A User Authentication Scheme Not Requiring Secrecy in the Computer

Arthur Evans Jr., William Kantrowitz
M.I.T. Lincoln Laboratory
and
Edwin Weiss
Boston University

In many computer operating systems a user authenticates himself by entering a secret password known solely to himself and the system. The system compares this password with one recorded in a Password Table which is available to only the authentication program. The integrity of the system depends on keeping the table secret. In this paper a password scheme is presented which does not require secrecy in the computer. *All* aspects of the system, including all relevant code and data bases, may be known by anyone attempting to intrude.

The scheme is based on using a function H which the would-be intruder is unable to invert. This function is applied to the user's password and the result compared to a table entry, a match being interpreted as authentication of the user. The intruder may know all about H and have access to the table, but he can penetrate the system only if he can invert H to determine an input that produces a given output.

This paper discusses issues surrounding selection of a suitable H. Two different plausible arguments are given that penetration would be exceedingly difficult, and it is then argued that more rigorous results are unlikely. Finally, some human engineering problems relating to the scheme are discussed.

Key Words and Phrases: operating system security, security, authentication, passwords, one-way encryption, cryptology

CR Categories: 4.39, 5.39

Introduction

Modern computer systems are frequently used to store information which is not to be divulged to unauthorized persons. A problem that arises in such a system is authentication: establishing that a would-be user is in fact the individual he claims to be. A common approach to solving this problem is to give each user a piece of information, usually called a password, which is known by no other person, and then to accept him if he knows that password.

For the purposes of this paper we ignore such very important issues as communication line security (against wire taps and such) and operating system security (for example, against reading data from system buffers). We confine our discussion to that part of the operating system which checks to determine whether or not the password entered by the user is the one recorded for that user.

The Usual Approach

The usual password scheme involves a hidden Password Table, stored in a file which can be read by the authentication program but not by the average user.[1] The file contains a table associating a password with each user. A user is accepted as being properly authenticated if he knows the password associated with his name in the table. The scheme depends for its success on being able to keep the Password Table secret: the operating system design must make it possible to prohibit access to this file for all users other than the System Administrator.

There are several disadvantages to this scheme. First, it depends for its success on the correct operation of a very large part of the operating system—the entire access control mechanism. Second, the System Administrator can know all of the passwords. There is no reason why he should know them, or even be able to know them. Third, any listing of the Password Table, even if obtained for a valid purpose, may be inadvertently seen by an unauthorized person. Fourth, anyone who can obtain physical access to the computer, such as an operator, may well be able to print the file. A final disadvantage is that it cannot be implemented at

This work was supported by the Advanced Research Projects Agency of the Department of Defense. Authors' addresses: William Kantrowitz and Arthur Evans Jr.: M.I.T. Lincoln Laboratory, P.O. Box 73, Lexington, MA 02173; Edwin Weiss, Boston University, Boston, MA 02215.

[1] Of course, there is a System Administrator—a person—who can both read and alter this file.

all in an environment whose file system security does not protect against unauthorized reading of files. The APEX Time Sharing System on the TX-2 computer at Lincoln Laboratory is such a system.

The Scheme Proposed

As a point of departure for an attempt to circumvent the disadvantages just listed, we have proceeded on the assumption that *nothing* about the scheme itself can be hidden. That is, we assume that all aspects of the scheme—except the passwords themselves—can be known by anyone attempting to intrude. Of course, we are assuming that suitable system security keeps the intruder from being able to read program temporaries, buffers, and other memory that holds some or all of the password while it is being processed.

The proposed scheme works like this. As in the usual system, a person requesting access gives his name and his password, P. The validation program computes the value of a function H at the point P, and it then looks in the Password Table for the entry corresponding to the name. It finds there some entry, E. If $E = H(P)$ the individual is accepted as having been authenticated, the assumption being that only he knows a value of P that gives rise to the tabulated E.

