NEW TECHNIQUES TO ENHANCE THE CAPABILITIES OF THE SOCKS NETWORK SECURITY PROTOCOL

Mukund Sundararajan and Mohammad S. Obaidat
Computer Science Department, Monmouth University, West Long Branch, NJ, U.S.A.

Keywords: Security protocols for computer networks, SOCKS, telecommunications, multicast, UDP tunneling.

Abstract: SOCKS is an industry standard network security protocol used in private networks to allow secure traversal of application layer traffic through the boundaries of the network. Standardized by IETF in Request for Comments (RFC) 1928 (Leech et al., 1996) as SOCKS Version 5, this protocol has found widespread use in various security frameworks to allow a variety of application layer protocols to securely traverse a firewall. This paper is the result of research performed on the usability of the protocol in application domains such as multicast. We discuss some of the shortcomings of the SOCKS protocol and provide a framework and the methods for enhancing the capabilities of the protocol in areas such as multicast and advanced TCP and UDP capabilities not addressed by the current standard of the protocol. The methods proposed are being implemented in a reference implementation by the authors.

1 INTRODUCTION

In today's global and geographically dispersed organizational world, network security is a key concern to organizations and individuals. With advances in technology, most of today's organizations have their key resources and data distributed across powerful transactional databases, Lightweight Directory Access Protocol (LDAP) directories and N-tier application servers. There is also an increasing use of tools for collaboration like instant messaging, voice and video conferencing over IP and unified communications portals. With a large amount of confidential data flowing through the private network, it is important to ensure that none of this data leaves the network in an un-intended way. It is also equally important to ensure that only trusted and reliable data enters into the private network from the public Internet.

The SOCKS protocol was developed to allow application layer protocols to securely leave and enter a private network. Typically deployed at the network boundary in a firewall, it allows an application gaining access into a network to authenticate itself and authorizes the application to access the network resource it is trying to access. It protects a wide range of TCP based applications like telnet, http, ftp and other TCP based application specific protocols.

Operating in a client server mode, application nodes or computers within a SOCKS protected network are 'socksified' by a socks client library that provides a transparent abstraction layer between the application and the kernel socket library and hides the implementation details of the socks protocol from the application. When an application attempts to make a socket connection to a node or computer outside the local network, the socks client library intercepts the call. It then sends request-response style messages to the socks server configured for the client to use. The socks server challenges the client to authenticate itself. The socks client can be pre-configured to provide the authentication credentials or can prompt the application or user to do so. The SOCKS protocol supports a variety of authentication modes from simple username-password authentication to sophisticated models like Kerberos. Once authenticated and authorized the socks server allows the client to connect to the remote node while acting as a proxy server for the connection.

2 RELATED WORK

There are certain inherent limitiations with the SOCKS protocol which makes it less suitable for securing certain applications like multicast and IP telephony. Fung and Chang (Fung and Chang, 2000)

Sundararajan M. and S. Obaidat M. (2008).
NEW TECHNIQUES TO ENHANCE THE CAPABILITIES OF THE SOCKS NETWORK SECURITY PROTOCOL.
In *Proceedings of the International Conference on Security and Cryptography*, pages 197-201
DOI: 10.5220/0001930001970201

in their work on allowing multimedia streams through a firewall have proposed one framework for allowing protocols like RTP and RTSP to securely pass through a firewall. Djahandari and Sterne (1997) discussed the problems with security in the MBone and with multicast nodes and provided an approach for Internet firewalls to pass trusted MBone traffic into and out of a local area network. Experimental studies have also been conducted (Vaidya et al., 2005) on securing audio streams (RTP over UDP) through IP tunnels and IPSec protocol that is used in VPNs.

3 IP MULTICAST

Multicast allows a sender to send one copy of a packet to several recipients without duplication of data in the network. In contrast to broadcast this optimizes the use of network resources and has widespread use in a variety of applications. Internet newsgroups, videoconferencing over a TCP/IP network, Internet games, streaming media and video on demand, distance learning and many more applications are built using IP multicast. Because of its nature, multicast is run over UDP and multicast packets are sent as UDP datagrams. Special multicast routers in the Internet, called the public M-bone, deal with the job of maintaining multicast group information and multicast routing. Special protocols like Internet Group Management Protocol (Cain, 2002) (IGMP) and Multicast Listener Discovery Protocol (Vida and Costa, 2004) (MLD) have been developed for end points or nodes to join multicast groups to send/receive multicast data and to leave groups.

