The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

PORT AUTHORITY: INTEGRITY VERIFICATION THROUGH CONNECTION MANAGEMENT

A Thesis in

Computer Science and Engineering

by

Christopher George Shal

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2009

The thesis of Christopher George Shal was reviewed and approved* by the following:

Trent Jaeger
Associate Professor of Computer Science and Engineering
Thesis Adviser

Patrick McDaniel
Associate Professor of Computer Science and Engineering

Raj Acharya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.

ABSTRACT

Distributed systems have become a standard method of operation for Internet-based services. While there exist methods to prove the authenticity and secure operation of a single remote party, it is not currently possible to efficiently obtain these same guarantees of a distributed system. Thus, in this work we present PortAuthority: a service which carefully manages connections and only authorizes input data from provably secure remote parties. To examine its effectiveness, we compare the performance of several different systems with and without the PortAuthority service. We show that PortAuthority is able to provide these guarantees without a significant overhead and independent of the communicating applications, with the only noticeable delay occurring at the initial connection setup.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank Joshua Schiffman and Thomas Moyer for their advice and constant willingness to answer questions; my adviser, Dr. Trent Jaeger, for his guidance throughout my research process; and Dr. Patrick McDaniel for his never-ending enthusiasm.

Chapter 1

Overview

1.1 Introduction

In the past several years, the Internet has increasingly become based on distributed systems rather than single servers. Virtualization and the performance and cost advantages of multiple commodity systems over a single super computer have hastened this change. In fact, most Internet and enterprise services today require interaction and coordination among multiple machines, each serving a potentially different purpose.

Take, for example, a simple distributed web application [8, 9]: a client may be directly interacting with a web server, but that web server in turn may obtain dynamic content from a series of database servers. Those database servers may be sending and receiving content from other machines, such as other web servers or management consoles. In trusting the main web server, a user is implicitly trusting all of these other machines as well - whether or not the user even knows of their existence.

While authentication protocols exist, such as TLS/SSL [10], the problem of verifying the underlying platform still exists. That is, a user may want to ensure that the received data is correct, not just that it came from the expected machine. The next step has been to use hardware-based modules for trusted computing, such at the Trusted Platform Module (TPM) [29]. These hardware-based attestations cryptographically bind the code on a machine to the underlying physical platform, allowing remote parties to

verify the boot-time integrity of a machine. However, these mechanisms have not scaled well to distributed environments and are unable to provide any guarantees about the applications actively running on the system.

In this work, we propose the *PortAuthority* service as a method for handling trusted connections. By using a service that can request and verify a hardware-based attestation of a remote party, PortAuthority is able to ensure that integrity-sensitive content is only received from verified parties. That is, PortAuthority provides mediation of network operations; it serves as a component in a system model that requires the mediation of all integrity relevant operations. As a result, PortAuthority provides the ability for remote parties to analyze the communication standards enforced by the system. In addition, PortAuthority uses message passing to minimize the amount of overhead required for maintaining integrity across a distributed system.

1.2 Problem

Consider a bank that allows online transactions, as most do. This online application will be interacting with other systems within the bank's corporation, as well as systems in other locations. For example, a user may request a transfer to another bank, or may pay a credit card bill directly from the bank's website, as illustrated in Figure 1.1. The user would like to ensure that all systems involved in this process are secure, since he will be using private bank account information.

However, the user does not (and perhaps should not) necessarily know all of the different systems that are involved in the transaction. For example, the web application

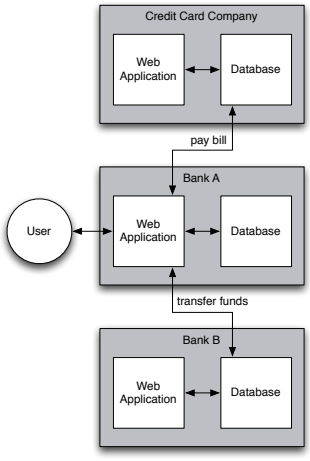

Fig. 1.1. Sample problem situation: how can the user trust all of the systems involved in a bank transaction?

may interact with a backend database. However, this database will likely not be accessible to the public for security purposes - it has no reason to be Internet-facing. Still, the user would like to ensure that the database is going to handle its information safely.

As a result, it becomes necessary to have a method of trusting all systems used in a distributed system from a single point - in this case, the user would like to be able to trust all systems used in the transaction based on his trust of the bank's web application server. However, at the same time, the method of proving trust to the user must not be too slow: the user should not need to repeat the whole verification process for each transaction, especially considering that the user may perform mutiple transactions at one time.

Likewise, from the bank's point of view, the bank should not have to recreate these proofs for each user on the web application: there may be thosands of simultaneous users.

At the same time, however, the bank would not like to continually verify remote systems that are not used very often, such as if a particular credit card company is only rarely accessed.

It then becomes clear that some sort of connection and proof management service is necessary in addition to a method of generating a cryptographic proof of a distributed system from a single system within it. While the proof generating system is handled in other work, such as the Virtual Machine Verifier (VMV) system [25], we aim to solve the problem of connection and proof management.

1.3 Contribution

It is with the VMV system described in Chapter 2 that we aim to further enhance system integrity via connection protection and enhance performance via smart connection management. Our contribution to this field is the PortAuthority service: a method of handling trusted connections in a distributed system environment. It is intended for integration with the VMV system archiecture. We consider PortAuthority a necessary improvement for scalability, due to the slow nature of each full system attestation.

1.4 Conclusion

We have proposed the PortAuthority service for managing trusted communications. In Chapter 2 we examine work related to integrity verification. In Chapter 3 we examine the design decisions behind the PortAuthority service. In Chapter 4 we explain how this design was implemented. In Chapter 5 we evaluate the performance of our implementation. Finally, a summary is provided in Chapter 6.

Chapter 2

Related Work

The notion of trust in a networked environment has several different avenues of research. Transport security mechanisms, such as IPsec [17, 12] and TLS/SSL [10], are able to provide network access control. However, they are unable to provide any guarantee about the secure creation of the content itself. Specific web access control mechanisms exist in web servers such as Apache, but these are specific to particular applications and are still unable to provide guarantees about the context in which content was created.

Research focus in establishing trust in a distributed system environment can be labeled in several different categories based on the proof they attempt to establish. For example, there are certificate-based systems for establishing the identity of machines in a network [4, 13, 5, 18]. Though successful in this regard, certificate-based approaches fail to verify that the underlying systems of these machines are performing as desired. For establishing trust in the secure operation of distributed systems, there has been work (e.g. CRISIS and Taos) in combining authentication, access control, and auditing [2, 15, 30, 1]. However these systems still fail to establish the integrity of each underlying system. That is, trust in distributed systems cannot be reduced to mere authentication, since there is still a concern about how data is handled in the network.

Current work in verifying the integrity of a remote system involves integrity measurement mechanisms [24, 16, 26, 14, 28, 20, 27, 23, 21], which typically involve using a cryptographic co-processor such as Trusted Computing Group's Trusted Platform Module (TPM) [29]. By using a hardware-based attestation mechanism, the proof of the system's integrity is cryptographically bound to the physical platform. The downside of these approaches is that a hardware-based attestation is time-consuming, taking about one second per proof generation. Thus, to be usable in practical applications, researchers must create a more intelligent approach to the use of these attestations.

Once the integrity measurement is developed, there is still the problem of having a remote party verify this measurement. The verification process varies dependent upon the particular integrity measurement system being used, but typically involves comparing the measurement against an expected value. Depending on the goal, successful validation of the measurement either allows further excution of the system or confirms trust in the execution of a remote party.

The next concern is extending these proofs of integrity of an individual system to that of a distributed system. Although the hardware-based attestation mechanisms can remotely prove the integrity of a system, there is still a concern that integrity-senstive information is flowing to or from other unverified systems. Thus, to extend this solution to distributed systems, there is the concept of shared reference monitors such as Shamon [22]. Shamon uses a network of mutually attesting reference monitors to establish a proof of the whole distributed system.

