Smokey: A User-Based Distributed Firewall System

Rachel Rubin

Department of Computer Science
University of California, Berkeley
Berkeley, CA 94704
rrubin@cs.berkeley.edu

Abstract

Traditional intranets have a central location on the network which easily allows the enforcement of a central security policy. They rely on the notion that there is one central entry point for all internal machines. Additionally, all machines that are considered logically inside the intranet reside behind the entry point. Increasingly this is not the case. We propose Smokey, a system that manages and distributes a central security policy to end clients on a network. Smokey installs a distributed firewall on a client based on a user's location and needs. The security policy that it distributes is based on the credentials the user provides and the lowest level of access needed for the user to complete their tasks is enforced.

Keywords:Firewall, Distributed, Network Security, User, Security ;

1 Introduction

Traditional corporate intranet security was based on the notion of "us vs. them". Anything inside the corporation's intranet was a trusted friend and anything outside was treated as a potential adversary. A firewall (a component placed between two sections of the network that filters traffic dictated by some security policy [Ioannidis et al., 2000]) was used to separate the intranet from the rest of the world.

Today intranet topology is changing. With the growth of extranets [Bellovin, 1999] that allow authorized users, such as telecommuters, access to the intranet from the outside(see figure 1) and the increasing existence of untrusted users, such as a corporate visitor, inside the intranet (see figure 2), the concept of a conventional firewall is becoming antiquated.

One solution is that central security policy be pushed to the endpoints. By allowing end clients the ability to enforce a centralized security policy, the intranet can essentially be spread over a large area and travel over untrusted links. Clients can be trusted based on providing authentication, not on their geographical location.

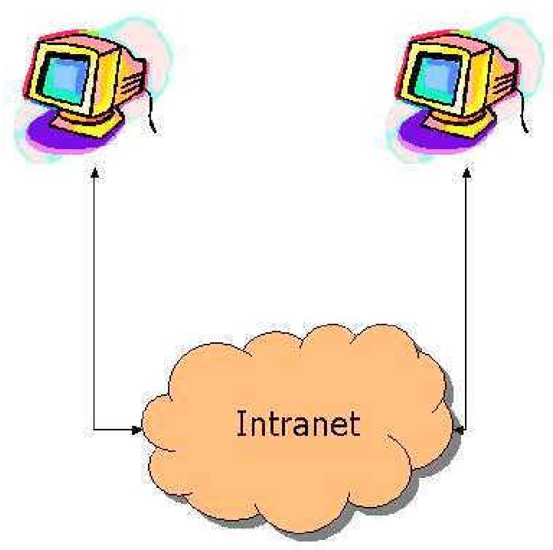

Figure 1: External single- and multi- user machines can belong logically to the intranet

However, it is difficult to distribute the security policy since adversaries can be found anywhere and may try to intercept it. By decentralizing the network, an attacker has more opportunities to break into the system. There is not one security bottleneck that can be tightly monitored. Each client has to be trusted to be secure so that the network will not be infiltrated.

We propose Smokey, a system that sets up a centralized security policy on clients in a distributed system. Smokey distributes the policy on a per-user basis, so that each user is allotted the amount of access that he or she can trusted with. On a multi-user machine, Smokey can retrieve and enforce separate policies for users with different needs.

