construct secure communication channels between two machines or individual ports on those machines. The administrator configures *IPSec policies* that are used to determine the security requirements (e.g., using the ipsec-tools program `setkey`). Such requirements may be negotiated by the two systems dynamically by a user-level IKE daemon (e.g., ipsec-tools `racoon`). IPSec represents secure communication channels by *security associations* that are stored in the kernel and used on each packet.

Previous work demonstrated that MAC control on network communication could be implemented by restricting socket access to IPSec security associations in the Flask system [20]. In this work, security associations are labelled such that a socket's ability to send or receive a packet using a security association can be authorized. We begin by applying this basic goal to the Linux kernel, but find that the Linux implementation of IPSec, called the `XFRM` subsystem (pronounced "transform"), requires some different design decisions to minimize intrusiveness and performance impact.

In this paper, we describe the design of IPSec-based MAC of network communication for Linux 2.6.12. We implement the changes necessary to support this design in the Linux kernel and in the ipsec-tools that implement user-level administration and policy negotiation. Our kernel changes support the two interfaces that may be used to manage IPSec policies `pfkey` and `xfrm_user`. With these hooks, we get a bonus that IPSec secure communications may be selected based on the access control labels of the processes which enables finer-grained usage of IPSec. We examine the utility of the IPSec MAC controls by using these hooks to enforce remote application-to-remote application access control and assist a system in labelling processes triggered by requests from remote clients (e.g., via `inetd`). The downside is that the performance impact of IPSec must be absorbed on communications, but the security offered by IPSec is commensurate with that required for the types of secrecy and integrity goals that the access control supports. We find that the impact of new authorization mechanisms is within the noise of the network communication.

In Section 2, we describe the Linux Security Modules framework and Linux IPSec implementation (XFRM) upon which our work is based. In Section 3, we outline the design approach and the resolution of some key design issues. In Section 4, we detail the implementation of network access control in Linux. We describe some applications of our mechanism in Section 5. We summarize our findings and future work in Section 6.

2 Problem

In this section, we define the problem that we aim to solve in this paper, leveraging IPSec to enable mandatory access control (MAC) across machines using the Linux Security Modules (LSM) framework. We first describe how MAC is implemented in Linux by the LSM framework. We then examine how IPSec is implemented by the XFRM subsystem in Linux and how the ipsec-tools suite of user-level programs support IPSec configuration and negotiation. We then outline the security functions that we envision that LSM should provide.

2.1 Linux Security Modules

The Linux Security Modules (LSM) framework defines a reference monitor interface in the Linux kernel as shown in Figure 1. A *reference monitor interface* defines where authorization decisions are requested of an *authorization module*, which implements the actual authorization mechanism using its access control policy. An access control decision determines whether a particular *subject* (e.g., process, user, etc.) may access a particular *object* (e.g., file, socket, etc.) to perform a specified *operation* (e.g., read, write, etc.). The LSM reference monitor

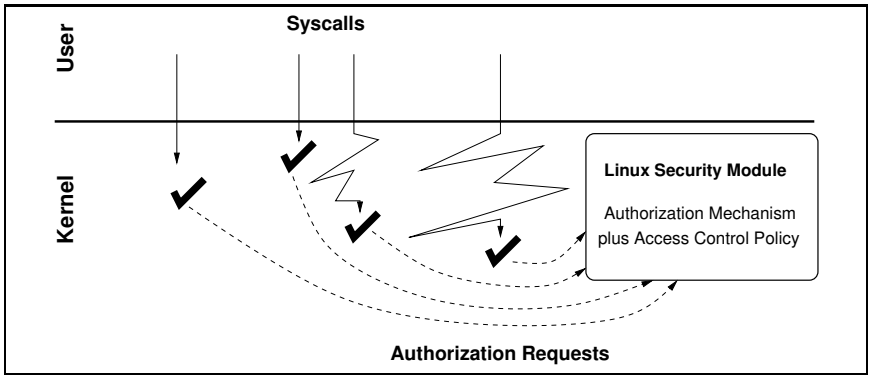

Figure 1: **Linux Security Modules (LSM) architecture.**

interface is implemented as function pointers that are placed at the location that such authorizations are required. 140 of such function pointers are defined in Linux 2.6.12.

The LSM design aims to meet the criteria of a classic reference monitor interface. Anderson identified that a reference monitor interface should have the following features [1]: (1) it must *mediate* all security-sensitive operations; (2) it must be as *simple* as possible to enable validation of correctness; and (3) it must be *tamperproof*. First, LSM mediates access to a wide variety of operations that are identified as security-sensitive operations in the Linux kernel. Independent validation of the mediation implemented by the LSM design has been done using static and runtime analyses [11, 24]. Second, the LSM design aims to place the authorization interfaces as close to the actual operation as feasible. This contrasts with the syscall interception approach [2, 6, 8] where the authorization may require redundant interpretation of arguments (e.g., to extract an inode from a file name) and be vulnerable to race conditions between time of check to time of user [3]. Third, the Linux kernel provides the tamperproof protection of all modules that use this interface.

