セキュリティ強化 OS によるログイン認証の強化手法

原田　季栄　　松本　隆明
（株）NTT データ

Chained Enforceable Re-authentication Barrier Ensures Really Unbreakable Security

Toshiharu Harada　　Takaaki Matsumoto
NTT DATA CORPORATION

1. はじめに

Microsoft Windows, Linux 等が標準で備える自由裁量のアクセス制御機構(DAC: Discretionary Access Control)が抱える脆弱性およびそこから生じるリスクを解消するものとして、強制アクセス制御機構(MAC: Mandatory Access Control)が考案された[1]。MAC は当初、軍事用途を含めた特殊な要件を持つシステムに適用、実装されたが、2004 年 11 月にオープンソースの Linux に MAC を実現するためのフレームワークである LSM [2]およびそれに対応した SELinux [3]が組み込まれたことにより一気に身近なものとなった。このようにセキュアなシステムを構築するための環境は大きく改善されたが、その一方で OS によるセキュリティ強化の目的と意味については、あまり正確に知られていない。SELinux の開発プロジェクトによる論文[4]においては、SELinux 導入の効果として次の 2 点が挙げられている。

- 「改ざんおよびアプリケーションレベルによるセキュリティ機構回避」といった脅威に対する対処
- 不正、あるいは欠陥を持つアプリケーションにより被る可能性のある被害の極小化

OS のセキュリティ強化により得られる利点は、まさに上記 2 点に集約される。それは決して過剰な要求条件ではなく、本来あらゆるコンピュータシステムが備えるべき条件である。しかし、MAC の導入はそれ自体があらゆる被害の防止を約束するものではない事について注意が必要である。MAC を採用したシステムでは、適切なポリシーさえ策定されていれば、バッファオーバーフロー等によりサーバプロセスを乗っ取られたとしても、そこからシステム管理者権限のシェルを起動されて無制限に被害を受けることはない。しかし、正規の手順（例えばユーザ名とパスワードによるログイン認証）を経ればシステムにログインできるため、パスワードが漏洩した場合システムにログインされるという脅威が存在している。定型的な業務や機能に対するポリシーの策定は可能でも、管理者がログインして行う操作に対してはポリシーによる厳密な制御は困難である。SELinux についても root を無力化したポリシーは策定できても[5]、そのポリシーを変更、反映するためのインタフェースは残っているわけで、その部分を守っているのは結局昔ながらのパスワードによる認証である。

本論文では、侵入者が正規の手順でログインするという脅威を、強制アクセス制御を応用して防御するためのアイデアについて、筆者らが独自に開発した TOMOYO Linux を例に紹介する。

2. ログイン認証における脆弱性について

コンピュータシステムにおいて、システムにログインするユーザを認証するために一般的に用いられている方法は、ユーザが入力するパスワードによるログイン認証である。ログイン認証は通常は一度しか行えない。そのため、辞書攻撃等でパスワードが割り出されたり、認証プログラムの脆弱性を攻撃されて認証を回避されたりするという脅威を常に抱えている。

従来のログイン認証には、以下のような課題がある。

- 認証は一度しか行われない。
- 多くの環境ではパスワードに基づいている。
- いつパスワードを破られるか判らず不安。
- 認証プログラムの脆弱性が不安。

以下にそれぞれの課題について説明する。

2.1. 1 回しか認証できない

ログイン認証は、システム管理者であるか否かによらず、通常システムの利用を開始する前に一度だけ行われる。特にセキュリティに配慮したアプリケーションの中には、アプリケーション独自の認証を強制するものがあるが、通常はログイン認証に成功した時点で、それ以降ほとんどの資源にアクセスできてしまう。ログイン認証が成功後、前回ログイン日時を表示するなど、不正なログインの可能性を知らせるための工夫の試みはあるが、それにより不正なログインがあったことに気が付いたとしても被害を受けた後では対処できない。また、事後となっては被害の範囲の特定も不可能である。

