Chained Enforceable Re-authentication Barrier Ensures Really Unbreakable Security

Toshiharu Harada Takaaki Matsumoto
NTT DATA CORPORATION

Abstract: Mandatory Access Control (MAC) is a powerful guard against non-authorized access, but is vulnerable to hackers logged in through proper procedure. This paper describes how to guard such case using MAC.

Keywords: Mandatory Access Control, MAC, Login Authentication, Anti-Spoofing

1. Introduction

The DAC (Discretionary Access Control), which Microsoft Windows, Linux and many other operating systems have built-in, holds vulnerabilities and there are risks caused by DAC's vulnerabilities. To solve DAC's problem, MAC (Mandatory Access Control) was invented [1]. At first, the MAC was implemented for the systems that have special requirements including military use. But LSM [2] (a framework to provide MAC to Linux) and SELinux [3] (one of implementations that provides MAC using LSM) were introduced into Linux (the open sourced operating system) on November 2004, and the MAC became closer to us at a stretch. While the environment to build secure Linux system has been greatly improved, the purpose and meaning of security enhancement by introducing the security enhanced OS are not very accurately known. According to a paper [4] by SELinux development project, the following two are the merits of introducing SELinux.

- Handling of the threats posed by "Tampering" or "Avoidance of security mechanisms at application level".
- Minimizing the damage caused by malicious or vulnerable applications.

These are the exactly merits gained by security enhancement at OS level. These are never excessive requirements and all computer systems should provide essentially. But you need to be careful that the introduction of MAC itself doesn't promise protection against all kinds of damage. On the systems that have MAC support, if MAC's policies are defined appropriately, the system won't get damaged indefinitely by invoking shell with administrator's privilege even if some process is hijacked due to vulnerability such as buffer overflow. But since it is possible to log into the system through proper procedure (for example, login authentication using valid username and correct password), there is a threat that a cracker logs into the system in case the password's secrecy is broken. It is possible to define strict policies for routine tasks and functions. But it is difficult to define strict policies for administration task that is done by administrator logged into the system. In SELinux, it is possible to define policies that deactivate root privilege [5], but there remains an interface to modify and reflect policies, and the interface is protected by conventional password authentication after all. This paper describes how to prevent crackers from logging in through proper procedure using MAC, using TOMOYO Linux (one of MAC implementations which the authors of this paper (hereafter, we) have originally developed).

2. Vulnerabilities of Login Authentication

The general method of login authentication used in many computer systems is password authentication that uses password supplied by user. Typically, login authentication can be performed only once. Therefore, login authentication always has threats such as password cracking using dictionary attack or avoidance of login authentication by attacking authentication program's vulnerability (for example, buffer overflow).

Conventional login authentication has the following problems.

- Login authentication can be performed only once.
- Passwords are used in many systems.
- Have to worry password's secrecy because you can't know the moment your password being cracked.
- Have to worry vulnerability of authentication programs.

These problems are described below.

2.1. Login authentication can be performed only once.

Normally, the login authentication is performed only once before a user logs into the system, regardless of the user is system administrator or not. Some security aware applications (for example, database software) enforce application specific authentications, but users can access to almost all resources if they passed the login authentication. There are some attempts to notify the possibility of illegal logins (for example, displaying last login time) after the user passed the login authentication. But even if the user can notice illegal logins, it's useless because the system is already damaged and the user can't respond. Moreover, the user even can't identify the damaged range of the system after the fact.

2.2. Passwords are used in many systems.

The only basis of password authentication is the correctness of the ordering of password string. Therefore, it is problematic that users have to keep their password's secrecy. Possible risks are, cracked by dictionary attack, stolen by eavesdropping or social engineering. It is possible to reduce these risks by introducing special systems (for example, one-time passwords, biometrics), but they are costly because administrators have to introduce special devices or special software.

2.3. Have to worry password's secrecy.

It is possible to detect that user's passwords are attacked (for example, using dictionary attack) by monitoring authentication failure log. But you can't

answer to the following questions.

"How many days does the cracker need to find my correct password?" (In other words, "When is the last day I can use my password safely?")

"Changing my password ALWAYS makes things safer, for I found an attempt to crack my password?" (In other words, "Changing my password ALWAYS makes the cracker need more days to crack my password?")

