SELinux Protected Paths Revisited

Trent Jaeger
Pennsylvania State University
University Park, PA 16802 USA
tjaeger@cse.psu.edu

Abstract

We revisit the notion of achieving a protected communication path for applications connected via the Internet using SELinux. Last year, we discussed the mechanism for integrating IPsec with SELinux security labels, but we did not consider the system goals for using such labels. Toward this end, we revisit early SELinux proposals for what is called a protected path. A *protected path* is a secure communication path that has the same security guarantees as if the two ends are directly connected on a trusted platform and mutually authenticated. If a protected path can be constructed over the Internet in a reliable manner, then distributed applications can be as secure as two applications running on the same, isolated environment. A variety of challenges remain in building a protected path system, but interestingly, efforts are underway in most areas, with the notable exception of secure windowing systems. This talk will outline an approach to protected paths in the context of a distributed computing example, what work is underway toward achieving protected paths, and what is required of that work to compose protected paths with SELinux.

1 Introduction

SELinux-based systems are reconsidered based on some of the original goals of the project. We define a system goal based on a distributed computing example, and argue that such a goal can be achieved in the near future.

1.1 System Goal

Our goal is based on the example of a desired, network application scenario sketched in Figure 1. In this example, a banking server processes transactions from its customers. The bank has a server system that implements its transaction processing. The customer has a client system that it uses to request these transactions.

We note that this combined banking and customer system must satisfy a number of requirements to complete the transaction securely. Traditionally, the main requirement is secure network communication between the bank and customer. Such communication must enable mutual authentication and guarantee that the messages transmitted over the network are confidential and unmodified. These requirements are well-understood and implemented by mechanisms, such as SSL and IPsec. However, even with secure network communication, it is still problematic to ensure that the communication is really between the customer and the banking application. On each end host, attackers may be able to modify messages or leak secret keys used for authentication. These may be malicious or compromised versions of trusted programs or other programs that may have unauthorized access. For example, a recent wave of attacks involve "crimeware" that is capable of reading a user's keystrokes, even targeting specific applications. Further, it is necessary for each end host to be able to prove the effectiveness of a complete secure communication path to enable the opposite host to confidently permit access based on the authentication. Without this, the corresponding host has no basis to trust that the communication channel is really connected to the authenticated principal.

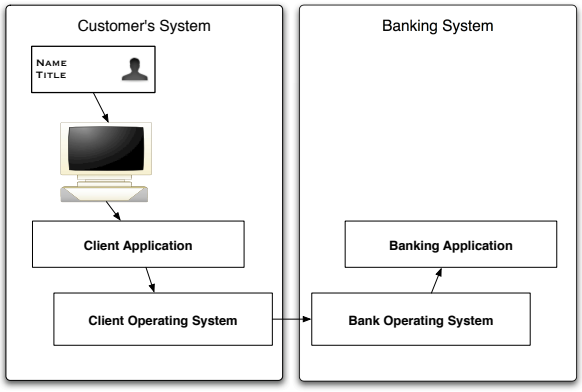

Figure 1: Simplified execution path of a networked, banking application

In general, we want to create a protected path between the customer and the banking application. A *protected path* provides secure communication guarantees of confidential and unmodified messaging across a mutually authenticated channel. In their paper that argues for SELinux-style operating system controls, Loscocco *et al*. [12] define the notion of a protected path and claim

that such an operating system is necessary to achieve a protected path. A protected path requires that the operating system control of all system information flow paths to prevent leakage of secrets (e.g., through unauthorized access to memory), prevent insertion of malicious programs, and protect the integrity of system processes. The SELinux system aims to achieve such goals. Further, it is necessary for each host to convey its security properties to others. For example, the customer must be able to convey the security of its end of the communication path before the bank can depend on it. Such function is emerging via network access control that conveys labels between SELinux hosts and attestation mechanisms that convey integrity status between hosts.

