Secure Partitioned Access to Local Network Resources over the Internet

Chris I. Dalton, D. A. Clarke
Extended Enterprise Laboratory
HP Laboratories Bristol
HPL-98-202
December, 1998

multilevel security, gateway, internet, network

A common problem faced by many organizations connected to the Internet is controlling precisely which hosts and services on their local network a user can access from outside the local network over the Internet.

One solution would group the local hosts and services into logical partitions (or segments), and allow users or classes of users access to particular partitions based on who they are or what functional group they belong to.

We have implemented such an approach using gateways running Multilevel Secure Operating Systems, such as HP-UX VVOS 10.24 or Trusted Solaris 2.5. The gateways run trusted versions of SSH and SOCKS that have been derived from publicly available sources.

This paper reviews the features of a Multilevel Secure Operating System relevant to its use in the role of an Internet gateway, describes our trusted implementations of SSH and SOCKS and shows how they may be used in combination to provide precise but transparent local network partitioning and access control.

We end with a concrete example that shows secure access over the Internet to file systems provided by WindowsNT servers on a local network.

The trusted SSH server software is currently being used by a number of banks in Scandinavia.

Internal Accession Date Only

1 Introduction

This paper looks at the problem of how to provide selective access on a per user basis to hosts and services on an internal network from clients on the Internet.

We show how this problem can be addressed by using hosts running Multilevel Secure operating systems as network partitioning gateways between the Internet and internal networks. The gateways run versions of the freely available SSH and SOCKS software that we have adapted for multilevel operation.

First, we introduce multilevel secure operating systems theory and concepts and discuss particular configurations that are suitable for use as network gateways. These configurations form the foundation of the rest of the paper.

We then present an overview of the architecture of the network partitioning gateway. We describe the theory of its operation and break down the key functionality the gateway must provide. We briefly analyze the security benefits we gain by basing the gateway on a multilevel secure operating system.

We go on to describe in detail the two software components we use to provide the key gateway functionality. We use a Secure Shell Protocol server to provide a user authentication service and a proxy service. Secondly, we describe our network access control mechanism which is based on multilevel modifications to the server component of the standard SOCKS toolkit.

We finish with an example of the network partitioning gateway in use. We show it being used to allow partitioned access to internal WindowsNT shares from clients out on the Internet. We discuss the advantages of using our network partitioning gateway.

2 Multilevel Secure Operating Systems

Hosts running a multilevel secure (MLS) operating system form the basic building block of our network partitioning gateways. In this section we define what we mean by a multilevel secure operating system by describing the functionality it must provide. We discuss suitable configurations of such operating systems when used as network gateways and end by briefly discussing two commercial operating systems that fit our requirements.

2.1 Our requirements

To support our needs, the operating system running on the gateway must support two extra features over and above the functionality found in general purpose multi–user operating systems such as UNIX. Firstly, it must implement a Mandatory Access Control policy based on the Bell–LaPadula formal model [1]. Secondly, it must support the idea of Least Privilege. Operating systems that support such functionality are typically called multilevel secure.

2.1.1 Mandatory Access Controls

The system should implement a Mandatory Access Control (MAC) policy alongside the more usual Discretionary Access Control (DAC) policy for controlling access to system resources such as files, processes and network connections.

With standard DAC schemes (such as the UNIX permission bits), a user who owns a particular resource can grant access to that resource to other users. On the other hand, MAC protection schemes enforce a particular policy of access control on the system. Users, even if they own a resource, cannot grant access to that resource to other users if that access would violate the MAC policy of the system.

Formal models for MAC were developed initially on behalf of military organizations in an attempt to develop systems in which the flow of information around the system was predictable based on a particular policy. A common policy adopted was that expressed in the Bell-LaPadula model.

In this model all resources (objects) are given sensitivity labels. The sensitivity label consists of two components: a classification and a set of compartments. The classification is used to specify how secret or sensitive a resource is. The compartment set is used to partition access to a resource at a particular classification. Compartments allow data to be partitioned according to project, department or organization, for example.

All users (subjects) are given *clearances*. Clearances have the same syntactic form as sensitivity labels. Users also have a *current level* associated with them. The current level also has the same syntactic form as sensitivity labels. The clearance a user holds and the

current level they are operating at dictates what access they have to resources based on the policy. Simplifying, the Bell–LaPadula model consists of a state machine and the following two rules for determining whether a particular state is secure:

– If a user has READ access to a resource then their *clearance* dominates the sensitivity label of the resource, i.e. the classification part of a user's clearance must be greater than or equal to the classification part of the resource's label. Additionally, the compartments set of the resource's label must be a subset of the user's clearance. This is the Bell-LaPadula *simple security property.*

– If a user has READ access to a resource then their *current level* dominates the sensitivity level of the resource, i.e. the classification part of a user's current level must be greater than or equal to the classification part of the resource's label. Additionally, the compartments set of the resource's label must be a subset of the user's current level. If a user has WRITE access to a resource then their current level equals that of the resource they are trying to write to. The classification part of the current level must be equal to the classification part of the resource's label. Additionally, the compartments set of the resource's label must be equal to the compartment set of the user's current level. This is the Bell-LaPadula **–property.*

Using MAC labels to protect resources

In order to add some clarity to the above discussion we present the following labeling scheme for handling data on an MLS system used to hold business data for an organization. Our example labeling scheme consists of three classifications, UNCLASSIFIED, CONFIDENTIAL, SECRET and a set of compartments, SALES, ACCOUNTING, ENGINEERING, MARKETING.

