

パスワードの話

- 自己紹介
- パスワードの保存
- パスワードの話題いろいろ (TODO)
 - 定期更新
 - ユーザ側での管理
 - 強度
 - マスキング
- まとめ

自己紹介

- ECナビ システム本部 春山征吾 @haruyama
- セキュリティ
 - OpenSSH (本x2, [OpenSSH情報](#))
 - [暗号技術大全](#)
 - 18章(ハッシュ), 20章(電子署名)翻訳担当
- 全文検索システム Apache Solrの勉強会開催

資料

- 本資料
 - TODO

パスワードの保存

- パスワード保存の常識(?)
- Unixのパスワード保存の歴史
- Unix的パスワード保存
 - 概要
 - ハッシュ
 - Salt
 - Strectch
- Webシステムでのパスワード保存
- Windowsのパスワード保存(?)

パスワード保存の常識(?)

Webシステムの記事では、パスワードの保存に

「saltを付けてハッシュ」

とよく言われている。

保存

- saltを付けてハッシュ化

- 保存された情報からはパスワードは復元困難

照合

- 入力値にsaltを付けてハッシュ化。保存情報と照合

Unixのパスワード保存の歴史

Unix的パスワード保存

GNU/Linuxの場合

形式

`idsalt$hashed`

例

`$6$3d1ahu0b$KiH....` (略)

- `id`: ハッシュ(後述)の識別子
 - 1 => MD5, 5 => SHA-256 6 => SHA-512
- `salt`: ソルト, お塩
- `hashed`: ハッシュ化されたパスワード情報

ハッシュとは？

TODD: 書き直す

暗号学的ハッシュ関数 - Wikipedia より

- 与えられたメッセージに対してハッシュ値を容易に計算できる。
- ハッシュ値から元のメッセージを得ることが事実上不可能であること。
- ハッシュ値を変えずにメッセージを改竄することが事実上不可能であること。
- 同じハッシュ値を持つ2つのメッセージを求めることが事実上不可能であること。
- 例: MD5, SHA1, SHA-256, 512

salt(ソルト, お塩)とは?

ハッシュの値をかきまぜる「お塩」.

- ハッシュ化するだけでは,
同じパスワードを利用する人が複数いるとき
同じパスワード情報が生成されてしまう
 - ユーザごとに異なる必要がある
 - ランダムでなくてもよい
 - 同時に多数のパスワード情報の解析を不可能に
- saltのサイズ
 - 伝統的なunix: 12bit / 現在のGNU/Linux: 96bit
 - CRYPTOGRAPHY ENGINEERING: ハッシュのサイズ

なぜ saltが必要か

TODO

Free Rainbow Tables » Distributed Rainbow Table
Generation » LM, NTLM, MD5, SHA1, HALFLMCHALL,
MSCACHE

なぜ salt はいくつも必要か

TODO

Free Rainbow Tables » Distributed Rainbow Table
Generation » LM, NTLM, MD5, SHA1, HALFLMCHALL,
MSCACHE

実際の処理

- CRYPTOGRAPHY ENGINEERING p304 の方式

PHP風の言語で記述

```
$x = '' ;  
for($i = 0; $i < $iter; ++$i) {  
    $x = hash($x . $password . $salt);  
}
```

- [crypt() アルゴリズム解析 (MD5バージョン)]
どちらも ハッシュを繰り返し利用している (stretch)

stretchとは?

- ハッシュを繰り返し利用することで、ハッシュ値を求めるのに必要な時間を増大させる
 - 攻撃に時間がかかるようになる
 - 実質的にパスワード文字数を伸ばす(stretchする)効果
- どれくらいやるのか
 - `crypt()` MD5の場合: 1000回
 - `crypt()` SHA-256, 512の場合: (デフォルト)5000回
 - CRYPTOGRAPHY ENGINEERING での例:
2²⁰(約100万)回

stretchの効果(1)

stretchの効果をはかるために、PHPの hash 拡張で SHA-256 を繰り返し呼ぶコードを用いた計測をした

- 方式は CRYPTOGRAPHY ENGINEERING のもの
- パスワード 10byte
- salt 32byte
- CPU 1コアのみ利用

Intel(R) Core(TM) i7 CPU 920 @ 2.67GHz で 1秒に
約50万回計算できた.