A wide variety of modules have been implemented as reference monitors behind the LSM interface (e.g., [9, 10, 16]). These modules define the policy model and authorization mechanism that will be invoked when the LSM interface is used. In this work, we extend the SELinux module [16] to enable network control. The SELinux module implements the LSM interface comprehensively using an access control policy representing in an extended Type Enforcement policy model [5, 19]. SELinux support 29 different kernel object types with about 10 different operations per type. SELinux is included in the Linux kernel mainline distribution, and it is enabled by default in RedHat's Fedora Core 3.

Using the LSM interface enables control of most objects in a fine-grained manner. For example, the extended attributes of the ext3 filesystem can be used to store labels on individual inodes, so that access to each file may be controlled independently. For most types of kernel objects, fine-grained labelling is possible, but not for network packets. When a network packet is sent or received, the current LSM hooks enable authorization based on the socket and the packet (i.e., `sk_buff`). However, IP packets do not contain a lot of information that is used for authorization currently. SELinux authorizes network communications as follows: (1) it authorizes the process's access to the socket and (2) it authorizes the socket's access to the network interface used and remote IP address of the packet. For the receiver of a packet, this information is not particularly helpful in authorization. IP addresses may be spoofed, and the source IP address does not indicate the source application. In general, more than the IP address is necessary for two SELinux machines to work together to manage information flow.

2.2 IPSec

IP Security protocol [12, 13, 14, 17] (IPSec) provides per-packet authenticity and confidentiality guarantees between peer machines communicat-

ing over an untrusted network. An *authentication header* (AH) may be used to provide a strong cryptographic checksum for a packet. An *encapsulating security payload* (ESP) encrypts packets to protect the confidentiality of their contents. IPSec is a kernel protocol that is implemented as part of IPv4 and IPv6. The choice of modes and algorithms is determined by an *IPSec policy*. Policy entries may refer to a directed pair of machines and, optionally, the communication ports. The kernel determines if an IPSec policy is present for a particular communication, and if so, it retrieves an *IPSec security association* for the communication compatible with that policy. If no security association has been created, the kernel can initiate a *negotiation* with the remote party which is implemented by a user-level Internet Key Exchange (IKE) daemon which creates a security association if the negotiation is successful.

In Linux 2.6, an IPSec implementation is part of the mainline kernel. IPSec is implemented in the XFRM (pronounced "transform") subsystem which is based on the USAGI prototype [15, 23]. The idea is that prior to transmitting a packet or prior to delivering a packet to higher protocol layers, the packet may be transformed by a specific algorithm.

The implementation of IPSec in the XFRM subsystem is shown in Figure 2. First, the `pfkey` and `xfrm_user` interfaces are defined which enable IPSec policies and security associations to be input to the kernel. The kernel stores IPSec policies in the Security Policy Database (SPD) and the security associations in the Security Association Database (SAD). Second, when a packet is sent, the kernel determines whether there is an IPSec policy for the endpoint pair. If so, it is used to retrieve an existing transform bundle which is a set of security associations. These correspond to security associations in the traditional IPSec sense. If no bundle can be found, then the kernel tries to *acquire* a bundle using the IKE daemon. If one is generated, then a bundle is created that includes all the security associations for this endpoint pair. Prior to transmitting the packet, the bundle of security associations is applied to transform the packet.

On the receiving end, the security associations in the transform bundle provided are applied first, then the IPSec policy is retrieved to check whether it is consistent with the bundle applied. The IPSec policy is retrieved using the same code as for the transmission case. Then, the retrieved IPSec policy is compared to the security associations in the bundle to verify that each algorithm in the policy is implemented by a security association with the same identifier (i.e., spi).

In both cases, bundles are attached to each packet, so it is possible to use the security associations in them to guide authorization decisions. A previous prototype on the Flask security architecture [7] demonstrated that socket access to security associations could be used to control packet sends and receives. The design of the Linux XFRM subsystem seems to enable such authorization, but we find that the existing LSM interface is not suitable to use such information correctly as described in the following subsection.

We must also consider changes to the user-level daemons that to support authorization via IPSec. We use the ipsec-tools programs `setkey` and `racoon` to manage IPSec policies and negotiate security associations, respectively. `setkey` takes policy specifications and generates IPSec policy and security associations objects that it submits to the kernel for entry into the SPD and SAD, respectively. `racoon` accepts negotiation requests from the kernel including an IPSec policy. It then negotiates transform bundles and enters them into the kernel SAD if the negotiation succeeds. Both daemons use the `pfkey` interface to communicate with the kernel. Note that the addition of access control information will require changes to both daemons: (1) to specify access control labels in the `setkey` configuration and (2) to account for access control labels in negotiation in `racoon` and generate security associations with appropriate access control labels.