There are discussions about "Password authentication and pass phrase authentication, which one is stronger?"[6], but neither password authentication nor pass phrase authentication can answer to these questions after all. Operations that forces users to change their passwords so frequently makes their passwords easier and, as a result, will lead to insecure system.

2.4. Have to worry vulnerability of authentication programs.

Even if you introduced recently spreading special devices (such as fingerprint authentication, iris verification), the authentication might be avoided if there is vulnerability (such as buffer overflow) in the program that handles these devices.

3. Security Enhanced OS

3.1. The concept of security enhancement at OS level.

The MAC is capable to forbid execution of unnecessary functions by controlling OS's behavior, although OSes are originally made available for generic purpose. The MAC's access control is applied to all processes and all users without exception, and can precisely restrict resources such as files and directories that processes and users can access. In normal Linux, DAC's access control is not applied to the system administrator (i.e. root). In general, an OS that supports MAC is called "Security Enhanced OS"[3].

The reason why security enhanced OS is helpful is described below with a simple example. Processes that provide services over network (such as ftp server, samba) are always configured to accept request from network. The crackers can hijack these processes and invoke shell with administrator's privilege if vulnerability exists in these programs. OSes that don't support MAC cannot prevent the invocation of shells or invocation of malicious commands from the invoked shells. But if MAC is supported and appropriate policies are defined by administrators, the OS can prevent the invocation of shells that are essentially unnecessary for these processes if the processes are hijacked.

3.2. Reinforcement of login authentication using security enhanced OSes

In general, the security enhanced OSes are introduced to reduce the damage of hijacking and to ensure the data integrity. But the login authentication can produce unexpected pitfalls, as described above. However, it is possible to solve this problem using MAC that security enhanced OSes support. The basic idea is "Multiplex the Login Authentications". The login authentication multiplexing itself is possible to OSes that don't support MAC, but has significant points on OSes that support MAC, for OSes that support MAC can enforce the multiplexed login authentications.

4. Login Authentication Multiplexing

4.1. Image of multiplexing

The Fig. 1 shows the conventional login authentication, and the Fig. 2 shows the multiplexed login authentications. The purpose of login authentication is to prevent crackers from reaching to the castle.

Fig. 1 creates a hole and places a guard. The guard means a program that performs authentication. There is only one wall. Without MAC, the wall could be broken and the cracker can reach to the castle without passing the guard (i.e. the cracker can log into the system without passing login authentication).

By introducing MAC, the wall becomes unbreakable (i.e. the cracker can't log into the system without passing login authentication). But since there is only one guard, the cracker can reach to the castle if the cracker could pass the login authentication through proper procedure (using valid username and correct password).

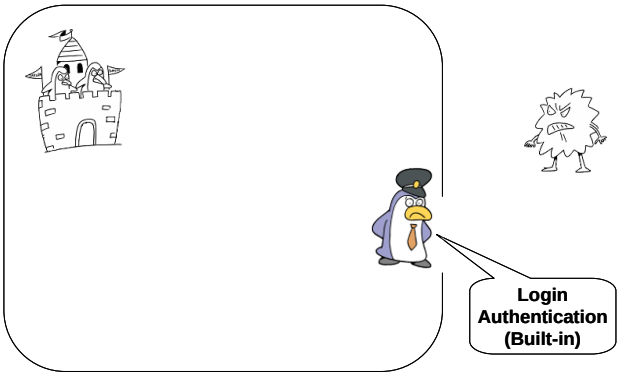

Fig. 1 Conventional Login Authentication

Fig.2 inserts two walls between the original wall and the castle, each wall has one hole and one guard. The cracker has to pass all guards to reach to the castle.