2.2. 多くの環境ではパスワードを使用している

パスワードによる認証ではパスワードとなる文字列の並びの正しさが唯一の判断の拠り所であり、常にその秘密を守ることが課題としてつきまとう。存在するリスクとしては、辞書攻撃によって突破されたり、盗聴やソーシャルエンジニアリングといった方法で盗み出されたりする場合が挙げられる。こうした問題点については、ワンタイムパスワード等のシステムの導入により改善が可能であるが、そのためには、専用のデバイス、認証プログラムを導入する必要があり、コストがかさむという欠点がある。

2.3. いつパスワードを破られるか判らず不安

辞書攻撃等パスワード破りのための攻撃を受けていることは、認証失敗等のログを監視することによりある程度は検知が可能である。しかし、「攻撃者が現在のパスワードを割り出すまでにあと何日を要するか」とか「パスワード破りを試みた痕跡が見つかったからパスワードを変更した方が必ず安全であるか」という疑問は解消できない。パスワードとパスフレーズのどちらが強固であるかという論争も繰り返し行われているが[6]、結局いずれの方法も、この疑問を解消できない。あまりに頻繁にユーザにパスワードの変更を強制するような運用では、パスワードの内容が簡単になり結果としてセキュリティが低下するということも考えられる。

2.4. 認証プログラムの脆弱性が不安

近年になって利用が浸透し始めた指紋認証、虹彩認証等のデバイスを導入したとしても、それらを制御するプログラムにバッファオーバーフロー等の脆弱性が存在した場合、認証を回避されてしまう恐れがある。

3. セキュリティ強化 OS について

3.1. OS セキュリティ強化の概念

強制アクセス制御は、もともと汎用の用途に供することを目的に設計された OS に対し、アクセス機能を制限することにより「必要でない」機能の実行を禁止することを可能とする。それは全てのプロセス・全てのユーザに対して例外無く適用され（標準の Linux においてはシステム管理者である root に対しては適用されない）、プロセスやユーザがアクセス可能なファイルやディレクトリ等の資源を詳細に制限できる。一般に、強制アクセス制御を導入した OS をセキュリティ強化 OS[3]と呼ぶ。

セキュリティ強化 OS が何故有効であるかについて簡単な例をあげて説明する。ftp, samba 等ネットワークに対してサービスを提供している場合、それらのサービスを提供するプログラムは常に外部からの通信を受け付けるよう設定されている。もし、そうしたプログラムにバッファオーバーフロー等の脆弱性が存在した場合、クラッカーはサーバプログラムを乗っ取り、管理者権限のシェルを起動させることが可能である。通常の OS はこのような攻撃を受けた場合、シェルの起動も、起動されたシェルからの悪意あるコマンドの実行も制限することはできない。しかし、もし攻撃されたサーバの OS がセキュリティを強化されたもので、管理者により正しくポリシーを設定されていれば、ftp や samba 等のプログラムを乗っ取ったとしてもそこから本来不必要なシェルの起動を拒否することができる。

3.2. セキュリティ強化 OS による認証の強化について

セキュリティ強化 OS は一般には OS 自体の乗っ取りによる被害の軽減、データの正しさの保証を目的として導入されるのが一般的であるが、前述のようにログイン認証が脆弱性となる危険性を孕んでいる。これについて、セキュリティ強化 OS 自体が備える機能を活用することにより解決することが可能である。基本的な考え方は、ログイン認証を多重化することである。認証の多重化自体はセキュリティを強化しない OS でも可能であるが、セキュリティを強化した OS においては、その多重化した認証の実行を強制することができるためより高い効果が得られる。

4. ログイン認証の多重化

4.1. 多重化のイメージ

従来のログイン認証のイメージを**図 1** に、多重化したログイン認証のイメージを**図 2** に示す。外部の侵入者の城への侵入を阻止することが目的となる。

図 1 では、壁に 1 箇所だけ穴を開け、門番を配置している。門番はユーザ認証を行うプログラムを象徴している。壁は 1 つしか存在せず、標準の Linux の場合、その壁を突破して侵入される（ログイン認証以外の経路でシステムにログインされる）危険がある。強制アクセス制御を導入すれば、壁を突破されることを防ぐ（ログイン認証以外の経路でシステムにログインされることを防ぐ）ことが可能になるが、門番が 1 箇所しか存在せず、正規の手順（つまり、ユーザ名とパスワード）によって門番の前を通過することができれば城まで到達できてしまう。