Securing multicast sessions is challenging because of the very nature of the traffic where there can be many senders and many recipients. The issues involved in securing multicast through firewalls are documented in RFC 2588 (Finlayson, 1999). Gong and Shacham (1995) discuss the security issues involved in multicasting especially in a mobile environment. For security reasons, most firewalls block multicast traffic from entering or leaving their network.

4 SECURING MULTICAST THROUGH SOCKS

One of the techniques we have identified to securely allow multicast traffic to traverse a network boundary is to provide multicast extensions to the socks protocol. While a draft proposal was made to add multicast support to socks, it was not well supported and was never implemented or standardized. The current standard of the SOCKS protocol, Version 5, as documented in RFC 1928 (Leech et al., 1996) does not have multicast support. We provide here a new technique that has not been previously implemented and can provide multicast capabilities to socks.

5 PROPOSED SCHEMES

5.1 Multicasting

Our proposed scheme for multicast applies to nodes in a network that are capable of multicasting and to a network that has multicast routing capabilities. Our technique consists of adding two new request methods to the original SOCKS protocol. Readers are referred to RFC 1928 (Leech et al., 1996) for a complete description of the original SOCKS V5 protocol and the request methods supported.

There are three phases in the SOCKS protocol: (a) authentication method negotiation, (b) authentication, and (c) request negotiation.

Our extension to the protocol alters the third phase, the request negotiation phase. We propose the addition of a new '**JOIN_MCAST_GROUP**' and a new '**LEAVE_MCAST_GROUP**' request method to allow clients to securely join and leave multicast groups through socks. All extensions we propose will require changes to the socks client and the socks server. Client computers or nodes will have to be re-socksified with the new enhanced client to take advantage of the proposed extensions. Our reference implementation provides a new socks client and a new socks server for testing the proposed extensions in a laboratory test bed. The socks client is being built using the new socket extensions proposed for multicast source filters in RFC 3678 (Thaler et al., 2004).

5.2 JOIN_MCAST_GROUP

When an application within a network firewall wishes to join a multicast group in the public Internet, the socks client must authenticate itself to the socks server and send a JOIN_MCAST_GROUP request packet to the socks server. The packet will contain the address of the multicast group. The structure of this packet will be as follows:

VER|CMD|RSV|ATYP|M.ADDR|M.PORT

Where:
VER – SOCKS protocol version.
CMD – 0x05 for JOIN_MCAST_GROUP
RSV – Reserved
ATYP – IPv4 or IPv6.
M.ADDR – Address of MCAST group
M.PORT – MCAST port.

When a client in the network wishes to join a multicast group, the socks server will first run security policies on the provided address. The security policies can allow the client to receive or send data to the group or deny both privileges. If denied access to the group, the socks server sends a REJECT as a response back to the client and closes the connection with the client. The structure of a response packet is shown below:

VER|REP|RSV|ATYP|ADDR|PORT|LEN|KEY

Where:
VER – SOCKS protocol version
REP – Response code –
0x00: Success.
0xB0: Un-authorized to send.
0xB1: Un-authorized to receive.
0xB2: Un-authorized to send and Receive.
RSV – Reserved
ATYP – Address type IPv4 or IPv6
ADDR – IP Address of multicast relay process if authorized.
PORT – Port for multicast relay process.
LEN – The length of the next field
KEY – A variable length encryption key.

If the security policies in place for the client and the group allow the client to send and receive data from the group, the socks server will send a response packet as shown above with the REP field populated with 0x00. This code indicates that the client is authorized to send and receive from the group. If the client is allowed to only send or receive a different response code as identified above, then it will be sent back.