The classification part of a user's clearance on the system is mapped to their grade within the company, i.e. a director of the company has a clearance of SECRET, a manager has a clearance of CONFIDENTIAL and a normal worker has a clearance of UNCLASSIFIED.

The compartments within the system correspond to departmental groups within the organization. So, for example, the full clearance (classification plus compartments) for a manager in the accounts department would be CONFIDENTIAL ACCOUNTS.

Based on this scheme, we would then protect files on the system as follows. Files that should be readable by anyone should be labeled UNCLASSIFIED. Files that any manager (or above) should be able to see are labeled CONFIDENTIAL. Files that only managers or directors within the accounts department should be able to read are labeled CONFIDENTIAL ACCOUNTING. A manager with cross-department responsibilities may be given several compartments as part of their clearance. For example, an engineering manager with the clearance of CONFIDENTIAL ACCOUNTS MARKETING ENGINEERING could view the management documents in both the accounts and marketing as well as the engineering department.

As well as allowing us to restrict which users get to see what data, the MAC policy also allows us to control the way data flows around the system. For example, the MAC policy restricts users to being only able to write documents at their current level. It also ensures they must be at the same or higher level to view documents. This prevents them from reading a document at TOP SECRET ACCOUNTING SALES and then writing it to UNCLASSIFIED SALES, for example. Had this been possible, then somebody with a clearance of TOP SECRET ACCOUNTING SALES would have been able to grant access to a file at that level to somebody with a lesser clearance. This would be a violation of the MAC policy.

Network gateway MAC configurations

Systems designed around the Bell–LaPadula model can be used outside their more usual data classification and protection role. Here we show how they can be profitably configured as network gateways.

When used in its traditional role the MAC policy is typically used to prevent the leakage of sensitive information. For network gateways, we are more interested in using the MAC controls to provide a read–only environment for processes and users and in the strong data partitioning between compartments that MAC systems offer.

The configuration shown in figure 1 below uses compartments to distinguish between data and resources accessible from the Internet, and those accessible from a company's internal LAN. This is the configuration used by Hewlett-Packard's VirtualVault product [2, 3]. The labeling scheme consists of one classification, SYSTEM and two compartments, INSIDE and OUTSIDE.

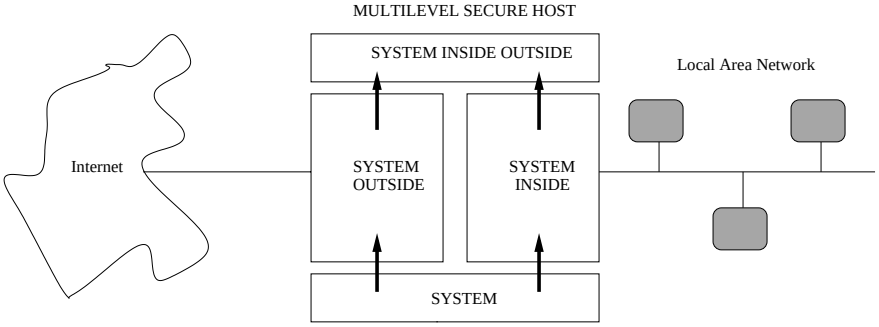

Figure 1: Simple MAC configuration

This host has two network interface cards, one connected to the external Internet and the other connected to an internal local area network (LAN). The networking is configured such that packets from the Internet are labeled SYSTEM OUTSIDE and packets from the internal LAN are labeled SYSTEM INSIDE.

In this configuration, a connection from the Internet can communicate only with processes

labeled SYSTEM OUTSIDE. Only processes running at SYSTEM INSIDE can communicate with hosts on our internal LAN.

If an attacker from the Internet were to gain access to a shell on the gateway machine, the current level of that shell process would be SYSTEM OUTSIDE. The Bell–LaPadula access control rules guarantee us two things. Firstly, the attacker cannot write to files labeled as SYSTEM. This in effect write-protects any files labeled as SYSTEM (such as the password file) as far as the attacker in the outside compartment is concerned. Secondly, the attacker cannot launch an attack on the internal LAN. The shell of the attacker's current level does not equal that of the internal LAN card so the attacker cannot write packets to that network. Further, the attacker's current level does not dominate the level of the inside network so the attacker cannot listen to packets on the internal network.

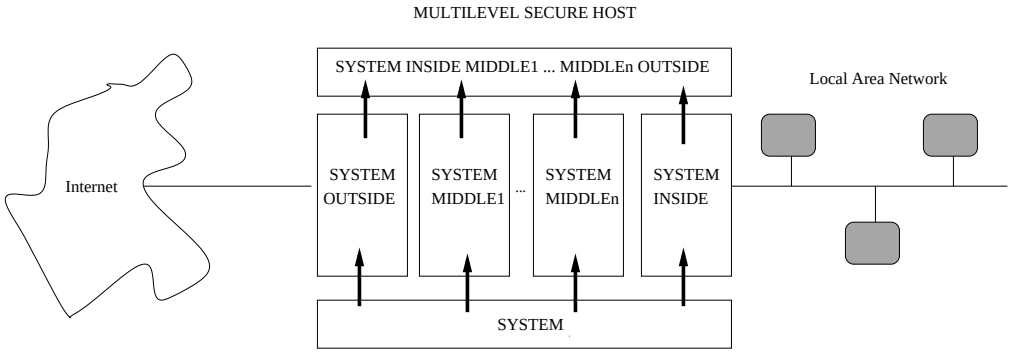

Figure 2: Multi-compartment MAC configuration

Figure 2 shows an extension of this model and is the one adopted as the foundation for our network partitioning gateways. A number of middle compartments have been added to the system. The middle compartments are used to provide isolation between processes and as we shall see later, so we can implement fine grained access control to hosts on our internal network from processes running in middle compartments. The use of this type of MAC configuration for also hosting multiple independent applications is introduced in [4].