For the purposes of the present discussion, we assume that anyone attempting to intrude has access to both the Password Table and the function H.[2] Since the integrity of the system is destroyed if the intruder can deduce a P that gives rise to any single E found in the table, what is needed is that H be very hard to undo, or invert. Note that the obvious brute force attack is not practical if the space is large enough; for example, at one try per microsecond it takes about 150 million years to try 2^{72} passwords. This is not an argument that the scheme is safe—only that an approach more sophisticated than mere brute force is needed to circumvent it. That there is no safety in numbers alone is a well-known cryptologic result.

One possible approach is to select a function H which is mathematically very hard to invert. One would then depend on mathematical analysis to try to establish the validity of the scheme.[3] Our approach is different; we have selected a function that is computationally hard to invert. Mathematically, we do not know how to analyze the function at all, much less invert it. We achieve this effect by having available an extremely large family of functions and selecting among that family depending on the password. That is, given a password P we determine a function F_P whose behavior depends on P in an extremely complex manner. Then the tabulated value in the Password Table is $F_P(P)$. Given the flexibility of the modern digital computer, the selection process can be "extremely complex" indeed!

[2] The function H is a computer program. By "accessing H" we mean that this program can be read.

[3] A solution using this method is described in [5]. The present scheme had already been developed when we first learned of Purdy's work.

Details of the Scheme

Before presenting the details of the scheme, a bit of overview seems in order. The computation of the function H is done in J cycles, each of which converts a value from the password space into a new such value. The first cycle starts with the original password P; each successive cycle starts with the value calculated by the previous one; and the last cycle produces the value compared with E. Each cycle is parametrized by a different value, so that each cycle is computationally distinct. The function calculated by a cycle is called F_x, where x is the parameter. The first cycle is parametrized by P itself, and the parameter for each successive cycle is calculated by applying the function *NextX* to the parameter of the previous cycle.

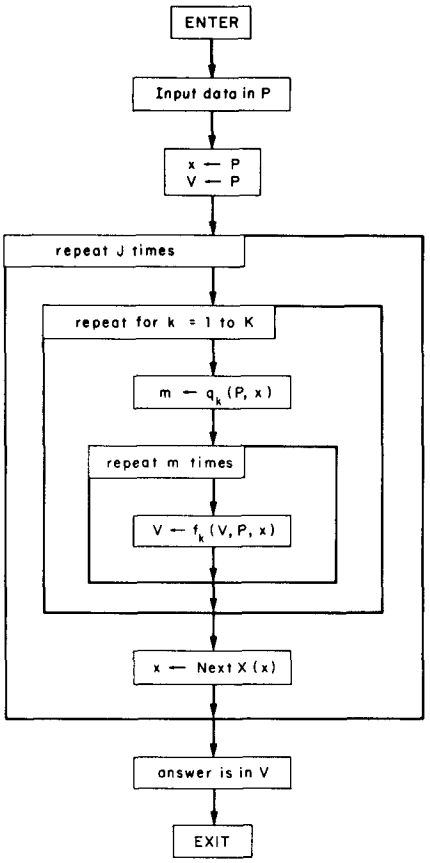
Each cycle consists of successive application of K scrambling functions f_k, each of which replaces the current value by a new one. Each scrambling function is parametrized by x (the parameter to the cycle) and/or by P (the original password). Further, each scrambling function is repeated m times, where m is a function of (the current) x and of P.

Note the nesting. There are J cycles, each consisting of K scrambling functions. Each cycle is parametrized based on a value x, and both x and P are used both to parametrize the scrambling functions and to determine the number of replications of each of them.

For convenience, all computations are performed on values from the space I of n-bit values. It is assumed that P is a value from that space determined in some convenient way from the password originally entered. The following terminology is used.