If the client is the first node to request access to the group, the socks server will spawn a new multicast relay process. This process will be given the address of the group that the client is interested in joining. The process will act as a multicast server on the intranet and pass back to the socks server a multicast address that the client must use to send and receive data. This multicast address is different from the address of the group on the Internet and is visible only to the intranet. The socks server passes back to the client the address and port of the multicast relay process in the ADDR and PORT fields of the response packet. The client must use this address and port to send and receive data. In addition, the socks server gives to the client in the same response an encryption key that the socks client must use to encrypt each multicast packet sent to the relay process. This is an additional security measure to prevent other nodes on the network from sending data directly to the relay process, bypassing the socks server. The relay process decrypts packets it receives from all the clients in the intranet that have joined the multicast group that the relay process represents. The actual length of the key, the type of key and encryption algorithm used can be implementation specific and can be configured in the system at run time. The choice of the encryption algorithm must be such that it does not cause a huge performance impact either on the server or on the client. The system can also be configured to use no encryption at all by specifying a key of length 0 bytes. In the reference implementation, a symmetric key will be used. The relay process proxies multicast traffic between the real multicast group on the Internet and its registered listeners on the local intranet.

The socks server actively manages the relay process. When all clients have left the group or all client sessions have timed out due to lack of activity for a long time, the relay process is terminated. The process is recreated when there is a request to join the same group again. Policies can also be put in place to control the number of clients that can join a group, control which clients can send data, which clients can receive data and which clients can do both. Policies can also control if and when a multicast relay should be terminated due to excessive use of network bandwidth or if spamming is detected.

5.3 LEAVE_MCAST_GROUP

A socks client joins a multicast group by opening a TCP connection to the socks server configured for the network. After the socks protocol performs handshaking and authentication, it sends a request packet to the socks server with the command set to JOIN_MCAST_GROUP. Once authorized to join the group, the socks client sends and receives multicast data from a multicast relay process setup by the socks server for all nodes in the network that have joined the group.

While the client has a separate channel of communication with the multicast relay process, the TCP connection with the socks server is kept active. The socks server can at any time terminate the control channel and relay process with an appropriate error message. Reasons for doing so could be a timeout due inactivity or that the client has exceeded the amount of time it can be in the group, as dictated by the security policy in place for the client or the group. This control channel also enables the client to send more request messages to the socks server. One such message that a client can send is the LEAVE_MCAST_GROUP message.

This message enables a socks client to terminate its multicast session by sending this message to the socks server on its TCP control channel. When the socks server receives this method it passes it on to the relay process, which removes the client from its cache of clients for the group it is representing. When the last client has left the group the relay process leaves the group by sending an IGMP or MLD message to the MBone and terminates itself.

5.4 UDP Tunneling

Another extension we propose for the SOCKS protocol is UDP tunneling. Here the socks server would relay UDP datagrams emanating from an application client on the local network securely through the public Internet to an application server or node on another protected network. This technique can be used to establish a TLS/SSL tunnel between two SOCKS servers implementing our proposed extensions avoiding the need for expensive VPN/IPSec tunnels. Figure 1 describes a deployment scenario.

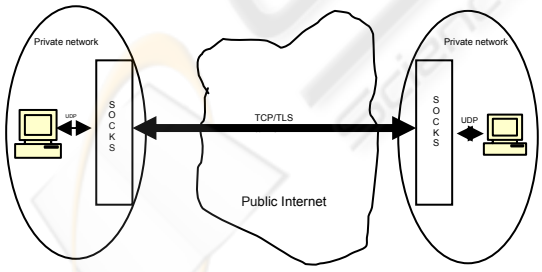

Figure 1: Deployment scenario.

This extension requires the addition of a new request method to the SOCKS protocol called SETUP_UDP_TUNNEL. The request packet will contain the address, port and transport protocol for the end point of the tunnel. When the socks server receives such a command in a request packet from a client, it first applies security policies on the client. If authorized to open a secure tunnel to another node on another protected network, the socks server will spawn a UDP relay process for the client.

This relay process will open a tunnel using the transport protocol specified in the request message to another socks server on the Internet. The local socks server appears as a client to the remote socks server and authenticates itself to the remote socks server. The credentials for authentication must be passed on to the local socks server by the socks client. Upon successful authentication the two socks servers would effectively be tunneling data between two application nodes on their local networks.