2.1.2 Least Privilege

In traditional UNIX based operating systems there is single account (root) that is privileged to override the security policy of the system.

To use a program that must override the security policy to do its job, a user must either assume the identity of root before running the program or alternatively the program must have its SUID bit set to root.

In the case of MAC systems this is particularly worrying as the breach of the root account or a process with root privileges could lead to all the MAC controls being circumvented.

Typically then operating systems that implement MAC controls also implement the concept of least privilege. Here, the power to override the system security policies is broken into a large number of individual privileges instead of the one root account. Processes (known as *trusted* processes) that need to override part of the system security policy are given just those privileges to do the job they are intended to. Users are given authorizations to use these trusted processes. An example privilege would be the one that allows a process to bind to a privileged network port (one less than 1024). Under UNIX, a process has to be running as root to do this. On a system that supports least privilege it is possible to grant that process only the privilege it requires.

2.2 Available implementations

The combination of MAC controls and the principle of least privilege are most often found in operating systems that are derived from the Compartmented Mode Workstation (CMW) specification [5, 6]. Two such operating systems are HP–UX VVOS 10.24 from Hewlett-Packard [3, 2] and Trusted Solaris 2.5 from Sun Microsystems [7]. They both provide a UNIX type user and programming interface. Trusted Solaris 2.5 is a full CMW implementation. Hewlett–Packard's HP–UX VVOS 10.24 is a based upon their full CMW operating system release (HP–UX 10.26) but has been streamlined for use on gateway systems. It forms the foundation of the VirtualVault product line.

3 The Network Partitioning Gateway architecture

In this section we give an overview of the architecture of the Network Partitioning Gateway. We explain the theory of its operation and identify the key services the gateway must provide. Our aim is to be able to allow client users out on the Internet to be able to connect to services running on our internal hosts. We wish to be able to strongly authenticate the client users before allowing them to connect. We wish also to be able to vary which internal hosts or groups of hosts (and hence their services) are visible to different users or groups of users.

3.1 Theory of operation

We proxy all connections through a MLS gateway that sits between the internal network and the Internet. No direct connections are allowed from the Internet to the internal network(s). The gateway is a multilevel secure system configured as in figure 2. This gateway provides three major services. Firstly, it can authenticate at the user level clients who wish to connect to internal services. Secondly, it provides a means of proxying connections from the (authenticated) client users through the gateway to internal services. Thirdly, it can control the visibility of internal hosts (and hence services) to external clients depending upon their authenticated identity. Figure 3 shows the overall architecture.

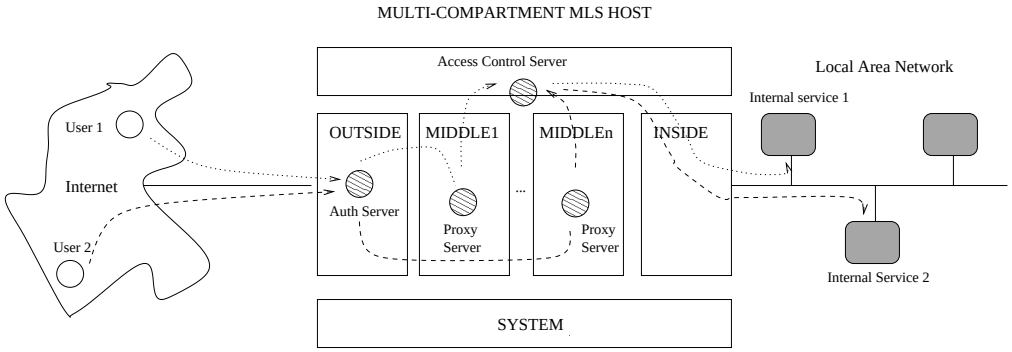

Figure 3: The Network Partitioning Gateway

3.1.1 Authenticating users

The only service exposed to the Internet is the authentication service of the gateway. Clients must initially connect to this server. When authenticated, the users are mapped to a particular sensitivity level. This corresponds to the level of one of the middle compartments on the MLS gateway. This is assigned to the user by the security administrator of the

system. Several users may be mapped to the same level in order to facilitate access on a group basis.

3.1.2 Proxying access to internal services

The key to the approach taken here is that we have adopted the the SSH method of forwarding connections to multiple TCP/IP ports over a single channel. Once authenticated, the user is given a direct (encrypted) channel to a proxy server that runs in the user's mapped middle compartment. Following the SSH model, the client sets up on their local machine dummy listening ports for particular services. These dummy ports correspond to the ports of services on the internal network that the client ultimately wishes to access. The client then instructs the proxy server running in its middle compartment to set up connections to the real services running on hosts on the internal network. To make use of a service on the internal network, the client application is configured to use the dummy service port on the local machine itself. Traffic to this port is then relayed to the proxy server in the user's middle compartment over the authenticated (and encrypted) channel. The proxy server relays this traffic to the real service on the internal network and passes any traffic from that service back to the local dummy service and hence back to the application client.

As an example, consider the case where we wish to allow external clients on the Internet access to an internal IMAP mail service that is sitting behind a gateway. This service runs on the well known IMAP port number (143) on a host within the internal network. The SSH model of port forwarding achieves access in the following way. Firstly, the user runs a client application (the SSH client program) that sets up a dummy listening daemon at the well known IMAP port on the user's local machine. The client SSH program also makes a connection to a SSH server process on the gateway. The SSH server authenticates the user and sets up a proxy to the real IMAP service host on the internal network. The external user configures their mail client to look for the IMAP service on their own machine instead of on the remote internal machine behind the gateway. The client SSH program intercepts any traffic to this local port and forwards it over the authenticated and encrypted channel to the SSH server. The SSH server simply proxies this traffic to the real IMAP host and passes any replies back.