- n The number of bits in the password and in the various values derived from it.
- I The space of all n-bit quantities.
- P The password entered, from I.
- E The associated table entry, from I.
- H The entire function that goes from P to E.
- F_P The version of H determined from a given P.
- J The number of cycles.
- F_x The function for a single cycle, parametrized on x.
- x The value used to parametrize the current cycle, from I.
- K The number of distinct scrambling functions f_k.
- q_k Functions from $(I \times I)$ to small integers. These are applied to P and the current value of x to determine the number of times to iterate the current f_k.
- f_k The scrambling functions. Each of the f_k scrambles a value from I, returning a new value from I. The set of values returned by each f_k includes all possible values from I. Some of the f_k may be paramet-

Fig. 1.



rized based on P or on x. This parametrization is indicated in the flow chart by showing each f as taking three arguments.

The overall transformation is described by the flowchart of Figure 1. The flowchart uses the identifiers listed above, as well as the following.

V The current value to be scrambled. Each application of an f_k replaces V by a new value.

m The number of times the current f_k is to be applied.

k A counter from 1 to K, to count through the scrambling functions.

NextX A function that goes from one value of the parameter x to the next one.

The inner loop represents the m-fold replication of f_k. The loop on k represents one cycle through all of the scrambling functions, and the outer loop represents the J cycles.

The Intruder's Problem

It is instructive to see, at least intuitively, why it is so hard to attack this scheme. Suppose that the table entry is E, that the last function used (i.e. the last f_k) is g, that the value to which g was originally applied was Q, and that g was executed m times. (That is, $E = g^m(Q)$.) We assume that the intruder knows E and that he knows all about g—including the inverse function g^{-1}. But this does not help him very much since he does not know m. Further, m is dependent on the *original* password and bears no discernible relation either to Q or to E. Thus applying g^{-1} repeatedly to E and looking at the successive results is of no help. The values that appear have essentially nothing to do with anything he knows, so when he gets Q he is unable to recognize it. There are no available stepping stones in going backward from E to the original P. The problem is harder as the number of different f_k gets at all large, and even harder as the number of cycles grows.

Relation to Cryptology

It seems useful to relate this work to well-known results from the field of cryptology. A *cryptosystem* (see [6] for a mathematical definition, or [3]) is a large family of *encrypting functions*. To encrypt a message, one function is selected from this family, the selection being specified by a *key*, and the message is transformed by this function to a *cryptogram*. The intended recipient, knowing the key, knows what encrypting function was used and applies its inverse to decrypt the cryptogram, thus retrieving the original message. It is usually assumed that a would-be intruder (usually called the "enemy" in cryptologic literature) knows all about the family of functions (the cryptosystem) but does not know the key used for a given message.

For a function to be useful for encryption, it must have an inverse, so that the intended recipient can retrieve the original message. For our purpose we require a function which we, the designers of the scheme, do not know how to invert. The closest analogy to cryptology would suggest that our password is the key, but this analogy seems rather forced when it is realized that the only message encrypted with that key is the key itself. Wilkes discusses this point briefly in [8], and suggests the term "one-way cipher" for this kind of scheme. The scheme somewhat suggests the autokey encryption technique.

The work of Horst Feistel and others at IBM's Thomas J. Watson Research Laboratory is related to our effort. This group has been working on schemes for secure communication between terminals and computers, using sophisticated encryption techniques due to Feistel. Our use of cycles is similar to Feistel's idea of "rounds." See [7] for an overview of the scheme, and [1] for a more detailed description of the encryption methods used. A less technical discussion of this and other issues of computer cryptology may be found in [2].

One idea of importance from cryptology is the "mixing function," a function with the property that

each bit of the output is dependent on every bit of the input. It is pointed out in [6] that mixing functions are very desirable cryptologically. An important design criterion of H has been that it be a good mixing function. One way we achieve this effect is to use each bit of P in at least one of the m. Proper selection of the f_k further insures good mixing.

It is mandatory that at least some of the f_k selected be nonlinear functions, in the sense that each bit in the output be a nonlinear function of the input bits. This point is developed further later.

Some Proposed Functions

Some possible families of functions for the f_k are now listed. Since the space I is of n-bit quantities, each function under consideration maps $A = (a_1, a_2, \ldots, a_n)$ to $B = (b_1, b_2, \ldots, b_n)$.

