Global Information Assurance Certification Paper

Authentication model that uses One Time Passwords

GSEC Practical assignment
1.4.b option 1

1 december 2003

Alberto Benavente Martínez, Spain

Authentication Model that uses One Time Passwords

Alberto Benavente Martínez

Abstract

The objetive of this paper is to design an authentication model that uses a one-time password (OTP) mechanism. A simple software application is proposed to take advantage of some security features. The result is an authentication arquitecture, closer to a single sign-on, where a user only needs to provide his credentials once to access a set of systems.

Applications of this model can enhance the manual process of a centralized password management infrastructure through the use of one-time password token based on authentication mecanisms.

This solution can be used in an hypothetical real world situation, for example, in a Customer Support Centre managing UNIX-like servers. Some benefits are:

- low implementation costs and small technical efforts
- improve end user productivity
- avoidance of human intervention.
- time and cost reductions in order to easily meet service level agreements.
- frustrate some malicious attacks such us “Passive Attacks” and the “authentication race attack” as a particular case of an “Active Attack”. All of them will be briefly explained later on.

1 Introduction

The case of study take place within an IT organization with a Customer Support Center. The objetive is to find a model that reduces efforts and increases security. This organization gives, among others, userid and password support to a big number of Unix-like servers. Not only do they provide these services to customers but also they provide these kind of services to its own workers. They can be physically spread in more than one location, some of them working in customer facilities and providing other services such us logical security, networking support, consultancy, operations and so forth.

A lot of situations require an efficient process for managing passwords. Some examples are: emergency situations, incidents, operator rotation, employee dismissal, business needs, job role changes, employees availability, separation of duties, password lost, etc. The frequency is proportional to the number of systems and accounts to be managed. In these cases it is important to deal with password availability and disclosure

Physical locations add complexity, more so if we want to design a simple model. This one will take into account that some servers are placed in different networks. It also depends on the customer topology. Some servers are not necessarily interconnected to each other. It could happen that, for these or other reasons, temporarily or permanently, some machines could be unreacheables. All could be important and should be taken into account when managing servers in a Customer Support Centre. Some of them could be easily overriden while others will be exceptions. In any case, it seems that password availability and password deployment is one of the most important issues to consider.

A simple procedure to change passwords includes: the reception of a notification through a 'secure' path, people processing these notifications, people 'sharing' privileged userids, people entering into systems with privileged accounts, execution of the command that changes passwords, set passwords with the appropiate syntax and, finally, send a notification back.

Human errors add risks that can be reduced. Let's imagine a group of people assinging passwords during the whole day. The following examples take place: mistakes when providing the correct password, different password policies depending on the customer and not necessarily implemented as a default operating system feature (password complexity), people without a real business need for knowing the value of passwords, provision of the same password within different customers in order to save time, password well known among workmates, etc.

2 Password and pass phrase authentication mechanisms

There is a type of authentication scheme that can be classified as "user to host" (or user to application). In this case an entity, such us an individual or program, initiates a request to the authentication program. This is usually done by entering his credentials and waiting for an answer. Userid and passwords strings are frequently used to identify and to authenticate these entities (in fact this method is widely accepted). The authentication program acts as a "server" requesting credentials and giving a particular criteria validating "clients", that is, controlling that an entity is really who he claims to be.

This model sounds fantastic but different implementations will show us different vulnerabilities. In the case where client and server programs reside in different machines (open networks), clear text passwords transverse communication channels, some of them public, some others semipublic or more secure. The fact is that clear text strings, called passwords or pass-phrases, could be found out by what are called sniffer tools. In this case an eavesdropper, or individual doing this malicious activity, will only have to catch it and then provide it again to the server application, impersonating the real owner.

Thus the problem has to be something to do with the communication channel and/or some characteristics of the string that validates userid's credentials. So

the first two things that could be addressed are: either we encipher communication channels between the server and the client so that a secure path is established between them, or we change some properties of the password. The first one still needs a previous exchange of some kind of information. This could take the shape of a secure exchange of a secret key that can be shared between these two parts or a public exchange of some kind of information (that could be associated to a secret on each sides of the communication). This seems to be the basis of a infrastructure that we could try to use to make the 'sniffed' information illegible. Different implementations based on secret and public keys algorithms were developed and some products that are *de facto* standards*,* like Kerberos, add stronger authentication and authorization methods. They take advantage of the best qualities of them. Different combinations of secret key, public key and hashing are mixed up to deal with confidenciality, integrity, non-repudiation and, in some cases, availavility. There is an analogy between IT evolution and these schemes, the more evolved the more complex.

3 Challenge/Response Scheme

There are two entity authentication methods called synchronous and asynchronous (e.g. challenge/response). Both of them use an irreversible mathematic function whose characteristic is that in spite of knowing its output and its definition it is not posible to calculate its input value. This functions are used on irreversible cryptographic schemes and are applied to authenticate entities.

One may think think that choosing these functions can be difficult but it is not. We only have to choose an appropiate cipher algorithm 'F' , then for a given a clear text 'Ti' and a key 'Kj' we proceed to cipher 'T' with 'F' . Doing this we have a different output C_i for every different input 'Kj'.

$$C_i = F(T_i, K_j) \forall K_j \in \Re$$

Fig.1

The challenge/response procedure uses this kind of irreversible cipher funtions to deal with one-time password authentication. One Entity Systrem E.S. (program, user, process, etc), Fig.1. ID, provides its credentials [1] to the Authentication Server A.S. The A.S will store all secret keys 'Kj' in a local database. All recognized entities will have an entry in this repository so that, given the userid string (ID), its secret key is retrieved (steps [2] and [3]). At this point a random number in generated and is given to E.S. That is called a Challenge [4]. The same ramdom number is kept internally. This number and the retrieved secret key will be the inputs to the irreversible cipher Function F [5']. At the same time the challenge arrives E.S. and it operates in the same way taking the proposed challenge and its well known secret key Kj and inserting them in a similar function F. The output is sent back to A.S as a response [5]. At this moment the Authentication Server only has to compare its internal value with the response [6]. If both results are the same, the userid is authenticated [8], if not another challenge is sent for another attempt[7].