2.3 Problem Statement

The problem is to determine how to extend the LSM interface using the IPSec (XFRM) implementation in Linux to enable the kernel to control an application's network communication based on its access control label. We envision that such a mechanism would enable two LSM-based systems in the same trust domain to limit communication between their applications. An additional benefit we envision is that security-aware applications can use

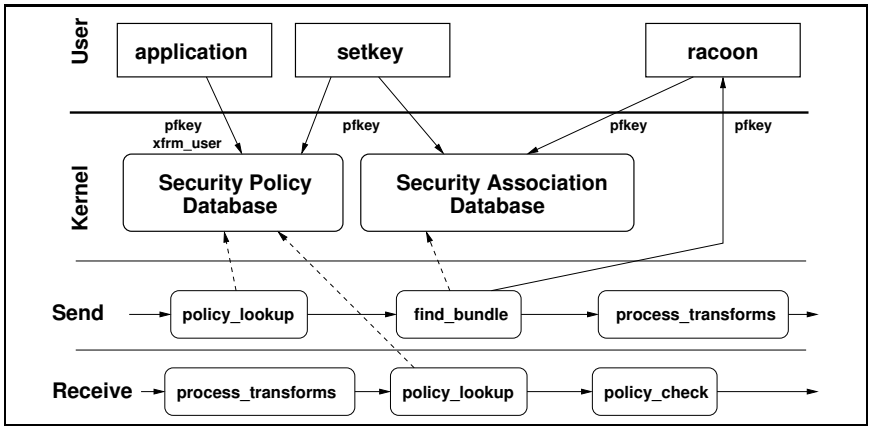

Figure 2: **Linux XFRM subsystem.**

the labels on the security associations to guide the selection of access control labels for the processing that such communications trigger (e.g., `inetd`-triggered processing).

IPSec is typically not envisioned as an application-level mechanism, but the ability for a LSM-based system to control information flows to applications enables application-to-application information flow control. The recent work of Yin and Wang demonstrate that IPSec may enable application-level provisioning [22]. That work relies on an ad hoc socket monitor to detect socket activity and trigger policy changes. In this work, we want to enable our security via small extensions to the LSM interface and minor changes to ipsec-tools to recognize access control labels. Distribution of credentials for IPSec is a significant problem for establishing distributed trust [4] as well. We do not address this issue specifically, but using IPSec to enable fine-grained access control will require that such issues be addressed effectively in the future.

The straightforward solution would be to use the existing LSM interface to authorize the packet's security associations used. The problem is that if the choice of IPSec policy does not account for access control labels, then the security associations will often not be authorized and communication will be terminated. Thus, new LSM hooks must be added that account for authorization in the choice of IPSec policies and ensure that this choice is propagated to security associations. Unlike the IP Security Options hooks previously proposed, the new LSM hooks should minimally impact unlabelled packets and any implementation should have minimal, additional per packet overhead.

3 Design

The overall design of the LSM and ipsec-tools extensions that enable packet-level access control based on labelled security associations is shown in Figure 3. First, IPSec policies and security associations are extended so that they may contain access control labels. We must both add labels to objects (`alloc`) and ensure that labels are removed when the objects are deleted (`free`). If no label is provided, the default label of *unlabelled* is assumed.

Second, when a packet is sent or received, an IPSec policy for that packet is retrieved by the kernel. A new LSM hook is added to authorize a selection (`lookup`), so only authorized policies are used subsequently.

Third, the authorized IPSec policy is used to ensure that the security associations used have the same access control label (`match`). No new LSM hooks are added, but functions are added in the XFRM subsystem that ensure that the access control label identifiers match. The identifiers are module-specific, but the matching function need not be – it can compare the module identifier and labels. For inbound communications, the security association used must have the same access control label as the retrieved IPSec policy. For outbound communications, we extend the kernel to ensure that only se-

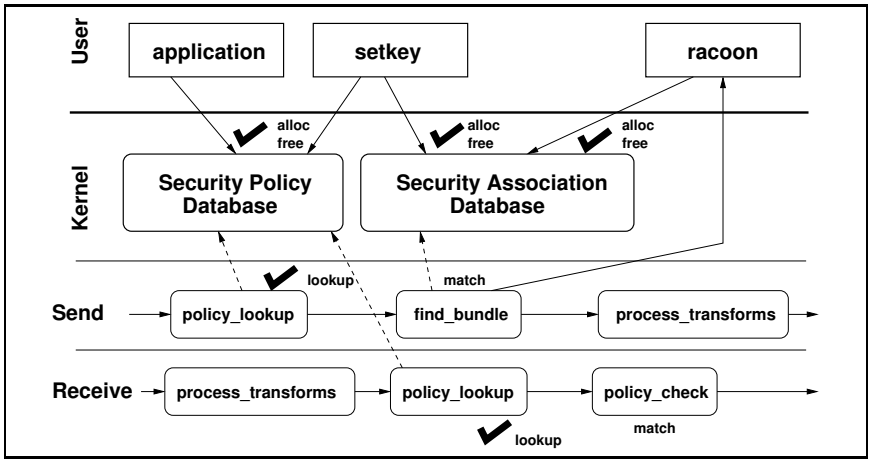

Figure 3: **LSM extensions for IPSec network control.**

curity associations with the same access control label are retrieved. If no security association matches, then the kernel requests that the IKE daemon (e.g., ipsec-tools `racoon`) negotiate a security association given the authorized policy. The IKE daemon must be capable of building a security association with a specified access control label.

In the remainder of the section, we discuss the details of the design of these three steps and the resolution of the design issues that were identified.