6 TCP BIND EXTENSIONS

SOCKS V5, allows a client to specify a TCP port for the socks server to bind and wait for a remote node to connect. This helps protocols like FTP which has a TCP control channel and requires the client to bind to another port for sending and receiving data from the FTP server. This method will enable applications to have one control channel and a reverse channel for the server to connect back to the client. This does not serve the needs of applications that require one control channel and several reverse channels for the server to connect back to the client. Furthermore these reverse channels could be TCP or UDP channels.

We propose an extension to the socks protocol to allow a client to request the server to bind to more than one TCP port or setup one or more UDP relay processes for the remote end point to connect or send datagrams to. This will help custom applications to have one TCP control channel and several other TCP or UDP channels for application sessions. A case in point is protocols like RTP and RTSP for multimedia streaming. While RTP and RTSP protocols run on top of UDP, the SOCKS version 5 call model does not have enough support for these protocols to seamlessly proxy through a SOCKS firewall. Allowing multiple TCP binds or UDP relays on behalf of a multimedia streaming application can greatly enhance the usability of the protocol for multimedia.

7 DISCUSSION AND CONCLUSIONS

Our proposed extensions to the SOCKS protocol could lead to a new version of the protocol, which when implemented, would greatly enhance the capabilities of the protocol in securing and auditing application layer traffic. With the convergence of voice, video and data and the global adoption of IP as a low cost medium for conducting multimedia communications, it is highly valuable to add multicast capabilities to a proven protocol like SOCKS. While the new features we are implementing will add value to the protocol, the performance of the system will only be limited by the network with no additional performance overhead added by the new extensions. The application of security policies and the initial setup of the channel through socks might cause some delay in the setup phase when an acpplication is reaching out to the socks server to cross the network boundary but once authenticated and authorized, there will be no further delays added by socks. We expect the proposed multicast relay process in socks to improve the performance of multicast routing within the network compared to other schemes where multicast packets would need to be routed in a unicast fashion to listeners in the protected network. We expect the benefits of adding UDP tunneling and multiple TCP and UDP bind support in the protocol to be well received in a variety of application domains, especially in the N-tier application server domains and IP telephony.

REFERENCES

Leech, M., Janis, M., Lee, Y., Kuris, R., Koblas, D., Jones, L.,"RFC 1928, SOCKS Protocol Version 5", www.ietf.org/rfc/rfc1928.txt, March 1996.

Fung, K.,P., Chang, R.,K.,C., "A Transport-Level Proxy for Secure Multimedia Streams", IEEE Internet Computing, pp. 57-67, November 2000.

Djahandari, K.,Sterne, D., "An MBone Proxy for an Application Gateway Firewall", Proceedings of the 1997 IEEE Symposium on Security and Privacy, pp. 72-78, 1997.

Vaidya, B., Kim, J.,W., Pyun, J.,Y., Park, J., Han, S., "framework for Secure Audio Streaming to Wireless Access Network", 2005 Systems communications, pp. 122-127, 2005.

Cain, B., Deering, S., Kouvelas, I., Fenner, B., Thyagarajan, A., "Internet Group Management Protocol, Version 3", RFC 3376, www.ietf.org/rfc/rfc3376.txt, October 2002.

Vida, R., Costa, L., "Multicast Listener Discovery V2 (MLDv2) for IPv6". Internet Engineering Task Force (IETF), RFC 3810, www.ietf.org/rfc/rfc3810.txt, June 2004.

Finlayson, R., "RFC 2588, IP Multicast and firewalls", www.ietf.org/rfc/rfc2588.txt, May 1999.

Gong, L., Shacham, N., "Multicast security and its extension to a mobile environment", Wireless Networks, ACM-Baltzer, Vol. 1, No. 3, pp. 281 – 295, August 1995.

Thaler, D., Fenner, B., Quinn, B., "Socket interface extensions for multicast source filters", RFC 3678, www.ietf.org/rfc/rfc3678.txt, January 2004.

Mazumder, A.S., Almeroth, K., Sarac, K., "Facilitating robust multicast group management", Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video, pp. 183-188, 2005.