1.2 Background

As of Linux version 2.6.16, Linux will contain an extension to the Linux Security Modules (LSM) interface that will enable per-packet access control based on labeled IPsec security associations. Each IPsec security association (in either transport or tunnel mode) can be labeled, such that the LSM can authorize whether the packet can be transmitted or delivered. Extensions to SELinux enable it to use this mechanism to control which packets can be sent and delivered by which sockets.

Our question is whether SELinux, including the LSM-IPsec networking controls, is now capable of implementing a protected path. In doing this, we will define a system approach to implementing protected path, identify the additional functions required, and determine how to best implement these functions.

2 Approach

The key segments on the protected path in our example are: (1) user to client application; (2) client application to client OS; (3) client OS to bank OS; and (4) bank OS to banking application. Each segment in the protected path must achieve the protected path goals: direct interaction that is mutually authenticated. We discuss how these goals are relevant to SELinux below.

1. **User to Client Application**: This is the traditional trusted path problem. Communication from a user to an application is not direct. The user uses a device (e.g., mouse or keyboard) to communicate with the computer where the *Xserver* receives the input and passes the input on to the appropriate X client. Typically, a *window manager* is the base X client, so it also gains access to the input prior to it reaching the client application. From a trusted path perspective, the Xserver and window manager must be trusted not to modify or leak the user input. This is analogous to having a secure communication channel between the user and the application. The Xserver and window manager's abilities to modify the input is obvious. Further, the window manager and Xserver each support multiple subjects, so they must be capable of enforcing access policies (e.g., to prevent leakage) and passing unforgeable security information to others (e.g. user identities from the window manager to the Xserver). If the client application is shared among multiple programs (e.g., a browser enables execution of multiple scripts), then this application must have the same enforcement requirements as the window manager and Xserver.

   At present, hooks have been proposed for the Xserver [11], but a significant amount of effort remains for constructing a security-aware window manager and secure-aware middleware to support multiple client applications (see [6] as a possible direction). For applications that do not use GUIs, then only application-level issues need to be considered.

2. **Client Application to Client OS**: The client application can directly interact with its operating system. The operating system can reliably identify the application's process based on its label, and applications assume that the trusted system operating system responds to their system calls, so mutual authentication is also presumed. The main task of the operating system is to ensure the integrity of these computations. The SELinux system policy must protect the integrity of the client application, as well as the window manager and Xserver, and prevent information leakage from them through system channels (e.g., access to application memory).

3. **Client OS to Bank OS**: Between machines, network communication security, authorization of delivery to applications, and integrity measurement of the client are necessary. Network communication security is implemented in the operating system by IPsec. Recent integration of labeled IPsec security associations with SELinux enables client identification [8]. The IPsec negotiation includes negotiation of a label for the security associations (i.e., both ends use the same label). The operating systems on each end are entrusted to insure that use of all security associations for packet transmissions and deliveries is controlled using those labels. Finally, attestation mechanisms enable verification of whether remote systems are worthy of trust.

4. **Bank OS to Banking Application**: The banking application receives packets from its operating system. The question is the authenticity of the packet. Typically, application-level protocols, such as SSL, are used to guarantee authenticity at this level. Unfortunately, SSL does not make sufficient protected

path guarantees. It does not determine if either the integrity of the client or even the integrity of the application that it is running within are preserved. SELinux is used to obtain integrity protections on the individual hosts, but a means is needed to convey the integrity of the remote client to the banking application. Further, the banking application must be able to determine the identity of the client as well to control accesses based on the requests.

First, attestation mechanisms aim to prove the integrity of one system to another [16]. Such mechanisms typically provide load-time guarantees [4, 13, 15] that enable verification by the banking application that desired code is loaded. Limitations of this approach are that it does not detect potential compromises due to malicious modification of dynamic data (e.g., the data in a database) causing false negatives and that it requires all loaded code to be trusted even if the client application is isolated from this code causing false positives. Recent work aims to remedy these limitations by attesting information flow integrity guarantees, using SELinux policies [9].

