COMPUTING SCIENCE

Simple Authentication

C.R. Snow and H. Whitfield

TECHNICAL REPORT SERIES

No. 459 November, 1993
Simple Authentication

C.R. Snow and H. Whitfield

Abstract

This paper describes a simple authentication protocol for use with terminal emulation programs running on small workstations, and communicating with a host computer over a (potentially) insecure network. The technique has the advantage that it requires no more network traffic than would be necessary using a standard user-id/password challenge, and also requires only a small amount of additional intelligence at the workstation. Some management issues are also discussed. The system has been implemented on two host systems and two workstation types.
Bibliographical details

SNOW, Charles Richard
Simple Authentication
[By] C.R. Snow and H. Whitfield

(University of Newcastle upon Tyne, Computing Science, Technical Report Series, no. 459)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE.
WHITFIELD, Harry

Abstract

This paper describes a simple authentication protocol for use with terminal emulation programs running on small workstations, and communicating with a host computer over a (potentially) insecure network. The technique has the advantage that it requires no more network traffic than would be necessary using a standard user-id/password challenge, and also requires only a small amount of additional intelligence at the workstation. Some management issues are also discussed. The system has been implemented on two host systems and two workstation types.

About the author

Dr Snow has been a lecturer in the Department of Computing Science since 1970. He was a Visiting Scientist with Xerox System Development Division in Palo Alto, California, between 1978 and 1979.

Harry Whitfield is Professor of Computing and Data Processing. He was Director of the Computing Laboratory from 1980 until 1992 when the two parts of the Laboratory (the Department of Computing Science and the University Computing Service) became independent. He is currently on secondment to the University Administration as Director of Management Information Services where he is responsible for the introduction of integrated office automation and management information systems. Over the years he has worked in the areas of Compiler Design, Operating Systems, VLSI Design and Computer Networking.

Suggested keywords

AUTHENTICATION DISTRIBUTED SYSTEMS ENCRYPTION SECURITY

Suggested classmarks (primary classmark underlined)
Dewey (18th): 652.8  001.64404
U.D.C. 355.405.2  519.687
Simple Authentication

C.R. Snow

Department of Computing Science
E-Mail: C.R.Snow@newcastle.ac.uk
Tel: 091-222-8064
Fax: 091-222-8232

and

H. Whitfield

Management Information Systems
E-Mail: Harry.Whitfield@newcastle.ac.uk
Tel: 091-222-6336
Fax: 091-222-8232

University of Newcastle upon Tyne,
Newcastle upon Tyne,
NE1 7RU.
U.K.
Abstract

This paper describes a simple authentication protocol for use with terminal emulation programs running on small workstations, and communicating with a host computer over a (potentially) insecure network. The technique has the advantage that it requires no more network traffic than would be necessary using a standard user-id/password challenge, and also requires only a small amount of additional intelligence at the workstation. Some management issues are also discussed. The system has been implemented on two host systems and two workstation types.

Keywords: Security, Authentication, Distributed Systems, Encryption.

Introduction

Many distributed systems are based on the principle of the user seated at a (more or less) powerful workstation, and facilities being available using some kind of client/server technique. These additional facilities are accessed over networks which are generally not necessarily secure, and thus authentication techniques (and systems) have been developed to provide appropriate checking of conversations between elements (or principals, in Needham and Schroeder's terminology [1]).

Despite these developments, it is still the case that many users wish to access the facilities of a mainframe computer system as though they were using a "dumb" terminal. Many personal computers and workstations provide terminal emulation and terminal-style connections over such protocols as TCP/IP's Telnet protocol [2].

In such conversations as these, it is typical for the host system to make no assumptions about the capabilities of the workstation, beyond the fact that terminal emulation is provided. In other words, the host computer interacts with the workstation precisely as it would with a "real" terminal of the emulated type, including asking for authentication checking, such as asking the user to enter an identifier and a password.
In former times, when hosts were accessed from "real" terminals, it was usually the case that the terminals were sited relatively close to the host computer, the connections being made by dedicated point-to-point links. With the advent of workstations, terminal emulation and Telnet-style conversations, possibly across many networks and large distances, the potential for eavesdropping on such conversations is great, and it cannot any longer be regarded as safe to pass the user's credentials (id and password) from the workstation to the host in cleartext.