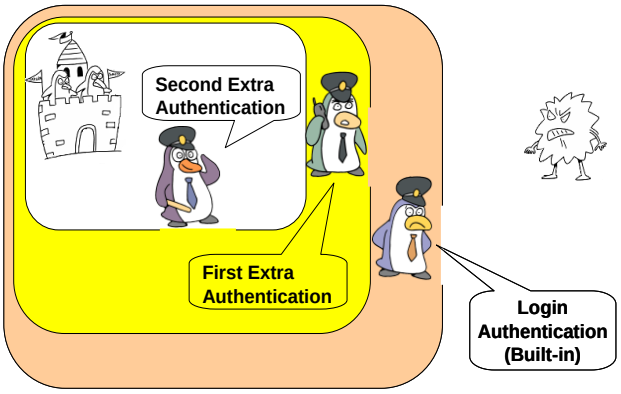

Fig. 2 Multiplexed Login Authentications

4.2. Example programs for extra authentication

You need to place newly developed authentication

programs for First and Second Extra Authentications shown in Fig. 2. Some examples using shell scripts are shown below. But you should develop your programs using non-scripting (for example C) language for production environment, for the content of shell script program is exposed if the environment variable "SHELLOPTS" is set with "verbose" flag.

(1) Simple password authentication

This program (Fig. 3) requires "SAKURA" as password. The authentication fails if the user entered wrong password for 3 times.

```
#! /bin/sh
for i in 1 2 3
do
  read -r -s -p 'Password: ' passwd
  echo
  [ "$passwd" = "SAKURA" ] && exec $SHELL
done
echo 'Incorrect password.'
```

Fig. 3 Simple password authentication

(2) Non-password authentication

This program (Fig. 4) authenticates the user by the existence of the file /data/rootauth . This program prompts users to enter password, but that is a dummy. The authentication always fails whatever passwords the cracker guesses unless the file exists. The file needs to be created (using touch commands, for example) prior to the execution of this program. (Since a terminal is supplied to the user after the conventional login authentication, the user can execute necessary command if granted by the policy.)

```
#! /bin/sh
for i in 1 2 3
do
  read -r -s -p 'Password: ' passwd
  echo
  [ -f /data/rootauth ] && exec $SHELL
done
echo 'Incorrect password.'
```

Fig. 4 Non-password authentication

(3) Never succeeding authentication

This program (Fig. 5) prompts users to enter passwords, but never succeed. This program is not for legal users, but for crackers who don't know how to pass this authentication. This program will confuse crackers.

```
#! /bin/sh
while :
do
  read -r -s -p 'Password: ' passwd
  echo
done
```

Fig. 5 Never succeeding authentication

Typically, the system looses protections against crackers when the cracker successfully passed conventional login authentication. But you can counter this threat by introducing extra authentications with various authentication rules and enforce them using MAC.

5. Advantages of Login Authentication Multiplexing

The following merits are derived by login authentication multiplexing.

5.1. You can enforce login authentication for arbitrary times.

You can enforce login authentication for arbitrary times depending on the resource's importance. For example, you can allow access to trivial resources after passing only conventional login authentication and allow access to critical resources after passing three extra login authentications.

5.2. You needn't to worry about vulnerability of authentication programs.

The vulnerability of authentication program is critical if the authentication can be performed only once. But since you can enforce multiple different authentications, it won't matter so much if one of the authentication programs has vulnerability.

5.3. You can use everything for authentication

Regarding conventional login authentication, the system can't know the process of supplying passwords and the authentication program authorizes the user using the supplied passwords. But after the conventional login authentication, a terminal (or a console) environment is provided to the user. This means that the authentication programs can know the user's behavior in great detail. You can use not only password strings but also all elements for authentication, for the authentication programs can know (for example) the speed of key typing or the user's behavior after the conventional login authentication and can use these elements for authentication.

Another example, you can use the existence of specific files (Fig. 4) or the contents of specific file as a password. You can use flags that always fail the authentication request like /etc/nologin , last modified time of specific file to test "Whether this authentication is started within 1 minute from the previous authentication".

The programs that perform authentication even needn't to be recognized at a glance that the programs are used for authentication. For example, a screen like card games appear when the program is executed and actually users can play with, but the authentication succeeds only when the specific key is pressed at the specific timing (like a kind of trapdoor programs). The requirement is that authentication programs are programs that only the legal users know the procedure how to pass that authentication.

You can create authentication programs in the same manner of developing normal application programs. Your idea makes strong authentication and the possible combinations of elements are infinite.

5.4. No damage unless all authentications are penetrated.

You can define policies that forbid access to critical resources unless the user passes all login authentications.

Specifically define policies that allow users who passed one login authentication do minimum operations that are needed to pass the next login authentication. You may append policies that allow users to execute dummy authentication program (like Fig. 5) to make penetration more difficult.