Second, the use of IPsec aims to provide a protected path between machines, but a question is how applications can use IPsec to identify clients. First, each acceptable packet must use IPsec for network communication security. This requires correct configuration of IPsec policies. Second, the identity of the source of each packet must be delivered to the application. Typically, this is not part of IPsec, as only the IP address is provided – the use of IPsec is largely transparent to the application. Third, the granularity of IPsec security associations must be fine enough to enable distinction between client application identities. For example, if two clients that require different labels use the same host, then IPsec should label them differently. This is limited by the network flows as represented by Linux, so two clients using the same network flow will have the same label. Thus, different clients must use different flows. The finest granularity for flows is ports, so each client application would have to use a different port.

To summarize the requirements to fulfill for a protected path between the client application and the banking (server) application, we need:

- **Client Application**: Protect secrets and integrity of critical code, Xserver, window manager, application from other client applications and adversaries.

- **Client OS**: Protect the integrity of the client application and prevent the leakage of secret or private data to unauthorized principals. Also, must provide evidence in support of trustworthiness to achieve these goals for the bank OS. Provide secure network communication that labels the IPsec security association that may be used by the client application.

- **Bank OS**: Provide secure network communication and authenticate the client OS based on evidence provided.

- **Banking Application**: Define IPsec policies are sufficient for ensure labeled, protected network communication with all valid clients that is sufficient to distinguish client identity.

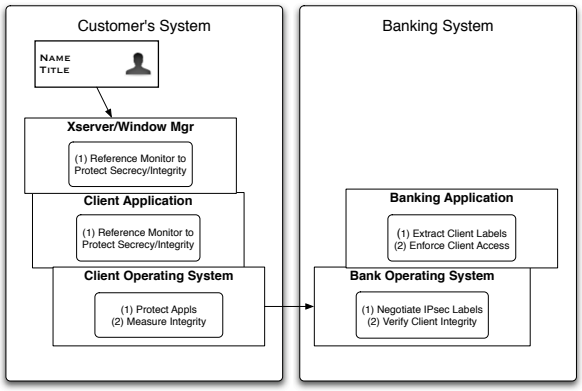

Figure 2: Protected Path System: Including security components for proposed protection

Since protected path requires mutual authentication, the same guarantees must be proven in the reverse direction as well.

3 SELinux Solutions

Figure 2 shows how the solution components proposed below compose a coherent protected path solution.

3.1 Client Application

The problem of ensuring a trusted path to the client application consists of: (1) defining reference monitor interfaces for the Xserver and window manager; (2) implementating such a reference monitor; and (3) specifying policies to control access to the Xserver and window manager objects. Work on two of these three areas is already underway.

First, a separate proposal examines the ability to generate reference monitor interfaces and their access requests from code and *idioms* describing conceptual operations [6]. While reference monitor generation is not conceptually complex, the task has proven time-consuming

and error-prone in practice. For example, bugs were found in the initial proposal for an LSM interface [17]. Since reference monitors will be needed for the Xserver, multiple window managers, and even some client applications themselves, the task will not be practical without some automated support.

Second, is the Tresys work defining an SELinux policy server [1]. It serves as a reference monitor for applications in the same way that the SELinux LSM implements the Linux kernel's reference monitor. The availability of an efficient, standard implementation for policy checking will also be necessary to make application-level access control broadly available.

Third, the development of application-level policies is naturally application-specific. To ease this process, specific policy metaphors are necessary. For example, we expect that such policies should start with secrecy (Bell-LaPadula) and integrity (Biba) information flow requirements initially. A runtime analysis can identify the areas where these security requirements are violated. For example, there may be untrusted user inputs that the application must filter to maintain its integrity.

3.2 Client OS

The client OS must protect the integrity and secrecy of the Xserver, window manager, and client application. Further, attestation mechanisms are necessary to prove the integrity of the client to the bank server.

