A Generic Proxying Facility for CMW Based on the SOCKS Protocol

Chris I. Dalton
Extended Enterprise Laboratory
HP Laboratories Bristol

HPL-98-100(R.1)
February, 1999

SOCKS, multi-level security, CMW

Our research into the application of *Compartmented Mode Workstation* (CMW)[1] technology has highlighted the need for a general method of allowing networked applications controlled but transparent access across CMW compartments.

This report looks at using the SOCKS TCP relay protocol [2] to provide such a generic proxying facility. We consider the case where a CMW host is situated between an internal and an external network and has network interface connections to both.

First, we show how SOCKS can be used to provide clients on the internal network with access to TCP application services residing on the external network.

Secondly, we consider a more complex configuration where the CMW platform is used to run multiple independent applications that require access to an organization's internal legacy systems. Access by applications to systems on the internal network is controlled using SOCKS.

We end with a discussion on the security implications of using SOCKS and SOCKS on CMW in particular.

1 Introduction

Our research into the application of *Compartmented Mode Workstation* (CMW) [1] technology has highlighted the need for a general method of allowing networked applications controlled but transparent access across CMW compartments.

This report looks at using the SOCKS TCP relay protocol [2][1]to provide such a generic proxying facility. We consider the case where a CMW host is situated between an internal and an external network and has network interface connections to both.

First, we show how SOCKS can be used to provide clients on the internal network with access to TCP application services residing on the external network.

Secondly, we consider a more complex configuration where the CMW platform is used to run multiple independent applications that require access to an organization's internal legacy systems. Access by applications to systems on the internal network is controlled using SOCKS.

We end with a discussion on the security implications of using SOCKS and SOCKS on CMW in particular.

We assume the reader is familiar with CMW concepts and terminology, in particular its use as a firewall type host. If not, then details can be found in [3, 4].

2 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[2]is an implementation of the SOCKS protocol. It consists of a proxying server process and a client library.

The SOCKS server process runs on the intermediate host. It's 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 plugin replacements for the standard TCP socket API calls[3]. 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

[1]SOCKS4.2
[2]Available from ftp://ftp.nec.com/pub/socks/
[3]connect, bind, accept, listen and select.

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 1 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 allowed 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.

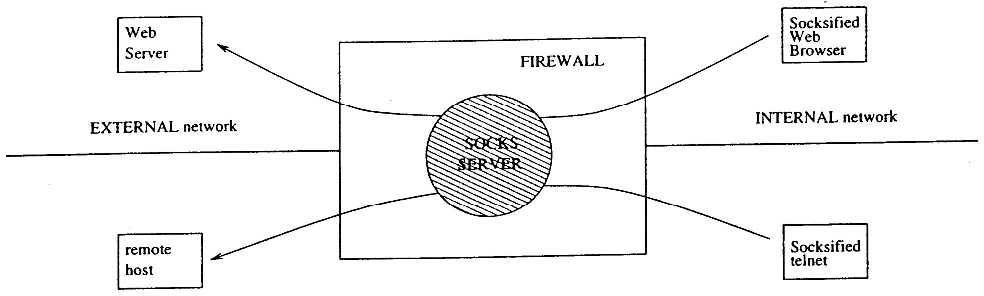

Figure 1: Common SOCKS configuration

2.1 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.

2.2 Logging

The SOCKS server has the ability to log all connection requests and also the number of bytes transferred across a connection.

2.3 Hostname resolution

The hostname to IP address resolution procedure done by the client is unmodified by the SOCKSifying process. This requires that any hostnames that a clients wishes to use must be resolvable by that client. It is not sufficient that the SOCKS server process can resolve the hostnames. The case where the intermediate host is a firewall separating internal and external networks usually implies that the internal clients have access to a DNS name server capable of resolving external addresses. See also section 6.

Further discussion on the use and configuration of the SOCKS toolkit can be found in [5, 6]

3 SOCKS on CMW

3.1 Provision of external application service access for internal clients

This discussion is based on the simple network architecture and CMW configuration of figure 2. It is comprised of a CMW host with two network interface cards, one connected to the external network and the other connected to an internal network. The networking on the CMW is configured such that packets from the external network are labeled SYSTEM OUTSIDE and packets from the internal network are labeled SYSTEM INSIDE. In this configuration, a connection from the external network can communicate only with processes labeled SYSTEM OUTSIDE. A connection from the internal network can communicate only with processes labeled SYSTEM INSIDE The use of CMW in this way creates a clean separation between the external and internal networks. The mandatory access controls ensure packets from one network can not reach the other network.