3.1 Adding Access Control Labels

The IPSec policy and the security association data structures are extended by the addition of an access control label, `xfrm_sec_ctx`. The structure has the following fields:

```
domain_of_interpretation
algorithm
SID
context_name
```

The *domain of interpretation* is used by the IKE daemon to identify the domain in which the negotiation takes place. The *algorithm* specifies the LSM for which the label is generated (e.g., SELinux). The *SID* is an integer representation of the label that is interpreted by the LSM. The *context name* is a string representation of the label, also interpreted by the LSM. Storing both an integer and string representation for the label is done to speed authorization, which uses the integer, and dumping contexts on user requests, which uses the string.

When an IPSec policy or security association is input (e.g., via `pfkey`), an LSM hook must be added to allocate the label and perform any LSM-specific processing (e.g., computing the SID from the context name). We have one hook each for IPSec policies and security associations, `xfrm_policy_alloc` and `xfrm_state_alloc`, respectively. Since `xfrm_sec_ctx` is dynamically allocated, it must also be freed, by hooks `xfrm_policy_free` and `xfrm_state_free`.

The choice of label must be authorized for the requesting process. First, we must control which programs can input IPSec policies and security associations in general. SELinux has a set of operations on the *security* of the system, so a new operation is added for IPsec configurations, `ipsec_label`.

Also, it is also possible for applications to set policies for their own sockets via `setsockopt`. For example, `racoon` must set a policy for its socket that handles negotiation to avoid causing a recursive negotiation request. We must ensure that the label chosen for such policies are permissible. We do not want a low secrecy process to create a policy that permits it to read high secrecy data. The process of adding policy is tantamount to relabelling a security association from the default label for the policy (e.g., entered via `setkey`) for the process to a new label. The process must be authorized to

perform such relabelling. In SELinux, this involves having permissions to `relabelfrom` the origin label and `relabelto` the specified label.

3.2 Authorizing IPSec Policy

Policy selection is the point at which authorization is done. In the XFRM subsystem, an IPSec policy that matches the source, destination, protocol, and ports (if specified) is retrieved. In addition, we require that the IPSec policy has a security context that the socket can use for the operation (i.e., send or receive). An LSM hook is added to authorize any matching policy, called `xfrm_policy_lookup`. If the policy is not authorized, then the XFRM subsystem continues to search for another match.

The result is that the IPSec policy selected for a socket is always the first one retrieved that both matches the selection criteria and is authorized. This is consistent with the prior case where the first policy that matched the selection criteria is returned. As a result, each combination of socket label and selection criteria can result in only one IPSec policy being selected, so the correctness of the IPSec policy database can be checked against this property.

The XFRM subsystem actually retrieves IPSec policies in two passes: (1) socket-specific policies added via `setsockopt` and (2) general machine and port policies. Thus, the `xfrm_policy_lookup` LSM hook for authorization must be added in both places.

With the addition of labels, IPSec policies may be specific to processes. In SELinux, sockets inherit the labels of their process by default. For example, a high secrecy service may be labelled *high service* and its sockets would inherit the same label. When the socket aims to send a packet to a remote computer, we can limit it to send only to other computers that we trust to deliver the communication to authorized processes (e.g., other processes labelled *high service*. This is done by giving *high service* sockets access only to security associations with labels accessible to remote *high service* sockets (e.g., *high data*). If the remote machine is truly compatible and trustworthy, it can restrict the delivery of packets only to sockets that can receive packets via the *high data* security associations. Since only these sockets running in *high service* processes can receive *high data*, no lower secrecy applications can intercept the data. We do not actively address covert channels in this design as we see it being outside the scope of hook placement (i.e., based on storage and timing channels in the drivers and protocol).

3.3 Using Security Associations

Once an authorized IPSec policy has been selected, we must ensure that the security associations used in the communication have the same access control label as the policy. Since security associations are used differently on inbound and outbound communications, we examine each separately.

For outbound communications, the XFRM subsystem uses the previously cached *bundle* of security associations for the policy. We ensure that any bundle retrieved matches the access control label of the policy. If none are cached, then security associations may be retrieved individually from the security association database. Once again, we ensure that these security associations have a matching access control labels with the policy. Finally, if there is no matching security association in the database with the same access control label, then the IKE daemon will be requested to negotiate one. Note that this is the existing behavior of the XFRM subsystem, so no additional code is necessary to trigger the negotiation. Once the negotiation is complete, we check that the security association built by the IKE daemon has an access control label that matches the policy.

The IKE daemon must be modified as well to ensure that the access control label is used in the negotiation. The IKE daemon generates possibly multiple proposal payloads that consist of a set of transform payloads. The idea is that one of the initiator's proposal payloads should match the proposal payload generated by the responder. If so, that proposal is returned to the initiator for acceptance. We must modify the proposal payload generation on both sides to ensure that security associations are built with the appropriate access control labels. On the initiator side, the access control label is extracted from the policy submitted to the IKE daemon, and it is added as an attribute to the corresponding security association in the proposal payload. On the responder side, the responder is

