IEEE Seventh International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE '98, June 17-19, 1998, Stanford University, California, USA, pp. 301–305. IEEE Computer Society. ISBN 0-8186-8751-7.

WebGroup: a secure group access control tool for the World-Wide Web

Fabien A.P. Petitcolas
Computer Laboratory, University of Cambridge
Cambridge CB2 3QG, U.K.
fapp2@cl.cam.ac.uk

Kan Zhang
UBILAB, UBS
Bahnhofstraße 45, 8021 Zurich, Switzerland
Kan.Zhang@ubilab.ch

Abstract

We present an integrated secure group access control tool to support workgroups on the World-Wide Web. The system enables user authentication, encrypted communication and fine-grained group access control. The tool comprises two proxies: one running on the server side and the other one on the client side. Typically the browser sends a query to the client side proxy which contacts the server side proxy for authentication, session key exchange and checking of access rights. The server side proxy finally forwards the request to the HTTP server. Our tool is completely transparent to the user and compatible with any Web server and browser. It can also become part of a firewall configuration.

1 Introduction

The World Wide Web has become the primary means of accessing information over the Internet. More recently, group based collaborative information sharing has received wide attention. In order to share information effectively, adequate security mechanisms are needed for Web access. Specifically, the security requirements are user authentication, fine-grained group access control and communication encryption. Currently, adequate and affordable security tools required for such use are lacking.

Existing approaches to secure access on the Web are either inadequate or unavailable to the majority of users. In addition, no single approach satisfies the above requirements in an integrated way. For example, firewalls protect a site from illegitimate access from outside. However, current firewalls technology do not provide fine-grained group access control. In addition, firewalls cannot be universally applied. Small companies or individual users may not be able to afford the installation and maintenance of firewalls.

Some solutions for authorizations on the Web employ advanced distributed computing infrastructure, which is not widely available to users. For example, DCE Web [4] provides a sophisticated authorization mechanism along with a structure for organizing groups of Web users and resources under coherent security and other administrative policies. However, DCE Web requires that both browsers and servers be DEC-capable, i.e. capable of using a set of sophisticated distributed computing technologies, based on OSF-DCE.

Some security solutions provide user authentication and encryption but not group access control, e.g. Secure HTTP [8] and SSL [2]. Others focus on group access control but lack encryption of communication, e.g. CERN httpd [5], a capability-based authorization model by Kahan [3]. No single approach provides adequate protection for group based collaborative information sharing.

Although one could use a combination of SSL and HTTP basic authentication, our solution provides more features that this simple combination. It can be used by people who do not have access to a secure web server or whose browser does not support SSL. It also provides access control at the file level rather than directory and much better users and groups administration since on the one hand authentication is done per user and it is very easy to define groups and subgroups of users.

In this work, we present an integrated secure group access management tool that provides user authentication, fine-grained access control and communication encryption in a seamless way. The tool is compatible with any Web server and browser. It is easy to use and available to everyone. Moreover, it can become part of a firewall configuration.

2 Design goal

The main purpose of the tool we present here is to establish a secure communication channel between a Web server and a Web browser and to provide secure group access facilities through the use of access control lists. This involves:

Authentication User-wise authentication

Access control Fine-grained access control to a single user and to particular portions of a HTML document tree. Unauthorised users cannot access certain pages. The administrator can group pages into convenient subdomains corresponding to the workgroups and to specify very precisely the access rights of each user: from an entire directory tree to a single file.

Confidentiality The entire HTTP transaction is considered private; thus, the HTTP headers and data objects of client requests and server responses are encrypted.

Transparency Apart from the need to login before using the tool, the behaviour of the program should be transparent.

Compatibility The tool should be able to communicate with all HTTP servers and browsers.

3 Assumptions

We focused on attacks by outsiders. We assume that the HTTP server is running on a trusted machine and that only a trusted administrator can change access rights to files belonging to the Web server. However other users may maintain the site, but it is up to the administrator to ensure that users can only read or write files they are authorized to.

The Web server must be configured such that it only replies to requests coming from our programs. Typically both the server and our program run on the same computer and only our program replies to requests coming from other computers. However if they do not run on the same machine one must ensure that the communication between these two processes will be secure (e.g. inside a LAN protected by a firewall).

Finally, the Web browser must use our program as a proxy. Basically, this proxy and the Web browser run on the same machine. The browser sends all its requests to the proxy and the proxy accepts only requests coming from a specified IP address (typically the local machine). There is no encryption between the browser and the proxy. Fig. 1, 4, 5 and 6

4 Design & implementation

The software has two different components (Fig. 1). The *Server Side Secure Proxy* (SSSP) communicates directly with the HTTP server that we want to protect and the *Client Side Secure Proxy* (CSSP) is used as a proxy for the Web browser. This latter component is able to communicate with standard HTTP servers (i.e. server #2 in Fig. 1) and, of course, with SSSP. Fig. 3 shows how a request from the browser is handled by these modules during a transaction. The program automatically writes in a log file all relevant information about its activity: connection accepted, connection refused, Web pages required, etc.