3.1.3 Controlling access to internal services

The proxy server runs in a middle compartment that maps to a particular user. It is not trusted with any privileges, therefore by default the MAC controls prevent it communicating with anything outside its compartment. So that the proxy server may be able to setup proxy connections to services running on internal machines on behalf of the remote clients, we provide an access control server that acts as TCP/IP relay. This relay consults a control file that lists permitted connections before allowing the proxy connection to be setup.

3.2 Summary and Analysis

The operating system MAC controls on the gateway ensure that a user can only connect to the services running in the OUTSIDE compartment of the gateway. They cannot connect directly to other compartments within the gateway or to internal hosts behind the gateway. We choose to run only the authentication server within the outside compartment. This ensures that the only service on the gateway that external users can connect to is the authentication service. The gateway authentication server maps users to a particular middle compartment level. Should attackers somehow manage to exploit a bug in the authentication server code, they are constrained within the outside compartment and cannot launch attacks onto the internal network.

Further, we can label any information used by the authentication server in making authentication decisions at the SYSTEM level. This in effect write-protects it as far as an attacker in the outside compartment is concerned. This protects against the attacker installing their own authentication information and so being able to circumvent our authentication controls.

The access control server dictates which hosts can be contacted on the internal network from particular middle compartments. The MAC controls ensure that the access control server is the only path from the middle compartments to hosts on the internal network.

With a combination of our authenticated user to sensitivity level mapping and the ability to control which hosts on the internal network are visible from which levels we can therefore limit what internal hosts are visible to external users. This allows us to group internal hosts into logical partitions and then allow access to these partitions based upon the identity of the connecting user.

The following two sections describe in detail how we have modified the standard SSH code base to provide our gateway authentication and proxy servers and how we have modified the standard SOCKS server code to provide the access control service functionality. We have based our services around these implementations since they are already widely used by the Internet community and thus well tested and likely to be free of any major flaws.

4 The Secure Shell SSH protocol

Our architecture requires that we have a means of authenticating a user and then mapping that authenticated user to a compartment on an MLS system. We also require a port forwarding server that runs on behalf of authenticated users and allows them access to internal services. The Secure Shell Protocol (SSH) provides both these functions and we have based the operation of the gateway around a SSH server front end. We establish the identity of the connecting user via the SSH key exchange. We then allow access to internal services via the built in port forwarding features of SSH. This section introduces the major features of SSH and it then describes on how we have modified it for multilevel operation. We concentrate on how we achieve the user to compartment mapping using the multilevel SSH. Further details of the multilevel SSH implementation and a description of using SSH for remote administration of MLS systems can be found in depth in [9].

4.1 The SSH protocol and implementation

4.1.1 Protocol

The SSH protocol [8] has been designed to enable secure remote access to hosts over an insecure network. It uses RSA–based host and user authentication as well as channel encryption using IDEA, 3DES or something similar.

The basic steps involved in the protocol are as follows. Initially, a client connects to the server SSH port[1]. A protocol exchange follows whereby the client system verifies the identity of the server system via two public RSA keys belonging to the server. The client then generates a random session key, encrypts it using the server keys and sends it to the server. From now on the channel between client and server is encrypted using this session key. Following this the client authenticates itself to the server via one of several methods[2]including RSA–based user authentication, RSA–based host authentication and simple password based authentication.

After authentication, the client usually requests a login shell or command to be run on the remote system. The protocol also supports X11 traffic forwarding and indeed arbitrary TCP/IP port forwarding both from client and server system over the encrypted channel.

It should be noted that the protocol has been designed so that it can be extended fairly easily.

[1] port 22

[2] The authentication methods accepted are controlled by the server administrator.

4.1.2 A Sample implementation

A sample implementation of the SSH protocol is available from *www.cs.hut.fi*. The implementation consists of two main parts: a client side program (ssh) and a server side daemon (sshd).