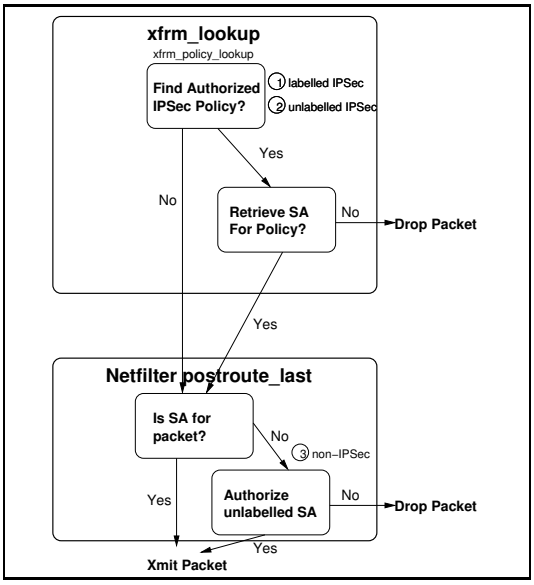

Figure 4: **Outbound packet processing.**

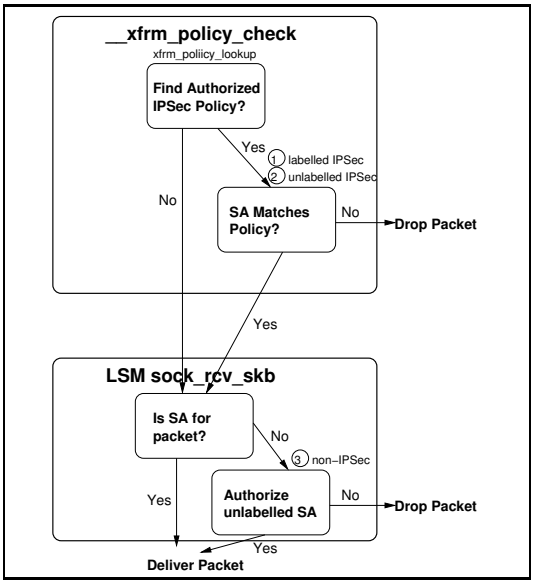

Figure 5: **Inbound packet processing.**

changed to peek at the initiator's proposal to extract the access control labels. There must be a policy in the initiator's database that matches this proposal and access control label, else the negotiation will fail.

For inbound communications, the XFRM subsystem compares the security associations of the received packets to the authorized IPSec policy for the socket. First, we extend the XFRM subsystem to ensure that only authorized IPSec policies are selected. The same function is used to select policies for both the inbound and outbound direction, so no new LSM hooks are required. Second, comparison between security associations and IPSec policies is based on a runtime identifier, `spi`. When the security associations are built, templates of each are associated with the IPSec policy from which they originated. The template spi and the security association spi must match. We extend this comparison to also verify that the access control labels match. This may not be strictly necessary since the kernel verifies consistency between security associations and policy on negotiation, but this does ensure correctness of behavior at runtime for low cost.

4 Implementation

In this section, we detail the implementation of the design described above in the Linux 2.6.12 kernel and ipsec-tools. The overall process is shown in Figures 4 and 5. There are three cases implemented by the LSM nethooks: (1) for packets that use IPSec policies with access control labels; (2) for packets that use IPSec policies without access control labels; and (3) for non-IPSec packets. In the first case, the socket is authorized to use the labelled IPSec policy at selection time. The second case is also checked at selection time, but the socket is authorized to process "unlabelled" packets. The third case applies only to packets that are not processed via the IPSec code path. In this case, we use the existing network control hooks to determine whether the IPSec authorizations have been performed. If not, then the socket is authorized to use "unlabelled" packets as in the second case.

Below, we describe the changes to the Linux kernel interfaces and XFRM subsystem, SELinux LSM, and ipsec-tools that are necessary to enable such authorizations.

4.1 `pfkey` and `xfrm_user` Changes

The kernel supports two interfaces by which IPSec policies (`xfrm_policy`) and security associations

(xfrm_state) can be input. Passing access control labels to the kernel requires that a data structure be defined for transporting this information to the kernel. Fortunately, the same information can be provided in the case, so a common data structure xfrm_user_sec_ctx is defined for both. The LSM hooks for allocation take these data structures and add access control labels to IPSec policies and security associations as defined below:

```
xfrm_policy_alloc(struct xfrm_policy *xp,
        struct xfrm_user_sec_ctx *ctx)
xfrm_state_alloc(struct xfrm_state *x,
        struct xfrm_user_sec_ctx *ctx)
```

These LSM hooks dynamically generate xfrm_sec_ctx objects that are stored in the xfrm_selector object of the IPSec policy or security association depending on the hook. The selector is used to match the IPSec policy or security association on subsequent retrieval.

Each command that enables adding an IPSec policy or security association via these interfaces must have an LSM hook to allocate the security data structure xfrm_sec_ctx. For the pfkey interface, IPSec policies can be added via the pfkey_spdadd command and by applications for individual sockets using setsockopt (pfkey function compile_policy). xfrm_user has corresponding functions for xfrm_add_policy and xfrm_compile_policy. Security associations can only be added via the interfaces (i.e., not via setsockopt) via commands pfkey_add and xfrm_add_sa.