5.5. You can advise to legal users.

You can know which authentication program was penetrated, and you can replace only the program that was penetrated.

You can notify to users by sending mail like "The login authentication of host XXXXX was penetrated, but the cracker was eliminated by extra authentication mechanism. To prevent another penetration, I changed your password to XXXXXXX."

6. Practical Issues and Solutions

6.1. Login shell

Login shell is a program that is executed when a user logs into the system, and is specified in the /etc/passwd file. In Linux, bash, ksh, tcsh, zsh etc. are available.

Shell is provided to execute external programs, but most shells have their internal (built-in) commands.

An example of shell's internal commands is "kill", which sends signals to processes. A cracker who passed the login authentication can forcefully terminate arbitrary process if appropriate privilege is given.

Of course, it is possible to restrict signal transmission using MAC's policy. But that is not enough.

A cracker can give high load using infinite loop using shell's internal command. For example, if the cracker gives internal command "while : ; do echo ; done" to bash, the system's response become slower. It is impossible to prevent this CPU consumption attack by infinite loop using MAC's policy.

Therefore, to apply this login authentication multiplexing method, it is important that login shells don't have unnecessary internal commands. The role of login shells is to provide interface to execute the next extra authentication. Less functional shells are better and suitable. Of course, you can use normal shells to start actual operations after passing all login authentications.

6.2. "scp" and "sftp"

There are two commands that are frequently used for server maintenance purpose, "scp" and "sftp". But it is impossible to apply this login authentication multiplexing method for these programs. The reason and solutions are described below.

A shell has two operation modes, one is "interactive mode" that prompts and waits for user's input, the other is "batch mode" that are invoked with "-c command list" command line parameter and process the given command list and then terminates. The method this paper describes invokes login shells in "interactive mode" and restricts user's behavior so that only operations that are necessary to pass the next authentication are allowed using MAC's policy; to prevent subversive acts unless the cracker succeeds all login authentications. Therefore, programs that invokes login shell in "batch mode" ("scp" connects to remote host using "ssh" and invokes remote host's login shell with "-c scp arguments" options. "sftp" connects to remote host using "ssh" and invokes remote host's login shell with "-c /usr/libexec/openssh/sftp-server" options.) can't recognize the extra login authentications; i.e. you can't use login authentication multiplexing for "scp" and "sftp". This means that resources that are accessible become vulnerable if the cracker passes ssh's login authentication.

The solution is that restrict resources that are accessible to such programs. Specifically, define policy that limits reading/writing to specific temporal directory, and move data between the specific temporal directory and the other directories from shells that are invoked after all extra authentication are succeeded.

7. Implementation using TOMOYO Linux

7.1. About TOMOYO Linux

TOMOYO Linux is one of MAC implementations that we have developed based on vanilla Linux kernels, and has "accept mode" that helps administrators defining MAC policies. Please refer to document [7] for abstract, and document [8] for implementation.

TOMOYO Linux defines DOMAIN (the unitary of granting ACLs) based on the process's invocation history, and lists ACLs that are allowed to each DOMAIN. The ACL consists of the access mode (read/write/execute) and the pathnames. For example, define the following line to allow /bin/bash which are invoked by /usr/sbin/sshd (i.e. a user logged into the system using ssh) to read /etc/passwd and execute /usr/bin/scp .

```
<kernel> /usr/sbin/sshd /bin/bash

4 /etc/passwd
1 /usr/bin/scp
```

The integer before pathnames corresponds to UNIX's permission. For example, "4" is "r--", "1" is "--x", "6" is "rw-", and "7" is "rwx". The name of DOMAIN starts with <kernel> , and the program's pathname is concatenated to the name of DOMAIN where the program is invoked. For example, the name of DOMAIN for /bin/tcsh that is invoked by /bin/bash that is invoked by /usr/sbin/sshd (i.e. a user logged into the system using ssh and invoked /bin/tcsh from the login shell) is represented as follows.

```
<kernel> /usr/sbin/sshd /bin/bash /bin/tcsh
```

The granularity of TOMOYO Linux's access mode is not high as SELinux. But since you can define policies using pathnames, it is easy to understand for administrators who have standard administration skill. And since DOMAIN is divided by invocation of a program and the ACLs are given for 1-file-at-a-time, you can specify more

precisely than SELinux.