This paper describes a method of simple authentication in which a user attempting to access a host system is required to respond to the normal id/password style challenge, but in such a way that the user's password is not required to be transmitted across the network in cleartext form. In fact, the user's password is not required to pass across the network in any form, and authentication is achieved by the encryption and decryption of a random number, the successful decryption of which indicates that the workstation encrypted the random number using the same key.

We show how this mechanism may be used as part of the initial login sequence, but it may also be used to provide additional control over access to sensitive activities at the host, again without the need for the password, in cleartext or encrypted, to be transmitted across the network.

The method is secure against password theft by eavesdropping, and it is also secure against replay attack. The encryption key remains constant for as long as the user keeps the same password, but since the encrypted messages are short, and relatively infrequent, it would take an unreasonably long time for an attacker to collect enough plaintext/ciphertext pairs to be in a position to mount a cryptanalytical attack on the key. It is to be hoped that users change their passwords sufficiently frequently for this form of attack to become impossibly difficult.
System Requirements

Clearly, if the user is employing a real "dumb" terminal, then it is not possible for any computation (such as encryption) to take place at the terminal. However, it is commonly the case that such a terminal would be connected by a point-to-point link, either to the host computer itself, or to a PAD (packet assembler/disassembler or terminal concentrator), which allows the terminal user to select the host with which communication is required. Clearly, the PAD would contain sufficient computational capability for encryption to be performed within it. Passwords would necessarily be transmitted in cleartext form from the terminal, but only as far as the PAD, and we would have to assume that the terminal to PAD connection was point-to-point, short and secure.

The minimal requirements for the workstation (apart of course from the terminal emulator and the corresponding communications capability) is the ability to set up an independent listener, separate from the terminal session. In addition, it must be possible to receive messages sent to the listener, to encrypt some or all of an incoming message using a password entered by the user, and to reply to this message. This could be done in a variety of ways, but the prototype system described here assumes that the workstation operates a windowing environment, and is capable of establishing a listener, so that when a message arrives, a dialog may be held with the user in order to obtain a password, and possibly other information. This implies that the workstation is sophisticated enough to manage at least two independent concurrent tasks, and to hold two independent conversations with the user.

The Basic Technique

The basis of the method is similar to, though simpler than, the techniques used by many authentication systems, of which the most well-known example is Kerberos [3],
whose encryption protocol is based on the method of Needham and Schroeder [1]. It would be clearly possible to achieve the same effect using the full Kerberos system, but this would have necessitated the installation of a large and complex system, whereas the task can be carried out by a relatively simple piece of additional software. An alternative simpler system is described by Feistel et al. [4], but this also requires rather more interactions (i.e. network traffic) than the method described here. The number of messages sent across the network using this method is actually slightly less than would be required for the traditional (insecure) user-id and password challenge.

On receiving a request for a login, the host essentially attempts to hold a separate conversation with the workstation, in order to ascertain the identity of the intended user. It does this by sending a random 64-bit number, called a nonce, to the listener on the workstation. For reasons which we shall deal with later, it is convenient for a part (at least) of this number to represent the time at which the message is sent. On receipt of the message, the workstation asks the user for his/her identity and a password. The password is then subjected to an algorithm to produce a 64-bit DES key [5]. (Actually, the key is parity corrected on a byte-by-byte basis, which effectively makes it a 56-bit key.) The resulting key is used to encrypt the nonce, which is then returned to the host, together with the id of the user. The host may now look into its tables for the entry corresponding to the user, extract the key, and use it to decrypt the nonce part of the reply. If this results in the original nonce, then the authentication is regarding as having succeeded, and the host believes that the user is indeed who he/she claims to be.