Implementations of these hooks have been developed for the SELinux LSM. The main tasks of these hooks are to: (1) authorize the addition of the access control labels (xfrm_sec_ctx) and hence the IPSec policies and security associations themselves; (2) generate the access control label object; and (3) compute and store LSM-specific information in the fields of the label. First, authorization is requires a permission to change system security by adding IPSec policies. This is defined by the operation ipsec_mod on the SELinux class security. Further, the subject must be capable of relabelling security associations from the origin label to the proposed label. We use the subject type of the socket as the origin label. Second, SELinux allocates the new access control label and attaches it to the IPSec policy or security association (see below). Third, SELinux uses the access control label string to compute a label identifier (e.g., SELinux SID) for use in subsequent authorization. The access control label string is an opaque value that can be interpreted by the module as necessary.

Since the xfrm_sec_ctx objects are dynamically allocated, they must be deallocated when deleted. The hooks below are added where IPSec policy and security association additions listed above fail and where they are freed, __xfrm_policy_destroy and xfrm_state_gc_destroy.

```
xfrm_policy_free(struct xfrm_policy *xp)
xfrm_state_free(struct xfrm_state *x)
```

4.2 XFRM Subsystem Changes

Once the IPSec policies in the Security Policy Database (SPD) and the security associations in the Security Association Database (SAD) are labelled, then the XFRM subsystem needs to be modified to utilize these labels. Below, we show how the following steps in IPSec processing are extended to ensure access control labels are enforced.

IPSec Policy Lookup For each packet, the first step is to retrieve the IPSec policy for the packet. IPSec policies are retrieved in two phases: (1) socket-specific IPSec policies are retrieved via xfrm_sk_policy_lookup and (2) general IPSec policies are retrieved via xfrm_policy_lookup. Both are changed to authorize the socket's access to use the IPSec policies using the following command.

```
security_xfrm_policy_lookup(struct sock *sk,
                  struct xfrm_selector *sel,
                  struct flowi *fl,
                  u8 dir)
```

The sock defines the subject, the xfrm_selector has the access control label, the flowi is used when there is no socket (e.g., ICMP), and the direction indicates a send or receive operation. This hook is used in both lookup functions as below.

```
xfrm_policy_lookup(struct flowi *fl,
                   struct sock *sk,
                   u16 family, u8 dir,
              void **objp,
                   atomic_t **obj_refp)
{...
for (pol = xfrm_policy_list[dir]; pol;
      pol = pol->next) {
   xfrm_selector *sel = &pol->selector;
   int match;
   ...
   match = xfrm_selector_match(sel, fl,
             family);

   if (match) {
      if (!security_xfrm_policy_lookup(sk,
                   sel, fl, dir)) {
         xfrm_pol_hold(pol);
      break;
      }
   }
}
```

The SELinux LSM has been modified to use this new LSM hook to authorize a socket's access to an IPSec policy covering the first two cases of authorization (i.e., labelled and unlabelled IPSec packets). If the `xfrm_selector` for the IPSec policy has an access control label (`xfrm_sec_ctx`), SELinux extracts the access control label identifier SID from the structure and uses it to authorize access. If not, the socket is authorized to access (i.e., send or receive depending on the direction input) an unlabelled security association.

This implementation covers the first two authorization cases shown in Figures 4 and 5. In the outbound case, `xfrm_lookup` is used to retrieve a policy which is now authorized. The security association bundle that is retrieved must match the authorized access control label. For IPSec packets (i.e., those with a security association), the existing Netfilter hook permits it to be sent, because it has already been authorized. In the inbound case, the authorized policy is retrieved by `__xfrm_policy_check` in the same way as for the outbound case. Matching the spi is done in `xfrm_state_ok` for each security association in the bundle for the inbound packet. Similar to the outbound case, the existing input packet filter hook `sock_rcv_skb` permits IPSec packets to be delivered without further authorization.

Retrieving Security Associations Once the IPSec policy has been determined, we must ensure that the security associations used have the same access control label. The function `xfrm_sec_ctx_match` ensures that two access control labels match. We use this function to ensure that the access control label of a security association matches that of the authorized IPSec policy. For outbound communication, the match is done on security associations cached on the policy (protocol-specific function, such as `__xfrm4_find_bundle`), security associations retrieved from the SAD (`xfrm_state_find`), and security associations returned by IKE negotiation (also `xfrm_state_find`). For inbound communications, the match is done when the identifier of the policy is checked against the input security association.

A problem is that packets are dropped during IKE negotiation, so an application that expects that the transmission will succeed may not attempt a retransmission. To address this, the packets should be queued in the kernel for a retransmission. An implementation of this mechanism has not been completed at the present time.

4.3 SELinux Module Changes

The SELinux LSM has been modified to implement hook functionality for allocation, freeing, and lookup that is described in the previous subsections. In this subsection, we detail two additional extensions to the SELinux LSM: (1) authorize access to non-IPSec packets per the third case in Figures 4 and 5 and (2) retrieve access control labels for active security associations for use by access control-aware applications.

Authorizing Non-IPSec Packet Access The SELinux LSM has existing functions for filtering inbound or outbound packets, `sock_rcv_skb` and Netfilter `postroute_last`, respectively. Currently, these functions authorize socket access to ports, IP addresses, and network interfaces. We extend these functions to also authorize non-IPSec packets. Since both IPSec and non-IPSec packets use this hook, we distinguish between them by checking whether the packet has any security as-

sociations attached to it. For inbound packets, a security path is associated with each packet, and all IPSec packets will have security associations in the security path. For outbound packets, a series of `dst_entry` objects may be associated with a packet, and all IPSec packets will have security associations attached to at least one of these objects.