7.2. Actual example policy

This section describes an actual example policy of login authentication multiplexing shown in Fig. 2. To help understanding, miscellaneous files like library files are omitted. The scenario for this policy is the following.

- Login using ssh and invoke (our custom made shell) /bin/falsh as the login shell. /bin/falsh has no built-in commands like "kill" or "while" to prevent attacks (for example, killing processes, infinite loop) using login shells.
- Invoke (our custom made authentication program) /bin/honey (which corresponds to First Extra Authentication in Fig. 2). /bin/honey prompts for password input, but this program checks not only the password string but also the time interval each letters are typed. The authentication fails if either password string or the time intervals (preset in this program) don't match.
- Invoke (our custom made authentication program) /bin/candy (which corresponds to Second Extra Authentication in Fig. 2). /bin/candy prompts for password input, but this program checks not only the password string but also the elapsed time from the invocation of the parent process. The authentication fails if either password string doesn't match or the elapsed time is longer than 10 seconds. (It is difficult to start /bin/candy after the invocation of /bin/honey within 10 seconds, for /bin/honey needs a several seconds. Therefore, /bin/falsh is inserted between /bin/honey and /bin/candy to reset the invocation time of the parent process.)
- Since "scp" and "sftp" need to be executed from login shell, the policy allows executing these programs from login shell, but these programs can access to only /data/scp.tmp directory.

```
<kernel> /usr/sbin/sshd /bin/falsh

1 /bin/honey
1 /usr/bin/scp
1 /usr/libexec/openssh/sftp-server

<kernel> /usr/sbin/sshd /bin/falsh /usr/bin/scp

6 /data/scp.tmp/¥*

<kernel> /usr/sbin/sshd /bin/falsh
/usr/libexec/openssh/sftp-server

6 /data/scp.tmp/¥*

<kernel> /usr/sbin/sshd /bin/falsh /bin/honey
1 /bin/falsh

<kernel> /usr/sbin/sshd /bin/falsh /bin/honey /bin/falsh
1 /bin/falsh

<kernel> /usr/sbin/sshd /bin/falsh /bin/honey /bin/falsh
/bin/falsh
1 /bin/candy

<kernel> /usr/sbin/sshd /bin/falsh /bin/honey /bin/falsh
/bin/falsh /bin/candy
1 /bin/falsh

<kernel> /usr/sbin/sshd /bin/falsh /bin/honey /bin/falsh
/bin/falsh /bin/candy /bin/falsh
1 /bin/bash

<kernel> /usr/sbin/sshd /bin/falsh /bin/honey /bin/falsh
/bin/falsh /bin/candy /bin/falsh /bin/bash
```

In addition to this, register the DOMAIN "<kernel> /usr/sbin/sshd /bin/falsh /bin/honey /bin/falsh /bin/falsh /bin/candy /bin/falsh /bin/bash" as trusted, and move data between /data/scp.tmp and other directories from this trusted DOMAIN.

7.3. Actual operation

This section describes the procedure for users. Fig. 6 is a screenshot that a user is connecting to a Linux server using ssh. In the screenshot, the user enters password strings to log in, as conventional.

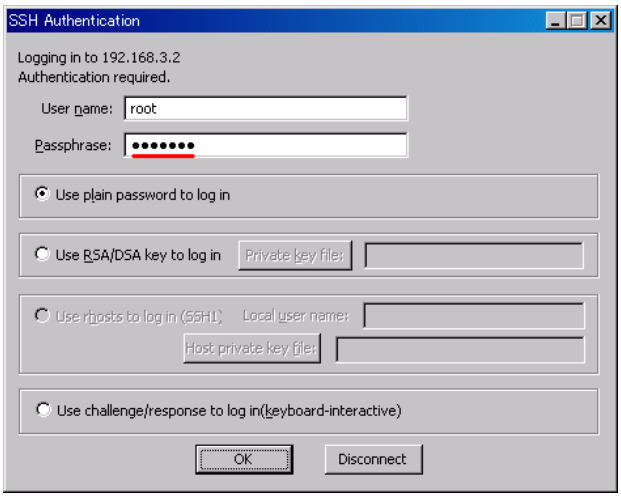

Fig. 6 Conventional login authentication

After the user passed ssh's login authentication, invoke "/bin/honey /bin/falsh /bin/candy" in this order (as defined in the policy) and precede the authentication. Fig. 7 contains authentication failures intently to show that the extra login authentications aren't simple password authentications. Also, the passwords supplied are visible, for this is a demonstration.