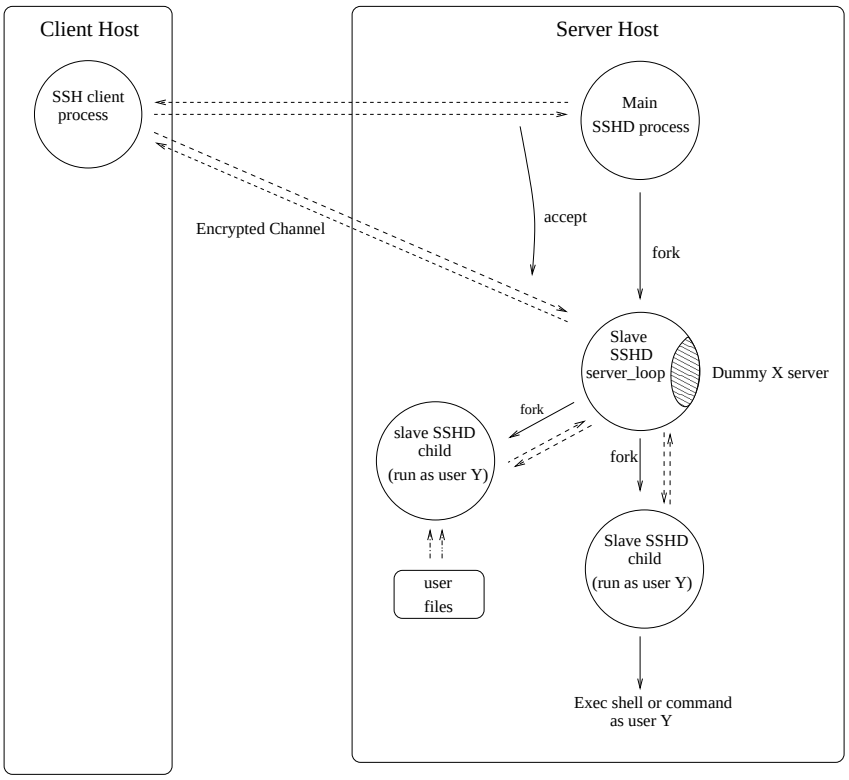

Figure 4: Client and server processes involved in an SSH invocation

The sample implementation works roughly as follows. When the server side process (sshd) receives a connection from a client it first spawns a child process to handle this connection.

This spawned process begins the SSH protocol exchange with the client over the channel (which is a bsd socket). Version strings are exchanged, followed by the server public RSA keys and some random data to guard against IP spoofing. The client returns this random data along with an encrypted session key. The server turns on encryption on the channel using the session key and begins the client authentication stage. The client tells the server which user it wishes to authenticate as on the remote system. The server forks a child as this user and carries on with the authentication process. The spawned child is used to provide the server with access to user owned files (such as private RSA keys) as and when needed as part of the authentication process.

If the authentication is successful, the server then enters a loop waiting for a shell or

command execution request from the client. At this point, the server may also set up any X11 and TCP/IP forwarding asked for by the client (section 2.2.1).

When the server receives a shell or command request, it spawns another child. This child has its input and output redirected to the server. The child execs the requested shell or command.

The server itself sits in a loop listening for any output from the child. This output is multiplexed onto the encrypted channel and sent back to the remote client. Similarly, any output from the remote client is sent to the child process.

The server also listens for any X11 or other forwarded TCP/IP traffic that needs sending over the encrypted channel.

Figure 4 shows the client and server side processes involved in an SSH invocation where the user on the client host wishes to login to the server as user Y.

X11 forwarding

The X11 forwarding is implemented by the server process creating a dummy X11 server. When logging in using SSH, the user's DISPLAY environment variable is set to this dummy X11 server. The server process listens for traffic on the port that it creates and sends this over the encrypted channel to the client host. The client then sends it to its real X11 server.

Arbitrary TCP/IP port forwarding

In section 3.1.2 we described the SSH model of proxying by client side port forwarding. SSH supports both server side and client side TCP/IP port forwarding.

The SSH server side port forwarding is a generalization of their X11 forwarding scheme. A dummy listening socket is set up on the server machine by the slave SSH daemon that is run on behalf of the connecting user. The slave SSH daemon listens on this socket and passes and traffic directed at it down to the client over the encrypted SSH channel. The client side SSH program then passes this to the real service that is running locally on the clients machine.

The client side port forwarding functions in the manner described in section 3.1.2. The client side SSH program sets up a dummy listening socket for a particular service on the client's local machine. The SSH client program passes data from that dummy listening socket over the encrypted SSH channel to the user's slave SSH daemon process. The slave daemon then forwards this traffic to the real service that is either running on the same machine as the SSH or on another host on the network.

4.2 SSH extensions to support multilevel gateway operation

Here we discuss the changes we have made to the SSH server code so that we can reliably map users to compartment levels within the gateway. For the purposes of this paper, we are most interested in the port forwarding features of SSH rather than its ability to run a shell or command on behalf of the user.