Retrieving Labels Using the `getsockopt` system call with the flag `SO_PEERSEC`, it is possible to retrieve the access control label of a peer in a UNIX domain socket communication. Since the UNIX domain sockets are on the same machine and both are used in authorization, the LSM can cache the label of the respective peer sockets, so that this can be retrieved using the system call.

We extend the SELinux LSM to also cache the label of a security association being used by a TCP socket. This can be retrieved using `getsockopt` as well via a new a new flag `SO_TCPPEERSEC`. Our patch adds LSM hooks that set the peer on TCP connection (`set_tcp_peer` establishment and remove the peer on closure (`remove_tcp_peer`). The former uses the security association on the packet at the time of connection to set the peer. This is determined by the socket's state being set to `TCP_ESTABLISHED`. The latter simply clears the peer value.

4.4 `ipsec-tools` Changes

To enable awareness of access control labels to the `ipsec-tools` suite, a data structure matching the `xfrm_sec_ctx` is added.

```
struct security_ctx {
 u_int8_t ctx_doi;
 u_int8_t ctx_alg;
 u_int16_t ctx_strlen;
 char ctx_str[MAX_CTXSTR_SIZE];
};
```

This new structure, `security_ctx` is added to the already existing, `policyindex` structure. The `policyindex` structure is used as a key to find policy. It is similar to the `xfrm_selector` structure in kernel.

When the IKE daemon `racoon` receives an *acquire* message from the kernel to initiate a negotiation, this message includes an index to an outbound IPSec policy. This index is used to retrieve the outbound policy from the `racoon` database. `racoon` uses the policy and `sainfo` from its configuration file (`racoon.conf`) to create a proposal.

In the ISAKMP negotiation protocol, a payload consists of a hierarchical set of *security association payloads* containing one or more *proposal payloads* that in turn contain one or more *transform payloads*. The transform payload consists of a set of attributes, such as the encryption or authentication algorithms, lifetime, etc. We added the access control label as another attribute. The access control label value from the retrieved policy is added to the set of transform attributes, if present.

The responder extracts the source, destination, and protocol to retrieve its own IPSec policy which is used to create a proposal. The responder then parses the SA payload sent by initiator and compares the newly created proposal with that sent by initiator. The responder derives a specific proposal which is sent to initiator for approval. If initiator approves, this becomes the new set of SAs.

We modified this sequence slightly to add `get_security_context(vchar_t *sa, struct policyindex *p)` which the responder calls before retrieving its policy. This routine peeks into the initiator's proposal to get the proposed access control label. Currently, only one access control label is used, although it may be that multiple options are proposed ultimately. If there is an access control label attribute, the contents are copied into the newly generated proposal.

When the responder compares its newly created proposal to initiator's proposal, the access control labels must match exactly, or the negotiation fails. This mode of negotiation means that the access control label for both the inbound and outbound security associations will be the same. This is not necessarily a requirement of IPSec if the policies are manually specified via `setkey`. We note that negotiated encryption/authentication algorithms also will always match, so having the access control labels match is consistent with current behavior.

5 Applications and Evaluation

We discuss two applications of IPSec-based MAC of network communication.

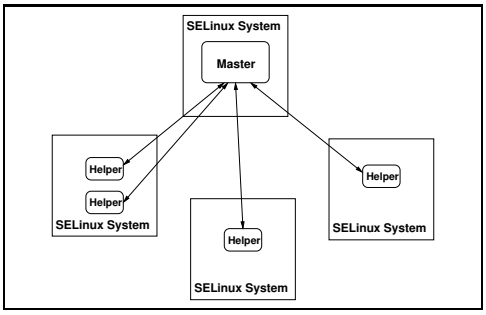

Figure 6: **Distributed Computation Example.**

5.1 Distributed Computation

Figure 6 shows a master computation process and helper processes that it distributes to remote machines that it trusts. All systems run our IPSec-based MAC Linux system with the extended SELinux LSM. The IPSec policy specifies an access control label of `master_computation` may be used for traffic between the helper machines and the master. The SELinux policies on each machine specify that the master process is labelled `master` and helper processes are labelled `helper`. Further, the SELinux policies on permit communications using `master` and `helper` sockets to use the `master_computation` security associations. As a result, the `master` and each `helper` can communicate. Note that the helpers cannot communicate unless an IPSec policy is added with the `master_communication` label for a pair of helper machines.

5.2 Labelled `inetd`

`inetd` can use the mechanism described in Section 4.3 to retrieve the access control label of a security association using `getsockopt` with the `SO_TCPPEERSEC` flag. We configure IPSec policies such that each subject from a different domain receives a different security association label. When `inetd` receives a remote request, it can retrieve the access control label of the security association used. It can then use the SELinux command `setexeccon` to cause the resultant helper process to be run under the same label.