In the first attempt of /bin/honey , the user entered the correct password, but the authentication failed since the typing interval was inappropriate. In the second attempt of /bin/honey , the user entered the correct password with appropriate typing interval, and the authentication succeeded.

In the first attempt of /bin/candy , the authentication failed due to incorrect password. In the second attempt of /bin/candy , the user entered the correct password, but the

authentication failed since /bin/candy has to be invoked within 10 seconds after the shell (/bin/falsh) starts. In the third attempt of /bin/candy , the user entered the correct password, and the authentication succeeded since /bin/candy is invoked within 10 seconds after /bin/falsh starts.

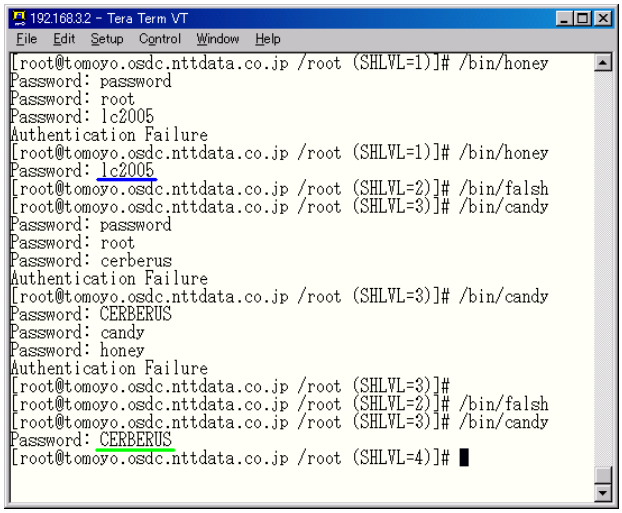

Fig. 7 Extra login authentications

After the user passed /bin/candy (i.e. the user has reached to the castle in Fig. 2), invoke /bin/bash and start normal operations.

8. Discussion

8.1. Comparison with PAM

It is possible to perform multiple authentication methods using PAM (Pluggable Authentication Modules) to reduce the risks of illegal login. But if PAM itself has vulnerability, the login shell could be started before performing all authentication modules specified as "requisite".

Also, there are typically only two input fields (username and password) like Fig. 6, it is impossible to use multiple passwords using PAM provided by the system. Therefore, people combine with other methods that use information other than password; for example, hours checking (pam_time.so) and the name of terminal device (pam_securetty.so).

If you WANT to use multiple passwords, you have to stuff all passwords into one input field, splitting by column number like Fig. 8. But the way of splitting password field (the way of interpretation) changes whenever new elements are stuffed into password field. This means you need to negotiate with all modules that share password field.

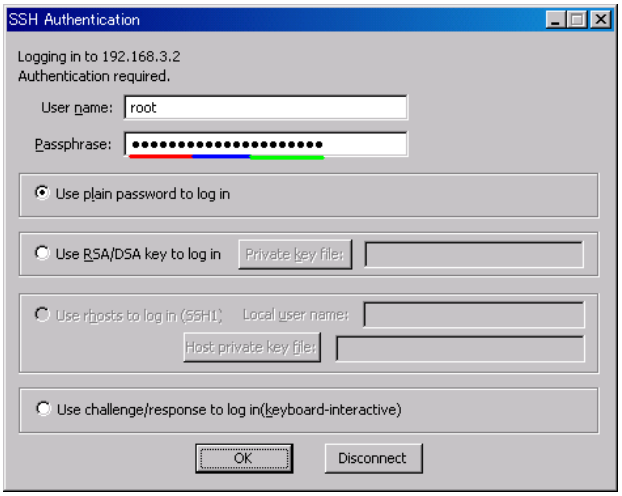

Fig. 8 Stuffing all passwords into one field

On the contrary, the way of multiplexing needn't to change the password field when the authentication method changes, for passwords are supplied on each step like Fig. 6 and Fig. 7. This means you needn't to change existent protocols and PAM configurations.

