Strongly Authenticated and Encrypted Multi-level Access to CMW Systems over Insecure Networks using the SSH Protocol

Chris I. Dalton
Extended Enterprise Laboratory
HP Laboratories Bristol

HPL-98-99(R.1)
February, 1999

SSH, multi-level security, CMW

This report looks at using the Secure Shell (SSH) remote login protocol to enable access to Compartmented Mode Workstation (CMW)[1] hosts from unlabeled clients such as WindowsNT hosts over insecure networks.

After giving a brief introduction to the SSH protocol and describing one particular sample implementation, we look at some of the issues involved in porting that implementation to CMW.

We then show how we have extended the SSH protocol to enable terminal access, file update and X11 from unlabeled clients not just at single but multiple sensitivity levels. We describe how we use the strong authentication properties of SSH to confidently map an external user identity to a sensitivity level or range of sensitivity levels within the CMW system.

1 Introduction

Traditionally, CMW systems have typically been used as part of a closed *trusted* network. The trusted nature of the network allows several assumptions to be made. It can be assumed, for example, that the network is protected from unauthorized snooping. Similarly, the identity of the hosts taking part in communication with each other does not need sophisticated verification. Additionally, the hosts forming such trusted networks generally have multi-level aware network stacks (such as MaxSix [2]) and are able to communicate with their peers at multiple sensitivity levels.

Our research work has involved using CMW technology outside of this normal environment where the above assumptions are not valid. We can't be sure that our network traffic isn't being listened to, neither can we assume that a host we are communicating with is who it claims to be. In general, clients communicating on this type of network are also not multi-level aware and do not have network stacks that support the concept of sensitivity labels.

This has highlighted the need for a way of enabling secure remote access to CMW systems from unlabeled systems over untrusted networks and for a way of allowing these unlabeled systems to interact at multiple sensitivity levels with the CMW servers.

The solution we present is based upon the SSH remote login protocol. The SSH encryption functionality is used to compensate for lack of a trusted network. Additionally, the strong user authentication mechanisms allow us to permit clients to interact with CMW systems at multiple levels and reliably map users to a sensitivity level or range of levels.

We begin with an introduction to the SSH protocol and discuss the design of one particular sample implementation. We then go on to highlight the issues involved in porting that implementation to a CMW operating system[1]. We discuss the extensions we have made to the SSH protocol and existing implementations to support our requirements for multi-level access from untrusted (and unlabeled) hosts.

Finally, we briefly mention some utilities (both WindowsNT and Unix) for remote file update and management that work on top of SSH. We have found them particularly useful in the situation where CMW machines are used to provide web based services.

We assume the reader is familiar with CMW concepts and terminology. If not, then details can be found in [3] or appendix A.

[1]We have done this for HP-UX CMW 10.16, HP-UX VVOS 10.24 and Trusted Solaris 2.5

2 The SSH protocol and implementation

2.1 Protocol

The SSH protocol [6] 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[2]. 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[3]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.

2.2 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).

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 informs the server as to which user it wants to try and 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.

[2]port 22

[3]The authentication methods accepted are server side administrator controllable