The advantage of including the time as part of the nonce is twofold. Primarily, it guarantees that the nonce is unique, but a secondary advantage is that the nonce can be used to determine the time at which the original challenge was made, and therefore can be used to check that there is not an unreasonable delay between the sending of the nonce, and the reply being returned by the user.
The absence of such a check might allow an intruder to receive the nonce, spend an arbitrarily long time attempting to obtain the key by cryptanalysis, and then reply when this had been successfully achieved. However, it would be unwise to use only a timestamp as the nonce, as this might be guessed at by a possible intruder, with a consequent diminution of security.

It is at least theoretically possible that an intruder could masquerade as the host computer, and send carefully chosen bit patterns to the workstation in order to mount a chosen plaintext attack on the key. This could be protected against by insisting that the time within the nonce corresponds reasonably closely to the workstation's own idea of the time, and that the workstation only responds to authentication requests from the host to which the original connection was made. Depending on the quality of the masquerade, this would either be sufficient to detect an imposter, or the network would discover (probably at some cost), that two hosts with the same address were on the network at the same time. The second factor which would make such attempts unlikely to succeed is that the (real) user would probably become suspicious of unsolicited requests for authentication long before the intruder had sent and received enough data to make a chosen plaintext attack feasible.

**Passwords and Keys**

The method by which passwords are translated into DES keys, or *strong keys*, is based on the technique used in the XNS authentication protocol [6]. It essentially consists of a one-way function taking strings of characters and converting them into a 64-bit key (which is then parity adjusted to produce what is effectively a 56-bit key, as described above).

Given a password, consisting of an arbitrary number of characters (usually printable
characters), the algorithm divides the password into groups of four characters. It then constructs a 64-bit pattern from the first group of four characters, and encrypts this pattern using the DES algorithm and a known key (actually a zero key). The resulting (encrypted) pattern is then used as the key for the next stage of the algorithm, in which the next group of four characters, regarded as a 64-bit pattern as before, is also encrypted using DES. This operation of using the output from each stage as the DES key for the following stage, continues until the last character(s) of the password has/have been used.

Symbolically, let the characters in the password be denoted by $p_0, p_1, p_2,$ etc., then the $i$th stage of the algorithm may be represented as:

$$key_i = \text{paradj} \left(\{P_i\}_{key_{i-1}}\right) \text{ for } i = 1...n$$

where $key_0 = \text{paradj} \left(0\right),$ and $\{P\}_k$ denotes the encryption of the message $P$ using the key $k$.

Parity adjustment, represented by the function $\text{paradj}$, is achieved by setting the least significant bit in each 8-bit byte, such that the parity of each byte is odd. Thus, for example, $key_0 = \text{paradj} \left(0\right)$ is a 64-bit pattern where each bit is zero, except for the least significant bit of each byte, which is set to one.

$P_i$ is an 8-byte (64-bit) number consisting of the following eight values:

$$0, p_i, 0, p_{i+1}, 0, p_{i+2}, 0, p_{i+3}$$

i.e. each character from the password has a zero byte prepended to it, and the $i$th group of four "extended" characters forms the 64-bit pattern $P_i$. The final group of characters may contain less than four characters, in which case, the password string is
padded out to a multiple of four characters by zero bytes.

It is clear from this algorithm that it is very easy to obtain the password from the strong key, if the password is less than five characters long. This is because it represents a single encryption using a known key, and decryption is immediately possible using the same key. However, obtaining the password from the strong key when two or more stages of the algorithm are employed would be computationally impossible.

It is, however, also the case that the strong key cannot be obtained by eavesdropping, since neither the password nor the strong key is passed over the network, neither is it susceptible to cryptanalytical attack, so that password cracking attempts have to rely on theft of the data from the data base. If this were to happen, however, then passwords would be as secure as any other scheme in which the intruder encrypts a (potentially) large number of likely passwords, testing the results against the "stolen" data base.

The security of the strong key data base is of paramount importance. Not only would the theft of this data allow the kind of attack mentioned in the previous paragraph, but it would be possible for an intruder to construct a "spoof" authenticator which only used the strong keys, and did not require the password at all.

