WIKIPEDIA

SOCKS

Socket Secure (**SOCKS**) is an Internet protocol that exchanges network packets between a client and server through a proxy server. **SOCKS5** additionally provides authentication so only authorized users may access a server. Practically, a SOCKS server proxies TCP connections to an arbitrary IP address, and provides a means for UDP packets to be forwarded.

SOCKS performs at Layer 5 of the OSI model (the session layer, an intermediate layer between the presentation layer and the transport layer). SOCKS server accepts incoming client connection on TCP port 1080.[1][2]

Contents

- **1 History**
- **2 Usage**
- **3 Use in cybercrime**
- **4 Comparison to HTTP proxying**
  - 4.1 SOCKS
  - 4.2 HTTP
- **5 Protocol**
  - 5.1 SOCKS4
  - 5.2 SOCKS4a
  - 5.3 SOCKS5
- **6 Software**
  - 6.1 Servers
    - 6.1.1 SOCKS proxy server implementations
    - 6.1.2 Other programs providing SOCKS server interface
  - 6.2 Clients
    - 6.2.1 Proxifiers
      - 6.2.1.1 Translating proxies
- **7 References**
- **8 External links**

History

The protocol was originally developed/designed by David Koblas, a system administrator of MIPS Computer Systems. After MIPS was taken over by Silicon Graphics in 1992, Koblas presented a paper on SOCKS at that year's Usenix Security Symposium, making SOCKS publicly available.[3] The protocol was extended to version 4 by Ying-Da Lee of NEC.

The SOCKS reference architecture and client are owned by Permeo Technologies,[4] a spin-off from NEC. (Blue Coat Systems bought out *Permeo Technologies*.)

The SOCKS5 protocol was originally a security protocol that made firewalls and other security products easier to administer. It was approved by the IETF in 1996.[1] The protocol was developed in collaboration with Aventail Corporation, which markets the technology outside of Asia.[5]

Usage

SOCKS is a *de facto* standard for circuit-level gateways.

Another use of *SOCKS* is as a circumvention tool, allowing traffic to bypass Internet filtering to access content otherwise blocked, e.g., by governments, workplaces, schools, and country-specific web services.[6]

Some SSH suites, such as OpenSSH, support dynamic port forwarding that allows the user to create a local SOCKS proxy.[7] This can free the user from the limitations of connecting only to a predefined remote port and server.

The Tor onion proxy software presents a SOCKS interface to its clients.[8]

Use in cybercrime

Hacked computers may be configured as SOCKS proxy software obscuring control of a botnet or to optimise credit card fraud.[9][10][11][12]

Comparison to HTTP proxying

SOCKS operates at a lower level than HTTP proxying: SOCKS uses a handshake protocol to inform the proxy software about the connection that the client is trying to make, and then acts as transparently as possible, whereas a regular proxy may interpret and rewrite headers (say, to employ another underlying protocol, such as FTP; however, an HTTP proxy simply forwards an HTTP request to the desired HTTP server). Though HTTP proxying has a different usage model in mind, the CONNECT method allows for forwarding TCP connections; however, SOCKS proxies can also forward UDP traffic and work in reverse, while HTTP proxies cannot. HTTP proxies are traditionally more aware of the HTTP protocol, performing higher-level filtering (though that usually only applies to GET and POST methods, not the CONNECT method).

SOCKS

Bill wishes to communicate with Jane over the internet, but a firewall between them exists on his network, where Bill is not authorized to communicate with Jane directly. So, Bill connects to the SOCKS proxy on his network, informing it about the connection he wishes to make to Jane; the SOCKS proxy opens a connection through the firewall and facilitates the communication between Bill and Jane.

For more information on the technical specifics of the SOCKS protocol, see the sections below.