In order to implement the authentication mechanism, a new HTTP method has been introduced. We call it: `AUTHENTICATE`.

```
Authenticate HTTP/1.0\r\n
ClientSide: <address>\r\n
ServerSide: <address>\r\n
UserID: <ID>\r\n
Content-length: <integer>\r\n
\r\n
<encrypted body>
```

The SSSP replies only to authentication requests, i.e. requests that use the `AUTHENTICATE` method and systematically sends an error message to all other HTTP requests. When an authentication request is received, the SSSP begins an authentication and session key exchange process with the CSSP. If successful, the subsequent communication between SSSP and CSSP is like a secure tunnel: every received encrypted message is decrypted and forwarded to the HTTP server; each answer of the HTTP server is encrypted and then forwarded to the CSSP.

In order to determine whether the CSSP is talking to the SSSP or to a HTTP server a trial and error method is used. The CSSP sends a first message in order to begin authentication (`AUTHENTICATE` method). If the server responds with an error status (basically an HTTP status code 4xx - Bad request) this means that the CSSP is talking to a standard HTTP server and not to the SSSP. In this case it simply forwards the request of the browser. In the other case, the authentication process goes on and a secure communication is set up.

4.1 Authentication Protocol

The authentication protocol is detailed below. This protocol is based on a shared secret (i.e. the key of the user, K_U) and does not require any third party as [6]. Thus, we assume that initially users can register their keys with the administrator in a secure way. In the following C, C_P, S_P and S represent, the browser the CSSP, SSSP and the server respectively.

$$\begin{array}{rcll}
C & \rightarrow & C_P & \mathit{Request} \\
C_P & \rightarrow & S_P & C_P, S_P, usrID, \{1, C_P, S_P, usrID, N_C\}_{K_U} \\
S_P & \rightarrow & C_P & \{2, S_P, C_P, N_C, N_S\}_{K_U} \\
C_P & \rightarrow & S_P & \{3, C_P, S_P, N_S, \mathit{Request}\}_{K_S} \\
S_P & \rightarrow & S & \mathit{Request} \\
S & \rightarrow & S_P & \mathit{Reply} \\
S_P & \rightarrow & C_P & \{\mathit{Reply}\}_{K_S} \\
C_P & \rightarrow & C & \mathit{Reply}
\end{array} \tag{1}$$

In the first message, the client sends in clear the user ID, thus the server can look up in its list of keys to decrypt the second part of the message. The fact that the keys are stored encrypted on the server is not an issue here since if an attacker can gain access to this file he can certainly get access to the web pages with the same ease. Fig. 2 shows how the keys are actually stored. A successful decryption means only that both the server and the client share the same secret; the user could indeed give his key and password to another person. We cannot do better anyway. Then, the following messages use nonces (i.e. random numbers generated for the purpose to be fresh [1]) to prevent replay attacks, that is attacks done by someone who recorded a previous authentication protocol. Nonces are also used to create a new session key: $K_S = N_C \oplus N_S$. In each message the addresses of the two parties are repeated as well as previously sent data. This also helps to defeat more subtle attacks [1]. The session key is then used to encrypt all other messages, i.e. the request of the browser and the reply of the server.

In this way, all the exchanged data between the CSSP and the SSSP will be encrypted (even the HTTP headers). Thus a third party would not know either the data or their name, location, time, size, etc. However we are aware that the communication is not perfectly private, at least, from the point of view of the user, his ID is sent unencrypted.

Given our assumption that users can register their secret keys with the administrator in a secure way, symmetric key crypto-system – triple DES in the current implementaiton – is sufficient for our two-party authentication and session key exchange purpose. We also use DES for message encryption. Our software is built in such a way that the encryption algorithm can be replaced easily.

4.2 Access Control

Web servers change continually to accommodate new facilities and to adapt to changing requirements of users. Secure group management must provide the flexibility of adapting to these changes. Thus it should be easy to change the profile of users and groups of users. A group model provides mechanisms for managing access control. In our case a group consists of a set of directories and files which define accessible domain for the group.