These are general comments on password schemes, with a recommendation that the data base itself be made inaccessible to all but those programs and system administrators who need access, and that additional information pertaining to users, which is required to be more generally available should be stored elsewhere. This is similar to the "shadow" password schemes used in some Unix systems [7]. Note that, as in many systems, a user's password is not held in cleartext form by the system, and since the mapping from passwords to strong keys is a one-way function, there is no
way that a password can be determined from the information held by the system. Neither is the password held at the workstation, since each time the challenge is issued the user enters the password, and once the strong key has been constructed and then used to encrypt the nonce, both the password and strong key are erased from the workstation.

**Functions and Packets**

The initial set of facilities which might be required are:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Name</td>
<td>install new user/function</td>
</tr>
<tr>
<td>Remove Name</td>
<td>remove existing user/function</td>
</tr>
<tr>
<td>Check Key</td>
<td>check password (strong key) for user/function.</td>
</tr>
<tr>
<td>Set Key</td>
<td>change password (strong key) for user/function.</td>
</tr>
<tr>
<td>Login</td>
<td>initial logging in of user</td>
</tr>
</tbody>
</table>

It is assumed that behind the authentication system, the host maintains a database which (minimally) provides a mapping between names and strong keys.

Within the database, there would not only be an entry for each user of the system, but also an entry for each function which requires protection. Thus, using the Check Key function above, the user could be asked to supply a password before being allowed to access the system's more sensitive facilities, such as adding or removing names from the data base, or changing the key for a name.

To turn now to the specific packet formats used, we describe the messages which pass over the network during the course of these operations. In what follows, the nonce is a 64-bit number, containing (at least in part) a timestamp; a key is also a 64-bit number which should have been parity corrected as described above, and a name is an
arbitrary character string, probably containing printable characters. In the case of each function, the host initiates the conversation, and it is assumed that the workstation has its listener awaiting messages. This being the case, the description of the messages is given from the point of view of the program running on the host.

Add Name

Message sent:

<nonce> "AN" <prompt>

Message expected:

<encrypted nonce> <encrypted key> <name>

In the message sent to the workstation, the <prompt> is a string which the workstation listener may wish to display to the user when asking for the password. While in practice this may be unnecessary, since this string could be deduced from the function code letters "AN", there is some merit in including the string in the message so that the workstation can use it in the same way as the <name> component of (for example) the Check Key function (see later). The same string is used by the host to find the appropriate strong key in the strong key data base.

On receiving the message, the workstation will be expected to ask the user for the password which effectively authorises the user to carry out this change to the data base, followed by requests for the name of the new user or function, and the password corresponding to that user. The workstation may choose to ask for the password to be entered twice, in order to guard against mistyping.

Using the first password, authorising the operation, the workstation creates the corresponding key, encrypts the nonce with this key, constructs a key corresponding
to the second password and encrypts this. Finally, it sends back to the host the encrypted nonce, the encrypted user key and the user's name.

The host may now extract from the data base the key corresponding to the "Add Name" function and uses it to decrypt the encrypted nonce. If this fails to yield the original nonce, then the request is rejected and the operation abandoned. Otherwise, the new key is decrypted using the "Add Name" key, and the resulting key is stored in the data base along with the username just received.

It would in principle be possible for the username itself to be encrypted using the "Add Name" key, but since the username is a string of (potentially) arbitrary length, an extended form of the DES encryption/decryption algorithm would be required, and it was felt that the cost of performing this additional operation was greater than the benefit that would accrue. Encrypting the new key is, however, absolutely crucial to the security of the system.

As mentioned previously, the workstation may ask the user to supply the new password twice as a precaution against mistyping. In the case where a mistyping has taken place, it is obviously desirable that the Add Name be aborted. Rather than require that a specific error code be returned to the host, it is sufficient for the workstation to alter the returned encrypted nonce slightly, and the host will then abort the operation in the same way as it would if the authentication of the operation had failed.