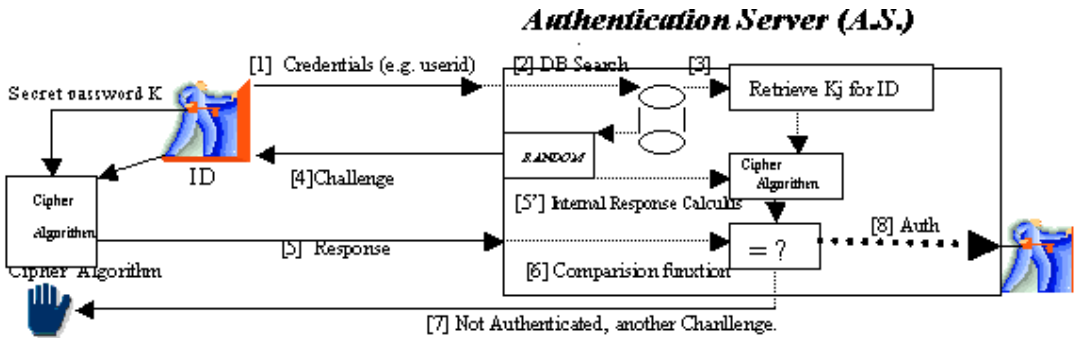

Fig. 2

This model was used in GSM (Global System Telephony) to authenticate userids. In this case the private key is kept in the end user SIM module and it is also stored in a remote authentication center that functions as an A.S. This one provides a 128 bits random challenge which is, in conjuntion with the private key, ciphered by an irreversible algorithm within the SIM module. The output value SRES is a 32 bit string that acts as a response ([1]Ribagorda, p10-5).

4. One-Time password algorithm

At this stage we need an algorithm that, independently of the F function, can provide a valid set of future passwords, all of them generated from an initial secret and being valid only once.

Let 'F' be an irreversible and appropiate function, 'K' the secret password and 'M' the number of one-time passwords we need. Let M be equal to 100. The Authentication Server has this information and calculates F(K) as many times as the number of one-time passwords are required (in this case 100 times). After that the AS keeps this value. Therefore the first time an entity wishes to be authenticated it will be given the sequence number 99. The entity will know what F function to use and, of course, his secret password K. At this point the entity calculates $F^{(99)}(K)$ (being this superindex the number of times the function F is applied) and provides its output to the A.S who also calculates $F(F^{(99)}(K)) = F^{(100)}(K)$. If this value is the same as the one previously stored in the A.S., the entity will be authenticated. It will also store the new value $F^{(99)}(K)$ and delete the old value $F^{100}(K)$. So, the next time the same entity requires this service, it will be given the sequence number decreased by 1. The process can be repeated up to the limit of generated sequences, 100 in this case. Changing the K secret in the 101st round will lead another 100 more one-time passwords and so on ([2]Lamport).

5 Passive Attacks vs Active Attacks

A 'passive attack' on an authentication system consists of monitoring information sent between two or more parties without inserting any kind of data. On the contrary, an 'active attack' modifies transmited data in order to be authenticated or to get authorization .

Traditional authentication technologies based on passwords are victims of passive attacks because they send them through networks and are potentially intercepted. One-time password are better authentication mecanisms because they are vulnerable to active Attacks but not to passive attacks. There is an active attack called 'replay attack' where the given password is intercepted by an unauthorized entity and then sent to the authentication system.

More complex authentication schemes are neither victims of passive nor active attacks. This technologies use mutual authentication between two or more entities. ([3] Haller).

6 The importance of a Challenge

When the authentication entity and the Authentication Server reside at different locations, information exchanged will travel through a communications network. Because information does not reside in a single location we are adding an extra insecure component. Lamport's algorithm is subject to attacks, one of them are known as the 'race attack'. ([4] William, p.104)

Imagine an eavesdropper in between the entity and the granting server that could read in clear text the information that flows over the network. He also realises that for a given sequence number, an entity answers each time with a different response. After analysing the response he learns that it ever has the same size and is made up of one kind of elements, (e.g.characters). So what he really needs in order to impersonate the entity, is to provide the next response before the entity does and to inhibit the correct response from reaching its destination. This has to happen just before the authentication event. So what we need is a program that functions as a 'relay'. While the entity is providing its response, the program is sequentially keeping one by one these elements and generating the same response up to the last character. At this moment the program must also have started as many sessions with the Authentication Server as many possible characters in the last position. Before the entity can enter its last character, the relay program sends all possible responses, each one over a different connection. Then, for sure, one will be correct and authenticated and the valid entity will be rejected.

This attack is possible because somebody can access the information and can force the access, it could be prevented by providing a different challenge on each potential connection. Another possible solution can be a small a PKI infraestructure where the entity receives the challenge in clear text, proceeds to

sign it with its private key and afterwars send the result back to the A.S. This one, by using the entitie's public key, ensures that the returned response belongs to that entity.

There are commencial products that uses this scheme, for example OmniGuard/Defender from Axent Technologies (www.axent.com).

7 Hash functions

These funtions take elements of a large Domain and transform them to a reduced, but still large Image. Given an output value, also known as a message-digest or hash value, is computationally infeasible to invert, in other words, to compute the element of the Domain that has generated this output. This also means that there can be collisions, that is more than one input value having the same output. Hash funtions are choosen so that two ramdom input values will have 'more or less' the same collision probability.

High performance hash functions were designed for 32 bit machines. One example is MD4 (Message Digest 4) and its successors: MD5, SHA-1 (Secure Hash Algorithm) and RIPEMD. MD4 and MD5 takes any number of bits as input and retrieves a 128 output string while SHA-1 and RIPEMD use160 bits. MD4 is not recommended in some circumstances because it has suffered a large number of collisions under specific circumstances. In these cases MD5 is stronger than MD4 and, moreover, NIST's SHA is stronger than MD4 and MD5 (against brute force attack). On the other hand the stronger the slower. MD5 is around 30% slower than MD4 and SHA around 70% slower than MD4. ([5] Menezes, ch. 9).

