第 8 講
パスワードは安全な認証方法か

パスワードはコンピュータを利用する際の認証方法として、長年使われてきた技術です。今も多くのシステムで採用されています。しかし、コンピュータ技術がこれだけ発達した現在でも、本当にセキュリティ的に安全な認証方法と言えるのでしょうか？

8.1 パスワードとは

そもそもパスワードとは何なのでしょう？　より正確に言えば「パスワードを用いた認証方法とは何か？」という問いになります。一般的な定義であれば「知識を共有し、その知識を持っていることを確認することで、正当な相手であることを確認する」というメカニズムがまずあって、その「共有している知識」がパスワードだと言えます。

たとえば、忠臣蔵の赤穂浪士が吉良邸討ち入りの際に相手を確認するために「山」「川」と言ったのも、共有している知識を確認しているので、一種のパスワード認証と言えます。「山」と「川」がパスワードです。

しかし、これは本人を認証しているわけではなく、知識を持っている者を確認しているに過ぎません。「者」と言っていますが、人間じゃないかもしれません。あるいはパスワードを試して、たまたま同じだったとしても同様です。偶然であっても「知識の共有をしている」とみなされてしまいます。

8.2 パスワードの実現方法

コンピュータでパスワード方式の実現方法をみた場合、外形的にはだいたいこんな感じです。

- 8 文字程度の文字列をユーザに入力させ、コンピュータ内部に事前に登録してある情報と突き合わせて同じかどうかを調べる。
- 同じであれば秘匿されている知識を共有しているとみなす。

どこまでの文字列の長さが使えるか、そしてどんな文字が使えるかはシステムに依存します。明確な基準はなく、かなり実装依存になります。UNIXのログインなどで使用するパスワードは、アルファベットの大文字と小文字、0～9までの数字、#や%といった何種類かの特殊文字が使えます。ですが、たとえばWebサイトのログインパスワードなどはアルファベット大文字小文字と数字のみというものも、かなりあります。過去にはPCのパスワードではアルファベット大文字小文字を区別しないというものがありました。

このように使える文字、長さ 1 つとっても、実現方法は強くシステムに依存しています。各々のシステムが採用しているセキュリティポリシーと言えば聞こえはいいのですが、「とりあえずやってみた。ないよりはマシ」という程度のものも散見されます。

COLUMN
PAMとlibpam-cracklib

GNU/Linuxのパスワード認証は、現在は「PAM（Pluggable Authentication Modules）」というメカニズムを使っており、古典的なUNIXより高度なパスワード認証の枠組みを備えています。

名前のとおり、プラグインするような形式になっています。現在おもなディストリビューションではパスワード認証のプラグインなどはデフォルトで設定されています。

たとえばDebian GNU/Linux 6.0.7 の場合、libpam-cracklibというプラグインが入っていない状態だと 6 文字パスワードを許しますが、libpam-cracklibを入れるとデフォルト設定で 8 文字になるといった具合になります。

パスワードの設定ファイルは「/etc/pam.d/common-password」です。libpam-cracklibが正しくインストールされていると次のような設定行があるはずです。

```
password requisite pam_cracklib.so retry=3 minlen=8 difok=3
```

これは「パスワードの試行回数が 3 回」「パスワードの最小文字列数が 8 文字」「新しくパスワードを設定するとき、それまでのパスワードより 3 文字以上違うことを要求する」という意味です。パスワードの最小文字列を引き上げるなど、サイトのポリシーによって変更することができます。

8.3 ひどいパスワード

次の 2 つは、「悪いパスワード」の例としてよく出てくるパターンです。

①Hironobu1963
②12345678

①は名前と生まれた年の組み合わせです。本人は忘れることはないでしょうが、簡単に類推されてしまいます。

②は誰が考えてもひどいパスワードでしょう。でも、現実にはそんなに笑ってはいられないのです。2012 年 7 月、CNETに「Yahoo hack reveals most-used passwords」という興味深い記事［39］が載りました。2012 年に米Yahoo!から 45 万件のパスワードが流出しましたが、この記事は、それを解析した結果が公開されたという内容です。わかったパスワードのうちで、筆者が興味を引かれたものをリストアップしてみました（**表 8-1**）。