HTTP

Bill wishes to download a web page from Jane, who runs a web server. Bill cannot directly connect to Jane's server, as a firewall has been put in place on his network. In order to communicate with the server, Bill connects to his network's HTTP proxy. His web browser communicates with the proxy in exactly the same way that it would directly with Jane's server if that was possible; that is, it sends a standard HTTP request header. The HTTP proxy connects to Jane's server, and then transmits back to Bill any data that Jane's server returns.[13]

Protocol

SOCKS4

A typical SOCKS4 connection request looks like this:

SOCKS client to SOCKS server:

- field 1: SOCKS version number, 1 byte, must be 0x04 for this version
- field 2: command code, 1 byte:
  - 0x01 = establish a TCP/IP stream connection
  - 0x02 = establish a TCP/IP port binding
- field 3: port number, 2 bytes (in network byte order)
- field 4: IP address, 4 bytes (in network byte order)
- field 5: the user ID string, variable length, terminated with a null (0x00)

SOCKS server to SOCKS client:

- field 1: null byte
- field 2: status, 1 byte:
  - 0x5A = request granted
  - 0x5B = request rejected or failed
  - 0x5C = request failed because client is not running identd (or not reachable from the server)
  - 0x5D = request failed because client's identd could not confirm the user ID string in the request
- field 3: 2 arbitrary bytes, which should be ignored
- field 4: 4 arbitrary bytes, which should be ignored

For example, this a SOCKS4 request to connect Fred to 66.102.7.99:80, the server replies with an "OK":

- Client: 0x04 | 0x01 | 0x00 0x50 | 0x42 0x66 0x07 0x63 | 0x46 0x72 0x65 0x64 0x00
  - The last field is "Fred" in ASCII, followed by a null byte.
- Server: 0x00 | 0x5A | 0xXX 0xXX | 0xXX 0xXX 0xXX 0xXX
  - 0xXX can be any byte value. The SOCKS4 protocol specifies that the values of these bytes should be ignored.

From this point onwards, any data sent from the SOCKS client to the SOCKS server is relayed to 66.102.7.99, and vice versa.

The command field may be 0x01 for "connect" or 0x02 for "bind"; the "bind" command allows incoming connections for protocols such as active FTP.

SOCKS4a

SOCKS4a extends the SOCKS4 protocol to allow a client to specify a destination domain name rather than an IP address;[14] this is useful when the client itself cannot resolve the destination host's domain name to an IP address.

The client should set the first three bytes of DSTIP to NULL and the last byte to a non-zero value. (This corresponds to IP address 0.0.0.x, with x nonzero, an inadmissible destination address and thus should never occur if the client can resolve the domain name.) Following the NULL byte terminating USERID, the client must send the destination domain name and terminate it with another NULL byte. This is used for both "connect" and "bind" requests.

Client to SOCKS server:

- field 1: SOCKS version number, 1 byte, must be 0x04 for this version
- field 2: command code, 1 byte:
  - 0x01 = establish a TCP/IP stream connection
  - 0x02 = establish a TCP/IP port binding
- field 3: port number, 2 bytes
- field 4: deliberate invalid IP address, 4 bytes, first three must be 0x00 and the last one must not be 0x00
- field 5: the user ID string, variable length, terminated with a null (0x00)
- field 6: the domain name of the host to contact, variable length, terminated with a null (0x00)

Server to SOCKS client:

- field 1: null byte
- field 2: status, 1 byte:
  - 0x5A = request granted
  - 0x5B = request rejected or failed
  - 0x5C = request failed because client is not running identd (or not reachable from the server)
  - 0x5D = request failed because client's identd could not confirm the user ID string in the request
- field 3: port number, 2 bytes (in network byte order)
- field 4: IP address, 4 bytes (in network byte order)

A server using protocol SOCKS4a must check the DSTIP in the request packet. If it represents address 0.0.0.x with nonzero x, the server must read in the domain name that the client sends in the packet. The server should resolve the domain name and make connection to the destination host if it can.

SOCKS5

The SOCKS5 protocol is defined in RFC 1928. It is an extension of the SOCKS4 protocol; it offers more choices for authentication and adds support for IPv6 and UDP, the latter of which can be used for DNS lookups. The initial handshake consists of the following:

- Client connects and sends a greeting, which includes a list of authentication methods supported.
- Server chooses one of the methods (or sends a failure response if none of them are acceptable).
- Several messages may now pass between the client and the server, depending on the authentication method chosen.
- Client sends a connection request similar to SOCKS4.
- Server responds similar to SOCKS4.