8 OPIE One-Time Password System (OTP)

OPIE stands for *O*ne-*T*ime *P*assword *I*n *E*verything and was released by the U.S. Naval Research Laboratory (NRL) in 1994. This product is based on S/KEY OTP released by Bell Communications Research (Bellcore) but with some major enhancements. For example Bellcore's 1.0 package was designed with 4.3BSD UNIX system in mind while OPIE covers a bigger spectrum of UNIX-like operating systems because it ensures portability through POSIX calls.

OPIE protect us only against passive attacks and against some active attacks. It does not protect us against dictionary attacks directed towards weaknesses of our secret keys. For this reason it has to be used as one of the multiple elements of our layered security infrastructure.

OPIE and S/KEY are based on a challenge/response mecanism where the challenge is made up of a sequence number and a seed. Responses (or one-time passwords) are not stored within the Authentication Server, only Secret Keys are kept.

One of the improvements from its ancestor is the cryptographic hash function. S/KEY uses a particular adaptation of the MD4 hash function because it takes a 64 fixed length character string for its input and also returned a 64 bit string (instead of arbitrary/128 input/output relationship) ([6]Haller). In contrast OPIE includes a MD5 hash function making brute force cryptanalysis harder and support MD4 to interoperate with its ancestor([7], [8] Rivest). OPIE also includes a FTP daemon and modifications of login (opielogin) and su (opiesu) programs in order for us to be able to choose the authentication scheme between challenge/response OTP or our traditional passwords.

9 OTP on the Internet

In order to download the source code from the Internet some web sites redirects us towards the NRL web site, ftp://ftp.nrl.navy.mil/pub/security/opie/opie-2.3.tar.gz. Unsuccessfully, at the time of the test, it was not be possible to download the code. Another site also offered us OPIE http://inner.net/opie. Their OPIE OTP portable source code could be found as well as some other pieces of software that interoperates with it. Examples of these one are some multiplatform client response generators, linux PAM module and so forth.([9] SGI) ([10]Berkheimer).

Once downloaded 2.4 distribution package (opie-2.4), a test server with RedHat 7.1 linux was choosen as our future OTP Authentication Server, so we proceeded to download the file 'opie-2.4.tar.gz' to a destination directory. There we decompressed the package (# tar -zxvf opie-2.4.tar.gz) so that the following files and directories were placed in the opie-2.4 directory (Fig. 4):

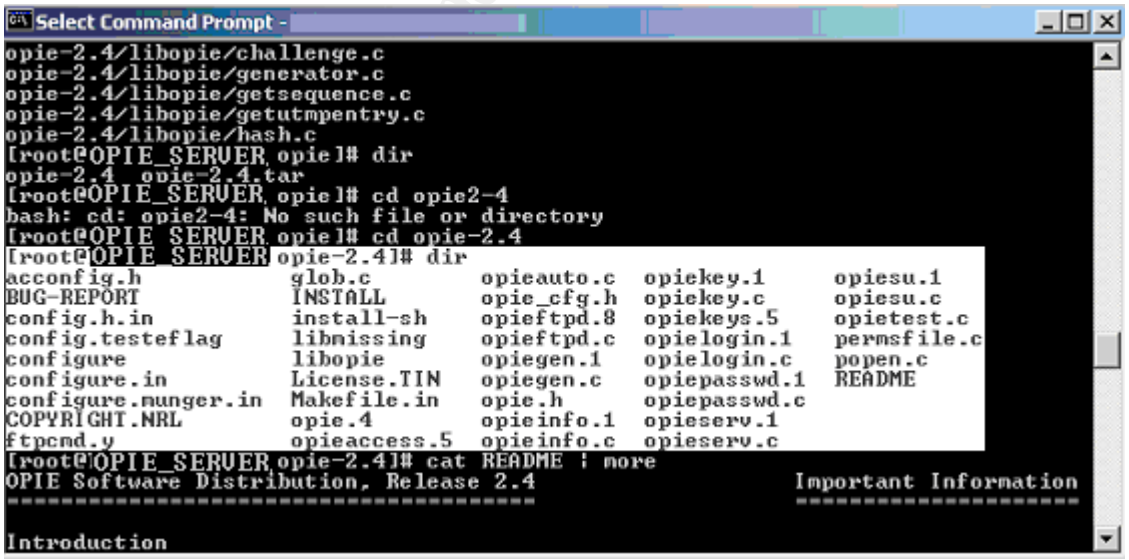

Fig. 3

The first useful resource is the README file. There we found a lot of information about bugs, fixes, enhancements as well as system requirements. Recommended system requirements are, literally, the following:

- A UNIX-like operating system
- An ANSI C compiler and run-time library
- POSIX.1- and X/Open XPG-compliance (including termios)
- The BSD sockets API
- Approximately five megabytes of free disk space

We can see the following source programs: *opieauto,* which will be the 'equivalent' of /hosts.equiv and contains the list of trusted hosts that bypass some authentication controls. In this exercise we do not use this facility. *Opieftpd* program is the opie's ftp daemon that will substitute our former ftpd daemon. *Opieinfo* is a function that let userids check their current challenge, sequence number and seed. *Opiekey* program is the response generator for a given sequence number and seed. The first time this function is called, it ask you to enter your secret password. *Opielogin* and *Opiesu* programs are OTP substitutes of the well known *login* and *su*. ([11] IETF) ([12] Metz)

10 Customization

OPIE has a configurable installation script *./configure* in order to make installation easier and, as far as possible, to fit the local machine characteristics. There are some configuration parameters which can be listed executing the *./configure –help* command. In this case we have choosen the following configuration: ([13]Craig), ([14] Neo).

```
[root@OPIE_SERVER opie-2.4]# ./configure –help
./configure --prefix=/usr --enable-insecure-override --enable-retype
```

The first parameter of this command *prefix=/usr* will place installation files in /usr file. In particular the following binary files:

```
/usr/bin/opiekey
/usr/bin/opiepasswd
/usr/bin/opieinfo
```

The second parameter *--enable-insecure-override* will try to determin by default which connections come from a secure source (local console or equivalents) and which not. This will be the most important configuration step. Some actions, such us entering our secret pass phrase, should not have to be done using insecure method.