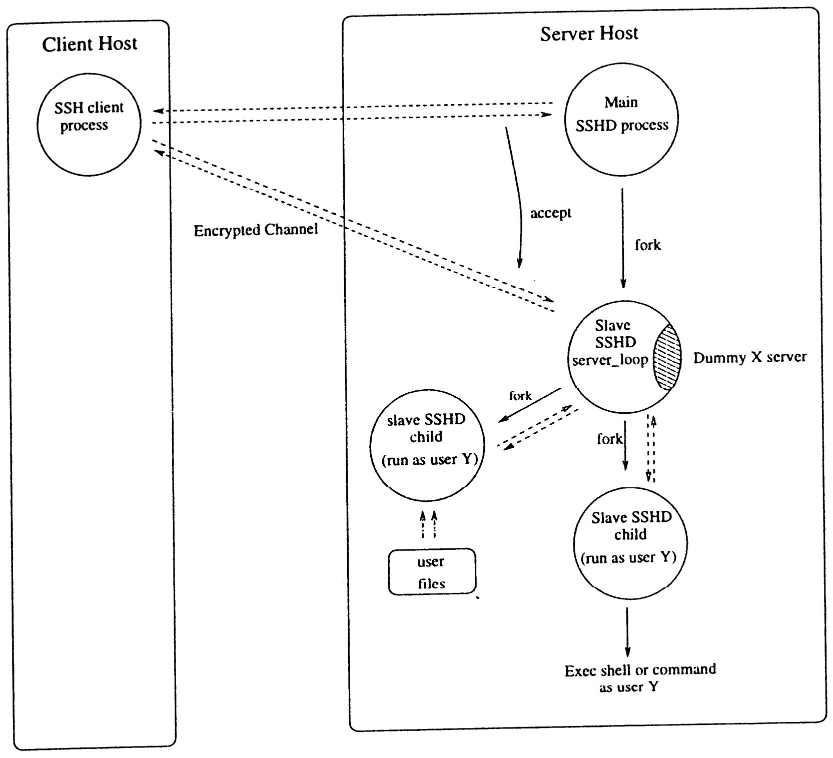

Figure 1: Client and server processes involved in an SSH invocation

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 1 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

As well as x11 forwarding over the encrypted channel, traffic to any TCP/IP port can be redirected. Again, this is achieved by the server process listening on the given port on the server and sending any traffic received there back to the client over the encrypted channel. The client passes it to the real service running on the locally on the client itself.

The client may also forward arbitrary TCP/IP port traffic to the server. Here the client listens on the given port. Any traffic received is passed down the encrypted channel to the server process which passes it to the real service for that port on the server. system.

3 A port of SSH to CMW

Unlike conventional Unix operating systems, CMW systems have some additional security relevant features that need to be taken into account when considering running an SSH server on one.

Passwords, clearances, user privileges and authorizations

The CMW machines use the normal unix scheme of user names and passwords to authenticate users before allowing them access to the system. There is also a profile file per user that contains additional information about a user such as their clearance and what command authorizations and privileges they hold. The user's encrypted password is also stored in this user profile file, not in /etc/passwd. The profile files are suitably protected to prevent non-privileged access. During login, the system reads the user's profile file and runs a shell for that user with a clearance, base privilege set and authorization list matching those given in the profile file. A port of SSH to CMW must take the user profile file into account.

Server privileges

The SSH daemon process (sshd) needs to be able to carry out privileged operations such as change user id to match that of a particular user, listen on a network port under 1024 and access user profile files. With the coarse grain privilege mechanisms available under standard unix implementations, this normally requires that the sshd process is run as root. Should a bug be found in the sshd code, then potentially this could be exploited by an attacker to compromise the machine. On the other hand, CMW operating systems have a finer grained privileged model and so it is possible to run sshd with only the privileges it needs, and also for those privileges to be active only when absolutely necessary (privilege bracketing).

Figure 2 shows an example of privilege bracketing around a call to the *bind* socket call. The process wishes to bind to a port below 1024; to do this requires the process to hold the *netprivaddr* privilege on CMW. To be as secure as possible, a port of sshd to CMW should include privilege bracketing.

Networking essentials

When communicating with systems that do not label network packets (such as a standard WindowsNT or unix host), the CMW applies a label to a packet as it enters one of its network interfaces.

```
#if defined(HPUX_CMW)
      /*
      ** Need to raise netprivaddr to allow the process to bind a port
      ** below 1024 (a privileged port).
      */
      if(forceprivs(privvec(SEC_NETPRIVADDR, -1), savePrivs)){
          fatal("could not raise netprivaddr.\n");
      }
#endif

      result = bind(listen_sock, (struct sockaddr *)&sin, sizeof(sin));

#if defined(HPUX_CMW)
      /*
      ** No longer need the netprivaddr priv to be active, so drop it.
      */
      if(seteffprivs(savePrivs, NULL) < 0) {
          fatal("couldn't reset privs.\n");
      }
#endif

      if(result < 0){
          error("bind: %.100s", strerror(errno));
          shutdown(listen_sock, 2);
          close(listen_sock);
          fatal("Bind to port %d failed: %.200s.", options.port,
                strerror(errno));
      }
```

Figure 2: Privilege bracketing

The label applied to a packet can be controlled at either the host or network interface level. For example it is possible to say that all packets arriving at lan interface 1, are labelled "TOP SECRET", irrespective of the particular host the packet came from. Equally, it is possible to say that packets comming from host A, say, will be labeled "CONFIDENTIAL".