Our access control mechanism relies on two files: the description of the groups and the definition of the users. Consider for instance the following entries respectively in the group-description file and in the user-description file:

```
Programmers   Development  /src/;/prg_news.html

Smith         Programmers  /users/smith/private/
```

They define a new group called *Programmers* which is a subgroup of *Development* and a user *Smith* belonging to *Programmers*. This user can read its private pages (in `/users/smith/private/` and subdirectories) and all the file readable by the members of *Programmers* (i.e. `/prg_news.html` and all the pages in `/src/` and its subdirectories). A special group name is reserved: *default*. It is used to specify files (usually images that decorate the pages) shared by all the users. All users belong to this group.

The procedure that checks the access rights of a given user is such that access is denied by default. Hence it follows the rule '*That which is not expressly permitted is prohibited*.'

5 Extensions and Future Work

The tool is designed to be easy to extend and to show that access control can be added in a simple way to an existing web site by the use of proxies. Some possible extensions are describe below.

Add a cache to the Access Control Module. Currently, each time a page is requested by a user the SSSP reads three files to fetch the user key and his profile. This could be improve by using a cache that would store the information about the last n connected users.

Currently HTTP open a new connection for each request. This is what we did. This implies the creation of a new socket at each time. A modification that could improve the speed of our program is to maintain the connection between the server and the client as long as it is required. Hence, more than one request could transit on the same channel. Since most Web pages contain embedded images, the HTML page and its images would transit during the same connection. Hence only one process would be required for several requests.

Another improvement could be the implementation of a 'pre-fetching' option. Instead of waiting the requests of the browser, the proxy could parse the HTML page, find out the embedded images and request them immediately to the server.

There are two aspects in group facilities: groups of users and groups of servers. Our current work focuses on groups of users on a single HTTP server. IBM Research proposed an extension to the HTTP that provides user authentication on several HTTP servers [7]. An interesting challenge would be to integrate these two approaches: authentication and access control of groups of users over groups of Web servers.

6 Conclusions

In this work, we implemented an integrated tool by which secure group based Web access can be added to existing Web servers. The tool enables user authentication, secure communication and fine-grained access control checking on per user, per group of users and per document bases. The tool developed here is transparent and compatible with any Web server and browser. It is readily usable for any Web user.

Acknowledgements

Some of the ideas presented here were clarified by discussion with Mark Lomas.

References

[1] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Technical report, Digital Systems Research Center, Feb. 1989.

[2] K. Hickman. The secure socket layer. Internet Draft, Mar. 1996.

[3] J. Kahan. A capability-based authorization model for the world-wide web. WWW3.

[4] S. Lewontin and M. Zurko. The dce web project: Providing authorization and other distributed services to the world wide web, www2.

[5] A. Luotonen. How to set up protected cern server, Jan. 1995.

[6] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21:993 Sq., 1978.

[7] A. H. qnd M. Kaiserswerth and P. Trommler. Secure world wide web access to server groups. Technical Report RZ 2812, IBM Research Division, Zurich Research Laboratory, Mar. 1996.

[8] E. Rescorla and A. Schiffman. The secure hypertext transfer protocol. Internet Draft, May 1996.

Appendix

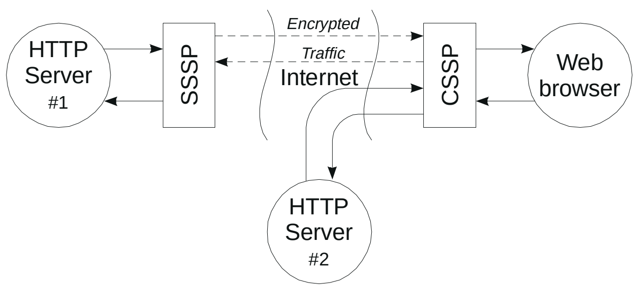


Figure 1. Security perimeter over the Internet.

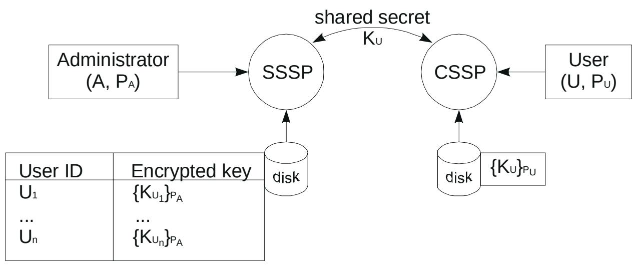


Figure 2. This figure shows how the different keys are stored. Each user has his own key which is encrypted with his password and stored on disk. Since the program is very small it can be copied on a floppy disk with the user key. On the server side, a list of user keys are stored in an encrypted file using a key only the administrator knows.