The third command is the *--enable-retype* which permit us to enter twice our secret pass-phrase instead of once each time it is needed.

The autoconfiguration process will check which programs, {login, ftp,su} are in place in order to replace them:

```
checking for su... /bin/su
checking for login... /bin/login
checking for ftpd... no
```

It also checks POSIX portability:

```
checking for sys/wait.h that is POSIX.1 compatible... yes
```

11 OPIE Server Installation

Once we have set and finished our configuration requirements we will start the compilation. Two different compilations are possible, server mode and client mode. Then we proceed to install the OTP server by executing the appropriate *makefile* command. Reviewing compilation results we could check, for example, that MD5 is used, not only MD4, as well as opie's program substitutions that take place during this process.

```
[root@OPIE_SERVER  opie-2.4]# make server-install
Installing OPIE server software...
...
 (8/13) testing opiehash(MD5)... passed
...
Copying OPIE user programs
Changing ownership
Changing file permissions
Installing OPIE system programs...
Copying OPIE login to /bin/login
Changing ownership of /bin/login
Changing file permissions of /bin/login
Copying OPIE su to /bin/su
Changing ownership of /bin/su
Changing file permissions of /bin/su
Making sure OPIE database file exists
Changing permissions of OPIE database file
Changing ownership of OPIE database file
Creating OPIE lock directory
mkdir: cannot create directory `/etc/opielocks': File exists
Installing manual pages
REMEMBER to run opiepasswd on your users inmediately.
[root@OPIE_SERVER opie-2.4]#
```

At this point we have successfully installed OPIE OTP Authentication Server in our system. If we had some problems we could execute *Make uninstall* command to reserve the installation, that is, a system without OPIE.

12 OPIE client installation

We have just installed the OPIE server. Then what we need is another system on which to install an OPIE client to generate the appropiate responses. The same operation as described before but, in this case, we have to execute the *make* command with a slight diffencence:

```
[root@OPIE_Client opie-2.4]# make client-install
make: Nothing to be done for `server'.
....
Installing OPIE client software...
Copying OPIE key-related files
Changing file permissions
Symlinking aliases to opiekey
Installing manual pages
...
```

OPIE's clients do not necessarily have to be installed in the same operating system as the OTP server nor in the same machine. The client side will only include the logic needed to properly answer server challenges. There is a client version for 32 bits Windows machines called Winkey32 released by Technologic (Fig. 4 and Fig. 5). ([15] Gaetz).

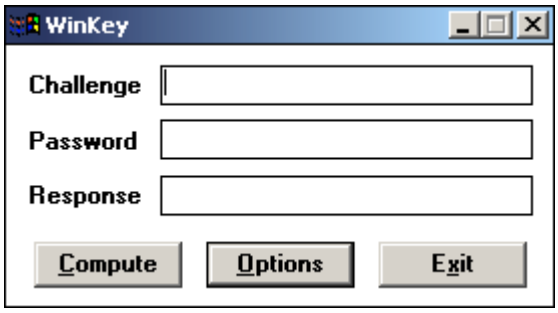

Fig. 4

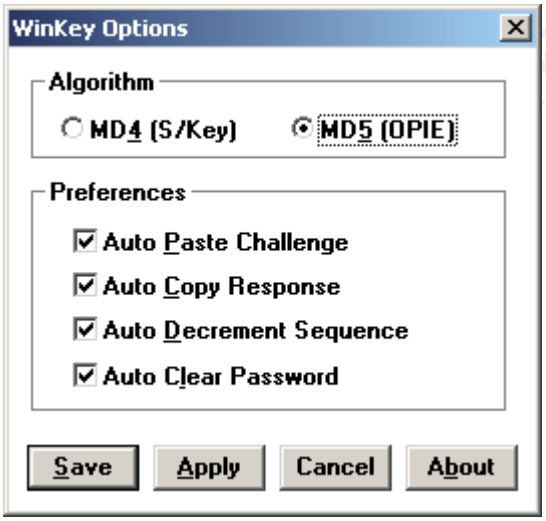

Fig. 5

There are some web pages that also offer JAVA 'clients' which calculate responses to a given challenge (made up of a sequence and a seed) and entering our secret pass-phrase. That sounds good because no matter wherever we are we could get our next non-reusable password but, because it is better not to disclose this important piece of information, another method should be used. Examples of these clients can be found in the internet. ([16] Mantakos).

13 OTP Usage

At this stage we have an OPIE Server installed so that we have two possible authentication methods. The old one, based on userid and password, and the new one based on challenge/response or challenge/pass- phrase. Next time an account tries to use an authentication function (login, su or ftp) we can notice some changes. For example, if we call the login function a prompt line will ask for selecting between a password or a token.

Let *spopie* be a system account that has never used challenge/response mecanism since OPIE Server was installed. Next time *spopie,* or another, wanted to login it will be asked to provide his traditional password or the new otp response password. We can then choose which method to use: the old method or the new one.

```
[spopie@OPIE_SERVER_]$ login root
otp-md5 492 TO1000 ext
Response or Password:
Last login: Wed Sep 17 09:47:16 on tty1
You have new mail.
```

If we decided to continue using passwords (/etc/passwd and /etc/shadow) there is no problem, the substituted *opielogin* program will act as the old one. On the other hand, if we decided to take advantage of one-time passwords, we will not know what to answer to the following question “enter a Response…”. To get the first response, for each account, we should ‘migrate’ the authentication scheme. The migration process introduce two programs *opiepasswd* and *opiekey*.