The CMW machine must be told in advance about any machines that the system manager wants it to communicate with. This is done by creating an entry for that host containing certain parameters in a database held on the CMW. When a packet arrives at an interface, the CMW first makes sure it has an entry for that host. If not then communication is refused. If it does, then it applies the label specified in that host's entry. If the entry doesn't specify a sensitivity level, then the default level of the interface the packet came in on is applied to the packet. Additionally, there can be a wild card host entry. Any host not having a specific entry picks up the parameters of this entry, including the sensitivity level.

The MAC policy and labeled networking features of CMW affect socket communication between an unlabeled host and a CMW SYSTEM at the process level. For example, if a server process wishes to listen for connections on a socket from multiple sensitivity levels, then it must possess a particular privilege.

To be able to communicate with a remote client, the server process must be at the same level as the incoming client connection (this may require the server process to switch sensitivity levels which is a privileged operation) or must possess the necessary privileges to override the system MAC policy directly.

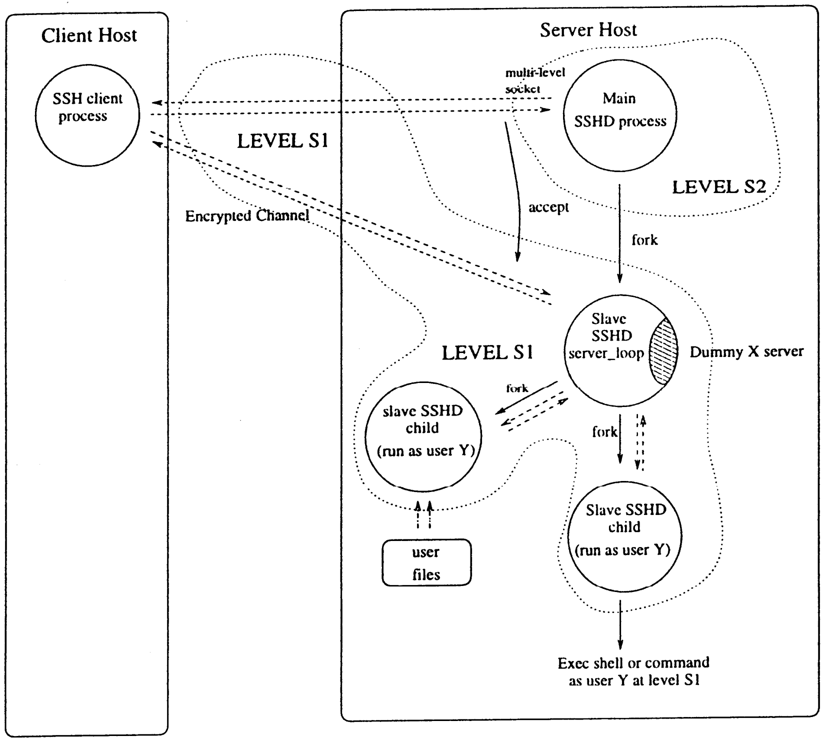

Figure 3: SSH on CMW

3.1 Implementation

Figure 3 shows how we have adapted the SSH implementation so that it can run on CMW. The sshd is given the privilege that allows it to create a multilevel socket. This allows it to accept connections from hosts at more than one level. The slave sshd process spawned to handle a request is done so at the same level as the incoming connection, i.e at the level the client host is labeled at. This allows the slave process to communicate with the client without the CMW MAC controls getting in the way.

When spawning the shell (or command), the system clears the privileges and authorizations it inherited from the initial server process, and instead bestows it with privileges and authorizations assigned to the users. The new shell runs with the clearance specified in the user's profile file.

4 SSH extensions to support multi–level access from unlabeled hosts

The port of SSH to CMW described in section 3.1, save encryption and strong authentication, behaves much the same as the standard CMW rlogin or telnet services. Here too, when accessed from an unlabeled host the user shell is run at the level the client host is labeled at by the CMW. However, to better support activities such as remote CMW management or content update, we can take advantage of the strong authentication and encryption properties of SSH to provide additional functionality.

For example we may wish to limit the levels of hosts a particular user can login from, e.g. if we have a CMW machine connected to both an internal and an external network, we may wish to allow a particular user to login from hosts on the internal network only and not from hosts on the external network

Additionally, to ease system management procedures, we may wish some users to be able to select for themselves what level their shell gets run at and not be bound to whatever level their host connects at. Ideally, the levels they can pick from would be administrator controllable. Alternatively, we may wish particular users to have their shells run at a fixed level but at a level different to the level their network connection came in it.