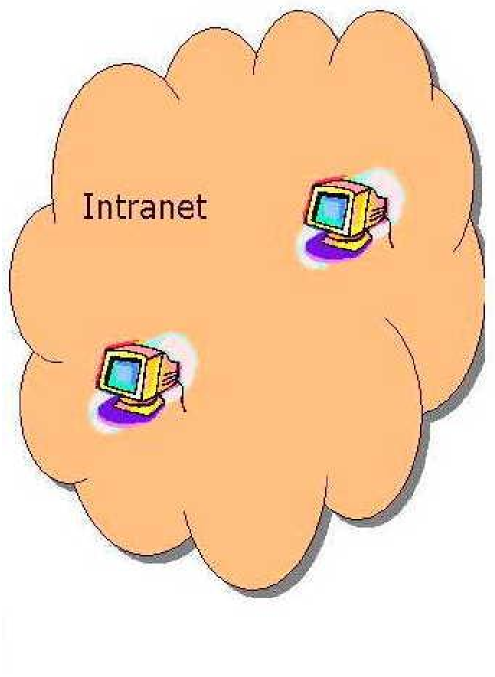

Figure 2: Internal machines may have different connectivity needs

1.1 Paper Organization

The remainder of the paper is organized as follows. Section 2 presents the related work. Section 3 describes the design of Smokey. Section 4 discusses the current implementation of the system. Section 5 outlines the future directions of the project. Section 6 defines the contributions Smokey has made to the problem.

2 Background and Related Work

2.1 Firewalls

Firewalls are a mechanism for policy control [Cheswick et al., 1994]. A traditional firewall consists of several different components the two most important being filters and gateways. Filters block the transmission of certain classes of traffic. Gateways provide relay services to compensate for the effects of the filters. Firewalls only block banned traffic from progressing. They provide no protection against problems with higher-level protocols. Traditionally the firewall has been placed as a barrier between an intranet and the outside world. It is a barrier that all traffic entering must pass through. More recently, internal firewalls have been instituted to isolate smaller, internal security domains.

Most of today's firewalls operate using a rule-based system [Bartal et al., 1999]. The rules instruct the firewall which inbound packets to let pass and which to block. Similarly, it monitors outbound traffic. The firewall's rule-base must be updated as new threats become known as rules are usually only written to combat known problems.

On some systems, the rules are grouped by roles. Roles are properties assumed by the different hosts in the network and define the capabilities needed by that host [Yialelis et al., 1996]. By assigning rules to roles and roles to hosts, each host is guaranteed protection of its needs. Roles can be hierarchically assigned with hosts inheriting rule sets from less secure ones [Thomsen et al., 1998]. This also has the side effect of separating the security policy from the network topology since hosts can have different rule sets.

2.2 Distributed Firewalls

A distributed firewall is a mechanism that enforces a centralized rule policy but pushes the enforcement of it toward the edges [Ioannidis et al., 2000]. In current implementations of the distributed firewall, access is controlled by machine identification. It is up to the host machine to fetch the security policy from a repository when an alarm is triggered. There is no way to automatically propagate the policy through the system or even to guarantee that a host has to update its policy on a regular basis. Machines are identified a via cryptographic certificate and are allowed to join the network based on this identification [Bellovin, 1999]. Local hosts download the security policy. When a connection is established, the policy daemon will screen the incoming traffic. If it is approved, the policy daemon will inform the kernel to proceed with the connection. If the daemon will try to establish validity through outside sources before, if needed, ultimately denying the connection.

2.3 Security Issues in Distributed Systems

There are several known threats to distributed systems that need to be analyzed when designing a security policy for one [Kemmerer, 1997]. Users must be verified and channels must be secured [Lampson et al., 1992].

A system, in order to be considered secure, has to provide several guarantees [Kemmerer, 1997]. These are:

- *Confidentiality:* The guarantee that sensitive information will not be disclosed to unauthorized recipients.

- *Integrity:*Data is modified in an authorized manner.

- *Availability:*Resources of the system must be accessible to an authorized user.

3 System Design

3.1 Case Scenario

This system would be useful in a plethora of situations. However, the key feature is that the security policy is assigned based on the user at the end host. Each user can load an individualized policy.

For example, there may be a situation inside a large company where they have noticed people misusing the IM clients and they have a suspicion that the clients are compromising their security. The system administrators decide to block these clients from being able to communicate with clients outside the corporation. This can be done with a simple addition to the firewall. However, it might not be pertinent for the policy to block internal communication. In this case, a user-based policy would be the best scenario to choose since the policy could be selectively applied.