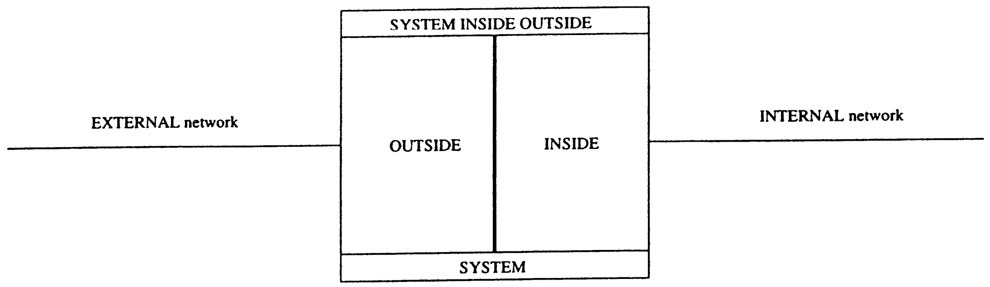

Figure 2: Simple CMW configuration

The problem addressed here is how to allow clients on the internal network access to application services on the external network without compromising the overall system security enhancements that the use of CMW provides. The solution described is based on use of the SOCKS protocol.

3.1.1 A trusted SOCKS server

Figure 3 shows the proposed solution of using SOCKS on CMW. A trusted version of the SOCKS server process runs on the CMW host. It has been trusted with the ability to switch between the SYSTEM OUTSIDE and SYSTEM INSIDE sensitivity levels.

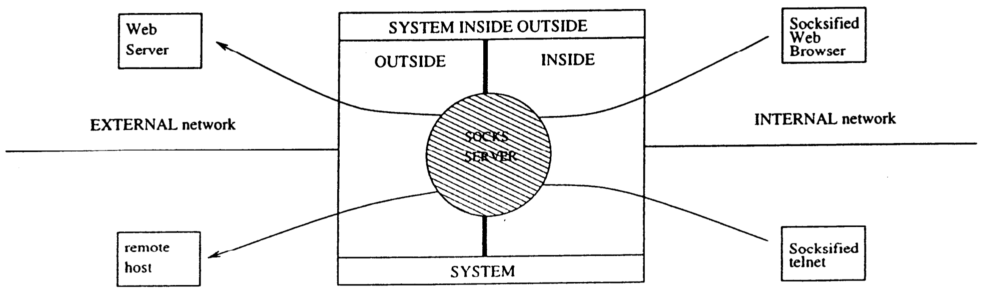

Figure 3: Simple SOCKS on CMW configuration

The socks server loops, listening for connections at the level of the internal network interface(SYSTEM INSIDE).

When it receives a packet from an internal client on this interface, it switches to the level of the external network interface (SYSTEM OUTSIDE) and relays the packet to the requested application server on the external network.

When a packet is received back from the application server on the external network, the SOCKS server switches level to that of the external network interface, reads the packet and switches level back to that of the internal network. The packet is then forwarded to the internal client.

Importantly, the SOCKS client libraries need no modification to work with the trusted CMW SOCKS server and are not trusted themselves. The internal clients are standard SOCKSified clients. For example, a standard web browser that has SOCKS support, such as the Netscape Navigator, can be used on the internal network to connect to web servers out on the external network. The fact that there is a CMW host between them is transparent to both ends of the connection.

3.1.2 Required code changes to the SOCKS server

The code below shows the main processing loop of the socks server. The CMW code that needs to be added to make it function as in figure 3 is contained within the *#ifdef HPUX_CMW* 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 HPUX_CMW
                                        if(setslabel(new_sl)  < 0){
                                            ...
                                            exit(1);
                                        }
#endif
                                        if (write(out, buf, n) < 0){
                                                goto bad;
                                        }
#ifdef HPUX_CMW
                                        if(setslabel(default_sl)  < 0) {
                                            ...
                                            exit(1);
                                        }
#endif
                                } else {
                                        goto bad;
                        }
                   }
                   if (FD_ISSET(out, &fds)) {
                            if(setslabel(new_sl)  < 0) {
#ifdef HPUX_CMW
                                ...
                                exit(1);
                            }
#endif
                            if ((n = read(out, buf, sizeof buf)) > 0) {
                                from_out += n;
#ifdef HPUX_CMW
                                if(setslabel(default_sl)  < 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 CMW initialization routines. However, the total amount of additional code is small.