To provide the functionality just described, we have extended the SSH protocol slightly. Two additional messages have been added to the SSH protocol to give a user the ability (once they have been authenticated) to select a level to run their login shell at, as happens when the user logs in via a terminal directly connected to CMW host. A server side authorization file controls(specifies) the levels a user can use.

```
user=cid:login_mode=FIXED:connect_levels=*:change_levels=UNCLASSIFIED
user=etd:login_mode=SPECIFY:connect_levels=*:change_levels=SECRET TOP SECRET
user=manager:login_mode=FIXED:connect_levels=SECRET:change_levels=TOP SECRET
user=fm:login_mode=CONNECT:connect_levels=*:change_levels=*
```

Figure 4: SSH authorization file

Figure 4 shows an example authorization file. Each line in the file specifies "login_mode", "connect_levels" and "change_levels" for a particular user.

A user's login_mode entry can be one of FIXED, CONNECT or SPECIFY.

In FIXED mode, the system runs the user's shell at the sensivity level given in the change_levels field. In this case, the change_levels field should specify only a single sensitivity level. The connect_levels field lists the permitted levels of hosts that the user can login from.

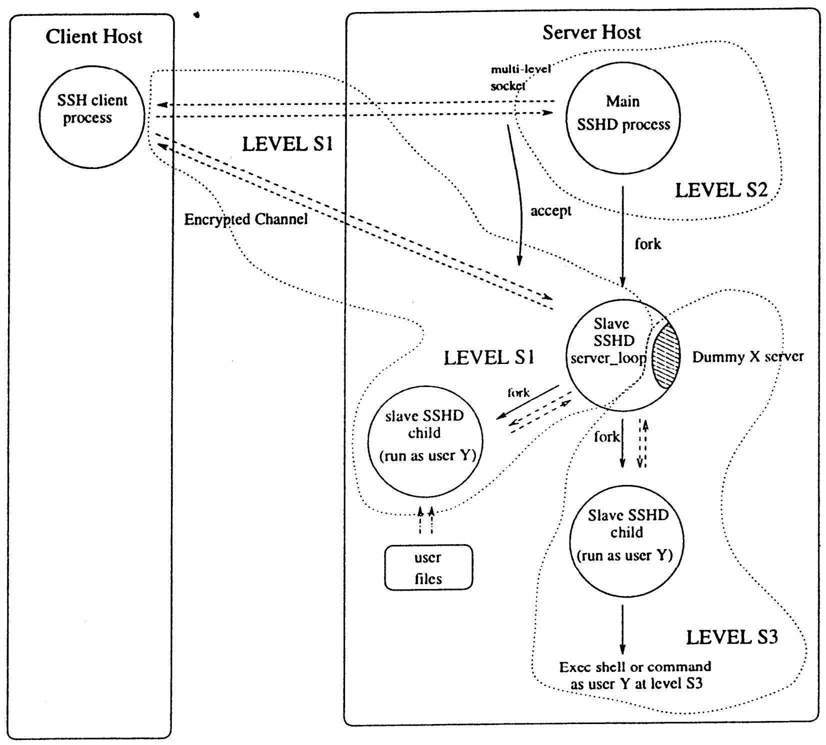

Figure 5: Extended SSH on CMW

In CONNECT mode, the system runs the user's shell at the level of the incoming connection, i.e. the level of the host that the user makes the connection from. This mimics the standard CMW telnet and rlogin behaviour.

In SPECIFY mode, the user is allowed to select the level their shell gets run at from the levels spcified in the change_levels field. Again, the connect_levels field lists the permitted levels of hosts that the user can login from. In order to take advantage of this option, the client side ssh software has to be aware of our additional protocol messages. We have made the necessary modifications to the client software supplied as part of the *www.cs.hut.fi* distribution. However, existing ssh clients that are not aware of the two additional protocol messages can still be used to connect to the CMW based sshd – they just won't have the interactive level query option.

Based on these rules, the example authorization file would force any login shells or commands run by the user "cid" to be run at UNCLASSIFIED, irrespective of the level of the host that user logged in from.

The user "etd" would be given the option of running their shells or commands at either TOP SECRET or SECRET and could also login from hosts with any label.

The user “manager” would have their shell or commands run at TOP SECRET. However, unlike user “cid” or “etd”, they could only log in from hosts labeled as SECRET.

Finally, the entry for user “fm” mimics the behavior of the standard CMW remote login and telnet services. Here the login shell or commands get run at the same level as whatever label the CMW system applies to the host the user connects from.