◆ 表 8-1　米 Yahoo! の 45 万件のパスワードの分析結果（一部）

	発見したパスワードの件数	パスワードの例
A	2,295	12345、123456、1234567……
B	160	111111、0000000、777777……
C	780	password
D	233	password1、password2
E	106	batman、superman……
F	27	ncc1701、ncc1701a……

A（2,295 件）の 12345……というのは、そのまま 1 から順に数字を並べたものです。B（160 件）は 111111 という具合に同じ数字を並べたものです。C（780 件）はそのままpasswordという単語です。D（233 件）はpasswordという単語の後ろに数字などを付けたものです。E（27 件）はbatmanやsupermanという誰でも思いつくような名前。筆者だけでなく、読者のみなさんも、「これはひどい」と思うでしょう。でも、これは現実に使われていたパスワードなのです。これだけで 3,000 アカウント以上が不正に使えるのです。

ところで、F（27 件）のncc1701、ncc1701aの文字の並びは何だと思いますか？　これを見て筆者は、「ほう！」と思いました。実はこれ、スタートレックのU.S.S.エンタープライズ号の登録番号なのです。つけたい気持ちはわかりますが、筆者が見て瞬時にわかる程度の情報は、すでにWikipedia上に存在します。そのWikipediaのテキストアーカイブをダウンロードし、すべての文字列パターンを抜き出し、ソートして重複文字列を消してしまえば、その中に入っているレベルの単語でしかありません。Wikipediaには大量の単語データが入っているといっても、それを抜き出すのは、UNIX演習の課題レベルで、それほど難しいことではありません。Wikipediaベースのパスワード類推用辞書などノートPCで簡単に作れます。

8.4 本来の役目を果たしていないパスワード

前述のCNETの記事で公開されているのは、漏洩したパスワードデータのすべてではなく、あくまでもパスワード探しを行って見つけられたものだけです。でもどれだけ見つけられたのでしょうか？

報告によれば 45 万件の漏洩したパスワードのうち 13 万 7,559 件、つまり全体の約 30%が判明したそうです。そして、そのうちの 10 万 6,873 件がGmailやHotmailに使いまわされていたそうです。

2013 年 5 月に、「ディノスに 111 万件の不正アクセス、1 万 5000 件の不正ログイン」というニュース［40］が報道されています。このディノスの大量の不正ログインも、先ほどと同様に、ほかのサイトから漏れたパスワードを使ってログインしていることを強く示唆しています。

最近は、Webサービスのアカウント名をメールアドレスにするものが多いです。その状況で同じパスワードを使いまわしていると、1 つのサイトでパスワードが判明されれば、芋づる式に次々にほかのサイトも不正ログインできることになります。

パスワードの本来の目的は、アカウントを使うユーザを正しく認識するためのものです。ですが、システムの全アカウントの 3 割がその役目を果たしていません。しかも、まったく情報を漏らしていないはずのサイトにも影響がおよびます。

ディノスの例では、「不正にログインできた割合は 1.35%」と低い率と考えるのではなく、「ディノスに大きな落ち度もないのに、不正に利用できるアカウントが 1 万 5,000 件も手に入る」という見方をすべきです。これだけ手に入れば何をするにしても十分過ぎるほどの数でしょう。

「パスワードがきちんとしていれば安心だ」と言うかもしれませんが、これだけ悲惨な数字を見ると、これまでのパスワード認証は期待どおりに役にたっているとは到底言えません。

8.5 適切なパスワードの管理とは

ここでもう一度、パスワード管理の方法とパスワードを不正に割り出す方法を整理してみましょう。そのほうが議論の全体像を整理できてわかりやすいと思います。

ですが、その前に「123456」、「password」のようなパスワードや、アカウント名が「admin」でパスワードも「admin」といったものでは、さしたる事前の準備も必要なく、いくつか試せば判明してしまいます。これから述べるようなパスワードの割り出しの作業すら必要ありません。これはさすがに「愚かなパスワード」としか呼びようがありません。しかし、これはユーザだけが悪いわけではなく、このようなパスワードを許容すること自体が、システムの欠陥だと筆者は思います。