6 Conclusion and Future Work

In this paper, we detailed an implementation of mandatory access control for Linux network communications that restricts socket access to labelled IPSec security associations. This implementation has been developed as a patch to Linux 2.6 which is under consideration for adding to the future Linux systems. The main patch consists of new LSM hooks for allocating and deallocating access control labels for IPSec and for authorizing IPSec policy selections. Also, the patch ensures that IPSec security associations have labels that match the authorized policy. Since the only functionality added to the runtime IPSec processing is the authorizing LSM hook and the matching code, the runtime overhead is negligible. We also described patches for SELinux and ipsec-tools that implement the authorization and IPSec configuration, respectively. We demonstrated these patches using two examples: (1) access control of network communications between processes running on different systems and (2) enabling security-aware applications to leverage access control labels for subprocesses that they spawn. The latter example requires two additional LSM hooks to update connection state.

The main challenge that emerges from this work is enabling management of policies and credentials for both IPSec and SELinux (or other LSM) across multiple systems. First, credentials for applications need to be securely distributed, such that IPSec communications can be established. Currently, IPSec is not used in such a fine-grained manner, but the complimentary work of Yin and Wang [22] shows that application-level sensitivity in IPSec is useful. Next, SELinux policies need to become multi-system aware. At present, SELinux policies have only a single system scope, mainly to protect that system from the outside world. However, application-level control of network communication means that SELinux will need to enable control at a scope that protects a distributed application from the network and from other low integrity processes on the individual machines.

Acknowledgements

Anonymized for review.

References

[1] J. P. Anderson. Computer security technology planning study. *ESD-TR-73-51, Air Force Electronic Systems Division*, 1972.

[2] A. Berman, V. Bourassa, and E. Selberg. TRON: Process-specific file protection for the UNIX operating system. In *Proceedings of the 1995 USENIX Winter Technical Conference*, January 1995.

[3] M. Bishop and K. Dilger. Checking for race conditions in file accesses. *Computing Systems*, 9(2), 1996.

[4] M. Blaze, J. Ioannidis, and A. Keromytis. Trust management for IPSec. In *ACM Transactions on Information and System Security (TISSEC)*, 5(2), May 2002.

[5] W. E. Boebert and R. Y. Kain. A practical alternative to hierarchical integrity policies. In *Proceedings of the* 8^{th} *National Computer Security Conference*, Gaithersburg, Maryland, 1985.

[6] S. Chari and P-C. Cheng. BlueBoX: A policy-driven, host-based intrusion detection system. In *ACM Transactions on Information and System Security (TISSEC)*, 6(2), May 2003.

[7] A. Chitturi. Implementing mandatory network security in a policy-flexible system. *Master's thesis, University of Utah*, 1998.

[8] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer. A secure environment for untrusted helper applications: Confining the wily hacker. In *Proceedings of the* 10^{th} *USENIX Security Symposium*, August 2001.

[9] Immunix Corp. Immunizing applications with AppArmor host intrusion prevention. `http://www.immunix.com`, 2005.

[10] LIDS Organization. LIDS: Linux Intrusion Detection System. `http://www.lids.org`, 2005.

[11] T. Jaeger, A. Edwards, and X. Zhang. Consistency analysis of authorization hook placement in the Linux Security Modules framework. *ACM Transactions on Information and System Security (TISSEC)*, 7(2), May 2004.

[12] S. Kent and R. Atkinson. Security Architecture for the Internet protocol. *RFC 2401, Internet Engineering Task Force*, 1998.

[13] S. Kent and R. Atkinson. IP authentication header (AH). *RFC 2402, Internet Engineering Task Force*, 1998.

[14] S. Kent and R. Atkinson. IP encapsulating security payload (ESP). *RFC 2401, Internet Engineering Task Force*, 1998.

[15] K. Miyazawa *et al*. IPv6, IPSec, and Mobile IPv6 implementation of Linux. In *Proceedings of the 2003 Ottawa Linux Symposium*, July 2004.

[16] National Security Agency. Security-Enhanced Linux (SELinux). `http://www.nsa.gov/selinux`, 2001.

[17] D. Piper. The Internet IP Security domain of interpretation. *RFC 2407, Internet Engineering Task Force*, 1998.

[18] M. St. Johns. Draft revised IP Security Option. *RFC 1038, Internet Engineering Task Force*, 1988.

[19] S. Smalley. Configuring the SELinux policy. NAI Labs Report #02-007, available at www.nsa.gov/selinux, June 2002.

[20] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Anderson, and J. Lepreau. The Flask Security Architecture: System support for diverse security policies. In *Proceedings of the* 8^{th} *USENIX Security Symposium*, August 1999.

[21] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman. Linux Security Modules: General security support for the Linux kernel. In *Proceedings of the* 11^{th} *USENIX Security Symposium*, August 2002.

[22] H. Yin and H. Wang. Building application-aware IPSec policy system. In *Proceedings of the* 14^{th} *USENIX Security Symposium*, August 2005.

[23] H. Yoshifuji, K. Miyazawa, Y. Sekiya, H. Esaki, and J. Murai. Linux IPv6 networking. In *Proceedings of the 2003 Ottawa Linux Symposium*, July 2003.

[24] X. Zhang, A. Edwards, and T. Jaeger. Using CQUAL for static analysis of authorization hook placement. In *Proceedings of the* 11^{th} *USENIX Security Symposium*, August 2002.