Remove Name

Message sent:

<nonce> "RN" <prompt>
Message expected:

<encrypted nonce> <name>

As before, the prompt is not strictly necessary, and could be deduced from the function code "RN".

As with the "Add Name" function, the workstation will ask the user for the authorising password, which will be converted into a key. The workstation will then ask for the name to be removed. The returned message will consist simply of the nonce, encrypted with the authorising key, and the name.

The usual checks will be carried out by the host on receipt of the response, and if the decryption of the nonce part of the reply yields the original nonce, the entry in the data base corresponding to the name will be deleted. Notice that the password (or key) corresponding to the name does not need to be supplied.

Check Key

This function is a general request from the host for the user to authorise him/herself to carry out a particular function. In this case therefore, the name of the function is sent by the host to the workstation for display to the user.

Message sent:

<nonce> "CK" <name>

Message expected:

<encrypted nonce>

The workstation will ask the user to enter the password corresponding to the given name. As usual, this will be used to construct a key, which in turn will be used to
encrypt the nonce. The successful decryption of the nonce by the host following the response from the workstation will indicate that the user has the necessary authorisation for the requested operation.

**Set Key**

Message sent:

```xml
<nonce> "SK" <name>
```

Message expected:

```xml
<encrypted nonce> <encrypted new key>
```

The "Set Key" function allows the key corresponding to a name to be changed. In order to do this, the user must know the existing password for that name. We distinguish between the case where a name may be removed from the data base without knowing the corresponding password but the "Remove Name" password must be known, and the "Set Key" function, where in order to change the key, only the existing password needs to be known. We justify this by the observation that removal of names from the data base is relatively rare, and may be regarded as an administrative function to be carried out by one of a (presumably small) number of system administrators. The changing of a key, on the other hand, might well be required on a fairly frequent basis, and it is not unreasonable to allow, or perhaps expect, that users should wish to carry out this operation on their own behalf. In fact, it would compromise security if users always had to inform the system administrator of their new passwords each time they were changed.

On receipt of the "Set Key" message, the workstation asks the user to supply the authorising password, i.e. the old password corresponding to the name, and then to supply the new password. For safety, the new password may be requested twice, as with the "Add Name" function. The old password is used to create the encryption key,
and the new password is used to create the new key. The nonce and the new key are then encrypted with the old key, and sent back to the host. If the nonce is successfully decrypted, the data base entry corresponding to <name> is updated with the new key. Again, as in the case of "Add Name" an error code is not necessary to reflect the failure of the operation due to mistyping of the password, since a perturbation of the encrypted nonce will automatically cause the host to abort the operation.

Login

Message sent:

<nonce> "LN" <prompt>

Message expected:

<encrypted nonce> <name>

The "Login" operation bears some resemblance to the "Check Key" operation, in that the user is being asked to authenticate him/herself by supplying a password. However, unlike the "Check Key" function, the host does not know at the time at which it sends the message, the name which it is being asked to authenticate. Thus, the "Login" function sends the login request, including a prompt which is simply a string which may be displayed to the user, and the workstation is required to return the name to be authenticated along with the encrypted nonce. In this case, it is clearly essential for the name to be sent in cleartext, since the host must be able to use this name to extract the appropriate key from the database. Having done so, it is then in a position to decrypt the returned nonce, and to check the result against the nonce sent out.
Practical Issues

To date, this simple authentication system has been implemented on two operating systems acting as hosts, Unix and Vax VMS, and on two workstations, the Apple Macintosh and the IBM-PC running Windows.

The communications sub-system uses the Internet Transmission Control Protocol (TCP), using MacTCP in the Macintosh, and the Winsock package [8] in the PC. Winsock packages are supplied in a number of implementations, the particular version in use in this work being supplied as part of the LAN Workplace for DOS [9] development kit.