The authentication methods supported are numbered as follows:

- 0x00: No authentication
- 0x01: GSSAPI[15]
- 0x02: Username/password[16]
- 0x03–0x7F: methods assigned by IANA[17]
- 0x80–0xFE: methods reserved for private use

The initial greeting from the client is

- field 1: SOCKS version, 1 byte (0x05 for this version)
- field 2: number of authentication methods supported, 1 byte
- field 3: authentication methods, variable length, 1 byte per method supported

The server's choice is communicated:

- field 1: SOCKS version, 1 byte (0x05 for this version)
- field 2: chosen authentication method, 1 byte, or 0xFF if no acceptable methods were offered

The subsequent authentication is method-dependent. Username and password authentication (method 0x02) is described in RFC 1929:

For username/password authentication the client's authentication request is

- field 1: version number, 1 byte (0x01 for current version of username/password authentication)
- field 2: username length, 1 byte
- field 3: username, 1–255 bytes
- field 4: password length, 1 byte
- field 5: password, 1–255 bytes

Server response for username/password authentication:

- field 1: version, 1 byte (0x01 for current version of username/password authentication)
- field 2: status code, 1 byte
  - 0x00: success
  - any other value is a failure, connection must be closed

The client's connection request is

- field 1: SOCKS version number, 1 byte (0x05 for this version)
- field 2: command code, 1 byte:
  - 0x01: establish a TCP/IP stream connection

  - 0x02: establish a TCP/IP port binding
  - 0x03: associate a UDP port
- field 3: reserved, must be 0x00, 1 byte
- field 4: address type, 1 byte:
  - 0x01: IPv4 address
  - 0x03: Domain name
  - 0x04: IPv6 address
- field 5: destination address of
  - 4 bytes for IPv4 address
  - 1 byte of name length followed by 1–255 bytes the domain name
  - 16 bytes for IPv6 address
- field 6: port number in a network byte order, 2 bytes

Server response:

- field 1: SOCKS protocol version, 1 byte (0x05 for this version)
- field 2: status, 1 byte:
  - 0x00: request granted
  - 0x01: general failure
  - 0x02: connection not allowed by ruleset
  - 0x03: network unreachable
  - 0x04: host unreachable
  - 0x05: connection refused by destination host
  - 0x06: TTL expired
  - 0x07: command not supported / protocol error
  - 0x08: address type not supported
- field 3: reserved, must be 0x00, 1 byte
- field 4: address type, 1 byte:
  - 0x01: IPv4 address
  - 0x03: Domain name
  - 0x04: IPv6 address
- field 5: server bound address of
  - 4 bytes for IPv4 address
  - 1 byte of name length followed by 1–255 bytes the domain name
  - 16 bytes for IPv6 address
- field 6: server bound port number in a network byte order, 2 bytes

Software

Servers

SOCKS proxy server implementations

- Antinat (http://antinat.sourceforge.net/) is a flexible SOCKS server and client library. It supports SOCKS 4, SOCKS 4a, SOCKS 5, authentication, CHAP, XML firewalling, Win32, server chaining, and UDP. It also contains very experimental IPv6 support.
- Dante (http://www.inet.no/dante/) is a circuit-level SOCKS server that can be used to provide convenient and secure network connectivity, requiring only the host Dante runs on to have external network connectivity.
- Srelay (http://socks-relay.sourceforge.net/) is a small SOCKS4/SOCKS5 proxy server.
- SS5 (http://ss5.sourceforge.net/) SS5 is a socks server that implements the SOCKS v4 and v5 protocol.
- Sun Java System Web Proxy Server is a caching proxy server running on Solaris, Linux and Windows servers that supports HTTPS, NSAPI I/O filters, dynamic reconfiguration, SOCKSv5 and reverse proxy.
- WinGate is a multi-protocol proxy server and SOCKS server for Microsoft Windows which supports SOCKS4, SOCKS4a and SOCKS5 (including UDP-ASSOCIATE and GSSAPI auth). It also supports handing over SOCKS connections to the HTTP proxy, so can cache and scan HTTP over SOCKS.

Other programs providing SOCKS server interface

- OpenSSH allows dynamic creation of tunnels, specified via a subset of the SOCKS protocol, supporting the CONNECT command.
- PuTTY is a Win32 SSH client that supports local creation of SOCKS (dynamic) tunnels through remote SSH servers.
- ShimmerCat[18] is a web server that uses SOCKS5 to simulate an internal network, allowing web developers to test their local sites without modifying their /etc/hosts file.
- Tor is a system intended to enable online anonymity. Tor offers a SOCKS server interface to its clients.