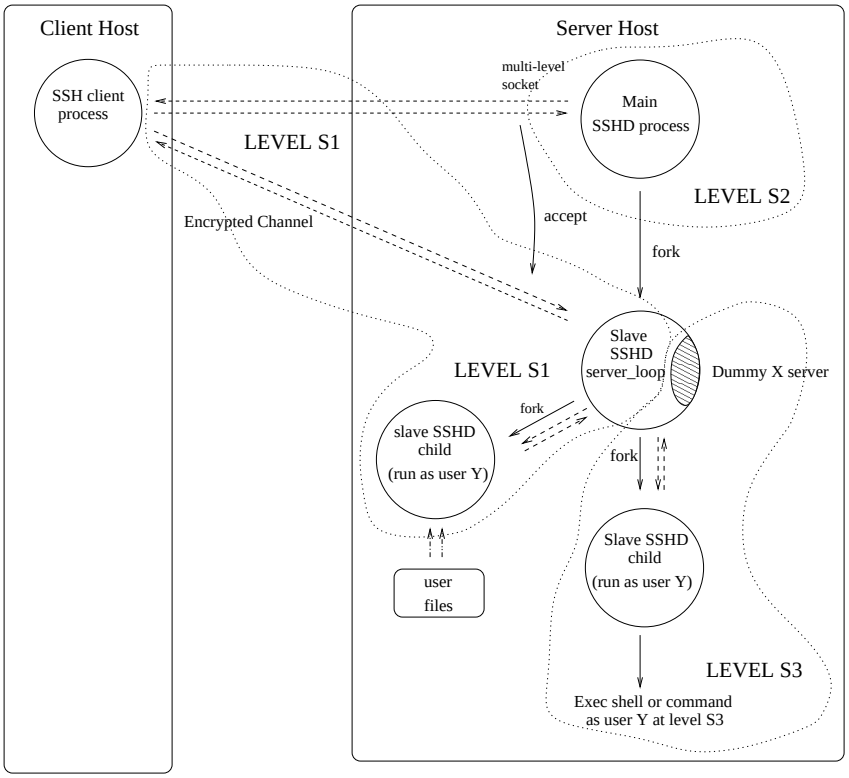

Figure 5: Extended multilevel SSH

The key new component in the multilevel sshd is a server side authorization file. This file associates a sensitivity level with each user. These levels correspond to middle compartments on the gateway machine. As with the non-multilevel version, when the user is authenticated, the SSH server spawns a slave server on behalf of that user. This slave server handles running a shell or command for the user and also dealing with port forwarding for that user. The level of any shell or command that gets run for the user is dictated by their entry in the server side authorization file. Similarly, and importantly, any forwarded connections the slave daemon is asked to make on behalf of the client are done so at the level specified for the user in the authorization file. Any server side port forwarding sockets are also set to listen only at the level specified for that particular user in the server side authorization file. Figure 5 shows the processes involved in the operation of the multilevel SSH server.

Some implementation details

Modification is required to the sshd code to permit a user's login shell to be run at a different level to the level that the user connects from. In our particular case, the client users will be always connecting at the SYSTEM OUTSIDE level. The slave server process is required to change levels between this level and the level that has been associated with that particular user as it reads and writes data to and from the remote client and the users shell process. It also has to switch levels when forwarding traffic.

5 Access control using SOCKS on the network partitioning gateways

Our network partitioning gateway architecture requires a server that controls access to internal hosts from processes (namely the proxy servers) within middle compartments. Here we discuss how we have adapted SOCKS [11] for this purpose. We do not look at how it can be used in its more traditional role of provide external application service access for internal clients that sit behind a firewall where the firewall is an MLS machine. A discussion of that can be found in [10].

First we give a brief overview of the SOCKS protocol and standard toolkit. We then look at the changes we have made to the standard implementation to support our requirements.

5.1 The SOCKS protocol and toolkit

The SOCKS protocol is a protocol for relaying TCP sessions through an intermediate host such as a firewall. It allows transparent proxying of application protocols such as telnet, ftp and HTTP.

The SOCKS toolkit[3]is an implementation of the SOCKS protocol. It consists of a relay server process and a client library.

The SOCKS server process runs on the intermediate host. Its role is to form a virtual circuit between the client and the application server and to relay packets between them. Access control mechanisms within the SOCKS server process allow a policy of network access to be established and enforced.

The client library provides direct plug in replacements for the standard TCP socket API calls[4]. The SOCKS protocol is carried out below the socket API layer and is therefore transparent to the application.

Creating a SOCKS*ified* client is usually only a two step process. First, the existing socket calls are renamed to their SOCKS equivalent. Secondly, the application is linked against the SOCKS library. Shared library versions of the client libraries are available for some UNIX platforms; in this case there is no need to recompile client source code. Client libraries are also available for Macintosh and Windows platforms.

Figure 6 shows a typical usage of SOCKS in a firewalled environment. Here the firewall host acts as a barrier between an external and internal network. The packet filtering function of the firewall ensures that there is no direct access between systems on the external and internal networks. A SOCKS server is run on the firewall host to allow (controlled) access to external hosts from internal clients.

[3]Available from ftp://ftp.nec.com/pub/socks/

[4]connect, bind, accept, listen and select.

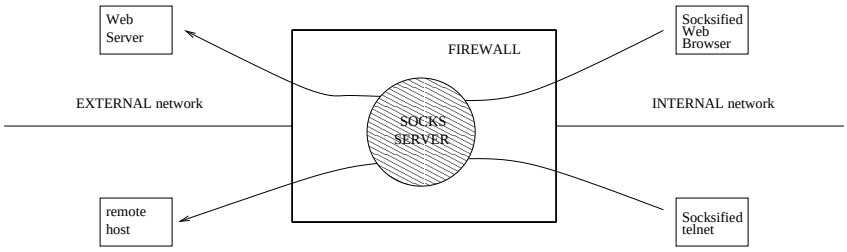

Figure 6: Common SOCKS configuration

5.1.1 Logging

The SOCKS server has the ability to log all connection requests and also the number of bytes transferred across a connection. This allows activity across the firewall to be monitored.

5.1.2 Access control in SOCKS

It is possible to restrict what hosts and application services a client can connect to through a socks server. Access control for a client using the SOCKS protocol is enforced by the SOCKS server process. The access control policy that any particular SOCKS server mandates is defined by rules within a configuration file for that server. The access control rules allow a policy based on a combination of user, requesting host, destination host and service port to be established. Figure 7 shows an example control file. This file would allow the host with IP address 147.143.2.2 to connect via the SOCKS server to host 147.143.3.1. It would also allow the host with address 147.143.2.10 to connect to any other host.

```
permit  147.143.2.2  255.255.255.255 147.143.3.1  255.255.255.255
permit  147.143.2.10 255.255.255.255
```

Figure 7: SOCKS control file

Further discussion on the use and configuration of the SOCKS toolkit can be found in [12, 13]

5.2 Adapting SOCKS for multilevel operation on the gateway

In this section we describe how the standard SOCKS server daemon can be modified to provide the access control and TCP relay function required of our gateway architecture. Figure 8 shows the basic architecture.