Policy analysis tools, like Apol, SLAT, and Gokyo [10, 7, 2], can provide evidence that SELinux policies protect specific subjects secrecy and integrity. Evolving such tools so that they can become an effective part of the policy development process will be key. Our experience is that deeper Linux systems knowledge will also be necessary to make such tools valuable (e.g., by eliminating false positives). For example, sharing fifo read-write privileges does not enable information flows between two clients, but rather it enables the creation of two separate pipes.

The problem of proving trustworthiness of the OS is two-fold: (1) prove that the OS code is acceptable and (2) prove that the client OS truly protects the client application. First, integrity measurement techniques based on trusted hardware have emerged that can enable load-time code integrity attestations where a remote party can prove the code's integrity to itself [13]. The Linux IMA module implements load-time code measurement and has been submitted to the Linux community. Although the final implementation of the IMA approach for Linux has not been determined, a functional load-time code measurement infrastructure will emerge in the near future.

We expect that the ultimate integrity measurement infrastructure will be different than IMA in that greater accuracy will be required. Information flow policy provides runtime information that enables improved accuracy. The PRIMA extension [9] to Linux IMA enable verification of an information flow guarantee called Clark-Wilson Lite [14]. Clark-Wilson Lite requires that high integrity programs discard or upgrade any low integrity inputs, as Clark-Wilson [3] requires, but only for a limited set of interfaces and without the requirements for complete, formal assurance.

3.3 Bank OS

The bank OS must verify attestation of the client's integrity to justify the use of IPsec labels and implement security communication using those IPsec labels. SELinux uses the IPsec-LSM extensions that will be widely available for Linux 2.6.16, described previously [8]. IPsec policies are defined such that when the client's request is made (i.e., a packet is to be sent to the bank system), SELinux can authorize the IPsec security association and label that the client application can use. For example, if the client is a preferred customer, a *gold* label may be authorized for the client.

The combination of client attestation and a certificate previously issued for the client by the bank CA enables the client to connect using a security association with this label. The addition of attestation to the IPsec negotiation (IKE) is future work. The verification of integrity measurements will probably be done by a service outside the operating system, much as user authentication is done now. The exact mechanism will depend on the integrity measurement function chosen. We expect that remote integrity verification will have to be simplified significantly by the requirement of privacy: it will not be acceptable to publish the identity of all the code running on a system. Likely, a few basic services will be measured (e.g., operating system, bootloader, BIOS, and local integrity verifier) along with the information flows among them.

3.4 Banking Application

Finally, the banking application must be able to leverage the IPsec labels used to authorize communication by the bank OS. Since the bank application will determine the access rights for the request, it also needs the labeling information to do its access control.

For servers using TCP sockets, the label of a remote peer (i.e., the client application) can be extracted using *getsockopt*. In a manner that is analogous to determining the peer of a UNIX domain socket, the application can request a *SO_PEERSEC* option. A proof-of-concept for TCP leverages the sock's *sk_dst_cache* to retrieve the security association labels for connected sockets.

For connection-less UDP, the question is whether the operating system enables an application to determine the

security label of its most recent input. Setting socket options *SOL_IP* and *IP_PASSSEC* via *setsockopt* results in the label being present in an ancillary message of type *SCM_SECURITY*.

Using the client label, the banking application must be able to control access for the client's request. Again, a reference monitor function is needed within the banking application to enforce this control.

4 Explorations

The labeled IPsec approach has been applied to a small number of prototype systems.

- **Label-aware Servers**: The banking application would be a label-aware server. We have extended *vsftpd* and *inetd* to use the IPsec labels for the subprocesses that they fork. Each server accepts connections from remote users that result in the creation of a new process for that user's request. The use of IPsec labels can associate labels with these new processes based on the label negotiated.

- **Label-aware Storage**: A somewhat different problem is that of controlling access to network-attached storage. Such storage uses protocols, such as iSCSI, that can use IP packets as the medium for storage requests. The storage system can authorize the IPsec security association to control access to storage.