However, another problem with many of these hardware-based attestation mechanisms (including Shamon) is that they solely focus on boot-time code measurements, and

fail to provide any guarantee regarding the runtime applications of the system. Users of the system would like to prove that the system was of high integrity *at the time* it generated integrity-sensitive data. Thus, work continues with the concept of an Integrity Witness and Virtual Machine Verifier (VMV) [25] to monitor integrity-sensitive operations and provide integrity proofs of a machine at a particular moment in time. Part of this requirement includes extending the concept of criteria beyond simply code measurements to include additional information about the system at that particular point in time, such as configuration options for integrity-sensitive applications. This system also relies on the creation of a Root of Trust for Installation (ROTI) to trace the creation of a machine back to a trusted installation media [6]. By using a static filesystem, this verification back to the installation can be performed more easily - the underlying code base of the minimal static system should remain the same. Our PortAuthority service is built into the VMV architecture, which uses the ROTI concept to establish trust in a system.

Chapter 3

Design

3.1 System Architecture

PortAuthority is intended to perform as a service on an architecture that provides system attestations as a measure of integrity. In particular, PortAuthority is installed on the virtual machines of a system constructed from a Shamon Core, or sCore [6]. An sCore contains a Xen hypervisor and Dom0 virtual machine, with the intention of being a small static system that is easy to verify. One of the goals of this system is to be able to tie the hash of its filesystem back to the installation, establishing a root of trust.

In addition, our installation of the sCore contains a Virtual Machine Verifier (VMV), which contains an Integrity Witness [25]. This component operates on the sCore to monitor all integrity-sensitive operations. It also generates an Integrity Enforcer on each installed virtual machine, and an Integrity Verifier to moderate all incoming connections. The Integrity Enforcer uses a modified SELinux LSM [19] to enforce PRIMA measurements [16]. PortAuthority is designed to fill this role as the Integrity Verifier.

Finally, an attestation daemon exists on both the sCore and the virtual machines to create and validate system attestations. This component was also generated as part of the VMV suite of services [25], and we utilize this tool in PortAuthority without further modification.

The criteria that are used in the creation and veification of attestations varies. The attestation daemon may wish to examine code hashes as well as configuration options to determine the security model enforced by a remote machine. The actual creation and verification of these attestations is in the domain of the attestation daemon and the VMV suite, and we will therefore not further explain that process here [25]. The important result is that these attestations are used to determine the approximate security model employed by the remote system, such as Biba [3] or Clark-Wilson [7].

We illustrate our system archiecture in Figure 3.1. In this example, we show two virtual machines on one sCore, each of which may be using different applications. They both contain a PortAuthority service for mediating the incoming connections. We also see communication between PortAuthority on VM 1 and a remote machine, a process which utilizes the attestation daemon to request and validate the attestation of that remote machine.

3.2 Goals

When designing the PortAuthority system, we had several goals in mind. These goals relate to its implementation in the system architecture as described in Section 3.1. We will proceed by describing in detail each of the design goals for PortAuthority.

1. **Mediation of High Integrity Network Communications**: Protecting the communication channels of a system is an important factor in maintaining its integrity state. As a connection manager, the PortAuthority service must be able

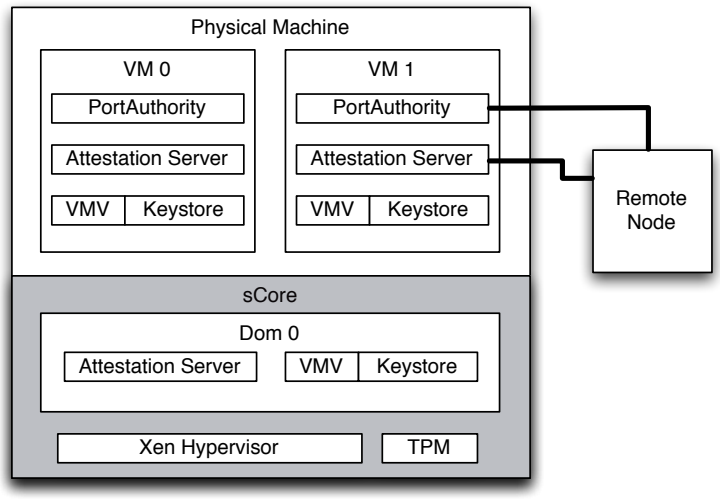

Fig. 3.1. System Architecture

to regulate integrity-sensitive inputs to the system. Integrity-sensitive inputs include those which have the potential to affect the integrity of data on the system itself. That is, while certain communication channels may not pose a threat to the integrity of the system, others require careful monitoring.

2. **Minimizing Performance Impact**: Performance is a crucial factor with any service, since the public is not likely to accept any solution that is too slow or unwieldy. A user should be able to use the system without any noticeable change in network or processor overhead.

3. **Easy to Overlay on Existing Systems and Applications**: Users and application developers also insist on easy integration with existing software. Ideally, existing applications should not break or need to be modified heavily to see the

benefits of the system, and the users should not have to be concerned with a complex setup or configuration procedure.

3.3 PortAuthority Design

We will now describe in detail our design of the PortAuthority service. For organization, we will examine the design decisions as they relate to the goals described in Section 3.2. In addition, for each design decision, we will identify the necessary protocols that we will implement in Chapter 4.

3.3.1 Mediation of High Integrity Network Communications

Protecting the data integrity of a system requires protecting the integrity-sensitive inputs. Towards this end, we recognize several actions that are necessary to protect these inputs, which we enumerate in the following sections.

3.3.1.1 Blocking Untrusted Systems

By directly blocking access of untrusted systems to integrity-sensitive ports, which we call *trusted ports*, PortAuthority is able to ensure that only trusted systems are able to modify data on the system. Thus the system is able to maintain its integrity state. This blocking needs to occur outside of applications, such that the applications do not need to be modified or updated to work with PortAuthority. However, ports that can be accessed without fear of affecting the data integrity of the system can be explicitly defined. This functionality allows the administrator to decide which ports, which we call *untrusted ports*, are sufficiently protected to allow untrusted access.

For example, a webserver is likely to allow an unverified client to request and receive a page, despite not knowing anything about the state of the client's system. Since this action does not affect the data on the webserver, it is not likely to affect the system's integrity. Conversely, a database that allows a remote system to perform database operations is at risk, since these database operations can directly affect the integrity of the database itself. Thus, such a communication channel would need to ensure that the received data is of high integrity.

We recognize the following protocols that will need to be implemented:

- **Initialize Blocking**: This function will configure and start the input blocking mechanism. Since it forms the foundation of the input regulation, it should be called immediately when PortAuthority is started, which should occur at boot.
- **Specify Untrusted Ports**: This function will allow the administrator of the system to decide which ports can be accessed from both trusted and untrusted systems. The applications that use the ports will thus need to be trusted to handle any input safely.

3.3.1.2 Authentication of a System

After establishing the trust of a remote system that seeks access to trusted ports, PortAuthority needs to verify the remote system's identity. PortAuthority is able to perform this authentication in a well-known manner. Once again, this authentication process should occur independent of which applications are being used, and should not require modification of existing applications.

We recognize the following protocols that will need to be implemented:

- **Set Initial Known Systems**: This function will specify which remote systems are initially known to our system. Simply being on this list will not grant the remote system access to trusted ports, since that system will first need to be attested, but it will enable the remote system to connect if it is successfully attested.

- **Authenticate Known System**: This function will perform the authentication of a remote system that has been seen before. Since PortAuthority will already have some knowledge about this system, the authentication process is more straightforward than for unknown systems.

- **Authenticate New System**: This function will perform the authentication of a previously-unkown remote system. PortAuthority will need to prove its identity, since it is not a system that was seen before. PortAuthority will also maintain some piece of knowledge about that system in order to verify its identity in the future. Once again, this should not be confused with the attestation process - authentication merely identifies the remote system, but does not necessarily grant it access to trusted ports.

3.3.1.3 Determining Trust of a System

Regardless of determining their identity, PortAuthority denies access to remote machines that are not sufficiently trusted. Thus, PortAuthority provides a method to detemine the trust level of a remote machine.