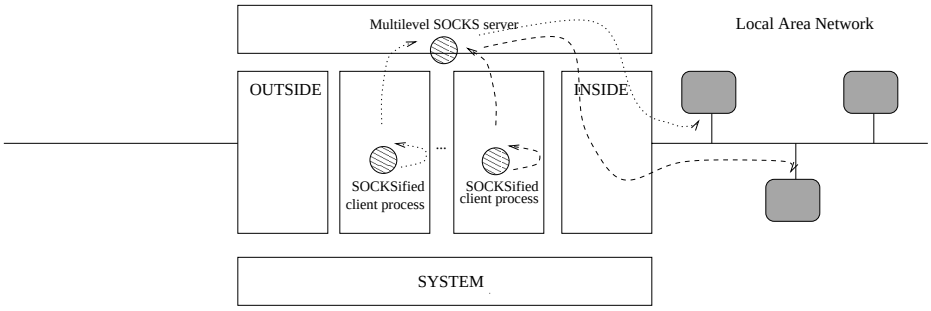

Figure 8: Multilevel SOCKS configuration

5.2.1 The SOCKS clients

In the case of our gateway, the SOCKS clients are the proxying server daemons that run on behalf of the users in middle compartments. We saw in section 4.1.2 that this proxying service is provided as part of the SSHD server daemon. An additional step we have taken is to SOCKSify the proxying code with the SSH daemon so that when it tries to set up a connection to a remote machine on behalf of a client, it does so through our SOCKS daemon on the gateway, not directly. Any direct connection would in fact be blocked by the MAC controls of the gateway.

When the proxy server tries to set up a connection to a particular internal host, the SOCKS server decides whether it should be allowed or not. If it should then it makes the connection to the requested service at the level of the internal network.

5.2.2 Multilevel Access control

As was seen in section 5.1.2, standard SOCKS access control decisions are based on the IP address of the client requesting a connection to a remote host and on the IP address of that remote host. This method has been retained and extended to support the concept of sensitivity levels.

As part of the standard SOCKS protocol, the client indicates to the server which host it wishes to contact. The server then initiates a connection to the requested host. For a multilevel operation SOCKS service, the server must also know at which sensitivity level

it should try to contact the requested host. In our simple gateway configuration we have only one inside network connection (labeled SYSTEM INSIDE). However, we may have more than one internal network connection, each at a different level. We may also wish to allow processes within one compartment to be able to communicate over TCP/IP with other processes in different compartments. Our multilevel SOCKS service must be able to cope with this situation. One way of ensuring this would be to add a sensitivity level field to the SOCKS protocol. A client SOCKS protocol message would then include both the remote host address and the level to contact that host at. However, this approach requires modification to the SOCKS client library and the client side code itself. The approach we have adopted instead uses the server side SOCKS control file to dictate what level connections requested by the client get made at.

The SOCKS configuration file has been extended to include two extra fields at the end of a permit or deny line. The two extra fields both contain sensitivity labels. The first one refers to the level a client is contacting the SOCKS server at, the second one dictates which level the SOCKS server should make its connection to the real server at. When the SOCKS server receives a connection from a SOCKS client[5]it checks that a client at that level is permitted to connect the host that they are asking to. If so, then the SOCKS server relays the connection to the host, switching level where necessary.

Importantly, the SOCKS client libraries need no modification to work with the trusted SOCKS server and are not trusted themselves. To provide the SOCKSified proxying service, all that is necessary is to SOCKSify the portion of the SSH server code that deals with forwarded connections and link it against the standard SOCKS library.

```
# MULTI_LEVEL  socks config file - cid 97.
# Allow connections from the middle1 compartment to the host 15.144.60.200
# client1 -> internal server1
permit 127.0.0.1 255.255.255.255 15.144.60.200 255.255.255.255 system middle1 system inside
# Allow connections from the middle2 compartment to  the host 15.144.60.203
# client2  -> internal server2
permit 127.0.0.1 255.255.255.255 15.144.60.203 255.255.255.255 system middle2 system inside
```

Figure 9: Multilevel SOCKS control file

5.2.3 Implementation details

The trusted SOCKS server has been trusted with the ability to switch sensitivity levels. Its clearance includes all classifications and compartments, so spans all compartment boundaries. It is trusted with enough privilege to listen and accept connections from clients at

[5]In our case when it gets a connection from the proxying portion of the SSH slave daemon running on behalf of a user at the level of that user's compartment.

any level / from any compartment. The following code shows the main processing loop of the socks server. The code that needs to be added to make it function as in figure 8 is contained within the

```
#ifdef MULTILEVEL
    ...
#endif
```

sections.

```
while (1) {
            ....
            if ((s = select(fdsbits, &fds, NULL,NULL, &tout)) > 0) {
                if (FD_ISSET(in, &fds)) {
                      if ((n = read(in, buf, sizeof buf)) > 0) {
                                      from_in += n;
#ifdef MULTILEVEL
                                      if(setslabel(dst->olevel) < 0){
                                          ...
                                          exit(1);
                                      }
#endif
                                      if (write(out, buf, n) < 0){
                                              goto bad;
                                      }
#ifdef MULTILEVEL
                                      if(setslabel(dst->ilevel) < 0) {
                                          ...
                                          exit(1);
                                      }
#endif
                                  } else {
                                          goto bad;
                      }
                  }
                  if (FD_ISSET(out, &fds)) {
#ifdef MULTILEVEL
                            if(setslabel(dst->olevel) < 0) {
                                ...
                                exit(1);
                            }
#endif
                            if ((n = read(out, buf, sizeof buf)) > 0) {
                                from_out += n;
#ifdef MULTILEVEL
                                if(setslabel(dst->ilevel) < 0){
                                    ...
                                    exit(1);
                                }
#endif
                                if (write(in, buf, n) < 0) {
                                          goto bad;
                                      }
                                } else {
                                      goto bad;
                                }
                      }
                } else
                    ...
                }
       }
```

Modifications to the existing SOCKS server code must also include calls to the general MLS initialisation routines and extensions to the control file so that we can specify labels as part of a rule. However, the total amount of additional code is small.