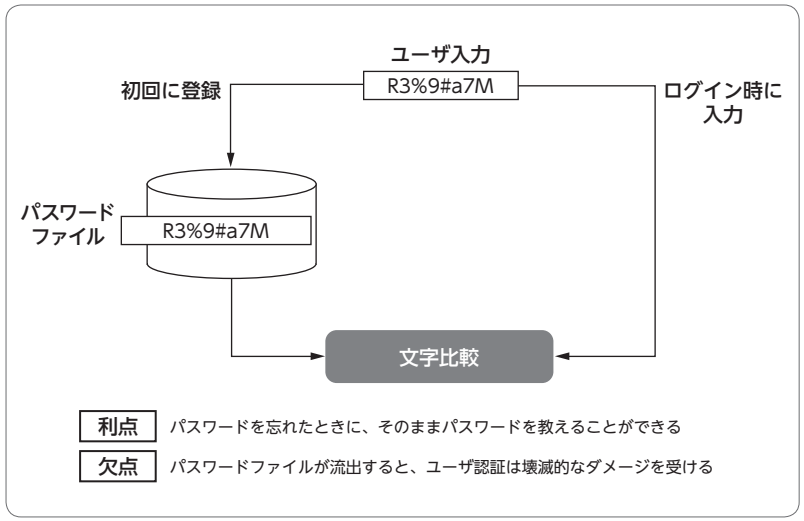
それでは本題に入りましょう。ある程度の複雑さを持ったパスワードに関しては、パスワード管理ファイルをサイトから流出させ、専用のコンピュータを用意してパスワードを見つける処理を行うというのが前提となります。サイトへの侵入方法やデータファイルの流出方法に関しては、本題から外れるためここでは省略します。

さて、以下に 3 つのパスワードの管理パターンを説明し、次に、どのような方法でパスワードを見つけていくかの説明を加えます。

パスワードを防御なしに管理

パスワード認証というと、ユーザが入力した文字列と、すでにパスワードとして登録してある文字列を突き合わせる（**図 8-1**）ことだと思っている人が意外と多いのではないかと思います。

◆ 図 8-1　保存されているパスワードは保護されていない



確かにこの方法だと簡単に実装できますが、パスワードファイルが外部に流出した時点で、システム全体のユーザ認証の枠組みが崩壊してしまい、サービスの存続が脅かされることになるでしょう。なぜならば、そのままの形ですべてのパスワードを利用できるからです。

中には、「これはファイルへのアクセスをコントロールすることで、安全性を保っている」と理解している人がいるかもしれません。しかし逆を言えば、それだけしか保護をしておらず、何かのタイミングで情報が流出する可能性についてはまったく考慮されていません。1 つのエラーがシステム全体をカタストロフィー（破局的）な状態に導く可能性がある脆弱な状態と言えます。

このシステムが大きな問題をはらんでいるのは、あらためて強調しなくとも読者のみなさんは理解されているだろうと思います。その一方で、このようなシステムは、それほど多くはないだろうと楽観的に考えるかもしれません。しかしながら、パスワードを忘れたときに親切に正しいパスワード文字列を教えてくれるタイプのシステムは、みなさんの周りを見渡せば結構あるはずです。