We recognize the following protocol that will need to be implemented:

- **Perform System Attestation**: This function will verify whether or not a remote machine is sufficiently trusted. PortAuthority will use this information to decide whether or not this remote machine can access trusted ports.

3.3.1.4 Maintaining Trust of a System

Simply attesting a remote system at connection-time does not allow us to verify that this system will remain one of high integrity. Thus, PortAuthority performs a periodic rechecking of each remote system that is connected to the trusted ports. The goal of these periodic rechecks is to ensure that the integrity-sensitive inputs are safe from machines that become untrusted. In addition, PortAuthority detects when a connected peer suddenly stops responding, since this situation may imply a problem with the remote machine. Additionally, this dead peer detection provides PortAuthority with protection against reset attacks, which occur when a machine is able to reboot into an untrusted state while maintaining an active connection.

We recognize the following protocols that will need to be implemented:

- **Recheck Procedure**: This procedure will need to be performed periodically, and will recheck each connected system. The administrator will be able to specify how often the recheck procedure is executed and how stale a verification may become before the remote system is no longer trusted.

- **Dead Peer Detection**: This functionality will alert PortAuthority whenever a connected peer goes dead or otherwise stops responding. PortAuthority will then destroy and clean up the connection.

3.3.2 Minimizing Performance Impact

Minimizing the performance impact of PortAuthority requires reducing the effect of the operations which show the most overhead. As described in Chapter 2, the attestation process is the slowest individual operation. This operation is on the critical path to establishing a trusted link, and trusted applications are unable to communicate until this link is established. As a result, avoiding an attestation when possible is crucial for a quick response time, which is necessary in any user-based or time-critical application. Additionally, the network bandwidth required for passing attestations is larger than for any other operation, since it requires sending code measurements and configuration options. Thus, many of the PortAuthority performance improvements aim to reduce the number of attestations required across the distributed system. We enumerate the steps taken to enhance the performance in the following sections.

3.3.2.1 Reduce Number of Directly Attested Links

In order to reduce the number of directly attested links, some amount of information must be communicated among the nodes in the distributed system. Take, for example, Figures 3.2 and 3.3. In Figure 3.2, we see that all nodes are mutually attesting each other, where the directed edges indicate trusted communication flow.

However, indirect attestations, as seen in Figure 3.3, can reduce the number of attestations required. In this example, Machine B has attested Machine C, and thus there is a trusted communication flow from Machine C to Machine B. Likewise, there is a trusted communication flow from Machine B to Machine A. As a result, there is an indirect trusted communication flow from Machine C to Machine A, as indicated by the

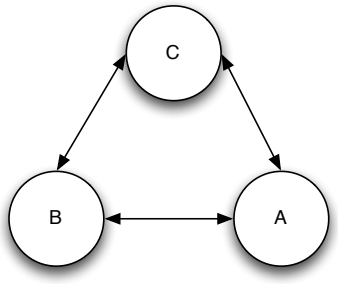

Fig. 3.2. A view of three nodes in which each is directly attesting each other, as indicated by the solid lines. Each directed edge indicates a trusted information flow.

dotted line. In this case, Machine A is able to implicitly trust Machine C as a result of his trusting of Machine B.

This form of indirect or transitive trust is possible through the attestation mechanism. The criteria to be verified contains information about what a system trusts, in addition to what that system itself enforces. Thus, successful verification of a system's criteria implies trusting that system's enforcement of its own connections. As a result, it is possible to trust any machines that are verified by a trusted party - in effect, we are trusting a remote machine to correctly verify new machines for us, which is checked and enforced through the criteria itself.

We recognize the following protocols that will need to be implemented:

- **Add Link**: This function will cause PortAuthority to initialize an active connection with a remote machine. PortAuthority will first need to attest this machine, if necessary. If successful, this remote machine will be able to access trusted ports

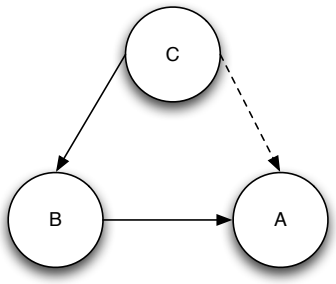

Fig. 3.3. A view of three nodes in which an indirect attestation has taken place. In this case, Machine A has indirectly attested Machine C through Machine B, as indicated by the dotted line. Directed edges indicate trusted information flows.

on the system, and PortAuthority will proceed to recheck the remote machine as necessary via the Recheck Procedure.

- **Remove Link**: This function serves as the inverse operation to Add Link. It will simply cause PortAuthority to drop the active connection with a specified remote machine.

- **Request Update**: This function will request information about trusted links from a remote party. It is through this function that PortAuthority receives information about trusted machines that were not directly attested.

3.3.2.2 Stop Attesting When Possible

Another way of reducing the number of directly attested links is to cease attesting links when possible. For example, if a remote machine requests a connection in order to send integrity-sensitive data, PortAuthority will first require a successful attestation of that machine and will then proceed to recheck that machine periodically. However, once

that machine is no longer sending data on the trusted ports, PortAuthority no longer needs to continue to recheck it.

We recognize the following protocol that will need to be implemented:

- **Check for Idleness**: The function will be executed periodically and will check whether or not all connected remote machines are still using the trusted ports. If not, PortAuthority will remove them via the Remove Link functionality.

3.3.3 Easy Overlay with Existing Systems and Applications

In order to easily integrate with existing systems and applications, PortAuthority needs to be able to automatically perform its necessary operations independent of the type of application. Ideally, applications will not need to be modified to work with the benefits of PortAuthority, and the users and/or system administrators will not need to perform a large amount of configuration ahead of time. Thus, PortAuthority provides several mechanisms to ease its integration with current systems, which we enumerate in the following sections.

3.3.3.1 Automatic Link Generation

Creation of attested links should happen only when necessary, and without explicit application or user request in order to be easy to use. As a result, PortAuthority provides a mechanism which, once initially configured with the list of trusted ports, will automatically detect an attempt to reach a trusted port and will establish a trusted link with that remote party if it is successfully attested. This automatic connection will

occur without modifications to the applications using these ports, and will not require user intervention.

We recognize the following protocols that will need to be implemented:

- **Automatic Connection Configuration**: With this function, the administrator will be able to specify which ports are trusted and should signal PortAuthority to attempt to create a trusted connection automatically.
- **Automatic Connection Initialization**: This function will cause PortAuthority to initialize the automatic connection process. Once initalizated, any connections to ports specified in the configuration will cause PortAuthority to attempt to generate a trusted link.

3.3.3.2 Configuration Options

For easy modification by the administrator, PortAuthority provides multiple configuration options. For example, PortAuthority will allow the administrator to specify parameters regarding time until attestation rechecks, dead peer detection, and the list of trusted ports.

We recognize the following protocol that will need to be implemented:

- **Set Parameter**: This function will simply allow the administrator to specify a configuration option for PortAuthority.

3.4 Conclusion

We have described the system architecture around the PortAuthority service, the design goals for the service, and the different protocols that will need to be implemented. We will now proceed to describe the implementation of these protocols in Chaper 4.

Chapter 4

Implementation

4.1 PortAuthority Overview

We implemented PortAuthority as a series of functions and daemons written in Python. We also utilized other technologies, most importantly iptables and Openswan IPsec [31], to help with the connection management.

Before we introduce the specific functions and daemons we built, we will first describe the primary data structures used for connection management. The PortAuthority service on each machine maintains two global tables: the *Active Connections Table* and the *Propagation Table*.

The Active Connections Table maintains the list of all remote machines that are currently connected to trusted ports on the local machine. By rule, these machines must have passed verification and have been successfully authenticated. Since these are the machines that have the authority to supply integrity-sensitive data, these are the machines that PortAuthority needs to regularly monitor to ensure they remain of high integrity.

The Propagation Table, on the other hand, contains the list of all verified machines, whether or not they are actively connected to the trusted ports on the current machine. This table is used by PortAuthority to potentially determine the integrity state of a connecting machine without directly performing an attestation and verification. The

table contains the verified machines, the machine that performed the verification, and the time that the verification took place (which PortAuthority uses to determine whether or not the recorded attestation is sufficiently fresh).