3.2 Controlled access to internal legacy systems

Figure 4 shows a more complicated configuration, where the CMW platform is used to run multiple independent applications, and applications which directly access an organization's internal legacy systems.

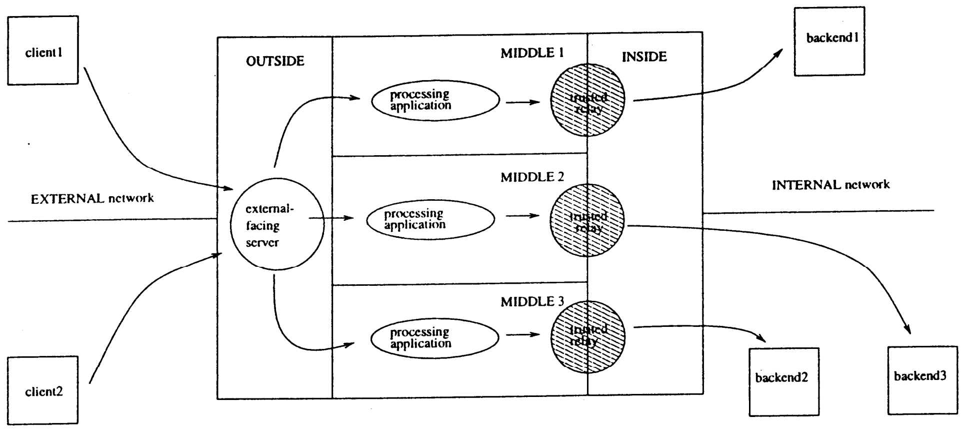

Figure 4: Controlled access to internal systems

By running each application in a separate compartment, it is possible to ensure that even if one application is compromised, it cannot be used to attack or interfere with the other applications running on the same machine. Where an application has to access systems on the internal network, it is not given direct access, but has to talk to them through a trusted relay. The relay can limit which internal machines the application communicates with and also log connection requests.

3.2.1 Using SOCKS to provide the trusted relay functionality

In this section we look at using SOCKS to provide the trusted relay functionality described above. Here, it is the processing applications that are SOCKSified. Access to internal systems by the processing applications is via a SOCKS server, which performs the role of the trusted relay.

We assume that the processing applications within the middle compartments have been designed to use the sockets API for their communications with the systems on the internal network.

It is possible to provide the trusted relay functionality with the modifications made to the SOCKS server code outlined in section 3.1.1. A separate SOCKS server is run at the level of each of the middle application compartments (or at least those that require access to the internal network). Each of these SOCKS servers listens on a different port and each has its own configuration file.

The separate configuration file for each server allows restrictions to be placed on which systems on the internal network a processing application at a particular level can connect to.

The main drawback of this approach is that the SOCKS client software in each compartment has to be modified to connect to a different SOCKS server port. Although this is workable on a system with only a few middle compartments requiring internal system access, it does not scale well.

In order to more flexibly support the required functionality of the trusted relay described above, further modifications must be made to the existing SOCKS server code.

The problem of having to modify SOCKS client software for each compartment can be avoided by running a single SOCKS server that has been trusted to listen for connections at multiple sensitivity levels.

If this is done, additions must be made to the SOCKS server access control mechanisms. The additional mechanisms must allow the SOCKS server to be able to base network access decisions on the sensitivity level of the connection request, as well as on the existing available decision criteria.

In the previous case where a SOCKS server per compartment level is run, no change is necessary to the access control mechanisms as there is a separate SOCKS configuration file per SOCKS server, and hence per compartment level.

3.2.2 Required code changes

Calling the *m6mlserver* [7] function once the listening socket has been opened allows the SOCKS server to accept connections on that socket at multiple sensitivity levels.

To support the additional access control mechanisms, two further changes have been made to the SOCKS server code. Firstly, the server now retrieves the sensitivity level of an incoming connection request. Secondly, an extra (optional) sensitivity label field has been added to the parameters allowed in a SOCKS configuration rule.