8.2. Elements available for authentication

Regarding conventional login authentication, the system can't know the process of supplying passwords and the authentication program authorizes the user using the supplied passwords. But by introducing login authentication multiplexing, the authentication programs can know (for example) the speed of key typing or the user's behavior after the conventional login authentication. This allows you to choose your favorite elements from infinite number of elements to create customized login authentication.

8.3. Burden increment on users

It is acceptable to provide multiple information for authentication on the systems that worth protecting from penetration by providing extra information other than password for authentication.

Our method is just supplying one information on each authentication instead of supplying all information at once. In the view of users, only the timing of supplying information is changed. There is no limitation for extra authentication program, so you can choose one that the users feel minimum burden.

8.4. Price for paying for login authentication reinforcement

Our method doesn't cause overall damage if there is vulnerability in one of the authentication programs. You can improve security for login authentication dramatically with just tens of lines code in C language.

8.5. Security Stadium 2004

We attended at Security Stadium 2004 held by JNSA on the defense side. We announced root's password so that the offence side can login via ssh (without cracking sshd). We received attacks by security experts, and turned out

that our method is very effective. Please refer to document [9] for details.

8.6. Applying to OSes that doesn't support MAC

It is possible to perform multiplexed login authentications for OSes that don't support MAC. But since the behavior of authentication programs can't be restricted from outside using MAC's policy, each authentication program has to restrict its behavior, and developers have to be very careful not to create security loopholes. If MAC is supported, the behavior of authentication programs are restricted from outside using MAC's policy, and developers can easily develop authentication programs without worrying security loopholes. Therefore, our method has significant points on OSes that support MAC.

9. Conclusion

The security enhanced OSes are invented to protect from unauthorized access and leakage of information, and are getting to spread. It is possible to reduce the risk of hijacking due to vulnerability such as buffer overflow and improve system security by defining appropriate policy. But how well access to system resources is controlled, the dependence on the password login authentication can produce unexpected pitfalls. The method of login authentication multiplexing described in this paper is easy to implement and doesn't require one-time passwords or costly biometrics technology.

Acknowledgment: We were supported from the technological study to implementation and evaluation on TOMOYO Linux by Tetsuo Handa, NTT DATA CUSTOMER SERVICE CORPORATION. We would like to thank Mr. Handa.

Bibliography

[1] "A research on information systems for e-Government based on OSes with well-considered security" (Written in Japanese)
http://www.bits.go.jp/inquiry/pdf/secure_os_2004.pdf
[2] Linux Security Modules
http://lsm.immunix.org/
[3] National Security Agency, Security-Enhanced Linux
http://www.nsa.gov/selinux/
[4] “Meeting Critical Security Objectives with Security-Enhanced Linux”
http://www.nsa.gov/selinux/papers/ottawa01-abs.cfm
[5] SELinux Play Machines
http://www.coker.com.au/selinux/play.html
[6] Pass Phrases vs. Passwords
http://www.microsoft.com/technet/community/columns/secmgmt/sm1004.mspx
[7] Toshiharu HARADA, Takashi HORIE and Kazuo TANAKA, "Towards a manageable Linux security." Linux Conference 2005
http://sourceforge.jp/projects/tomoyo/document/lc2005-en.pdf
[8] Toshiharu HARADA, Takashi HORIE and Kazuo TANAKA, "Task Oriented Management Obviates Your Onus on Linux." Linux Conference 2004
http://sourceforge.jp/projects/tomoyo/document/lc2004-en.pdf
[9] Security Stadium 2004 (Written in Japanese)
http://www.jnsa.org/active/press/vol12pdf/4_report4.pdf

Notes

This is a translation of the original paper, which was written in Japanese and published in Workshop on Informatics 2005 held in Japan. You can obtain the original paper from the following URL.

http://sourceforge.jp/projects/tomoyo/document/winf2005.pdf

TOMOYO Linux was released on November, 11, 2005. You can get more information at the following URLs.

http://tomoyo.sourceforge.jp/
http://sourceforge.jp/projects/tomoyo/