In the case where we attempt to remotely login using telnet, for example, we will be only prompted for a response for a given ramdon challenge. Because this userid is still using passwords he has to be migrated to the new scheme. *spopie* will try to get its first challenge, to do this he uses *opiepasswd* function, Fig. 7 (used in conjuntion with –c argument will not let us acces from an ‘insecure’ method) :

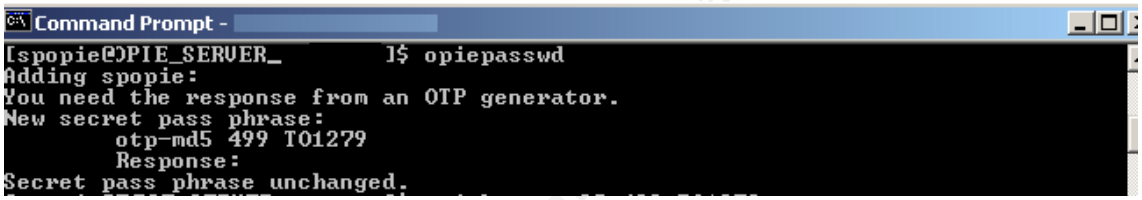

Fig. 6

In the figure the challenge contains three different parts and it starts with *otp-md4* or *otp-md5* this indicates the hash funcion in use. The next parameter is the sequence number (by default set to 499) and the last one is the seed (or salt) for this specific challenge. So we are asked to enter a “response” but we have not generated any response.

We must first choose a private pass-phrase (or private key) that will be stored in the OPIE server. Due to security matters it is highly recommended to enter this value locally at system console or through a secure path. In this particular case we call *opiekey* function by opening a console in the OPIE server. Arguments passed to this functions are: –n 5 the number of OTP with its related sequence number, the sequence number (499) and the seed (TO1279) Fig. 7.

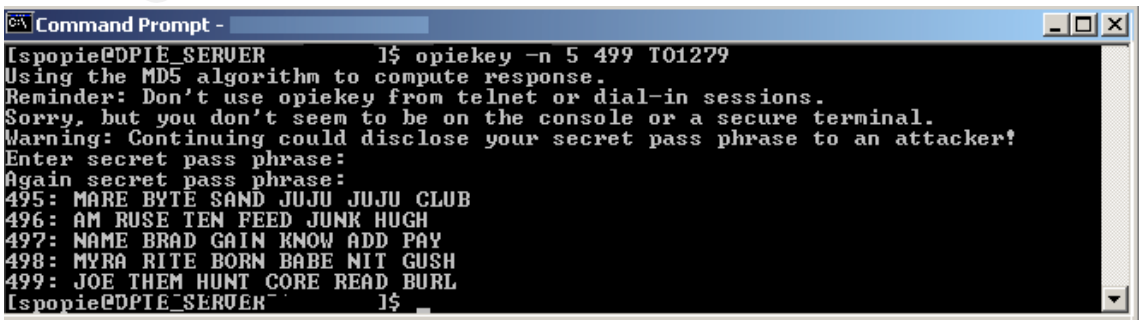

Fig. 7

We are asked to enter our secret pass-phrase twice, which can be up to 127 characters long, and then a new entry for our account is added to /etc/opiekeys ; also in this example the first next five OTPs are shown. Next time *spopie* tries to login into the system it will be prompted with the 498th challenge and seed and the response or pass phrase OTP will be MYRA RITE BORN BABE NIT GUSH. Now we will proceed to use telnet from a remote machine to access using this pass phrase:

Red Hat Linux release 7.1 (Seawolf)
Kernel 2.4.20-19.7 on an i686
login: spopie
otp-md5 **498 TO6890 ext**
Response: **MYRA RITE BORN BABE NIT GUSH**
Last login: Tue Nov 11 19:27:48 from X.X.X.X

At this point we could migrate all userids of a system from the old password authentication scheme to the new one. One of the aspects a system administrator could consider, is the case where all userids in the system lose or forget their secret passwords so that they could not use *opiekey* function. In this case if we are logged on as root, the *opiepasswd* function let us manage all userids.

The following sequence also shows that, for different authentication sessions, different seeds are generated for the same sequence number. This would prevent a possible 'race attack' started with parallel opened sessions as explained before.

```
[root@OPIE_SERVER_]# opiepasswd
Adding root:
You need the response from an OTP generator.
New secret pass phrase:
        otp-md5 499 TO7391
        Response: albertobenaventemartinez
        That is not a valid OTP response.

        otp-md5 499 TO7391
        Response: ALBERTOBENAVENTEMARTINEZ
        That is not a valid OTP response.

        otp-md5 499 TO7391
        Response: alberto1benavente
        That is not a valid OTP response.

[root@OPIE_SERVER_]# opiepasswd
Adding root:
You need the response from an OTP generator.
New secret pass phrase:
        otp-md5 499 TO4457
        Response: fhdhhfydsfjfdd

        That is not a valid OTP response.

        otp-md5 499 TO4457
        Response: xxxxxxxxxxxyyyyyyyzzzzzz
        That is not a valid OTP response.

        otp-md5 499 TO4457
        Response:
Secret pass phrase unchanged.
[root@OPIE_SERVER_]# exit
```

14 Bypassing OTP

Some products permit us bypass OTP challenge/response scheme. Let's see for example Webmin software, where we could still change old passwords, that is modifying /etc/passwd and /etc/shadow entries, although these accounts do not use the old scheme anymore (Fig. 8).