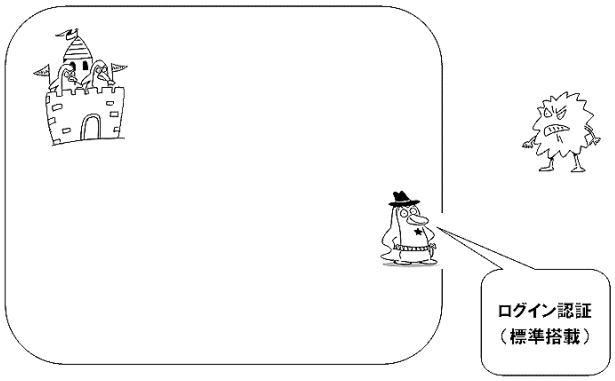

図 1　従来のログイン認証

図 2 では、**図 1** の壁と城の間に新たに 2 つの壁を配置して、それぞれの壁に 1 箇所だけ穴を開け、門番を配置している。城に到達するには全ての門番の前を通過しなければいけない。

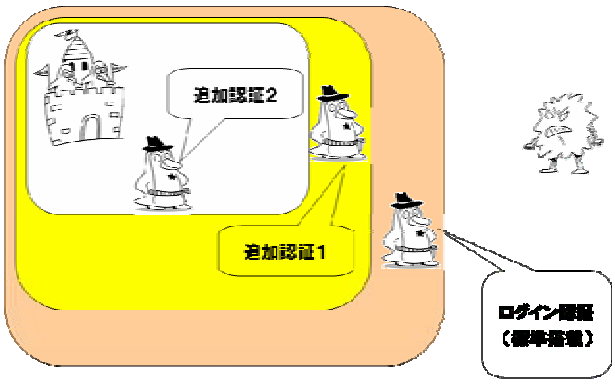

図 2　多重化されたログイン認証

4.2. 追加の認証を行うプログラムの例

図 2 における追加認証 1 および追加認証 2 には、新規に作成した認証プログラムを配置する。ここではシェルスクリプトで作成した例を紹介する。ただし、シェルスクリプトの場合、環境変数 SHELLOPTS に verbose が設定されているとシェルスクリプトの内容が表示されてしまうため、実際の運用においては C 言語等で作成すべきである。

(1) 単純なパスワード認証
図 3 は、”SAKURA”というパスワードを入力してもらう認証である。3 度間違えると認証は失敗する。

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

図 3　単純な認証プログラムの例

(2) パスワードによらない認証
図 4 は、/data/rootauth というファイルの存在により認証判断を行う。ユーザにパスワード入力を求めるがそれはダミーであり、いくらパスワード文字列を推理しても当該ファイルが存在しない限り認証は失敗する。当該ファイルはこの認証プログラムを実行する前に touch コマンド等で作成しておく。(ログイン認証を通過した時点でターミナルが提供されるため、ポリシーで許可してやれば touch コマンドを使用できる。)

```
#! /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.'
```

図 4　パスワードを使わない認証の例

(3) 決して成功しない認証
図 5 の認証は、パスワードを入力してもらうためのプロンプトを表示するが、決して成功しない。正しいユーザが実際に使用するためのものではなく、何も知らない侵入者に使用させることで侵入者を困惑させるためのものである。

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

図 5　決して成功しない認証の例

通常であれば、ssh の認証が破られた時点で侵入者に対する防御を失うが、強制アクセス制御により多段階の認証を追加し、その中で多様な認証基準を設けることで、侵入者に対する脅威に備えることができる。

5. ログイン認証多重化の利点

ログイン認証を多重化できるようになることで、以下のようなメリットが生じる。

5.1. 好きなだけログイン認証を強制させることが可能

保護したい資源の重要度に応じて、任意の回数だけログイン認証を強制させることができる。

5.2. 認証プログラムの脆弱性を気にする必要が無い

認証が 1 回しか行えない場合は、認証プログラムの脆弱性は致命傷となる。しかし、複数の異なる認証プログラムを用いて認証を強制させることができるので、認証プログラムの 1 つに脆弱性があったとしてもほとんど問題にならない。