The actual usage of these tables will be described in detail in the protocol implementations in Section 4.2. However, consider the following example of how the tables are used:

Let Machine A request an attestation of Machine B, which is successfully verified. Machine A adds machine B to its Propagation Table, with itself listed as the verifier. Machine A proceeds to pull Machine B's Propagation Table, which it implicity trusts since it just verified Machine B. Machine A merges Machine B's Propagation Table with its own. Say Machine B's Propagation Table includes Machine C. Now if Machine A attempts to connect with Machine C, it will find Machine C in its local Propagation Table with Machine B as the verifier. Assuming that that particular attestation is still fresh (checking the time in the table), Machine A will be able to connect to Machine C without first performing a direct attestation of Machine C. However, since Machine A did not directly attest Machine C, it will not pull Machine C's Propagation Table in the process - which is desirable, since anything in Machine C's Propagation Table is necessarily in Machine B's, since Machine B verified Machine C. This example is exactly the situation pictured in Figure 3.3.

4.2 Protocol Implementation

The implementation of each protocol described in Chapter 3 provided us with some challenges. We will detail these challenges and solutions in the following sections, ordered by protocol.

4.2.1 Initialize Blocking

One of the basic requirements of PortAuthority is that it protects the integrity-sensitive applications from unauthorized access. PortAuthority is preconfigured with the list of integrity-sensitive ports, or trusted ports (see Section 4.2.15). PortAuthority performs this initial layer of protection via iptables, which works without modification of the applications: all incoming traffic is subjected to the iptables firewall before being forwarded to the applications themselves. Due to this implementation decision, all trusted communications to and from trusted processes must occur via Internet Protocol (IP).

PortAuthority initializes the iptables rules immediately when the service is started. Certain ports that are known not to affect the data integrity of the machine are given full access - that is, a remote machine can connect to these ports without being verified. These ports include the PortAuthority daemon itself and the Attestation Daemon, since untrused remote machines will need to access these daemons to become verified initially (see Section 4.2.9). In addition, multicast DNS for network setup and the IKE daemon for IPsec are allowed: both of these need to be used prior to the establishment of trusted

links. In all of these cases, the open applications do not affect data on the system; thus, the integrity of the system is able to be maintained even with access to remote parties.

The remainder of the ports are blocked, aside from the list of trusted ports - these ports can still be accessed, but only by verified systems. As will be described in Section 4.2.4, IPsec is used to perform the authentication of verified systems - our system will only create an IPsec tunnel with a system that is first verified. Thus, we can restrict access to trusted ports as desired simply by only allowing IPsec traffic to these ports, and we have built our iptables rules accordingly.

4.2.2 Specify Untrusted Ports

In addition to the ports that are automatically accessible to both verified and unverified remote parties, the administrator can specify additional ports to operate in this fashion. These untrusted ports should only be used for applications that cannot affect the integrity of the system itself.

For example, a web server would likely want to allow clients to retrieve webpages without first verifying each client, since retrieval of webpages should not affect the integrity of the web server. Thus the web server administrator would likely define HTTP port 80 to be untrusted. Then it is up to the clients to decide whether or not they will require an attestation of the web server - regardless of their decision, the web server will not require an attestation of the client in return.

4.2.3 Set Initial Known Systems

As will be described in Section 4.2.4, we will be using IPsec to perform authentication of systems. However, in order to perform this authentication, some previous knowledge of the connecting system is necessary. We use certificates to serve this function. Each virtual machine, at the time it is created, will be given a certificate signed by underlying sCore for that virtual machine.

Thus, this initialization process involves obtaining the certificates of all the virtual machines that we wish to know initially, as well as any necessary sCore root certificates. We obtain these certificates from a trusted source and place them in the appropriate directories for use by Openswan IPsec. For testing purposes, we simply copied these certificates to different systems as necessary; however, in practice, a public key infrastructure would be necessary.

4.2.4 Authenticate Known System

Authentication in our setup occurs via establishment of IPsec tunnels with shared public keys, or certificates. Note that this authentication process is separate from trusting the remote party's underlying system, which occurs prior to the establishment of the IPsec tunnel (see Section 4.2.9). Using these shared certificates is a well-known method of establishing the identity of nodes in the distributed system [5, 18, 13].

For this process, we assume that we already have the certificate of the connecting machine: we may have retrieved this certificate either by the initial setup, described in Section 4.2.3, or by obtaining the certificate of a new system, described in Section 4.2.5.

Having these certificates allows us to cryptographically prove the identity of the connecting system. We then configure Openswan IPsec to create a tunnel secured with null-encrypted ESP. Our choice to use null encryption was due to our reason for using IPsec: we only want to authenticate the source, not encrypt the traffic between the systems. Since all application traffic will pass through the IPsec tunnel once it is established, we did not want to negatively affect performance through encryption. If an application uses its own form of encryption, it will proceed uninhibited and simply pass through the IPsec tunnel without being doubly encrypted. However, if the application chooses not to encrypt transmitted data, we do not wish to enforce this decision upon it.

4.2.5 Authenticate New System

Authenticating a new system provides an additional challenge beyond that of authenticating a known system: we need to obtain the certificate of this new system. If we already have (and trust) the root certificate of the underlying sCore of this new system, the certificate can merely be transmitted over the network - since it is signed by the private key of the sCore, that sCore vouches for the identity of the system.

The situation is more difficult when the new system is contained on an unknown sCore. In this case, we first need to obtain the sCore's certificate in a secure fashion. Since this new sCore's certificate would also need to be stored on our own sCore, which does not contain the PortAuthority service, we leave this process to the sCore to handle.

Once the certificate of the new system and its sCore (if necessary) are installed in the appropriate locations, the authentication process proceeds exactly as described for known systems in Section 4.2.4.

4.2.6 Perform System Attestation

When PortAuthority needs to request a system attestation, it first checks whether or not that system needs to be verified. The administrator is able to specify specific systems that do not need attestations via the PortAuthority configuration (see Section 4.2.15).

If the system does need to be verified, PortAuthority then checks the Propagation Table to determine if a fresh verification already exists. PortAuthority is thus able to reduce the number of actual attestations by using prior knowledge of verified systems. However, if no fresh verification exists in the Propagation Table, then PortAuthority will need to request a new one.

PortAuthority does not directly perform the verification of remote systems. However, it interfaces with the Attestation Daemon to make requests to perform system attestations. The Attestation Daemon requests the attestation from the remote system, and performs the verification of the criteria to determine whether or not the remote system is trusted. The result of this verification process is returned to PortAuthority, which then determines whether or not to allow the connection. If the verification was successful, PortAuthority will proceed to request an update of propagation information, as described in Section 4.2.11.

This system verification process is reiterated in the Figure 4.1.