Implementation

Further modification is required to the sshd code to permit a user’s login shell to be run at a different level to the host level that the user connects from. Figure 5 shows this. The server process is now required to change levels as it reads and writes data to and from the remote client and the users shell process.

The child that execs the user shell is forked at the level specified by the user (or given in the authorization file). The server process when it has data to send to or read from that child does so at the child’s level. When the server process has data to exchange with the remote client, it does so at the network level of the client.

Similarly, when the server process has X11 or other TCP/IP forwarding traffic to send and receive, it must change to the appropriate level.

5 Remote file update

As well as a need for remote terminal and X11, we have also found the need to be able to remotely update files on a CMW host, e.g. when a CMW machine is being used to provide web based services.

The sample implementation discussed in this report comes complete with *scp*. This is a version of the unix file copy program (rcp) that runs on top of ssh. scp can be used to remote copy files from one system to another over an encrypted SSH channel. For more sophisticated file copying and updating (including the ability to delete files on the remote host) the rsync [4] utility can be used in conjunction with SSH. Details and source code for rsync can be found at *ftp://sunsite.auc.dk/pub/unix/rsync/*.

Both scp and rsync work with the extended SSH server presented here. Depending upon on the contents of the server authorization file, the user can either select a level for the files to be copied at, have a level forced on them or have the files given the level matching the incoming host connection.

A WIN32 port of the ssh and the scp programs for WindowsNT and Windows95 is available from *http://bmrc.berkeley.edu/people/chaffee/winntutil.html.*

Additionally, rsync has been ported to the WindowsNT platform by the authors with the aid of the UWIN [5] libraries.

6 Conclusion

We have found that SSH and the extensions we have made to SSH allow us to use CMW systems outside their more usual trusted network environment. With our modifications, we can confidently allow remote login and file copying / update from unlabeled systems to CMW servers at multiple sensitivity levels. If wished standard, unmodified SSH clients can be used with our enhanced SSH server.

The trusted versions of the SSH programs described in the report have been implemented in an experimental form at Hewlett-Packard Laboratories, Bristol.

References

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

[2] HEWLETT-PACKARD CO. (1995). HP-UX CMW MaxSix Administrator's Guide.

[3] 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.*

[4] TRIDGELL, A and MACKERRAS, P. (1996). The rsync algorithm. *The Australian National University Technical Report TR-CS-96-05*

[5] KORN, D.G. (1997). Porting UNIX to WindowsNT. *Proceedings of the Anaheim Usenix conference.*

[6] YLONEN, T (1995). The SSH (Secure Shell) Remote Login Protocol. *RFC nnnn.*

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

[8] NATIONAL COMPUTER SECURITY CENTER (1985). Department of Defense Trusted Computer System Evaluation Criteria. *DOD Standard 5200.28-STD.*

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

A The Compartmented Mode Workstation

Contents of this appendix are taken from "Applying Military Grade Security to the Internet" [3].

The Compartmented Mode Workstation was originally developed for military and government use according to the CMWEC criteria [8] for evaluating trusted systems. The CMW class is an entirely separate but related set of criteria to the more familiar Orange Book criteria [7]. In Orange Book terms, CMW has all of the B1 level security features, and includes a number of B2 and B3 features. A number of the CMW features are relevant to Internet firewalls/application gateways, in particular, mandatory access control (MAC); discretionary access control (DAC); privileges; command authorizations; audit.

The combination of these security features makes CMW especially suitable as an application gateway. Some features make it easier to administer and maintain the gateway machine in a secure state and to detect attempts at attack: the detailed auditing, the command authorizations allowing separation of duty and retirement of the root account, and the trusted execution path combating Trojan horses. Other features make it possible to build and run applications securely: MAC and privileges in particular. This section explains these security features; the remainder of this paper concentrates on the use of these features to develop applications to run securely on CMW while providing access from the Internet to sensitive resources and information.

A.1 Mandatory Access Control

Mandatory access controls are enforced consistently by the operating system – users cannot choose which information will be regulated. On CMW all information has associated with it a sensitivity label. The sensitivity label comprises a 'classification' and a number of 'compartments'. The operating system labels files, processes and network connections. In general, to have read access to some data, a process must have a sensitivity label which 'dominates' the label of the data. A sensitivity label is said to dominate another when its classification is higher or equal to the other's classification, and when it includes all compartments included in the other label. For write access, a process's label must exactly equal the data's label.