5.3. パスワード以外の認証も可能

通常のログイン認証では、パスワードを入力する過程等をシステム側が知ることはできず、認証プログラムは「入力結果」に基づいてのみ、本人かどうかを判断する。しかし、通常のログイン認証後には、ターミナル（またはコンソール）環境が提供されるので、認証プログラムはユーザの挙動を詳細に知ることができる。キーボードをタイプする速度や、それまでに行った操作内容等の「入力過程」も判断基準に使えるので、パスワード以外の要素を認証に用いることが可能である。
他にも、**図 4** に示すように特定のファイルが存在するかどうかや、ファイルの内容をパスワードとして利用することもできる。あるいは、/etc/nologin のような「認証を常に失敗させるようなフラグ」や、ファイルのタイムスタンプを使って「直前の認証に成功してから 1 分以内であるか」といった条件を使うこともできる。
また、一目で認証を行うプログラムであることが判る必要さえも無い。例えば、実行するとカードゲームのような画面が表示され、実際に遊べるようになっているが、その中で特定のタイミングで特定のキーを押した場合だけ認証に成功するような仕組み（いわゆるトラップドア）でも構わない。とにかく、正規の利用者だけが認証に成功するための正しい手順を知っているプログラムであれば十分なのである。
このように、認証プログラムといっても、通常のアプリケーションと同じ感覚で作成でき、実現できる組み合わせは無限である。

5.4. 全ての認証を突破されない限り、被害を受けない

全てのログイン認証に成功するまでは重要な資源へのアクセスを許可しないようなアクセスポリシーを策定することで実現できる。
具体的には、あるログイン認証プログラムからは、次のログイン認証に必要な最低限の操作のみを行えるようにアクセスポリシーを策定する。**図 5** のようにダミーの認証プログラムも実行できるようなアクセスポリシーを策定することで、侵入をより困難にさせることが可能である。

5.5. 正しい利用者にアドバイスをすることができる

第何段階の認証まで突破されたかを知らせることで、突破された認証プログラムのみを変更するという対処を行える。
また、「ホスト XXXXX の最初のログイン認証が突破されましたが、追加認証機構により排除されました。再度の侵入を予防するために、パスワードを XXXXXXX に変更しました。」という内容のメールを送信したりすることもできる。

6. 実運用上の課題と対策

6.1. ログインシェル

ログインシェルとは、ユーザがシステムにログインした時に起動されるプログラムで、/etc/passwd ファイルで指定することができる。Linux においては bash, ksh, tcsh, zsh 等が利用できる。

シェルは外部コマンドを実行するためのものであるが、ほとんどのシェルは内部コマンド（ビルトインコマンド）を備えている。

内部コマンドの例として、プロセスに対してシグナルを送る kill が挙げられる。ログイン認証を突破した侵入者は、(適切な権限さえあれば）この内部コマンドを使用することで、任意のプロセスを強制終了させることが可能になってしまう。

もちろん、シグナルの送信をポリシーで制限することは可能であろう。しかし、それだけでは不十分である。

シェルの内部コマンドを使用すれば無限ループで負荷を与えることが簡単にできてしまう。例えば、bash に対して侵入者が while : ; do echo ; done というコマンドを与えるだけで、システムの応答を鈍らせることができる。この無限ループによる CPU 資源の浪費をポリシーで阻止することは不可能である。

よって、本論文の手法を適用する場合、ログインシェルが備えるべき要件として「不要な内部コマンドを持たないこと」が重要である。ログインシェルの役割は次の認証を行うプログラムを起動するためのインタフェースを提供するだけであり、機能が少なければ少ないほど本手法の利用に適している。なお、全ての認証に成功して実質的な操作を開始する際にはもちろん普通のシェルを使用して構わない。

6.2. scp と sftp

サーバのメンテナンスなどで頻繁に利用されるコマンドとして、scp と sftp が挙げられる。しかし、scp や sftp に対しては、本論文の手法を適用できない。その理由と対策を以下に述べる。