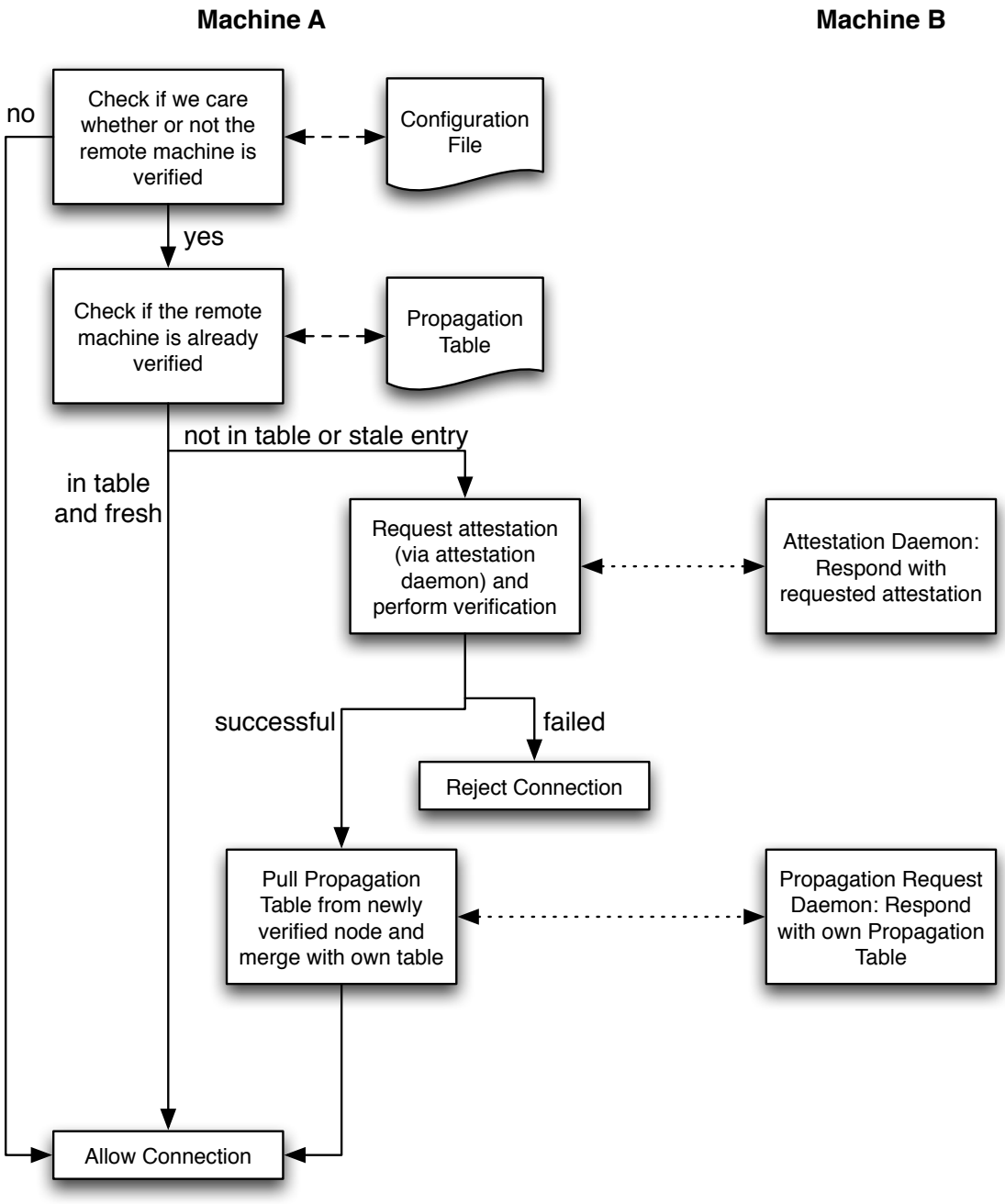

Fig. 4.1. System Verification Procedure

4.2.7 Recheck Procedure

The recheck procedure occurs at a regular time interval, which can be specified in the configuration options (see Section 4.2.15). Rechecking remote machines is a multi-step process, which is illustrated in Figure 4.2.

First, this procedure will obtain the list of all active connections via the Active Connctions Table - these are the machines that need to be rechecked, since they have the authority to provide integrity-sensitive data. For each of these connections, the verification procedure (Section 4.2.6) is executed - if any system fails this recheck, the connection is immediately dropped and its entry is removed from the Active Connections Table.

Next, for each connection that remains, the idle connection daemon is used to determine whether the connection is still actively supplying integrity-sensitive data. This daemon is described in detail in Section 4.2.12. If any connection is idle, the connection is dropped and it is removed from the Active Connections Table to avoid unnecessarily verifying the system repeatedly.

4.2.8 Dead Peer Detection

To handle dead peer detection, we had to create and integrate a daemon in PortAuthority with a modifiable script in Openswan. Openswan has dead peer detection functionality contained within it, and offers a script to run arbitrary shell commands upon detection of a dead peer. We use this script to communicate dead peers back to PortAuthority via a Unix domain socket.

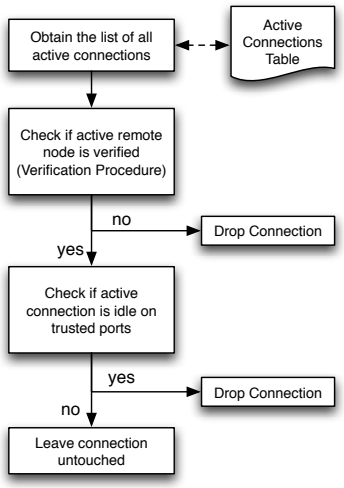

Fig. 4.2. Recheck Procedure

Thus, the dead peer detection daemon within PortAuthority waits for indication from Openswan that an IPsec connection is broken. When this occurs, the connection is broken and the remote machine's entries in the Active Connection Table and Propagation Table are removed. Thus, the remote machine will not be able to connect again until it has a fresh entry in the Propagation Table, which can only happen if the current machine or another trusted machine attests the dead peer once it returns to activity.

4.2.9 Add Link

Ensuring incoming data is of high integrity requires both verifiying the source's underlying system and authenticating that source. Thus, adding a link to a remote system combines two of the previously described mechanisms: performing a system attestation (Section 4.2.6) and authenticating a source (Section 4.2.4). If a source is

successfully verified via the verification procedure (Figure 4.1), then the creation of the IPsec tunnel can proceed.

We implement this process with the creation of two interacting daemons, which we illustrate in Figure 4.3. Namely, we implemented a Command Daemon, which communicates with the Listener Daemon on the other machine to facilitate the creation of a connection.

The Listner Daemon waits for requests for trusted connections. It will request attestations if necessary via the verification procedure, and will then set up an IPsec tunnel if the verification was successful. This daemon acts as the server for the Command Daemon.

The Command Daemon, in turn, waits for requests for commands from the local machine. The only command is to set up a trusted connection with a remote machine. This daemon communicates with the Listener Daemon of the remote machine after performing an attestation, if necessary. The Command Daemon can be invoked manually on the local machine with a script available to administrative users, and it is also invoked by the Automatic Connection Detection Daemon (see Section 4.2.14).

In both cases, the daemons add the remote host to their respective PortAuthority's Active Connections Table if the procedure was successful.

4.2.10 Remove Link

This function serves as the inverse to the Add Link function. Since the cleanup procedure for a removed link is mostly the same regardless of the reason for removal, we created this function to simplify that process. This function will tear down the