6 Using the Network Partitioning Gateway

In this section we give an example of how to give users on the Internet selective access to WindowsNT file system shares on internal machines.

6.1 The SMB protocol and services

WindowsNT uses the Server Message Block protocol (SMB) to share file systems between machines. The SMB protocol implementation has three well known ports associated with it. The two of primary interest are its name service on port 137 and its session service on port 139.

6.2 Running SMB over SSH

Some sites wish to allow access to file system shares on their internal machines to client users out on the Internet. One way of doing this is to allow direct connections to the internal machines from the Internet. However, SMB traffic is not encrypted and its authentication methods can be circumvented. An alternative approach taken by some sites is to run the SMB connections over an authenticated and encrypted SSH channel in the following manner.

A gateway machine running the SSH server daemon sits between the user on the Internet and the internal file system share. If the user has an SSH identity on the gateway machine then they can connect to that and set up a forward to the internal host that is hosting the file system. The user sets up dummy listeners for the SMB ports (137 and 139) on their local machine. They also instruct their SSH server process on the gateway to forward any traffic passed to these ports to the real machine on the internal network. Instead of mounting the share as:

```
net use g: \\internal-machine.some.domain\sharename
```

they redirect it to the local dummy SMB ports with the command:

```
net use g: \\localhost\sharename
```

The client side SSH program then intercepts traffic directed to the dummy ports and passes it to the user's SSH process on the gateway which then passes it to the real services on machine on the internal network.

Using SSH the SMB connections are encrypted and before a user can get access to a machine on the internal network they are strongly authenticated. Although a fair amount of plumbing is required to set up the service the advantages are very clear[6].

6.3 Making use of the MLS gateway

Using SSH on a standard gateway to facilitate access to internal machines is a definite improvement over allowing direct connections to internal machines from the Internet. However we can achieve security enhancements by using our MLS gateway. Further, our network partitioning gateway gives us the ability to easily, precisely and securely control exactly which users get access to which internal shares.

The setup procedure as far as the client is concerned is the same as before. Now however because we map a user to a compartment level and can restrict what internal machines can be contacted from a particular compartment with guaranteed MAC controls, we can vary easily just what internal machines are visible to each user.

[6]The amount of plumbing required can be considerably reduced by using a network redirector that automatically makes SMB file system connections over SSH. This is currently being implemented by the authors.

7 Conclusion and Future Work

We have seen how we can profitably use suitable configurations of multilevel secure operating systems in the role of network partitioning gateways. This allows us to achieve our aim of controlled access to internal systems at the user level from clients on the Internet.

We identified the major components such a network gateway must have and showed how we implemented those components using standard publicly available software packages. Whilst we hope that the tried and trusted public sources are free of major flaws, we showed how the mandatory access controls of the operating system guard against vulnerabilities in those software components.

Our experimental implementations have shown the potential value in using multilevel secure hosts as network gateways. We believe there is further useful work that can be done in this area. For example combining the concept of TCP/IP connection splicing [14] with multilevel secure gateways and pushing cross–compartment access control decisions into the kernel.

References

[1] BELL, D. and LAPADULA, L. (1975). Secure Commputer Systems: unified Exposition and Multics Interpretation. *MITRE Technical Report MTR-1997, The MITRE Corporation.*

[2] HEWLETT-PACKARD CO. (1996). Virtual Vault Transaction Server Concepts Guide.

[3] HEWLETT-PACKARD CO. VirtualVault *http://www.hp.com/security/.*

[4] DALTON, C.I. and GRIFFIN, J.F. (1997). Applying Military Grade Security to the Internet. *Computer Networks and* ISDN *systems, volume 29, number 15, November 1997.*

[5] MILLEN, J.K. and BODEAU, D.J. (1990). A Dual-Label Model for the Compartmented Mode Workstation. *MITRE Paper M90-51, The MITRE Corporation.*

[6] DEFENSE INTELLIGENCE AGENCY (1991). Compartmented Mode Workstation Evaluation Criteria. *Report DDS–2600–6243–91.*

[7] SUN MICROSYSTEMS INC. Trusted Solaris 2.5.1. *http://www.sun.com/trustedsolaris/.*

[8] YLONEN, T., KIVINEN, T., SAARINEN, M., RINNE, T. and LEHTINEN, S. (1998). SSH Protocol Architecture. *Internet Draft, daft-ietf-secsh-architecture-02.txt.*

[9] DALTON, C.I. (1998). Strongly authenticated and encrypted multi-level access to CMW systems over insecure network using the SSH protocol. *HP Laboratories Technical Report HPL-98-99.*

[10] DALTON, C.I. (1998). A generic proxying facility for CMW based on the SOCKS protocol. *HP Laboratories Technical Report HPL-98-100.*

[11] NEC USA, Inc. Introduction to SOCKS. *http://www.socks.nec.com/introduction.html.*

[12] CHAPMAN, D.B. and ZWICKY, E.D. (1995). Building Internet Firewalls, pp 278–296. *O'Reilly and Associates, Inc.*

[13] CHESWICK, W.R. and BELLOVIN, S.M. (1994). Firewalls and Internet Security, pp 137–208. *Addison Wesley.*

[14] SPATSCHECK, O., HANSEN, J.S., HARTMAN, J.H. and PETERSON, L.L. (1998). Optimizing TCP Forwarder Performance. *Dept. of Computer Science, University of Arizona Technical Report TR 98-01.*