シェルの動作モードには、プロンプトを表示してユーザがコマンドを入力するのを待つ”対話的モード”と、起動時に"-c コマンド群"というパラメータで渡されたコマンド群を実行して終了する”バッチモード”が存在する。本論文で述べている手法は、ログインシェルを対話的モードで起動し、ログインシェルからは次の認証に必要な操作しか行えないようにポリシーで制限することで、全ての認証に成功するまで侵入者が破壊活動を行えないようにしている。そのため、scp や sftp のようにログインシェルをバッチモードで起動するようにハードコーディングされているプログラム（scp は ssh を使用してリモートホストに接続し、リモートホストのログインシェルに対して"-c scp 引数"というパラメータを渡すことで動作する。また、sftp は ssh を使用してリモートホストに接続し、リモートホストのログインシェルに対して"-c /usr/libexec/openssh/sftp-server"というパラメータを渡すことで動作する。) の場合、追加認証を強制させることができないため、ssh ログイン認証さえ突破されてしまえばそれらのプログラムがアクセス可能な範囲は無防備になってしまう。

対策は、これらのプログラムがアクセス可能な範囲を制限することである。具体的には、特定のディレクトリ以下のみ読み書き可能となるようなポリシーを記述する。そして、そのディレクトリと目的のディレクトリとの間の移動は、全ての認証に成功した後に起動されるシェルから行うようにする。

7. TOMOYO Linux での実現方法

7.1. TOMOYO Linux について

TOMOYO Linux とは、筆者らが Linux カーネルをベースとして開発した「強制アクセス制御」の一実装であり、ポリシーの生成を支援するための学習機能を含む点に特長がある。TOMOYO Linux の概要については文献[7]を、実装については文献[8]を参照されたい。

TOMOYO Linux では、「起動履歴（親子関係）を含めたプロセス」をドメインと呼ばれる単位として扱い、それぞれのドメインで認める操作内容を列挙する。列挙する内容は、アクセスモード（読み/書き/実行）とその対象のファイル名である。例えば、ssh でログインした状態の bash に対して、/etc/passwd ファイルの参照と/usr/bin/scp の実行を許可する場合には次のように記述する。

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

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

ここで、/etc/passwd の前の”4”および/usr/bin/scp の前の”1”はそれぞれ Linux における”r—“、”—x”に対応しており、”rw-“では”6”、”rwx”では”7”になる。ドメイン表記については、<kernel>が仮想的な基点であり、それ以降は起動されるプロセスの並びを半角スペースを区切りとして文字列として結合する。例えば、「ssh でログインした状態の bash から起動した tcsh」については、

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

となる。TOMOYO Linux のアクセス許可は SELinux ほど詳細ではないが、ファイル名をそのまま用いることができるので Linux/UNIX に関する標準的な管理スキルを持っていれば理解は容易である。また、権限を指定する単位であるドメインはプログラムの実行を機に分割され、アクセス許可を与える対象も 1 ファイルを単位としているため、SELinux よりも詳細に指定できる。

7.2. 実際のポリシー例

図 2 で示した多重化されたログイン認証について、具体的なポリシーにより説明する。解りやすくするために、ライブラリ等は省略している。以下のような手順でログインを行う。

- ssh によりログインし、ログインシェルとして/bin/falsh を起動する。/bin/falsh は、ログインシェルを使用した攻撃を防ぐために、kill 等の内部コマンドを持たず無限ループを実行できない(筆者らが作成した)シェルである。
- その状態で/bin/honey（追加認証 1 に対応）という(筆者らが作成した)プログラムを実行する。/bin/honey ではユーザに対してパスワードを求めるが、/bin/honey の内部で規定された正しいタイミングでパスワードを入力しないと認証に成功しないような仕様になっている。
- /bin/honey の認証をパスした状態において、/bin/candy（追加認証 2 に対応）という(筆者らが作成した)プログラムを実行する。/bin/candy ではユーザに対してパスワードを求めるが、親プロセスが起動してから 10 秒以内に実行しないと認証に成功しないような仕様になっているため、親プロセスとして/bin/falsh を起動してから実行する。

・ scp および sftp はログインシェルから実行できるようにする必要があるため、通常通り実行許可を与えるが、これらのプログラムは/data/scp.tmp ディレクトリのみアクセスができるようにする。