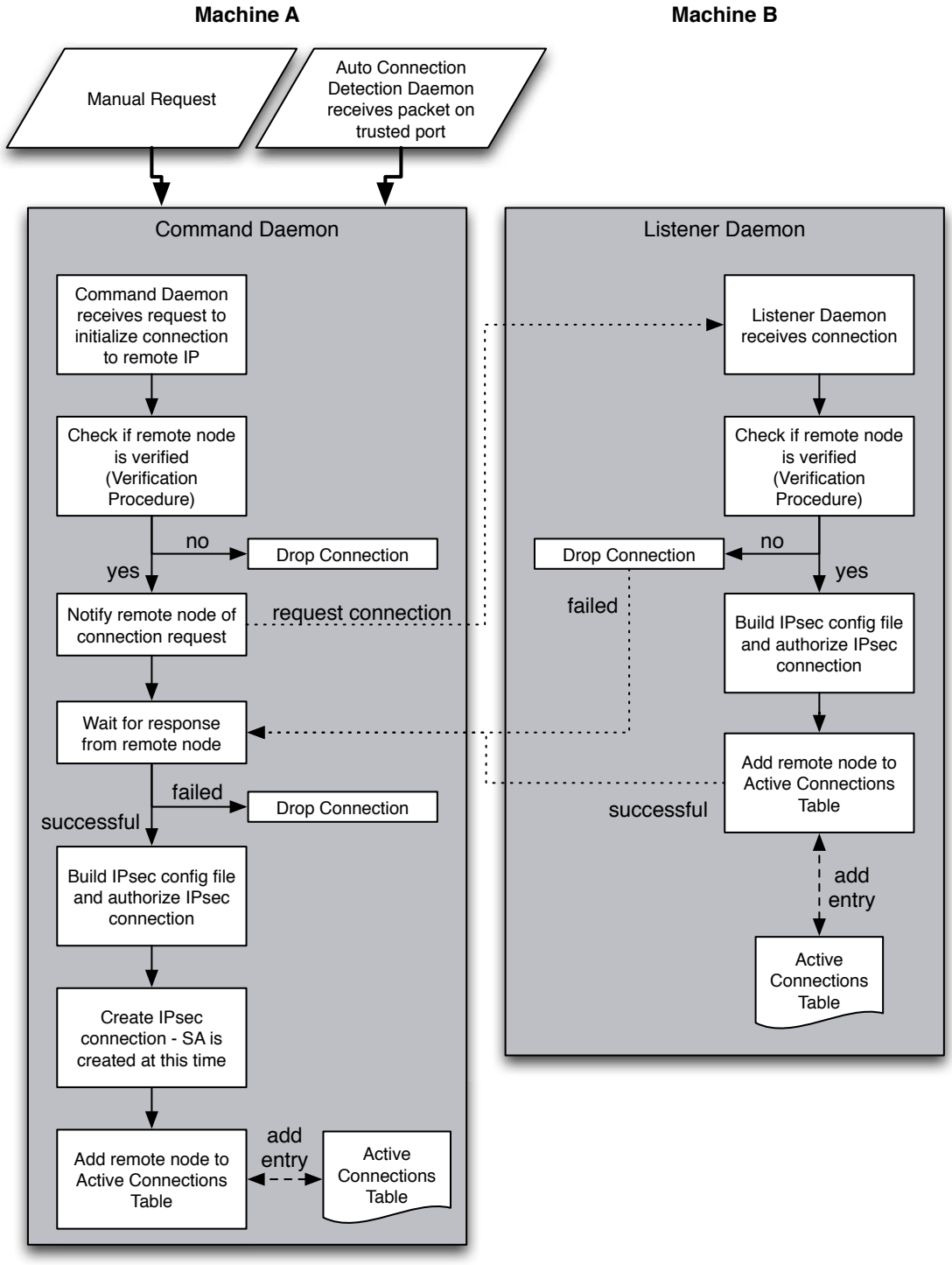

Fig. 4.3. Connection Creation Procedure

IPsec tunnel and remove the entry from the Active Connections Table and Propagation Table. By also removing the entry from the Propagation Table, we prevent the remote machine from being able to connect again without verification by simply initializing the connection procedure itself. While the remote machine will be able to attempt to connect again, it will first need to have a fresh verification created.

This function is invoked by the verification procedure (Figure 4.1) if the verification fails, and also by the dead peer detection daemon (Section 4.2.8). The idle connection detection daemon will also invoke this function (Section 4.2.12), but in this situation the cleanup process is slightly different: the entry from the Propagation Table is not removed. Thus, if a connection is considered idle and torn down, it can be reestablished quickly if the verification in the Propagation Table is still fresh. As a result, PortAuthority is able to efficiently handle the case where a connection temporarily appears idle but then continues to transmit data shortly thereafter.

4.2.11 Request Update

After a remote machine is successfully verified, this function is invoked to request the Propagation Table of that remote machine. It is through this function that PortAuthority is able to significatly reduce the number of full system attestations required, since information about verified machines is transmitted through the current distributed system.

Once the remote Propagation Table is pulled, it is merged with the current local Propagation Table. This process simply keeps the most recent verification of each verified machine, which only requires a comparison since the times of verification are

included within the Propagation Tables. The only other implementation requirement was a lightwight Propagation Request Daemon, which receives requests for the local Propagation Table and transmits it to the requesting party.

4.2.12 Check for Idleness

PortAuthority checks and removes idle connections in order to reduce the number of required attestations and verifications. For all connections that have not been used recently, the connection is dropped. Recent connections are detected via netstat, which lists all actively connected machines and which ports are being used. A connection is maintained only if the remote machine has recently accessed one of the trusted ports, since that is the only traffic that requires a verified link. We cannot simply check for any traffic between the two systems, since there will be always be IPsec and reattestation traffic even if there is no traffic on the trusted ports. If the connection is determined to be idle, then this observation is sent to the remote system, which performs the same check. If the remote system agrees that the connection is idle, it will drop the connection via the Remove Link function (Section 4.2.10). As described in Section 4.2.10, the IPsec connection is dropped and the machine is removed from the Active Connections Table, but remains in the Propagation Table. Thus, if the remote machine requests to connect again to a trusted port, and the attestation is still fresh, the connection can continue without an attestation.

In order to listen for idle observations, we had to implement another daemon to handle these messages. Both systems must mutually agree that the connection is idle before dropping the connection since it may be the case that only one is concerned

with the incoming traffic on the communicating ports. Thus, one machine may detect that the connection is idle, even though the other machine is still actively concerned about the incoming traffic: in this case, the connection should not be dropped. Our implementation of idle detection handles these situations.

4.2.13 Automatic Connection Configuration

Since we want PortAuthority to automatically determine when creation of trusted links is necessary, we must first define what these trusted links should be. Via a configuration option, as described in Section 4.2.15, an administrator is able to define the ports that should automatically create a trusted link when they are accessed by remote parties. These are, by definition, the same trusted ports used to receive integrity-sensitive data. The automatic connection implementation occurs as described in Section 4.2.14.

4.2.14 Automatic Connection Initialization

Using the list of trusted ports defined in Section 4.2.13, PortAuthority is able to detect connection attempts to these ports. PortAuthority then automatically initiates the process to create a trusted link without an explicit user request or application modification. The implementation of this functionality is again done through iptables: all non-IPsec traffic that is sent to these ports is instead forwarded (via DNAT forwarding on the PREROUTING table) to a separate daemon that listens for any incoming traffic. When this daemon detects incoming traffic, it makes a request to the Command Daemon (described in Section 4.2.9) to create a trusted link with the remote machine. If successful, an IPsec tunnel will be created and future traffic from this remote machine to the

trusted ports will proceed uninhibited. This implementation meets the design goal of easily overlaying PortAuthority with existing systems, since the applications themselves do not need modification and the user does not need to intervene aside from initially defining the trusted ports (Section 4.2.13).

4.2.15 Set Parameter

PortAuthority allows the administrator to define several parameters that determine how it will operate. Since some of these configuration parameters directly affect how well the system will monitor other systems, they will be included in the criteria used in the attestation process. For example, a poorly-informed administrator could set the verification recheck time to be several days, even though a more reasonable time would likely be less than a minute long. This setting would expose that administor's system to attacks in which a remote system is trustworthy at one time but later becomes of low integrity (such as by suddenly allowing any remote user to connect without authentication or authorization).

We have defined the following parameters that an administrator may wish to modify:

- **Recheck Time**: This parameter defines how often PortAuthority rechecks the verification of every actively connected system. That is, this parameter defines how often the Recheck Procedure is executed (Section 4.2.7).

- **Stale Time**: This parameter defines how old a verification may be in the Propagation Table before it is determined to be too old to use, and thus another verification

must be performed. This parameter is thus used in the System Attestation Procedure (Section 4.2.6).

- **Dead Peer Detection Time**: This parameter defines how long a remote peer may go unresponsive before it is determined to be dead. It is used in the Openswan IPsec configuration files that are created for each connection, since Openswan provides the indication of dead peers to PortAuthority (Section 4.2.8).

- **Trusted Machines**: This parameter contains a list of all of the machines that are allowed to connect to trusted ports even without verification. This parameter is described in the System Attestation Procedure (Section 4.2.6).

- **Trusted Ports**: This parameter defines the list of all the trusted ports that PortAuthority must protect. This list is used in the creation of the initial firewall rules (Section 4.2.1) and in the daemon that automatically detects connections to trusted ports (Section 4.2.14).

4.3 Conclusion

Through this implementation of the PortAuthority service, we were able to create and test our implementation on actual distributed systems. We will proceed to evaluate the performance of this implementation in Chapter 5.

Chapter 5

Evaluation

5.1 Introduction

In order to gague the effectiveness of the PortAuthority service, we constructed several systems to test its performance. These tests aim to measure several aspects of PortAuthority: its ability to initialize connections quickly, its ability to work in a large-scale distributed system without a major performance impact, and its ability to handle large connection sets.