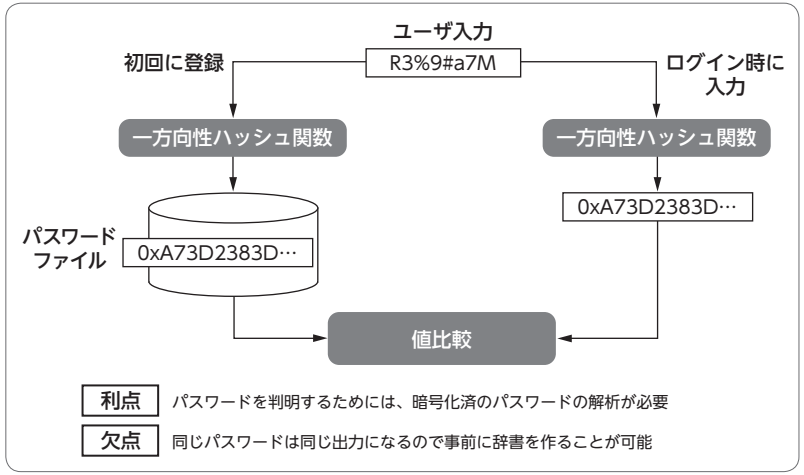
一方向性ハッシュ関数で計算した値を管理

一方向性ハッシュ関数とは、値xに対しハッシュ関数Hを用いて計算した値H(x)から、逆をたどってxを見つけることは極めて困難であるという性質を持つ関数です。一方向性ハッシュ関数としては、MD5、SHA-1、SHA-256 といったものだけではなく、暗号化関数を使ったメッセージ認証コードなども同様に使えます。たとえば古典的UNIXのパスワード生成にはDES暗号を使っているDES-CBC-MACというメッセージ認証コード法が使われています。また、一方向性ハッシュ関数を使用したメッセージ認証コード法HMACも使用されます。

よく「パスワードを暗号化」するという表現を使うので、暗号化するなら復号もできるだろうと類推しそうですが、これは一方向にしか計算できません。ですので、一方向性ハッシュ関数なのです。

ユーザが入力した文字列を一方向性ハッシュ関数で計算します。その値と、パスワードを一方向性ハッシュ関数で事前に計算し登録しておいた値とを比較します（**図 8-2**）。たとえばWindows XPなどで使っていたLMハッシュ（LAN Manager hash）は、この方式です。

◆ 図 8-2　一方向性ハッシュ関数を導入する



生の文字列を持っている**図 8-1** の方法よりは安全になってはいます。しかし、この方法は、同じ入力に対しては同じ出力値を返してしまうという弱点を持っています。パスワード攻撃用の辞書に載っているようなパスワードであれば、事前に処理してハッシュ化済み辞書を作成できます。つまり、辞書攻撃には極めて脆弱なパスワードシステムだと言えます。

ソルトを加えたパスワード管理

一方向性ハッシュ関数だけだと逆引きの攻撃辞書が使えるので、それを困難にするために、ソルト（salt）と呼ばれるユーザごとに異なるランダムな値を加えたのちに一方向性ハッシュ関数に入力する方法をとります（**図 8-3**）。

このソルトは隠さなくてもかまいません。長ければ長いほど逆引きの攻撃辞書のサイズが巨大になります。ソルトはランダムデータですが、それほど大きなものは必要ありません。ここ数年のパスワードの安全性でかまわなければ、32 ビット（4 バイト）程度でも十分に役目は果たします。70 年代は 12 ビット程度が使われていました。将来的にも使うと考えるならば、80 ビット（10 バイト）程度あればかなりの確実性をもって安全と言えるでしょう[注1]。

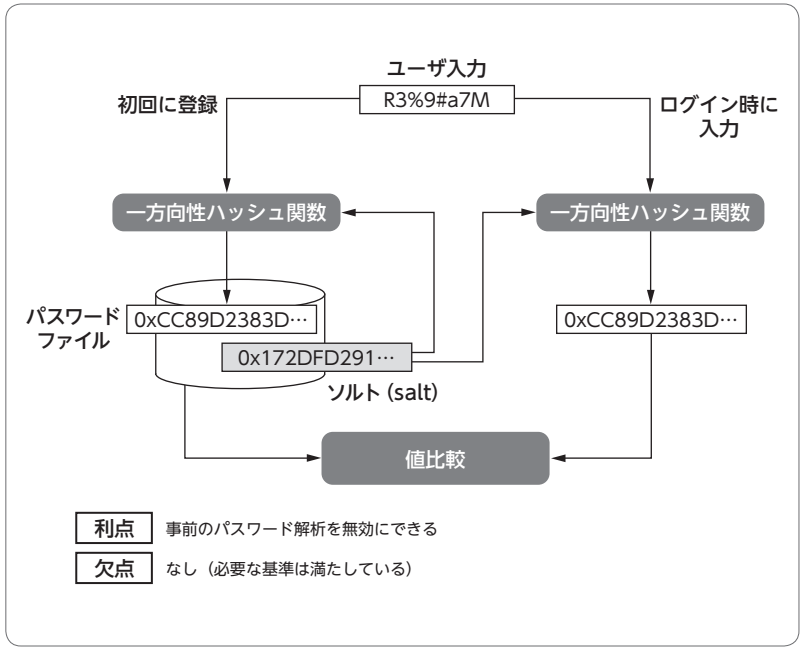
UNIX のパスワード処理では、さらに一方向性ハッシュ関数を複数回かけて計算時間をより多くかけさせるという手法を使っています。ただし、一度に多数のユーザの対応をしなければならない Web サイトで行うのは、認証サーバ側に負荷がかかり過ぎる可能性があるので、本当にこの手法を選択すべきかどうかは、考える必要があります。