3.2 Design Overview

Smokey is a user-based distributed firewall system for use on single- or multi- user machines. The system is designed in three large components which reside in user space and kernel space on the individual system and the server base located inside the traditional intranet. Each of these areas holds different sections of the system.

- *User Space*: The items in user space include the Login Client (see section 3.2.1), the Policy Manager (see section 3.2.2) and the Stored User Information (see section 3.2.3). These components are either in need of user-level input, like the login client, or need to access items, such as the file system, which is done from user space.

- *Kernel Space*: The items in kernel space include the Policy Handler (see section 3.2.5) and the Firewall (see section 3.2.6). These components are placed in the kernel because they have need of access to the low-level processes the kernel has control of.

- *Intranet*: The intranet contains components that hold sensitive information, such as passwords, and needs to be in an area where it can be closely monitored and be under tight control. The servers(see sections 3.2.1 and 3.2.4) are located in this space.

Smokey was designed with security in mind. It follows the basic rules of system design that underly a secure system. It follows the principles of separation of privilege and least privilege [Saltzer and Schroeder, 1975]. The Trusted computing base [ora, 1985] includes the kernel, the firewall, the policy server and handler and the policy and login servers. The servers and client communicate using public key encryption over a verified secure channel.

3.2.1 Login Client and Server

The login client and server are a part of a basic authentication system. The user enters his or her name and password into the login application. The user name and password are used to obtain a ticket and a user identification number (uid) which will allow the user to gain access to the policy server. [Bryant, 1988]

3.2.2 Policy Manager

The policy manager has two different states times that it is called to act. It is called by the login application at login time. It is also accessed when the user logs out.

At login time, the system is passed the ticket and the uid by the login client. The policy manager uses this uid to retrieve the user-specific rules if they are stored on the computer (see section 3.2.3). The time the rules were last stored along with the ticket, the uid and the type of client the user is on are passed to the policy server (see section 3.2.4). The policy server will then inform the policy manager if it has the most recent version of the security policy. If not, or if the rule base is not on the machine since the user is new to that client, the policy manager will request the new rule set from the policy server and it will be sent back in encrypted form and stored. The policy manager will then request the key to decrypt the policies it has retrieved from storage. When this is received, it will decrypt the policies and pass them to the policy handler (see section 3.2.5) along with an install instruction.

On logout the policy handler has a much simpler task It is passed the uid and it will pass that to the policy handler with a remove instruction. However, it must authenticate that this removal