1. Permute. The bits of B are some permutation of the bits of A. Some possible permutations are the following.
   - — Shift c bits. Here each $b_i = a_{i+c}$, the addition in the subscript being modulo n. The value of c could well be the associated m.
   - — Permute, fixed. The bits of B are some fixed permutation of the bits of A.
   - — Permute, calculated. Calculate some permutation of the first n integers, the particular permutation selected being dependent on P and/or x. The bits of B are that permutation of the bits of A.
   - — Permute bytes. This is a variant of either of the above, with whole bytes being permuted. It is mentioned separately because it could be significantly easier to implement on some computers.
2. Scramble. Deduce some n-bit quantity Q from P and/or x, and then let each bit of B be the (noncarrying) binary sum of the corresponding bits of A and Q. The m corresponding to this operation must be exactly one.
3. Table lookup. Suppose $n = c \times d$, so that we can think of A as being made up of d-bit bytes, c of them. Given an arbitrary table with 2^d entries, let each byte of B be calculated from the corresponding byte of A by treating the latter as a d-bit integer and using the corresponding table entry. To make the result one-to-one, the 2^d table entries must include each integer from zero to 2^d-1, so that the table is in fact a permutation of the first 2^d integers. An obvious variation is to use an integer derived from P and/or x as the seed for a pseudo-random-number generator, the latter being then used to generate the 2^d entries of the table.[4] The above comments about one-to-one apply.
4. Add bytes. A constant (either built in or calculated from P) can be added to each byte, with the carry (if any) ignored. A different constant could be used for each byte.[5]
5. Assemble. Select some "suitable" machine instructions from a pool, the selection being based on P and/or x, and calculate a value Q by executing these instructions starting with some bit pattern (perhaps dependent on P and/or x). Then B is the binary sum of A and Q. The result appears to be rather formidable to analyze.

Variations

The scheme just defined uses $f_1, f_2, \ldots, f_K$ in each cycle in a predetermined order. Instead one could compute from P and/or x a permutation $(s_1, s_2, \ldots, s_K)$ of the integers from one to K, and use $f_{s1}, f_{s2}, \ldots, f_{sK}$, in that order. This seems to make inverting H somewhat harder. It is of course also possible to use some of the f_k more than once—perhaps parametrized differently.

Analysis

Let H be the entire transformation just defined:

$$E = H(P) = F_P(P). \tag{1}$$

Since E and P are each n-bit quantities, we can write them in vector notation as

$$E = (e_1, e_2, \ldots, e_n)$$
$$P = (p_1, p_2, \ldots, p_n)$$

where each p_i and e_i is a binary quantity. Then an alternate way to view the transformation H is as a collection of n functions h_k, each taking n binary arguments and yielding a binary value. Thus eq. (1) can be rewritten as

$$\begin{aligned} e_1 &= h_1(p_1, p_2, \ldots, p_n), \\ e_2 &= h_2(p_1, p_2, \ldots, p_n), \\ &\ldots \\ e_n &= h_n(p_1, p_2, \ldots, p_n). \end{aligned} \tag{2}$$

The intruder knows the e_k; and the h_k are implicit in H, which he knows. His task is first to determine the h_k explicitly, and then to solve the system of eqs. (2) for the p_k.

It is immediately obvious that the designers of the system must select H so that the transformation represented by eqs. (2) is nonlinear, since the solution of n linear equations in n unknowns is straightforward. Thus it is important that H be highly nonlinear. From our point of view, this requires that at least some of the f_k selected must be nonlinear. The first two functions listed above are linear, but the last three are not.

[4] Such generators are discussed at length in [4].

[5] This scrambling function is not quite one-to-one in a computer unless the addition is done using the two's complement convention, since otherwise +0 and −0 lead to the same value.

[6] For reasons to be explained below under the section Human Engineering, n must be 64 or 72 or more, so at best the problem is nontrivial.