We performed these tests on multiple Dell PowerEdge M605 blades. These machines have 8-core 2.3GHz Dual Quad-core AMD Opteron processors, 16.0GB RAM, and 2x73GB SAS Drives (RAID1). These systems were connected on an isolated network over a Gigabit Ethernet switch. Each blade served as the Xen hypervisor, and hosted multiple virtual machine domains for testing. The exact organization of virtual machines was dependent upon the experiment, so we will elaborate on these setups in the following sections. However, in all cases, the configuration for PortAuthority remained the same: active connections were rechecked for fresh verifications every 20 seconds, and a verification older than 45 seconds needed to be completed again.

We begin with a series of microbenchmarks in Section 5.2 to evaluate the time required for individual steps in the connection process. In Section 5.3, we build a large distributed system to compute macrobenchmarks: these describe the performance of

PortAuthority from the view of a full distributed system. Finally, in Section 5.4, we determine the load-handling capability of PortAuthority by requiring a single node to handle multiple simultaneous connections.

5.2 Microbenchmarks

We generated a series of microbenchmarks within the PortAuthority service to measure the performance impact of the various operations. We applied these benchmarks in the connection procedure (see Section 4.2.9), since this procedure is on the critical path to initially establishing a verified link. While some significant operations, such as attestation generation and verification, occur during other procedures, these procedures occur in parallel with the ongoing communication between two nodes; thus, the impact of these procedures is better evaluated in the macrobenchmarks in Section 5.3.

To obtain these results, our experimental setup included two virtual machines which created a mutually attested and verified link. That is, one machine requested a trusted connection with the other, and both sides verified attestations prior to authorizing the connection. Table 5.1 displays the real computational time of the operations in the Command Daemon and Listener Daemon, as well as the computational time of the daemons as a whole.

The total time required for the Listener Daemon includes an attestation and a generation of an IPsec configuration file and connection specification. These three operations account for 1.504 (80.0%) of the total time of the Listener Daemon. That is, the majority of the time delay for the Listener Daemon occurred for the attestation and for IPsec setup - neither of which can be controlled by PortAuthority.

Operation	Computational Time (seconds)
Attestation Time	1.087 ± 0.010
Generating IPsec Configuration File	0.00027 ± 0.000022
Generating IPsec Connection Specification	0.417 ± 0.020
Generating IPsec SA	0.588 ± 0.026
Listener Daemon	1.880 ± 0.097
Command Daemon	4.349 ± 0.095

Table 5.1. PortAuthority microbenchmarks: the time required to compute different phases in the connection process.

Similarly, the Command Daemon requires an attestation and all of the IPsec operations (including the creation of the IPsec Security Association). However, the Command Daemon is also required to wait for the Listener Daemon to execute, and is thus effectively required to wait for both attestations and all of the IPsec initialization commands. These commands account for 3.597 (82.6%) of the total time required for the execution of the Command Daemon.

These results indicate that while the time to initialize a verified link is fairly large, the majority of the time required is due to attestations and IPsec setup: both of which cannot be heavily influenced by PortAuthority itself. Thus, PortAuthority is able to manage connections with minimal overhead beyond the required waiting periods.

5.3 Macrobenchmarks

In order to evaluate the large-scale performance of the PortAuthority service, we generated a large working distributed system. To this end, we implmented a 40-node Tor anonymity network [11] that was used to request webpages from an Apache webserver. We use this example since Tor inherently provides a large degree of interconnectivity

- that is, many Tor nodes connect to many other Tor nodes when multiple clients are using the network.

Thus, our experimental setup consisted of forty Tor nodes, two of which were the required authoritative servers. These nodes were split over four of the blade servers described in Section 5.1, with ten nodes per server. An additional server housed a single application virtual machine with the Apache webserver, while another server contained ten client virtual machines. These client machines would make requests for an 8kb file to the webserver via the Tor network. In addition, these clients would make these requests from multiple threads to simulate additional clients. Each simulated client would generate its own three-hop route through the Tor network - this is the default Tor route length.

In terms of verification, the Tor nodes verify each other through a chain of mutually-attested links. Thus, each Tor node is attesting two neighboring Tor nodes. However, due to the message passing mechanism, each Tor node is implicitly able to trust each of the other Tor nodes due to indirect verification (see Section 3.3.2). Additionally, the clients are able to verify the whole of the Tor network simply by attesting a single node in the network. Note that the clients themselves do not need to be verified by the Tor network, since the operations they perform are unable to affect the integrity of the Tor network itself. For the purposes of this test, we also assume the webserver is trusted and does not require it to be verified - the goal is to determine the performance impact of PortAuthority on the distributed Tor network.

For this test, we measure the average requests per second and the average latency of client requests to the webserver in three cases: a vanilla Tor network, Tor using IPsec

only (with PortAuthority not running), and Tor with PortAuthority managing IPsec and verifications. These results are displayed in Figure 5.1 (average latency) and Figure 5.2 (average requests per second).

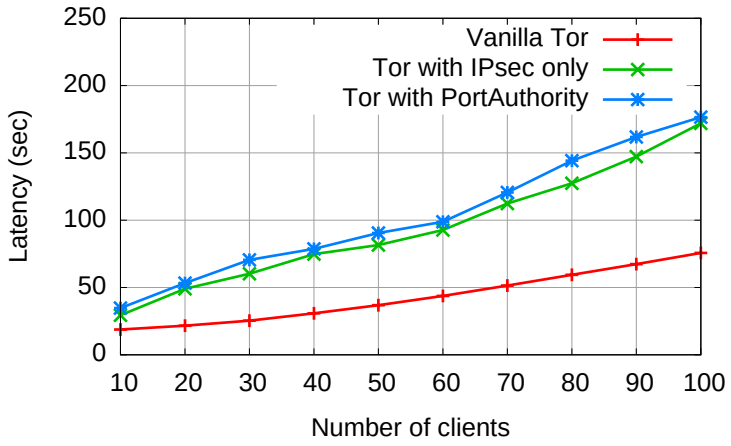

Fig. 5.1. Average latency for the Tor network under three different scenerios

These results indicate that the majority of the network-wide performance impact of PortAuthority is simply due to IPsec tunnels. Even though we use null encryption for the IPsec tunnel, we can see that IPsec still has a significant impact on performance. Since each web request and subsequent response are required to travel through three intermediary Tor nodes, the impact of IPsec is magnified: each packet from client to webserver (and webserver to client) effectively travels through four IPsec tunnels rather than one. The additional network overhead of PortAuthority is a result of the system attestations that are sent across the network. However, this additional overhead is clearly

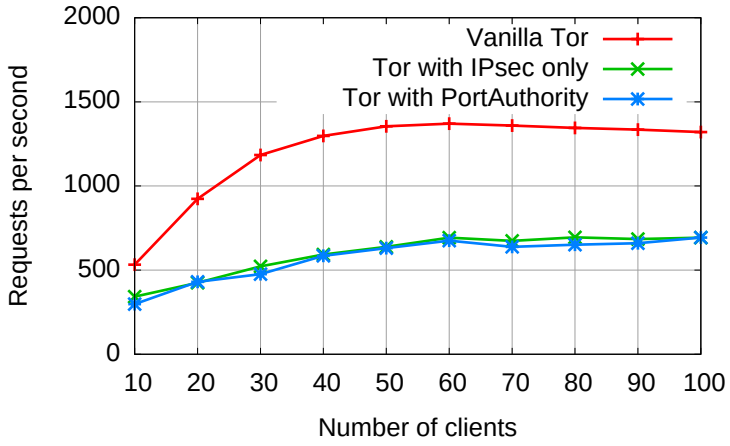

Fig. 5.2. Average requests per second for the Tor network under three different scenerios

minimal compared to the overhead of IPsec. Thus, relevant future work would include finding another mechanism for security-labelled connection authentication.

5.4 Stress Testing

Finally, we chose to evaluate the load capabilities of PortAuthority - that is, how many simultaneous connections PortAuthority is able to handle before performance begins to degrade significantly. For this experiment, we used a single virtual machine and began to create mutually attested links with more and more remote machines, creating a hub-and-spoke network model.