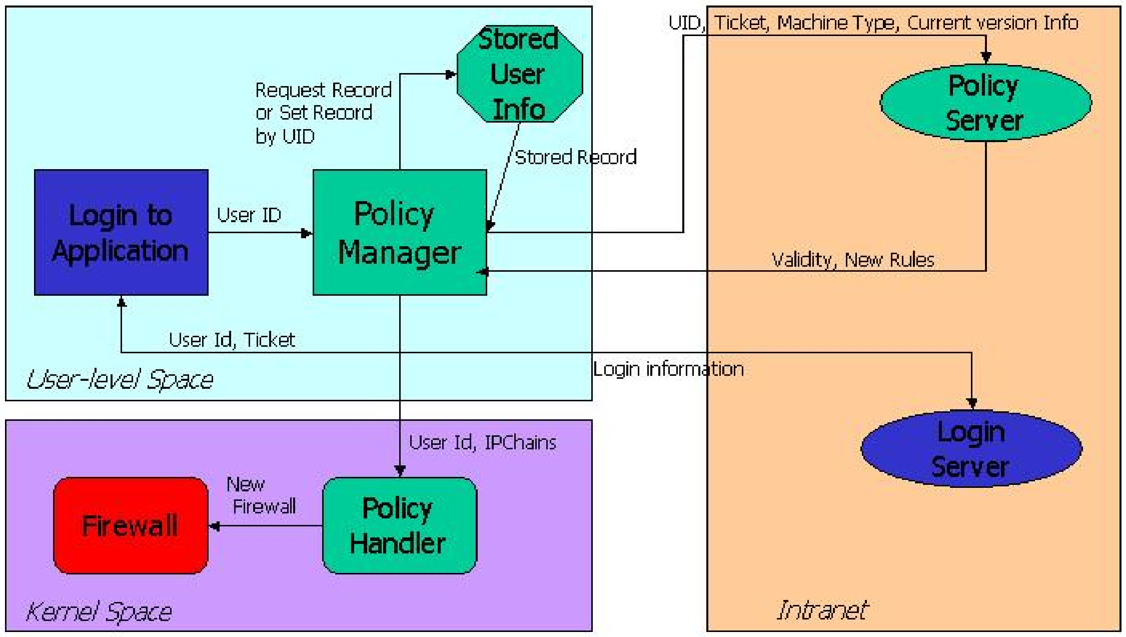

Figure 3: Diagram of the system design

request is valid by being given a certificate of assurance by the application that the user has indeed been disconnected.

3.2.3 User-Level Storage

The user-level storage stores a user's individual security policy on the client. These rules are stored in an encrypted form because if the client is compromised they cannot be read thus allowing knowledge of policies used across the system. The uid and date they were last modified is stored unencrypted along with the record. The key to decrypt the rules is not stored on the machine so they cannot be read without authentication by the server. The rule set is stored by hashing over the uid. Any user who has logged onto the client and had the security policy set up for them will have their rules stored under the assumption that a user will make a pattern of using that machine. Currently, there is no mechanism for removing the stored data of users who have not appeared on the client for a while.

The choice was made to store data on the system in order to reduce the latency of configuration. Passing large amounts of data across the network is not trivial and would slow down the time it will take to configure the system. Since we anticipate the security policy will evolve slowly, the decision was made to store the data locally.

3.2.4 Policy Server

The policy server is the central machine where the rules are stored and delivered to clients requesting them.

The security rules are stored in usable form so they can be directly installed onto the client without modification. When a rule is added, the system administrator goes to the policy server and adds the rule in this form along with the groups of users and types of machines that it applies to.

The rules are stored in a database which is keyed under the groups and types of machines. When a request comes in from the policy manager to see if its rules need updating the policy server will first check the ticket and make sure that it is valid. If it is, it will retrieve the records that correspond to the group the uid belongs to and the machine type that have been added since the rule bundle has last been updated. If this query is null, then the client has the most recent set of rules and is informed of the fact. If not, the server will gather all of the rules applicable to the uid on that type of machine into a bundle and encrypt them. Upon request from the client, this bundle will be sent to the client to store.

Once the client is guaranteed to have the most recent version of the rules, it will request the key to decrypt them. The key is different for each uid so a user will not be able to access any other user's rules with that key in order to preserve the separation of privilege. The key is a combination of the servers key and the uid. This key is sent back to the policy manager.

3.2.5 Policy Handler

When the policy handler receives rules along with an install command, it integrates the new rules into current firewall. It receives the rules in the correct form to be interpreted by the network stack. It also receives the uid and tags the rules with these so that they are only applied to network traffic that is arriving destined for that user. If the rule already exists on the firewall, the uid tag is added to that rule to prevent duplication.

When the policy handler receives a remove command with the uid, it will remove the uid tag from the rules. If the rule is only tagged with the current uid, it is removed from the firewall The rest of the firewall is left intact.

3.2.6 Firewall

The firewall, like a traditional one, will block transmission and receipt of certain classes of traffic [Cheswick et al., 1994]. However, there are some key changes made to Smokey's firewall. The firewall is not static: it has to be able to be modified on the fly while the computer is still online. The other key change to note in the implementation of the firewall is the presence of uids. Rules, when bound to the uids, should only be applied to traffic bound for applications owned by that user.