MacTCP offers the programmer a set of function calls which are closely related to the actual packets sent. Thus, it provides calls such as TCPActiveOpen, TCPPassiveOpen, TCPSend, etc. The Winsock interface, on the other hand, draws heavily on the ideas of sockets, as used in the Berkeley (BSD) versions of Unix [10]. Although this provides a general framework for the facilities available to the programmer, the handling of the so-called "blocking" calls, connect, accept, read, etc., can make use of the notion of Windows messages. Thus, a call to (for instance) the accept function, which would normally block until the remote end issues a connect request, can inform Windows that it wishes to be notified by a Windows message when the expected event has occurred. This then allows the Windows system to expedite the processing of other tasks which the PC might be performing.

At the host, both the particular version of Unix in use, and the Vax VMS system, also employ the Berkeley sockets approach to data communications.

The four functions, AddName, RemoveName, CheckKey and SetKey, have all been
set up as user programs. In normal operation, only the SetKey function would be habitually used by ordinary users. The CheckKey function might be used by the supplier of a sensitive function, who wished to ensure that only those users to whom the password had been given could employ those sensitive functions, and AddName and RemoveName would normally only be called by system administrators. However, it does no harm to allow them to be generally available, since the protection is afforded by the authentication mechanism itself.

In principle, the login function might also be offered as a user callable function, but it would probably be necessary to insist that this function should be called during the standard login exchange. Minimally, it could be called from within a start-up script (such as the Unix `.profile` file, a shellscript invoked as part of the login procedure), which would require the user to authenticate him/herself in the usual way, and then to be subjected to the secure login challenge as well.

A rather better solution is to modify the login program itself, and this has been done in the case of the Unix implementation. The strategy adopted by the modified login program is to try to connect to the workstation's listener process by attempting a connection to a specific port. If the listener responds, then the host may assume that the workstation is capable of performing the secure login, and if not, the host understands that access is being attempted from an ordinary terminal, or via an ordinary Telnet (or other) connection, and the login challenge takes the usual (insecure) form.

With this arrangement, the system administrator is free to control access to the system by appropriate entries in the two access control "data bases". The ordinary password file (usually known as `/etc/passwd` in Unix), will control the usual terminal-oriented password challenge. The special data base of strong keys controls the secure password scheme. A user may have entries in either or both of these. In practice,
because the password file contains more information than just the encrypted password, each user of the system must have an entry in this file, in order that the system may set up the user's home directory, preferred shell, etc. However, it is possible for the entry corresponding to a user to include an "encrypted password" for which there is no corresponding plaintext password. Thus, any attempt by this user to log in using the normal method would fail. However, this entry would be used to provide the necessary additional information, if the user were to be successful in gaining access through the secure route.

It follows, therefore, that each user who has an entry in the strong key data base, must also have an entry in the standard password file, even though the encrypted password field might be such that a standard login by this user is not possible.

There could also be a case for giving each user a "dual identity", by essentially duplicating the /etc/passwd file for use with the secure login mechanism. In this way, it would be possible to allow a user with certain access rights to be denied those rights, but to be allowed access with reduced privileges if the insecure authentication method is being used. Ideally, a user should be able to access all those facilities for which s/he has permission while logged in using the secure authentication method, and a subset of those same facilities when the insecure method is being used. Sadly, the Unix access control mechanisms are not sufficiently discriminating to allow such a fully hierarchical arrangement of access rights to be employed.

Throughout the discussion, we have assumed that the DES encryption algorithm has been used. Despite the restrictions imposed by the U.S. Government on the exportation of DES code, implementations of the DES algorithm abound, in various languages, throughout the world. No doubt the publication of the algorithm in several books (e.g. Tanenbaum [11], Davies and Price [12]) have contributed to this. We have chosen to use the DES method, partly because the programs to implement it were
available to us and partly because it is believed to be secure. However, the system has been designed in such a way that the substitution of any encryption/decryption technique can be easily achieved.

The DES algorithm has been criticised for the time it takes, particularly when implemented in software. However, since this technique requires the algorithm to be applied once for each group of four characters in the password, and once more for the encryption of the nonce, we do not regard the speed of the algorithm as a serious drawback to the method.
References