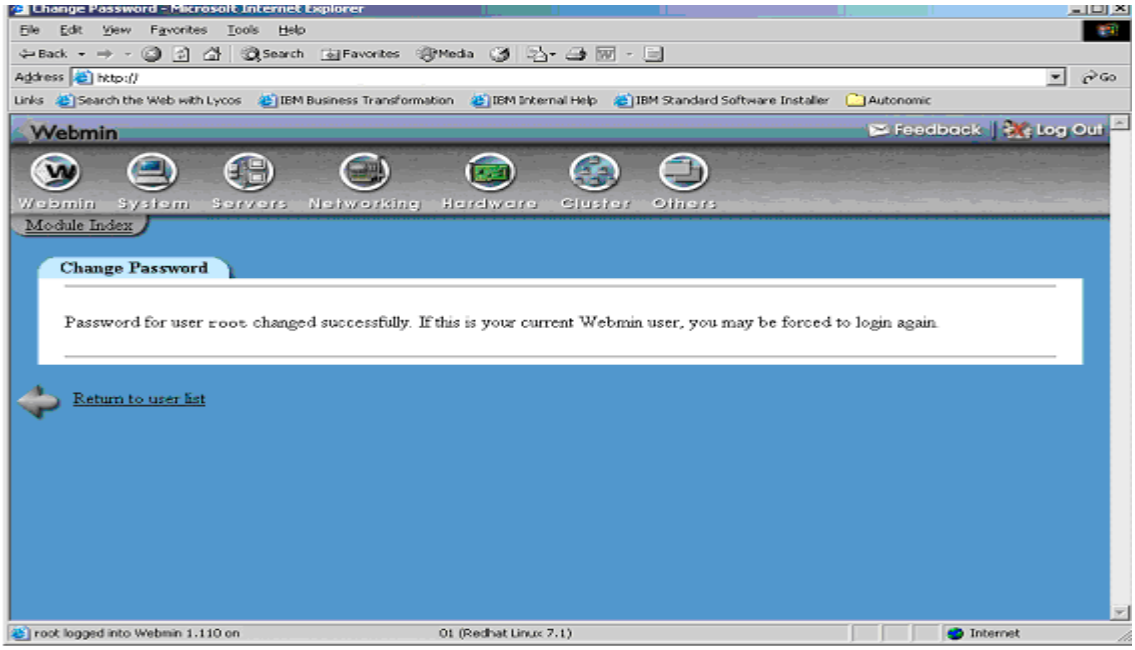

Fig. 8

15 Uninstalling the product

In addition to executing *Make Uninstall* command to return to the initial state, we will also have to restore those programs that were substituted during the installation phase such us login and su. Once the initial situation is restored we could proceed to reboot the system to let changes take effect. The commands to do this are:

```
rm –f    /opt/opie/opie-2.4/opielogin; mv  /bin/login.opie.old; chmod  755 /bin/login.opie.old
rm –f    /opt/opie/opie-2.4/opiesu;    mv /bin/su.opie.old;  chmod  755 /bin/su.opie.old
```

16 OTP *de facto* standard

Some Web sites speak about OTP as a *de facto* standard proposing independent implementations as a standard. ([14] Haller, Metz.) ([17] Haller), ([18] Newman).

'Surfing the internet' we can also check that some well known commercial firewalls support OTP such as IBM, Checkpoint, Cisco, etc, as well as some handheld devices.

17 Proposed Model: case of study

With some knowledge of the OTP scheme we can return to the initial case of a big Customer Support Centre. The model which take into account two different aspects, on one hand it will have to increase efficiency, on the other it should enhance our former authentication scheme. The objetives are: reduce response times when providing and reseting passwords and to reduce costs or initiate a business expansion by accepting more customers.

The model needs a central authentication process that manages all privileges for all systems. The model needs: a server program interface between OTP server and client, a user program interface and a secure database that stores 'credentials' for each account and for each system; that is, a global mecanism of password scrow and authentication.

The following figure shows the three elements of this model. One in a public zone, one at the interface and another in a secure zone. The information is 'refracted' from one media to another.

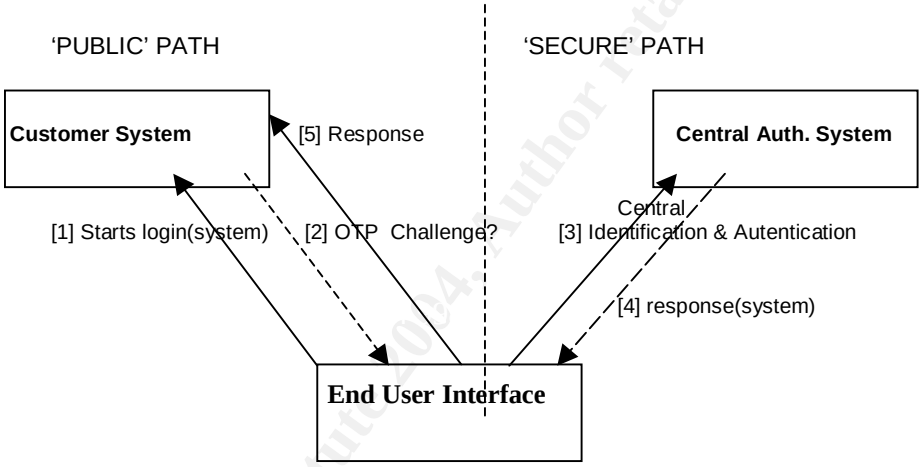

Fig 9

[1] The end user program starts the identification and authentication sequence by sending userid credentials to the target system
[2] The target system sends back a challenge to the user interface.
[3] The user program entity and the Central Authentication system have their own and unique authentiction mecanism through a secure path.
[4] The Central System has enough information and logic to compute the next OTP response for this userid for that system
[5] The program entity answers the Customer System with the appropiate response through the 'public' network

From the end user point of view the OTP authentication process will be transparent. He will only have to provide his unique userid, password and system name to an application, and he will be authenticated in a customer system. The process will be the same if he wants enter to another system.

The following diagram shows all these hidden events. Here 'user' is the end user and program logic accesing a customer system. 'Application Interface' here is the program logic and program interface. 'Opie client', 'Opie Server' and 'Application Interface' are the only elements of the Central Authentication

System. All the events whose labels start with the 'opie' are functions defined in the default OPIE C library. ([19] Daniel, Randall and Craig.)