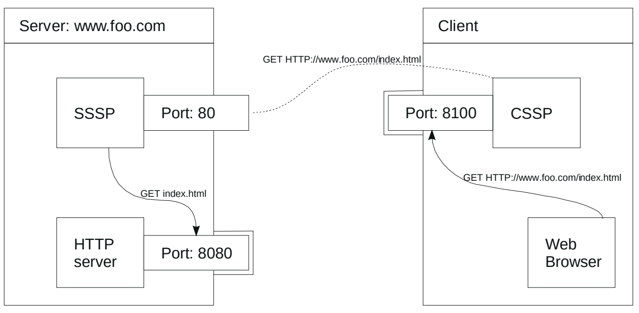
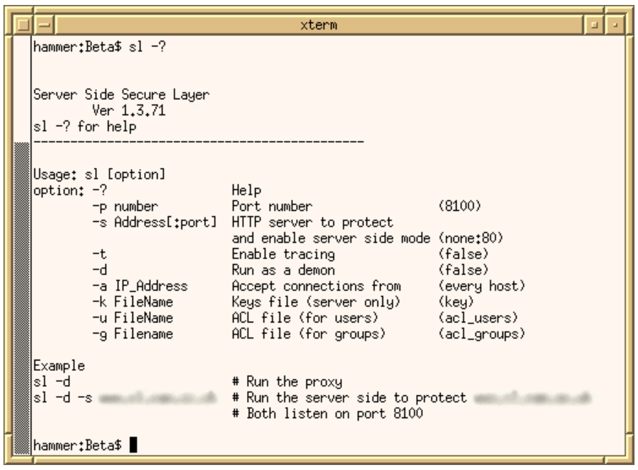
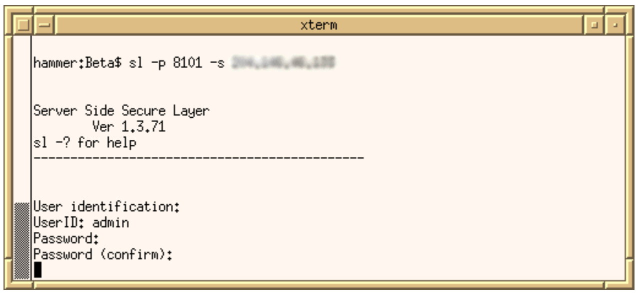


Figure 3. This drawing shows how a HTTP request goes from the Web browser to the Web server. The HTTP server and the CSSP should be configured such that they will not accept any connection from the outside. We represented the listening ports only. During the transfer the request is modified and only the relative path remains in the URL when it reaches the server.



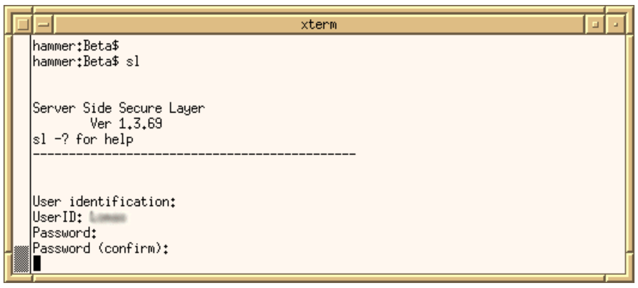
```
hammer:Beta$ sl -?


Server Side Secure Layer
        Ver 1.3.71
sl -? for help
---------------------------------------------

Usage: sl [option]
option: -?                 Help
        -p number          Port number                (8100)
        -s Address[:port]  HTTP server to protect
                           and enable server side mode (none:80)
        -t                 Enable tracing             (false)
        -d                 Run as a demon             (false)
        -a IP_Address      Accept connections from    (every host)
        -k FileName        Keys file (server only)    (key)
        -u FileName        ACL file (for users)       (acl_users)
        -g Filename        ACL file (for groups)      (acl_groups)

Example
sl -d                      # Run the proxy
sl -d -s [redacted]        # Run the server side to protect [redacted]
                           # Both listen on port 8100

hammer:Beta$ █
```

Figure 4. Several command line options are available. The -? option displays them.



```
hammer:Beta$ sl -p 8101 -s [redacted]


Server Side Secure Layer
        Ver 1.3.71
sl -? for help
---------------------------------------------


User identification:
UserID: admin
Password:
Password (confirm):
█
```

Figure 5. An example of launching: the user is the administrator. The password is asked twice in order to prevent misspelling. In this example the SSSP is set so that it listens on port 8101 and protects the server whose address is xxx.xxx.xxx.xxx and which listens on port 80.



```
hammer:Beta$
hammer:Beta$ sl


Server Side Secure Layer
        Ver 1.3.69
sl -? for help
---------------------------------------------


User identification:
UserID: [redacted]
Password:
Password (confirm):
█
```

Figure 6. When launching the program, the user is prompted for an ID and a password. The password will be used to decrypt the user's key.