To evaluate performance, we used the response time for attestation requests. Since the central node will be bombarded with multiple reattestation requests, we can evaluate its ability to successfully function within the distributed system by its ability to respond to requests from peers. In an actual distributed system, a node may choose to break

its connection with another if that node does not respond to a request in a reasonable amount of time - thus, reponse time is an appropriate measure of the ability of a node to function in a verified network.

We see the results of our experiment in Figure 5.3. In order to create more attestation request collisions earlier, the attestations were rechecked every 5 seconds and were considered stale if older than 9 seconds.

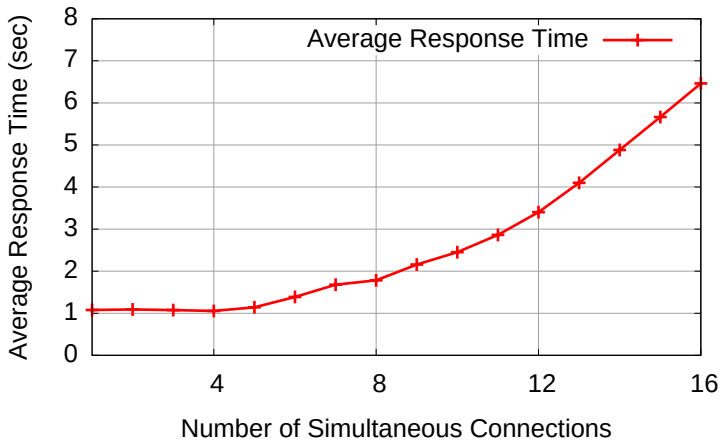

Fig. 5.3. Average response time for attestation requests

The major performance problems caused by multiple simultaneous verified links from a single node justify our goal of reducing the number of attestations and directly verified links required. Although no performance gains would be possible via link management in this particular network model, other distributed system structures would be able to perform better with indirectly attested links.

Chapter 6

Synopsis

In many of the systems used today, users implicity trust a number of machines to be performing correctly. Authentication of a remote machine is possible, and a user can validate that a single remote machine is in fact operating correctly, but extending this operation to distributed systems has proven difficult. Certain solutions exist for distributed systems, such as Shamon, but these solutions only provide load-time measurements of code and fail to prove the integrity of a system at a particular time.

The VMV work builds upon the Shamon archiecture to prove the integrity of a distributed system at a particular time. This solution only requires the validation of a single machine in that distributed system. However, it requires a smart method of connection management to ensure protection of integrity-sensitive inputs and to improve performance.

In this work, we have proposed the PortAuthority system as a method for handling trusted connections in a network. We have shown an efficient mechanism for enforcing that connections are only made to trusted remote parties, without affecting the applications that use these connections. In addition, we have shown how to tweak the design for further performance enhancements, such as by passing messages about trusted links. Finally, we have created and evaluated an implementation of our PortAuthority service design to show its negligible impact on performance.

References

[1] M. Abadi, E. Wobber, M. Burrows, and B. Lampson. Authentication in the taos operating system. In *In Proceedings of the ACM Symposium on Operating System Principles*, 1993.

[2] Eshwar Belani, Amin Vahdat, Thomas Anderson, and Michael Dahlin. The crisis wide area security architecture. In *SSYM'98: Proceedings of the 7th conference on USENIX Security Symposium*, pages 2–2, Berkeley, CA, USA, 1998. USENIX Association.

[3] K. J. Biba. Integrity Considerations for Secure Computer Systems. Technical Report MTR-3153, MITRE, April 1977.

[4] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote Trust-Management System Version 2. RFC 2704, IETF, Sep 1999.

[5] M. Blaze, J. Feigenbaum, and Jack Lacy. Decentralized Trust Management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173, November 1996. Los Alamitos.

[6] Luke St. Clair, Joshua Schiffman, Trent Jaeger, and Patrick McDaniel. Establishing and sustaining system integrity via root of trust installation. In *Proceedings of the 2007 Annual Computer Security Applications Conference*, December 2007. To appear.

[7] David D. Clark and David R. Wilson. A Comparison of Commercial and Military Computer Security Policies. *sp*, 00:184, 1987.

[8] Michele Colajanni, Tor Vergata, Philip S. Yu, and Daniel M. Dias. Analysis of task assignment policies in scalable distributed web-server systems. *IEEE Transactions on Parallel and Distributed Systems*, 9:585–600, 1998.

[9] Michele Colajanni, Philip S. Yu, and Valeria Cardellini. Dynamic load balancing in geographically distributed heterogeneous web servers. In *Proceedings of International Conference on Distributed Computing Systems*, pages 295–302, 1998.

[10] T. Dierks and C. Allen. The TLS Protocol Version 1.0. *Internet Engineering Task Force*, January 1999. RFC 2246.

[11] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *In Proceedings of the 13th USENIX Security Symposium*, pages 303–320, 2004.

[12] N. Doraswamy and D. Harkins. *IPSec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks.* Prentice Hall, First edition, 1999.

[13] C. M. Ellison, B. Frantz, B. Lampson, R. L. Rivest, B. M. Thomas, , and T. Ylonen. Spki certificate theory. IETF RFC 2693, September 1999.

[14] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: a virtual machine-based platform for trusted computing. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 193–206, New York, NY, USA, 2003. ACM.

[15] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. Peerreview: practical accountability for distributed systems. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 175–188, New York, NY, USA, 2007. ACM.

[16] T. Jaeger, R. Sailer, and U. Shankar. PRIMA: Policy-Reduced Integrity Measurement Architecture. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT)*, June 2006.

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

[18] Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. Delegation Logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security (TISSEC)*, 6(1):128–171, February 2003.

[19] Peter Loscocco and Stephen Smalley. Integrating Flexible Support for Security Policies into the Linux Operating System. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, pages 29–42, Berkeley, CA, USA, 2001. USENIX Association.

[20] J. Marchesini, S.W. Smith, O. Wild, and R. MacDonald. Experimenting with TCPA/TCG Hardware, Or: How I Learned to Stop Worrying and Love The Bear. Technical Report Computer Science Technical Report TR2003-476, Dartmouth College, 2003.

[21] H. Maruyama, F. Seliger, N. Nagaratnam, T. Ebringer, S. Munetoh, S. Yoshihama, and T. Nakamura. Trusted platform on demand. In *IBM Technical Report RT0564*, 2004.

[22] Jonathan M. McCune, Trent Jaeger, Stefan Berger, Ramon Caceres, and Reiner Sailer. Shamon: A System for Distributed Mandatory Access Control. In *ACSAC '06: Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference*, pages 23–32, Washington, DC, USA, 2006. IEEE Computer Society.

[23] Microsoft Corporation. Next generation secure computing base. `http://www.microsoft.com/resources/ngscb/`, May 2005.

[24] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *Proceedings of the 13th USENIX Security Symposium*, San Diego, CA, USA, August 2004.

[25] Joshua Schiffman, Thomas Moyer, Christopher Shal, Trent Jaeger, and Patrick McDaniel. No Node Is an Island: Shamon Integrity Monitoring Approach. Technical Report NAS-TR-0103-2009, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, February 2009.

[26] Elaine Shi, Adrian Perrig, and Leendert Van Doorn. BIND: A Fine-Grained Attestation Service for Secure Distributed Systems. In *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 154–168, Washington, DC, USA, 2005. IEEE Computer Society.

[27] Santosh Shrivastava. Satem: Trusted Service Code Execution across Transactions. In *SRDS '06: Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems*, pages 337–338, Washington, DC, USA, 2006. IEEE Computer Society.

[28] Sean W. Smith. Outbound Authentication for Programmable Secure Coprocessors. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, October 2002.

[29] TCG. Trusted Computing Group: Trusted Platform Module (TPM) Specifications. `https://www.trustedcomputinggroup.org/specs/TPM/`.

[30] Von Welch, Frank Siebenlist, Ian Foster, John Bresnahan, Karl Czajkowski, Jarek Gawor, Carl Kesselman, Sam Meder, Laura Pearlman, and Steven Tuecke. Security for grid services. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, page 48, Washington, DC, USA, 2003. IEEE Computer Society.

[31] Xelerance, 2008. `http://www.openswan.org/`.