3.3 Threat Model

Smokey has several places where it could be attacked. If Smokey was broken into, the attacker could set the security rules or stop them from being applied on a machine. This is a dangerous situation. An attacker could bypass the security by knowing what back doors are there because they put them there! There are several locations where Smokey is at risk. However, there are safeguards in place to prevent the attackers from gaining control

- The attacker could pretend to be the policy server. When Smokey checks to see if the rules it has are up to date, the attacker could respond negatively and replace the user's rules with its own and be able to gain control of that user's account on the machine. However, Smokey uses secure authenticated channels to prevent this.

- The attacker may try to spoof a policy manager to gain access to the rules applied on a machine. It would request the rule bundle and key as if it were a new user on a client. He or she could use their knowledge of the security policy to break into the system later. The tickets and authenticated channels should prevent this attack.

- A malicious user may have access to a multi-user machine. However, since the security policies stored on the machine are encrypted using a unique key generated by the server, the attacker will not be able to get the key to decode the rules without a valid ticket.

- An attacker may gain access to a user's account. The attacker will be able to decode the rules that are stored since he will have a valid ticket. Since rules are the for that user and others in the same group on the same type of machine, he will be able to use his knowledge to break into those machines as long as he can detect where they are. This can currently only be stopped by detection and changing the rules to plug holes. In order to stop this attack, Smokey would need to be modified to allow a uid access to the key only once per ticket issued. (Although, if the user has already gained access to the account, it is unlikely he would need to examine the security policy to discover holes!)

- If an attacker gains access to the machine, they may be able to spoof the application and tell the policy manager that a specific uid has logged out, thus removing the security protection from the user while they are still connected to the network. However, since the communication between modules requires authentication, this is protected against.

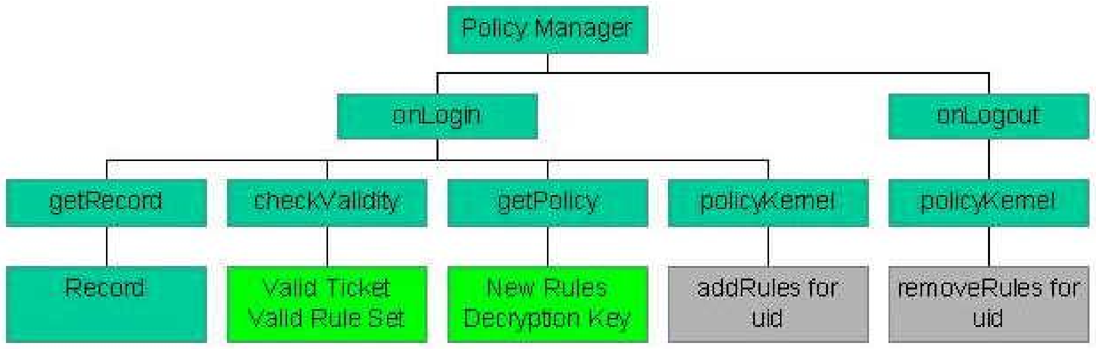

Figure 4: Flowchart of the Policy Manager. Teal boxes are a part of the Policy Manager. The access some external functions. Green boxes are a part of the Policy Server. Grey boxes are part of the Policy Handler.

These safeguards are put into place to maximize assurance that the security policy will not fall into malicious hands.

4 Implementation

The current status of Smokey is that it contains a fully implemented policy manager and a skeleton-code implementation of the policy handler and policy server. Smokey is written in JAVA and runs on a Linux platform. The rules are implemented in an IPChains format [Russell, 2000], which is a toolkit that adds rules to an existing firewall on a Linux platform.

The policy handler is implemented using several small modules(see figure 4). All data is private and inaccessible from the outside. The modules are divided into the following:

Main Interacts with the application. It receives the uid and whether this is a login or logout request. It is the only module that is able to receive instructions from the local machine. If the ticket is invalid, it returns the error to the login application and exits. It calls the module manages the rest of the modules and makes sure they are acting correctly.

Director Manages the rest of the modules and directs the flow of the data.