基本的には、複雑さを増すにはパスワードに使える文字種類を多くしたり、文字数の数を増やしたりすることが重要です。これでやっと普通[注2]のパスワード管理です。

注 1　ただし、パスワードが 12 ～ 16 文字分のランダム文字列といった安全性が非常に高いものでなければ、ソルトだけビット数を増やしても意味はありません。

注 2　ここでの「普通」とは、ただ単純に「統計的に多くあるパターン」という意味ではなく、「理想に近い」という意味での普通です（参考：「Dream Fighter」by 中田ヤスタカ）。

◆ 図 8-3　ソルトを加えて一方向性ハッシュ関数にかける



8.6 安全なパスワードなのだろうか?

よくある「安全なパスワードを運用しましょう」という説明には、「複雑な並びであること」「長いこと」「頻繁に変更すること」という条件が推奨されています。しかし、次のポイントを指摘したいと思います。

- **複雑な並びであること**

  人が考えるとバイアスが入るので安全なランダム性を確保できない。

- **長いこと**

  人間の記憶能力を過大評価している。結果として思い出しやすいパス

ワードを選択するバイアスがかかる。

- **頻繁に変更すること**

  上の 2 点の問題を何度も繰り返すことになり、結果として、覚えやすいパスワードか、同じパスワードが繰り返し使われることになる可能性が大きい。

たまに「“i love you”ならば“1 l0v3 y0u”といった具合に規則性をもってほかの文字列に置き換えをすると、英語の辞書に載っていないので、辞書攻撃は避けられる」とか、「“i want to eat cake”といった好きな文章の頭文字をとって“iwtec”とすると良い」などと解説するWebサイトを見かけますが、この程度の乱雑さは、コンピュータの前では乱雑さにすらならず、ほぼ無力です。

パスワードのみで抵抗するならば、十分に長いランダム文字列を機械で生成するしか方法はありません。当然、数十個とか百個とかを越えるランダム文字列を人間は覚えきれませんので、パスワード管理のためのツールを使うことになります。コンピュータが簡単に推定できるパスワードを使うくらいなら、十分に長いランダム文字列を紙に書いて、その紙を物理的にしっかり管理したほうがよほど安全です。

あるいは、もうあきらめて、自分も次回はログインできないかわりに誰もログインできないくらい複雑なパスワードを考えて、使うときはパスワードを再設定するかです。

これくらい割り切って使わないと、パスワードが本来与えるはずであろうユーザ認証の能力を引き出せません。

8.7 パスワードに代わる認証方法

さて、現実を眺めると、多くの場面で導入されているこれまでのパスワード管理／運用方法は、すでに寿命がつきかけていると言わざるを得ません。しかし一方で、いまだにパスワードは全盛です。

この現状ですが、先端企業は徐々に変わりつつあります。すでに、Googleのアカウントは二重認証を取り入れています。筆者はすでに使っているのですが、それほど煩雑というほどではありません。銀行などのように想定される被害金額が大きい場合、ワンタイムパスワードのセキュリティトークンを使うのも合理的な方法だと思います。

筆者は公開鍵方式を利用したユーザ認証方法を積極的に取り入れるべきだと考えています。この技術は、UNIXユーザにとってはSSHの公開鍵認証方式などで身近に使われていますが、一般にはなかなか普及できていないのが実状です。

今は、これまでの古典的なパスワード認証から次の世代の認証へと変化する端境期（はざかいき）だと言えるのかもしれません。ですが、まだしばらくは、これまでのパスワードとの付き合いは続きそうです。