<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

この他に、「<kernel> /usr/sbin/sshd /bin/falsh /bin/honey /bin/falsh /bin/falsh /bin/candy /bin/falsh /bin/bash」を信頼済みドメインとして登録し、そのドメインから/data/scp.tmp ディレクトリのファイルにアクセスする。

4.3. 実際の操作画面

ユーザから見た実際の操作手順を説明する。**図 6** は TeraTerm Pro というソフトウェアを使用して Linux サーバに ssh で接続する際に表示される画面である。この画面では、従来と同様にパスワードを入力してログインする。

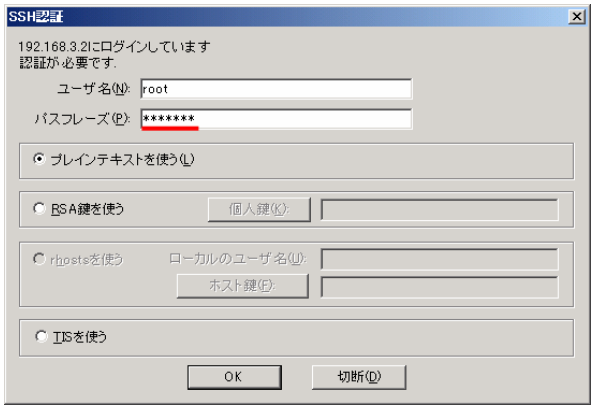

図 6 ログイン認証画面

その後、ポリシーで規定されたとおりに/bin/honey /bin/falsh /bin/candy という順序でコマンドを実行し、認証を通過する。

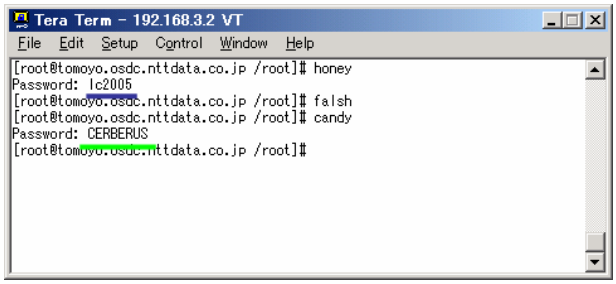

図 7 追加認証の画面

/bin/candy を通過した後(図 2 で城に到達した状態に対応)は、bash を起動して通常の操作を行う。

8. 考察

8.1. PAM との比較

従来のログイン認証が抱える脆弱性について、PAM(Pluggable Authentication Modules)の機能を用いて複数の認証メソッドを組み合わせることは可能である。しかし PAM 自体に脆弱性が存在した場合、requisite で指定した全てのモジュールが実行されるより前にシェルを起動される可能性が残る。

また、通常は**図 6** のように「ユーザ名」と「パスワード」しか入力欄が無いため、システムが提供している PAM で複数のパスワードを用いて認証することはできない。そのため、時間帯（pam_time.so）や端末のデバイス名（pam_securetty.so）等、パスワード以外の情報と組み合わせて認証を行うようになっている。

どうしても複数のパスワードを用いて認証を行いたい場合、**図 8** のようにパスワード欄を桁位置で区切るなどして全てのパスワードをその入力欄に詰め込む必要がある。この方法だと、新しい要素を追加する場合には、パスワード欄の区切り方（解釈の方法）を変更する必要が生じてしまう。

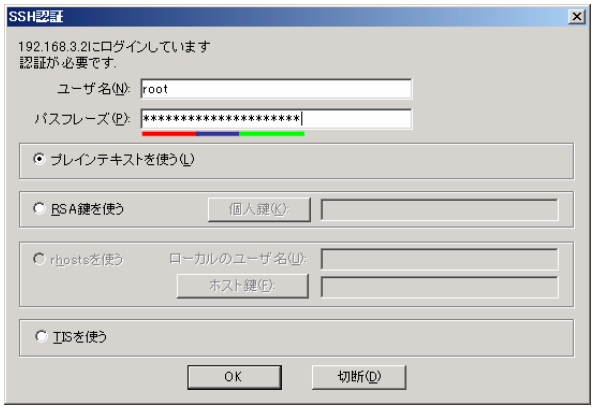

図 8　多重化をせずに要素を増やそうとした場合

多重化による方式では、**図 6** および**図 7** のように、複数回に分けている為、認証方式が変更されてもパスワード欄の区切り方に手を加える必要がない。つまり、既存プロトコルや現在の PAM の構成を変更せずに実現できるというメリットがある。