[7] For example, the user might have to identify himself in person to the System Administrator and then log-in from the System Administrator's console and set his own password. It is easy to think of other possibilities.

[8] This fact justifies the earlier statement that a 64 or 72 bit password would be used. The exact length would be selected to be computationally convenient.

The basic intractability of the scheme should now be apparent. Given a set of equations such as (2) a mathematician can attempt a solution using various tools.[6] The present scheme is of such complexity that it is not at all clear how to reduce it to this form. That is, the transformations defined by some of the f_k, especially when raised to an unknown power, would be very hard to put into the neat form of (2). Thus the intruder has two intractable problems: reducing the algorithm for H to the n functions h_k, and solving the resulting system of equations.

Does this mean we claim the scheme cannot be broken? No—what it does mean is that we, the creators of the scheme, have thought long and hard about the matter, and have no idea how to reduce our algorithm to the form of a manageable system of boolean equations. Further, we know of no suitable tools for solving such a system if it is highly nonlinear and if n is big enough.

Further Comments

Since the System Administrator has no way to learn a user's password, some mechanism is needed to accommodate the user who forgets it. Safeguards are of course needed, but these are procedural in nature and not relevant to this paper.[7] If the scheme is as good as we intend it to be, there is no way the user could ever retrieve his original password. All he could do is to select a new one.

Presumably anyone attempting to defeat this scheme will (among other things) implement the function H on his own computer so that he can study it empirically. Thus there is an advantage to be had by an implementer of this scheme, particularly if his computer is an uncommon one, if he codes it using characteristics of the computer that would be hard to simulate on other hardware. To the extent that such features are used, the brute force part of any attempt to break in is made that much more expensive. TX-2 has some uncommon features which are hard to simulate, such as word permutation and reconfiguration, bit operations combined with rotations, parallel arithmetic on bytes, and others.

The Issue of One-to-One

If H is the entire function produced by the scheme just described, then it seems desirable that H be one-to-one. Otherwise, there could be distinct passwords α and β such that $H(\alpha)$ is equal to $H(\beta)$. This seems to be undesirable, since then more than one password produces a given value of E, and the intruder's task of finding such a value might be eased.

Is H as we have defined it one-to-one? Probably not, although we do not expect ever to be able to decide with assurance. We have required that each of the f_k be one-to-one. It then follows that the function F_P derived from a given password P is one-to-one, since the composition of one-to-one functions is one-to-one. Thus $F_P(\alpha)$ can equal $F_P(\beta)$ if, and only if, α equals β. But that is not what is in question here. Could there be distinct passwords P and Q such that $F_P(P) = F_Q(Q)$? Nothing in our discussion precludes this possibility; indeed, H could collapse horribly. However, it appears that it is just as hard to investigate whether or not H is one-to-one as it is to defeat this scheme analytically, and we do not know how to do either.

It would be desirable if we could argue convincingly that the collapse is not too bad, but even that seems beyond our means. More investigation is needed, but the discussion above under Analysis should make it clear that results will be very hard to come by. Perhaps some experimental investigation into collapse would be useful, but the space is so large that meaningful experiments seem excessively costly in computer time.

It should be noted that computing $F_P(K)$ instead of $F_P(P)$, where K is some constant, is no better with respect to being one-to-one. We are quite unprepared to argue that $F_P(K)$ differs from $F_Q(K)$ if P differs from Q, even though each of F_P and F_Q are one-to-one.

Even though the mapping from P to $F_P(P)$ cannot be proved to be one-to-one, it still seems to be desirable that each of the f_k be one-to-one. Since this keeps the number of possible values at each step as large as possible, it makes harder the analysis involved in breaking in.

Human Engineering

We have discussed a password as an n-bit quantity, but it is clear that human beings cannot memorize such quantities conveniently. Presumably, the user will think of his password as a sequence of letters and digits, there being standard programming techniques for transforming such a sequence to a bit vector.