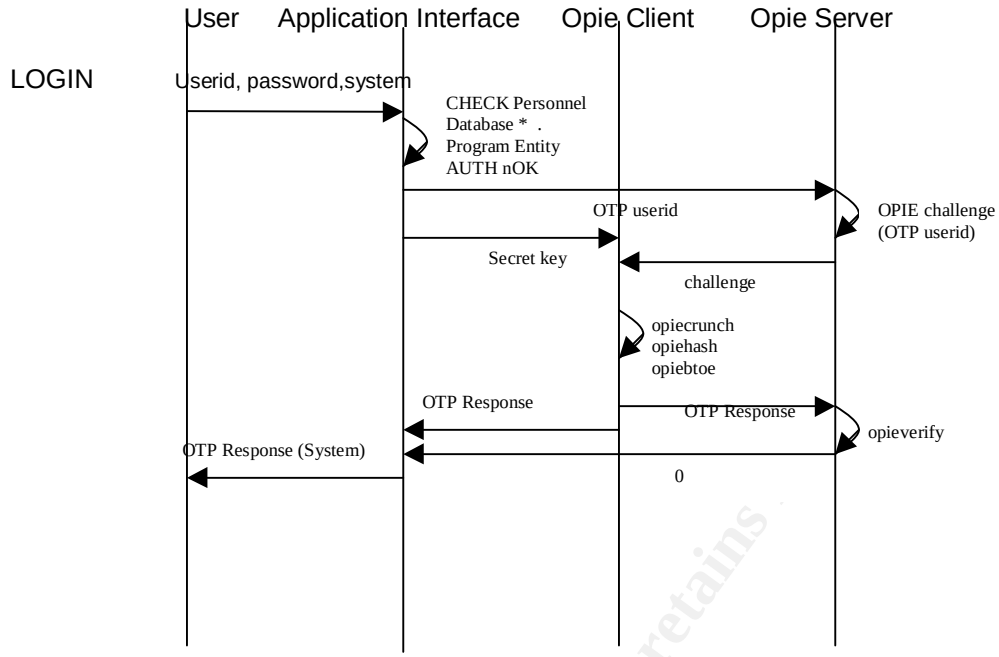

Fig. 10

This framework reduces the number of passwords to be managed by the Customer Support Centre. In fact there will be only one password per user accesing to a lot of systems. This not a single sign-on access control mecanism but it take some advantages of it. This model saves time because a lot of work is left to an application that makes password management transparent. It also enhances the authentication mechanism because it uses non reusable passwords, moreover, it does not let the user know its values.

In Figure 10 there is a call to a database refferenced as personnel database. The following figure shows the information stored there for a single user. There are as many entries as systems the user is authorized to access. In this example the user is authenticated using its intranet userid and password (Intranet_Account and Intranet_password fields). Secret Key and OTP userid are required to centrally reproduce the same scenario that will take place in the target system. Here the database and the application server are elements of the central arquitecture and are placed in a secure environment.

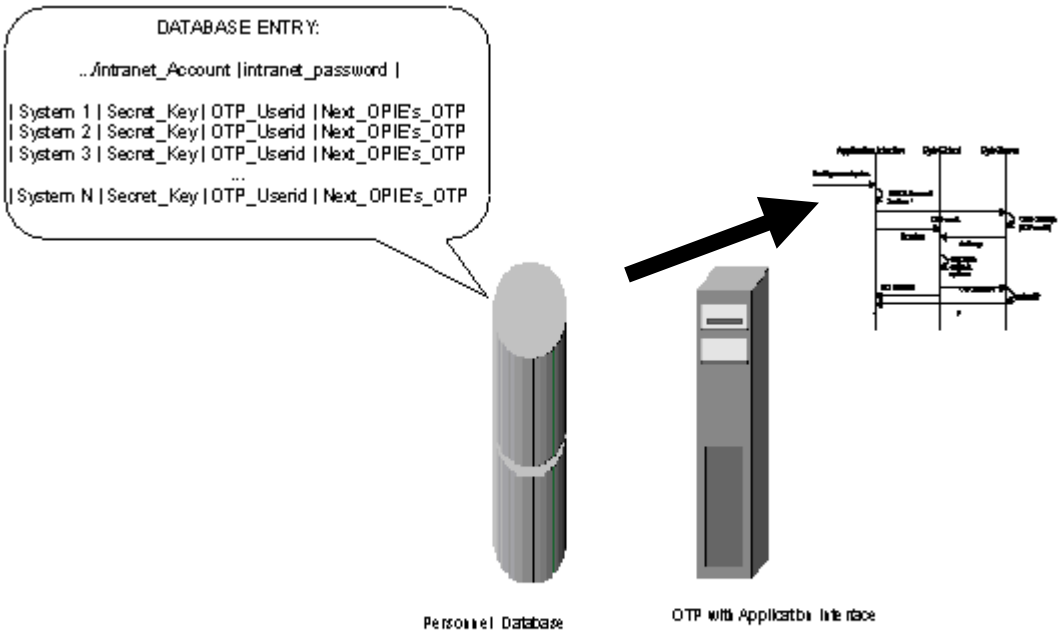

Fig. 11

The following figure shows an architecture that uses this model. Let's take , for example, a worker at home. He connects his laptop, through PSTN, to his corporate LAN in order to access to the Centralized Authentication System. Once he is correctly authenticated he starts a connection to a customer system through the internet. At this moment a second authentication process takes place as described before. At the end he is authenticated by the target customer system. This process can be repeated for all the systems with an entry in the centralized database (red line in Figure 12).

The proposed model is not, for sure, a complete solution but it could enhance the authentication process in some circumstances in our layered security design.