Clients

Client software must have native SOCKS support in order to connect through SOCKS. There are programs that allow users to circumvent such limitations:

Proxifiers

- Comparison of proxifiers

Translating proxies

- Polipo, a fast, lightweight, forwarding and caching proxy server with IPv6 support. Open Source running on GNU/Linux, OpenWrt, Windows, Mac OS X, and FreeBSD. Almost any Web browser can use it.
- Privoxy
- socat

References

1. RFC 1928
2. "Service Name and Transport Protocol Port Number Registry" (https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?search=1080). Internet Assigned Numbers Authority. 19 May 2017. Retrieved 23 May 2017.

3. Darmohray, Tina. "Firewalls and fairy tales (https://www.usenix.org/publications/login/february-2005-volume-30-number-1/firewalls-and-fairy-tales)". ;LOGIN:. Vol 30, no. 1.
4. Archived (https://web.archive.org/web/*/http://www.socks.permeo.com/) index at the Wayback Machine.
5. CNET: Cyberspace from outer space (http://news.cnet.com/Cyberspace-from-outer-space/2100-1001_3-244725.html)
6. "2010 Circumvention Tool Usage Report" (http://cyber.law.harvard.edu/sites/cyber.law.harvard.edu/files/2010_Circumvention_Tool_Usage_Report.pdf) (PDF). The Berkman Center for Internet & Society at Harvard University. October 2010.
7. "OpenSSH FAQ" (http://www.openssh.org/faq.html#2.11).
8. "Tor FAQ" (https://www.torproject.org/docs/faq.html.en#TBBSocksPort).
9. "How to chain socks with Tor" (https://www.deepdotweb.com/security-tutorials/chain-socks-tor/). Retrieved 23 January 2017.
10. Graham, James. *Cyber Fraud* (https://books.google.co.uk/books?id=BZLLBQAAQBAJ&pg=PA45&lpg=PA45&dq=socks+proxy+fraud&source=bl&ots=x6VDwUxEct&sig=H8gsSdPljFhdefp17-Tmg-E8W5g&hl=en&sa=X&ved=0ahUKEwjh6Lzx6tjRAhVcFMAKHco4AxwQ6AEIRjAH#v=onepage&q=socks%20proxy%20fraud&f=false). p. 45.
11. Krebs, Brian (16 October 2016). "IoT Devices as Proxies for Cybercrime" (https://krebsonsecurity.com/2016/10/iot-devices-as-proxies-for-cybercrime/). Retrieved 23 January 2017.
12. van Hardeveld, Gert Jan; Webber, Craig; O'Hara, Kieron. "Discovering credit card fraud methods in online tutorials" (https://www.researchgate.net/publication/303418684_Discovering_credit_card_fraud_methods_in_online_tutorials). Retrieved 23 August 2017.
13. "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing" (http://tools.ietf.org/html/rfc7230#section-2.3). Retrieved 2014-08-01.
14. Ying-Da Lee. "SOCKS 4A: A Simple Extension to SOCKS 4 Protocol" (http://www.openssh.org/txt/socks4a.protocol). OpenSSH. Retrieved 2013-04-03.
15. "RFC 1961" (http://tools.ietf.org/html/rfc1961). Tools.ietf.org. Retrieved 2009-06-19.
16. "RFC 1929" (http://tools.ietf.org/html/rfc1929). Tools.ietf.org. Retrieved 2009-06-19.
17. IANA.org (http://www.iana.org/assignments/socks-methods)
18. "Easy Net with SOCKS5" (https://www.shimmercat.com/en/info/articles/socks5-swift/). *shimmercat.com*. ShimmerCat. Retrieved 20 April 2016.

External links

- RFC 1928: SOCKS Protocol Version 5
- RFC 1929: Username/Password Authentication for SOCKS V5
- RFC 1961: GSS-API Authentication Method for SOCKS Version 5
- RFC 3089: A SOCKS-based IPv6/IPv4 Gateway Mechanism
- Draft-ietf-aft-socks-chap (http://www.tools.ietf.org/html/draft-ietf-aft-socks-chap), Challenge-Handshake Authentication Protocol for SOCKS V5
- SOCKS: A protocol for TCP proxy across firewalls (http://ftp.icm.edu.pl/packages/socks/socks4/SOCKS4.protocol), SOCKS Protocol Version 4 (NEC)

Retrieved from "https://en.wikipedia.org/w/index.php?title=SOCKS&oldid=797957059"