stretchの効果(2)

- ・パスワードの文字種を64bitとすると

文字数	総パスワード数
n	64^n
3	26万
4	1677万
5	10億
6	687億
7	4兆
8	281兆

stretchの効果(3)

1CPU(8コア)のPCでパスワード解析する場合を考察

- 1日3456億回 計算可能
 - stretch がないと...
 - 6文字が 0.2日, 7文字が 13日
- 1000回 stretch すると
 - 1日3.5億回パスワードを計算可能
 - 5文字が 3日, 6文字だと 199日

stretchの効果(4)

MD5だと.. (TODO)

stretchの強度は, (回数) x
(1回あたりの実行時間)で比較

方式の保存

現在は問題なくとも、将来問題になるかもしれない

- ハッシュ関数自体
- ハッシュ化の方法
- stretch回数

長く運用するシステムでは、

パスワード保存方式(のID)をパスワード情報と共に保存する

なぜUnixはこの方式なのか？

先に歴史で説明する？(TODO)

- なぜ可逆な暗号化ではないのか？
- 鍵を管理するのが難しい。
 - 以下からパスワード情報と鍵が漏れるかもしれない
 - バックアップファイル
 - システムの脆弱性
 - 別のOSでブート
 - 物理的な攻撃

Unix的パスワード保存まとめ

- パスワードはハッシュ化して保存
 - この時 salt と stretch を利用
- メリット
 - 鍵管理が不要
 - 生パスワードを復元できない
- デメリット
 - 弱いパスワードが記録された情報だけで破れる

Webシステムでは？

- 通常WebサーバとDBサーバは物理的に分離されている(されていない場合もあるが).
- Unixよりもパスワード情報と鍵が共に漏洩するリスクは低いだろう.
- もちろん、鍵管理のコストは無視できない
 - 漏洩，改竄，紛失

鍵を用いる場合の手法案

- (共通鍵)暗号
- ハッシュ+暗号
- 鍵付きハッシュ

(共通鍵)暗号

- メリット
 - ちゃんと暗号化し鍵が安全ならば、
弱いパスワードもパスワード情報だけでは破れない
- デメリット
 - 鍵があればパスワードを復元できる
 - 鍵の管理の必要がある

ハッシュ+暗号

常識(?)通りにハッシュ化したあとで暗号化

- メリット
 - ちゃんと暗号化し鍵が安全ならば,
弱いパスワードもパスワード情報だけでは破れない
 - 鍵を保持するものでも生パスワードを復元できない
- デメリット
 - 鍵の管理の必要がある

鍵付きハッシュ(1)

- saltの一部を固定の鍵に?
 - 単純に鍵と平文を文字列連結をしたものをハッシュするMACは期待通りの強度がないという論文

On the Security of Two MAC Algorithms

- hash(\$key . \$salt . \$password) などは避けよう

鍵付きハッシュ(2)

- HMACには前述の問題はない
 - CRAM-MD5はHMACを元にしたパスワード情報保持をしている。
 - チャレンジレスポンス認証用の情報保持なので、応用していいかは不明

鍵付きハッシュ(3)

- メリット
 - ちゃんとしたアルゴリズムを用いて鍵が安全ならば、弱いパスワードも記録された情報だけでは破れない
 - 「ちゃんと」しているかは「ちゃんと」した人に確認してほしい
 - 鍵を保持するものでも生パスワードを復元できない
- デメリット
 - 鍵の管理の必要がある

まとめ

方式	弱パスワードの保護	生パスワード	鍵管理
ハッシュ	stretchで対応	復元不可能	不必要
暗号	可能	復元可能	必要
ハッシュ+暗号	可能	復元不可能	必要
鍵+ハッシュ	可能	復元不可能	必要

参考文献

man 3 crypt

Manpage of CRYPT

CRYPTOGRAPHY ENGINEERING

ISBN-13: 978-0470474242

認証技術 パスワードから公開鍵まで

ISBN-13: 978-4274065163