- **Virtual Machines**: Recent work explores the use of labeled IPsec to build *coalitions* of isolated virtual machines [5]. Since virtual machines communicate via network APIs, the use of labeled IPsec is also natural here. A set of distributed virtual machines may communicate based on the access policy that they all share. Labeled IPsec negotiates the labels between the attested virtual machine monitors (VMMs), such that the VMMs can control access between virtual machines using SELinux.

5 Conclusions

In this paper, we discussed the efficacy of establishing a protected path over the Internet using SELinux and labeled IPsec. A protected path is a path that has the same security guarantees as if the two ends are directly connected and supports mutual authentication. Such a protected path was identified as a key function of secure operating systems, so we want to examine how far we are from such a goal. It turns out that a variety of functions including application-level reference monitor construction, policy design, policy analysis, integrity measurement, and labeled security communication are necessary, but significant efforts are underway in each area. Practical combination of existing solutions into a total package is necessary to achieve the goal of protected path.

References

[1] SELinux Policy Server. Tresys Corp., 2005. Available at www.tresys.com/selinux/.

[2] SETools policy tools for SELinux. Tresys Corp., 2005. Available at www.tresys.com/selinux/.

[3] CLARK, D. D., AND WILSON, D. R. A comparison of commercial and military computer security policies. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy* (1987).

[4] ENGLAND, P., LAMPSON, B. W., MANFERDELLI, J., PEINADO, M., AND WILLMAN, B. A trusted open platform. *IEEE Computer 36*, 7 (2003), 55–62.

[5] J. MCCUNE *et al*. Bridging mandatory access control across machines. Tech. Rep. RC23778, IBM Research, 2005.

[6] GANAPATHY, V., JAEGER, T., AND JHA, S. Automatic placement of authorization hooks in the Linux Security Modules framework. In *Proceedings of the ACM Conference on Computer and Communications Security* (Nov. 2005).

[7] HERZOG, A., AND GUTTMAN, J. Achieving security goals with Security-Enhanced Linux. MITRE, 2002. Available at www.mitre.org/tech/selinux/.

[8] JAEGER, T., HALLYN, S., AND LATTEN, J. Leveraging IPSec for mandatory access control of Linux network communications. Tech. Rep. RC23642, IBM Research, 2005.

[9] JAEGER, T., SAILER, R., AND SHANKAR, U. PRIMA: Policy-Reduced Integrity Measurement Architecture, Jan. 2006. In submission.

[10] JAEGER, T., ZHANG, X., AND EDWARDS, A. Policy management using access control spaces. *ACM Transactions on Information and System Security 6*, 3 (2003), 327–364.

[11] KILPATRICK, D., SALAMON, W., AND VANCE, C. Securing the X Window System with SELinux. NAI Labs, 2003. Available at www.nsa.gov/selinux/papers/.

[12] LOSCOCCO, P., SMALLEY, S., MUCKELBAUER, P., TAYLOR, R., TURNER, S. J., AND FARRELL, J. F. The inevitability of failure: The flawed assumption of security in modern operating systems. In *Proceedings of the National Information Systems Security Conference* (1998), pp. 303–314.

[13] SAILER, R., ZHANG, X., JAEGER, T., AND VAN DOORN, L. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the USENIX Security Symposium* (2004), pp. 223–238.

[14] SHANKAR, U., JAEGER, T., AND SAILER, R. Toward automated information-flow integrity for security-critical applications. In *Proceedings of the* 13^{th} *Annual Network and Distributed Systems Security Symposium* (2006), Internet Society.

[15] SHI, E., PERRIG, A., AND VAN DOORN, L. Bind: A fine-grained attestation service for secure distributed systems. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy* (2005), pp. 154–168.

[16] SMITH, S. W. Outbound authentication for programmable secure coprocessors. In *ESORICS '02: Proceedings of the 7th European Symposium on Research in Computer Security* (London, UK, 2002), Springer-Verlag, pp. 72–89.

[17] ZHANG, X., EDWARDS, A., AND JAEGER, T. Using CQUAL for static analysis of authorization hook placement. In *Proceedings of the 11th USENIX Security Symposium* (2002).