In practice, classification is generally used to indicate how secret or sensitive data is. Compartments, however, are often used to partition data so that access to separate sets of data is given to different groups of users, e.g., members of different departments in a company. The configuration shown in figure 6 below uses compartments to distinguish between data and resources accessible from the Internet, and those accessible from a company's internal LAN.

CMW supports MAXSIX trusted networking [2]. When communicating with hosts that are not trusted or do not support labelling, the system automatically attaches sensitivity labels to all packets arriving from or sent to the remote host. The label can be applied according to which interface card the packets arrived on or the Internet Protocol address of the remote host. This combines with the MAC features, so the operating system prevents the remote host communicating with processes at other sensitivity levels and accessing inappropriate information.

A.2 Discretionary Access Control

CMW has discretionary access controls on similar lines to most Unix systems, including read–write–execute protection based on user and group id's and access control lists. The work described here concentrates on the use of mandatory controls; discretionary controls are less relevant. In particular, the DAC features seem less suitable precisely because they are under the control of the user.

A.3 Privileges

On CMW, the root account's special powers are replaced by a large set of individual privileges. The relevant privilege is checked by the kernel whenever a process tries to make a system call which could in some way compromise security. There is a total of approximately 50 different privileges, ranging from fairly harmless to very dangerous.

Some of the most dangerous privileges are those which allow a process to override the MAC, and these must be carefully granted to allow selected traffic to cross the firewall. For safety, we grant privileges only to small relay programs which are specially designed and carefully reviewed. These 'trusted' programs allow information to cross compartment boundaries, so that large pre–existing applications can be safely accessed from sensitivity levels other than their own. The trusted programs must follow the 'least privilege' principle: they raise a privilege only while it is needed for a particular operation and lower it again immediately afterwards.

A.4 Command Authorizations

Command authorizations are the sisters to privileges. They are given to users, whereas privileges are granted to programs. Authorizations allow control over which users are allowed to invoke which trusted programs. By allocating different sets of authorizations to different users, we can achieve separation of duties. No single user has absolute control of the system; rather there are a number of administrative roles with complementary powers.

A.5 Audit

The trusted kernel audits system calls, and trusted applications can audit their own actions using a standard auditing subsystem interface. This auditing cannot be overridden without special privilege. It will normally be configured to log any access denial or insufficient privilege for an attempted operation. Trusted programs can log their actions directly in an easily understood form, so an administrator can track any suspicious behaviour involving overriding MAC without having to decipher long sequences of system calls.

A.6 An Example Configuration

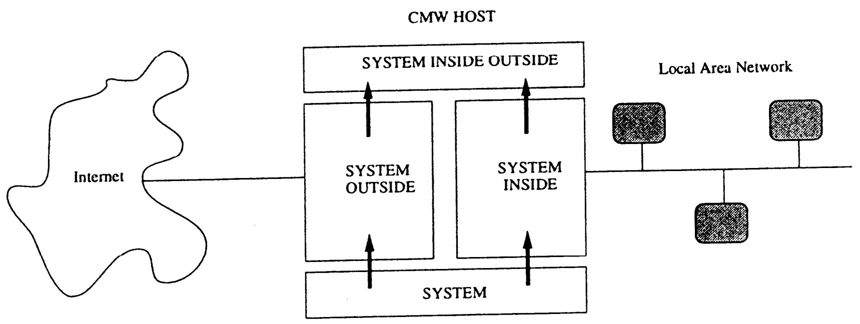

Figure 6: Simple CMW configuration

Figure 6 shows the permitted flows of information in a simple configuration. In this example, there is a single classification level, SYSTEM, and two compartments, INSIDE and OUTSIDE. On such a system, a process labeled SYSTEM OUTSIDE would have read–write access to information labeled SYSTEM OUTSIDE, read–only access to information labeled SYSTEM and would have no access to any other information on the system. A process at SYSTEM INSIDE would have read–write access to an entirely separate set of information labeled SYSTEM INSIDE and read–only access to the information labeled SYSTEM.

As shown in the figure, this configuration can be used for a firewall host with two network interface cards, one connected to the external Internet and the other connected to an internal local area network (LAN). The MAXSIX 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. Even if an attacker from the Internet were to gain access to a shell on the CMW machine, he could not modify files labeled SYSTEM, and could not access files or services labeled SYSTEM INSIDE.