The SOCKS client libraries do not require modification.

3.2.3 Hostname resolution issues

A process running in one of the middle compartments must be able to resolve any hostnames that it wishes to use. If the number of hosts on the internal network to which access is required are small, then using entries from a coded hosts table (/etc/hosts) would probably be sufficient. For larger scale systems, the multi-level CMW dns software [3] developed by the author can be used to provide each compartment level with access to a DNS service.

4 Security issues associated with the use of SOCKS

4.1 SOCKS in general

The major security vulnerability that the use of SOCKS introduces is that of allowing potentially unsafe application protocols into the internal network where client platforms may then be compromised.

Since SOCKS is a circuit level proxy, it can do nothing to provide protection at the application level, other than refusing to proxy a particular application protocol.

By using SOCKS, therefore, clients can become vulnerable to both command driven attacks and data driven attacks. Command driven attacks are possible where a protocol, such as X11, allow inherently dangerous commands to be passed in the protocol stream to the application client. SOCKS cannot check for this. Data driven attacks are possible, for example, where data is sent to the client and is then executed on the client platform. Protocols such as http allow this. Again SOCKS cannot protect against this.

Allowing internal clients access to external DNS information can also create a security vulnerabilty [3].

Despite the vulnerabilties and threats associated with the use of SOCKS, many sites consider the risks acceptable given the security safeguards that the use of SOCKS introduces.

The main safeguard is ability to have only a single host directly exposed to a potentially hostile external network. The use of SOCKS enables this whilst still allowing internal clients transparent access to external application services with the associated minimal inconvenience to users.

4.2 SOCKS on CMW

The issue discussed here is that of whether the inherent security that the CMW platform offers is significantly compromised by running a SOCKS server on it. The problems with application protocol based attacks still apply when the SOCKS server host platform is a CMW machine. Client side use of CMW can help protect against these attacks but is not considered here.

When run on the CMW platform, the SOCKS server process needs to be given certain privileges. The most potentially dangerous privilege it requires is the ability to switch sensitivity levels. This is needed so that it can talk to both the external and internal networks; it allows the process to circumvent some of the mandatory access controls. It becomes the responsibility of the SOCKS server to uphold a security policy. Should an attacker find a way of exploiting a bug in the server code, then that attacker can make use of any privileges the code has, and is not bound by the SOCKS server security policy.

SOCKS is the most widely used proxying package on the Internet. It is a relatively small amount of code and has been well tested by the Internet community. The potential for exploitation of bugs in the SOCKS server code by an attacker appears small.

Only minor changes to the SOCKS server code are needed for it to be able to support CMW operation. This should not increase the potential for exploitation of bugs in the SOCKS server code. Therefore, it is thought that running a SOCKS server on a CMW platform does not significantly degrade the overall security of that platform.

5 Conclusion

We have found that the use of SOCKS as a base for providing a generic proxying facility for CMW is feasible. The code changes required to the existing SOCKS toolkit are small and therefore verifiable. The inherent security of the CMW platform is not significantly reduced by running a SOCKS server on it.

The trusted versions of the SOCKS server described in the report have been implemented in an experimental form at Hewlett-Packard Laboratories, Bristol as part of the E2S European collaborative.

6 Future work

This report discusses software developed based upon the SOCKS4 protocol. However, an extended, though less widespread, version of the SOCKS protocol, version 5, is also available [8]. Work currently being carried out by the author includes development of a trusted SOCKS server based on the SOCKS5 protocol. The SOCKS5 protocol has several major enhancements over SOCKS4. It includes support for UDP as well as TCP network clients. Hostnames can be passed by the SOCKS clients to the SOCKS server for resolution, thus alleviating a major problem with SOCKS4. Support for authentication schemes such as kerberos is included. Shared libraries for most platforms are available, hence avoiding the need for client re-compilation. Performance has also been improved by the use of multithreaded code.

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

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

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

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

[7] HEWLETT-PACKARD CO. (1996). HP–UX 10.16 CMW Security Features Programmer's Guide.

[8] LEACH, M., GANIS, M., LEE, Y. and KURIS, R. (1996). SOCKS Protocol Version 5. *RFC 1928.*