8.2.　認証に使用できる要素

通常のログイン認証においては、パスワードを入力する過程等をシステム側が知ることはできず､認証プログラムは｢入力結果｣に基づいてのみ本人かどうかの判断を行う｡しかし、本論文で紹介したような多段階の認証を行うことにより、パスワード文字列を入力する時間間隔や、ログイン後にユーザが行った操作内容等の情報も認証のための判断情報として活用することができる。これにより事実上無限に近い組み合わせのカスタマイズされた認証を実現することができる。

8.3.　利用者側の負担増について

パスワード以外に認証のための情報を提示してでも不正アクセスから保護するべき価値のあるシステムでは、複数の認証のための情報を提示するのは妥当だと考える。

本論文で紹介した手法は、１回の認証で全ての認証のための情報を提示するのではなく、複数回の認証で１個ずつ提示しているだけである。利用者から見れば、情報を提示するタイミングが分かれただけである。また、認証方式は自由に決定することが可能なので、利用者が負担に思わない方式を利用できる。

8.4.　ログイン認証を強化するための代償の評価

本論文の手法では、複数の認証プログラムの１つに欠陥が含まれていたとしてもそれが全体的な被害に直結することはない。C 言語で僅か数十行程度のコードで、システムのログイン認証のセキュリティを飛躍的に高めることができる。

8.5.　セキュリティ・スタジアム 2004

本論文で紹介したログイン認証の強化について、JNSA が主催したセキュリティ・スタジアム 2004 にて防御側として出展を行った。 root パスワードを公開し、セキュリティ技術のエキスパートによる攻撃を受けたが、容易に被害を与えることができないことが実証された。詳しくは文献[9]を参照されたい。

8.6.　強制アクセス制御機能を持たない OS への適用について

ログイン認証を連続して複数回行わせることは強制アクセス制御機能を持たない OS でも可能である。しかし、プログラムの振る舞いを外部からポリシーにより制限できない場合、それぞれのプログラムが自身の振る舞いを制限する必要があり、抜け道が発生しないように細心の注意を払わなければいけない。強制アクセス制御機能があれば、外部からポリシーにより制限できるので、抜け道を心配する必要が無く認証を行うプログラムを容易に作成できるのである。従って、本手法は強制アクセス制御機能を持つ OS に対して適用してこそその意味がある。

9.　おわりに

不正アクセスを防ぎ、情報漏洩を予防するためにセキュリティ強化 OS が登場し、その普及が始まりつつある。適切なポリシーを策定することができれば、バッファオーバーフロー等による乗っ取りのリスクを低減し、システムのセキュリティを高めることができる。しかし、いかにファイル等へのアクセスを制限できたとしても、パスワードに基づくログイン認証だけに頼っていると、それが思わぬ落とし穴になり得る。本論文で紹介した、強制アクセス制御を使用したログイン認証の多重化手法は、ワンタイムパスワード認証や高価なバイオメトリクス認証を必要とせず、容易に実装できる。

謝辞　本論文で紹介した内容について、方式の検討から TOMOYO Linux 上での実装と評価まで NTT データカスタマサービス半田哲夫氏の多大な支援を受けました。半田氏に感謝致します。

参考文献

[1] 「電子政府におけるセキュリティに配慮した OS を活用した情報システム等に関する調査研究」
内閣官房情報セキュリティセンター
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] パスフレーズ vs. パスワード
http://www.microsoft.com/japan/technet/community/columns/secmgmt/sm1004.mspx
[7] 原田季栄、保理江高志、田中一男 「使いこなせて安全な Linux を目指して」
Linux Conference 2005
http://lc.linux.or.jp/lc2005/02.html
[8] 原田季栄、保理江高志、田中一男 「TOMOYO Linux-タスク構造体の拡張によるセキュリティ強化 Linux」
Linux Conference 2004
http://lc.linux.or.jp/lc2004/03.html
[9]セキュリティ・スタジアム 2004
http://www.jnsa.org/active/press/vol12pdf/4_report4.pdf