Retrieve Record Retrieves the encrypted user data.

Check Validity Calls the policy server and checks the validity of both the stored data and the ticket.

Retrieve Policy Requests and receives the new rules from the policy server and stores them locally. It also retrieves the key which is used to decrypt the rules and then is destroyed.

Call Kernel The unencrypted rules are passed to the policy handler and then destroyed,

Logout Requests the policy handler remove rules related to the uid

This division of labor into smaller modules makes the application more secure [Saltzer and Schroeder, 1975].

5 Future Work

There is still work left to be done on Smokey.

The first priority is to finish the implementation. Once there is a fully working prototype, it would be useful to deploy Smokey on a network and test it in a real environment.

There are several ways that Smokey could be changed that would add functionality. The clients need to have a way to report any security breaches back to the server. This way any breaches or attempted breaches could be dealt with by a system administrator. Another step would be to add the functionality to the system that the policies could self-configure based on these reports. They could be configured into the format for the rules and added to the database. The could be applied to other groups and types of machines that are similar to the one where the breach was noticed.

Currently, there is no way for the system to distribute new rules to clients that are already running. If a rule is added it may be because of a potentially harmful system breach which needs to be patched immediately. However, rule sets currently can only be modified on login. Smokey could implement a way to propagate rules throughout the system as they are added by keeping track of who is currently running that needs the patch and sending an update to the manager which would install it.

Several organizations may have access to a machine running outside the intranet. It would be useful to allow a secure multi-organization scenario.

Finally, it would be interesting to test Smokey on other types of security policies. It would be useful to be able to propagate any type of policy throughout the network and have the clients be able to adjust.

6 Conclusions

Smokey is the prototype of a system that distributes a centralized security policy to clients. Smokey can set individual security policies based on the user, location and type of the machine. This customability maximizes the usefulness of the security system since it can be geared toward the current situation. Although firewalls have come under criticism as not the most useful security policy, Smokey can be generalized to the distribution of other policies. Smokey has shown how to centrally manage and securely dynamically distribute a security policy,

References

[ora, 1985] (1985). The department of defense trusted computer system evaluation criteria. Technical report.

[Bartal et al., 1999] Bartal, Y., Mayer, A. J., Nissim, K., and Wool, A. (1999). Firmato: A novel firewall management toolkit. In *IEEE Symposium on Security and Privacy*, pages 17–31.

[Bellovin, 1999] Bellovin, S. M. (1999). Distributed firewalls. *;login:*, 24(Security).

[Bryant, 1988] Bryant, B. (1988). Designing and authentication system: A dialogue in four scenes.

[Cheswick et al., 1994] Cheswick, W. R., Bellovin, S. M., and Rubin, A. D. (1994). *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley.

[Ioannidis et al., 2000] Ioannidis, S., Keromytis, A. D., Bellovin, S. M., and Smith, J. M. (2000). Implementing a distributed firewall. In *ACM Conference on Computer and Communications Security*, pages 190–199.

[Kemmerer, 1997] Kemmerer, R. A. (1997). Security issues in distributed software. In *Proceedings of the 6th European conference held jointly with the 5th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 52–59. Springer-Verlag New York, Inc.

[Lampson et al., 1992] Lampson, B., Abadi, M., Burrows, M., and Wobber, E. (1992). Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310.

[Russell, 2000] Russell, R. (2000). Linux ipchains. http://www.netfilter.org/ipchains/HOWTO-1.html.

[Saltzer and Schroeder, 1975] Saltzer, J. and Schroeder, M. (1975). The protection of information in computer systems. In *Proceedings of the IEEE*, pages 1278– 1308.

[Thomsen et al., 1998] Thomsen, D., O'Brien, D., and Bogle, J. (1998). Role based access control framework for network enterprises. In *14th Annual Computer Security Application Conference*.

[Yialelis et al., 1996] Yialelis, N., Lupu, E., and Sloman, M. (1996). Role based security for distributed object systems.