18 Proposed Model: architecture

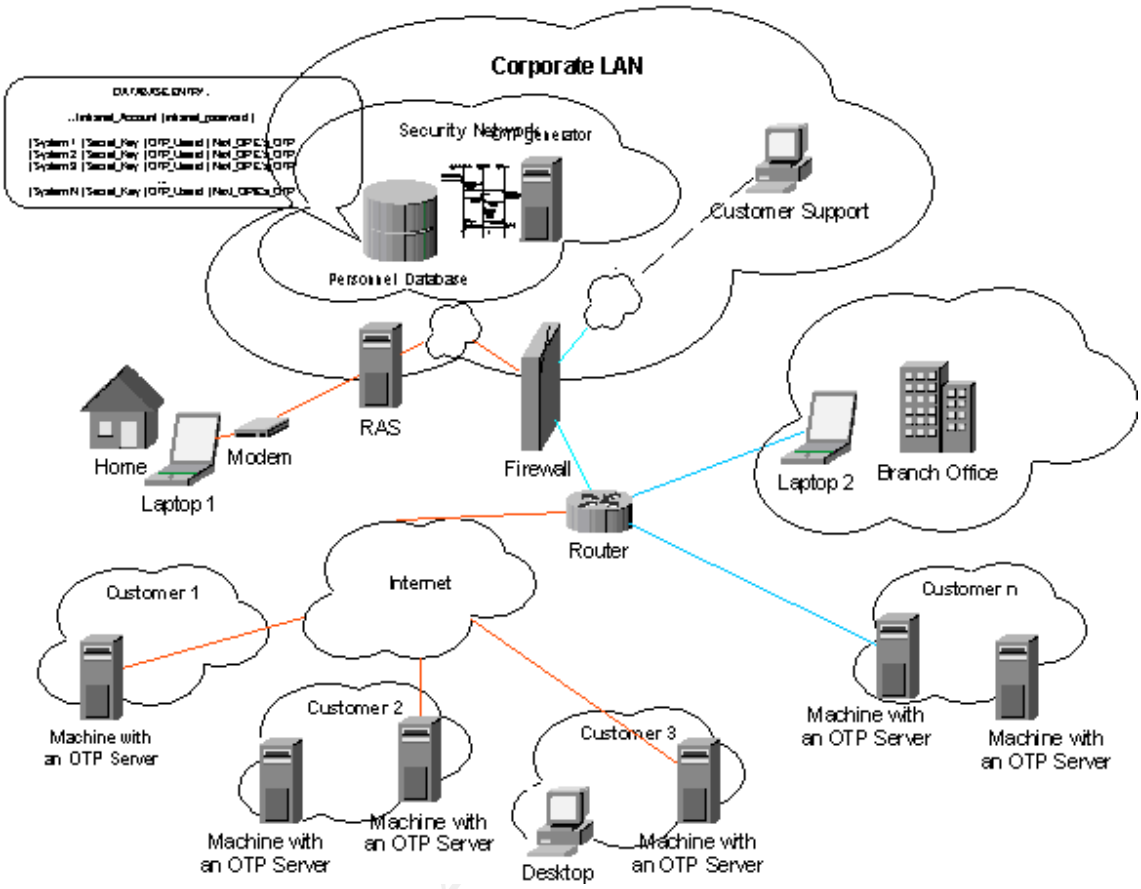

Fig. 12

18 List of References

[1] Ribagorda, Arturo. Seguridad de la Información: Criptografía. Reading: Curso de. Universidad Carlos III de Madrid. Escuela Politécnica Superior. Chapter 10 "Autenticación de Entidades". p10-0, p10-5. 10 April 2000.

[2] Lamport, Leslie. Password authentication with insecure communication. Communications of the ACM, nov 1971. "the basis for the Bellcore S/KEY system" -Lamport, 1981

[3] Haller, N., Atkinson, R. "On Internet Authentication".FAQs. RFC 1704. October 1994
URL: http://www.faqs.org/rfcs/rfc1704.html

[4] William R. Cheswick, Steven M. Bellowin Aviel D. Rubin. Firewalls and Internet Security, Second Edition "Repeling the Wily Hacker". Reading: Addison Wesley, 2003. p.104, 146-147.

[5] Menezes, J. Alfred. Handbook of Applied Cryptography. Reading: CRC Press. 5th printing. August 2001. Chapters 9 and 10.

[6] Haller, N. "The S/KEY One-Time Password System". FAQs. RFC 1760. February 1995.
URL: http://www.faqs.org/rfcs/rfc1760.html

[7] Rivest, R. "The MD4 Message-Digest Algorithm".FAQs. RFC 1320. April 1992.
URL: http://www.faqs.org/rfcs/rfc1320.html

[8] Rivest, R. "The MD5 Message-Digest Algorithm" FAQs. RFC 1321. April 1992.
URL: http://www.faqs.org/rfcs/rfc1321.html

[9] SGI. "Opie 2.4: description +notes". SGI Freeware. July 2003
URL: http://freware.sgi.com/Installable/opie-2.4.html

[10] Berkheimer, Andy. "OPIE module for Linux-PAM". The Linux Kernel Archives. 10 nov 1999.
URL: http://tho.org/~andy/pam-opie.html

[11] IETF Working group "One Time Password Authentication (OTP)". Internet Engineering Task Force (IETF). 29 June 1999.
URL: http://www.ietf.org/proceedings/99nov/46th-99nov-ietf-118.html

[12] Metz, Craig. "One Time Passwords in Everything". The Inner Net.
URL: http://inner.net/opie

[13] Craig, Hunt. TCP/IP Network Administration. Reading: O'REILLY. April 2002
Chapter 12. Network Security.

[14] Neo, "Esempio di implementazione di OPIE". Open Skills. 28 february 1997. URL:
http://www.openskills.info/view/boxdetail.php?IDbox=617&PHPSESSID=265a9d115a939306d2cf2d7d68afc413

[15] Gaetz, Owen, Walsh. Basen in "Improving Remote Security with S/Key". Mc Cormik School of Engineering and Applied Science. Industrial Engineering and management Sciences.
URL: http://www.iems.northwestern.edu/labs/opie.html
URL: ftp://ftp.iems.nwu.edu/pub/skey/winkey32.exe

[16] Mantakos, Harry. "S/Key Calculator". Rice University.Ownlet Computing.
URL: http://www.owlnet.rice.edu/skey/

[17] Haller, Metz. "A One-Time Password System" FAQs. RFC 2289. February 1998.
URL: http://www.ietf.org/rfc/rfc2289.txt

[18] Newman, C. "The One-Time-Password SASL Mechanism". Internet Engineering Task Force (IETF). October 1998. URL: http://www.ietf.org/rfc/rfc2444.txt

[19] Daniel, Randall, Craig. "One Time Passwords In Everything (OPIE): Experiences with Building and Using Stronger Authentication". Chacs of the Naval Research Laboratory. June 1995. URL: http://chacs.nrl.navy.mil/publications/CHACS/1995/1995mcdonald-USENIX.pdf