The intruder will find this scheme much easier to experiment with than the usual one. Having first implemented H on his own computer, he is able to study it independently of the system being attacked. In the usual scheme it is possible to test a password only by attempting to log in, but it is easy to program the log-in program to warn the operations staff of too many unsuccessful attempts. An implication of this ease of experimentation is that a rather long password is needed.[8] If the password were selected from some fairly small set (perhaps a few thousand items), the intruder could easily try each possibility on his computer.

There are some serious issues of human engineering that follow from the requirement for a long password. Faced with the need to select (say) a 12-character password, a person might well select one, such as his spouse's name or "qwertyuiop", that could be too easily guessed. One possibility is to assign random passwords, but most

people find "cFq38,Te" rather hard to live with. This issue is not pursued further in this report, but it deserves more attention before an implementation is installed.

There are some requirements on the function that converts the entered characters into a bit vector for internal processing. All possible values of P should be produced by this function. Further, it seems desirable that no bit of P be dependent on any single character of the original password.

Acknowledgments. The impetus to study this problem came from a suggestion from Lawrence G. Roberts that a password scheme be implemented for the TX-2 computer. After the scheme had been developed, Jerome H. Saltzer pointed out that the basic idea had been presented by Maurice V. Wilkes, in [8]. Wilkes gives credit to R.M. Needham for the initial idea and the first implementation, but provides no discussion of the details of the method used.

Saltzer has also pointed out that the Multics operating system uses a similar scheme, in addition to keeping the Password Table secret. The point of view seems to be that it is the integrity of the file system that provides the crucial aspect of security, while this scheme circumvents the second, third, and fourth of the disadvantages of the usual scheme as listed above.

Received August 1973; revised February 1974

References

1. Feistel, Horst. Cryptographic coding for data-bank privacy. Res. Rept. RC-2827. T. J. Watson Res. Lab., IBM, 1970.
2. Feistel, Horst. Cryptography and computer privacy. *Scientific American 228* (May 1973), 15–23.
3. Kahn, David. *The Code Breakers.* Macmillan, New York, 1967.
4. Knuth, Donald E. *The Art of Computer Programming.* Vol. 2. Addison-Wesley, Reading, Mass. 1969.
5. Purdy, George. Security code. U. of Illinois, Center for Advanced Computation, 1973.
6. Shannon, Claude E. Communication theory of secrecy systems. *Bell System Technical J. 28* (1949), 656–715.
7. Smith, J. Lynn, Notz, William A., and Osseck, P.R. An experimental application of cryptography to a remotely accessed data system. Proc. ACM 1972 Annual Conf., ACM, New York, 282–297.
8. Wilkes, Maurice V. *Time-Sharing Computer Systems.* American Elsevier, New York, 1972.

Operating Systems

C. Weissman
Editor

A High Security Log-in Procedure

George B. Purdy
University of Illinois at Urbana-Champaign

The protection of time sharing systems from unauthorized users is often achieved by the use of passwords. By using one-way ciphers to code the passwords, the risks involved with storing the passwords in the computer can be avoided. We discuss the selection of a suitable one-way cipher and suggest that for this purpose polynomials over a prime modulus are superior to one-way ciphers derived from Shannon codes.

Key Words and Phrases: operating systems, time sharing systems, security, cryptography

CR Categories: 4.35

1. Introduction

One of the managerial aspects of time sharing systems is the need to protect the system from unauthorized users. Access may be controlled by restricting certain users to certain terminals at certain times only, limiting access to the building, or by the use of passwords. Passwords, if used, must be carefully guarded; the most vulnerable part of a password system is usually the list of passwords stored in the computer.

M.J. Wilkes [1, p. 91] describes a device, due to R.M. Needham, implemented at Cambridge, England,

This work was supported in part by the Advanced Research Projects Agency of the Department of Defense and was monitored by the U. S. Army Research Office-Durham under Contract No. DAHC04-72-C-0001. Author's address: Center for Advanced Computation, University of Illinois at Urbana-Champaign, Urbana, IL 61801.