Lecture 10: Passwords

ENGG5105/CSCI5470 Computer and Network Security
Spring 2014
Patrick P. C. Lee

How (Stupid) People Are With Their Passwords?

- 4 in 10 respondents shared passwords with at least one person in the past year.
- Nearly as many people use the same password to log into multiple Web sites
- Almost half of all users never use special characters (e.g. ! ? & #) in their passwords
- 2 in 10 have used a significant date, such as a birth date, or a pet's name as a password – information that's often publicly visible on social networks.

Reference: http://www.securityweek.com/survey-reveals-how-stupid-people-are-their-passwords, October 2010

In Hong Kong….

- 70% use the same password with an average number of 7 accounts
- 60% never update their passwords

http://hongkongbusiness.hk/information-technology/news/70-hong-kong-netizens-use-same-password-in-multiple-accounts, August 2011

Roadmap

- How passwords work in Linux
- Password cracking: John the Ripper
- How passwords work in Windows
- Password protection
- One-time password

/etc/passwd

- Information about all local Linux accounts is stored in **`/etc/passwd`**
- Traditional Linux systems without Shadow Suite store passwords in **`/etc/passwd`**
- Format of **`/etc/passwd`**

```
username:passwd:UID:GID:full_name:directory:shell
```

Non-Shadowed Passwords

- Example (non-shadowed password):

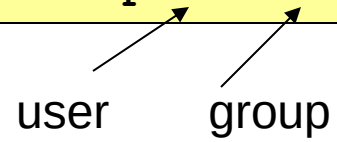
```
pclee:Npge08pfz4wuk:503:100:Patrick Lee:/home/pclee:/bin/bash
```

- **Npge08pfz4wuk** is the password in encoded format (discussed shortly)
- See for details:
  - http://tldp.org/HOWTO/Shadow-Password-HOWTO-2.html

Non-Shadowed Passwords

- The passwd file contains UID/GID, and must be readable by all users to allow name service switch
  - e.g., when a user lists file names in his folder

```
[pclee@localhost]$ ls -a a.txt
-rw------- 1 pclee lec 301 2010-10-01 22:41 a.txt
```



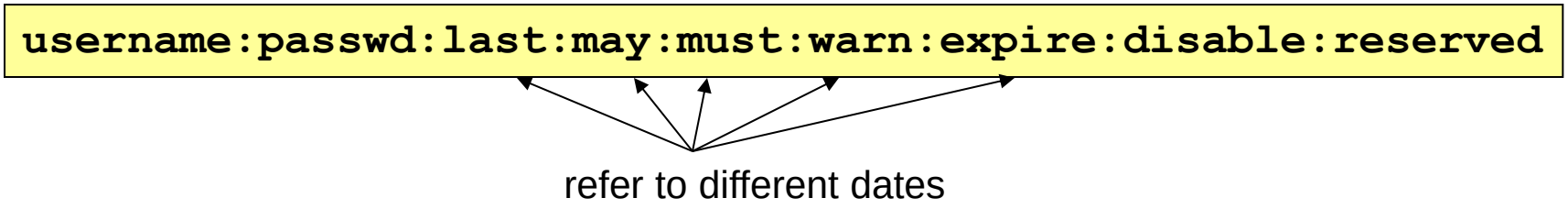
- Attackers can look at the encoded passwords and hack them (e.g., by brute force)

Shadowed Passwords

- Install Shadow Suite, move password to **`/etc/shadow`**, while **`/etc/passwd`** only stores user information
  - password field in **`/etc/passwd`** shows a character such as '*', or 'x‘
- **`/etc/shadow`** is only readable by root

Shadowed Passwords

- /etc/shadow contains password related information, e.g., password aging:



- Now, the default use in today's Linux/Unix
- See for details: http://tldp.org/HOWTO/Shadow-Password-HOWTO-2.html

How Passwords are Stored?

- Passwords are encoded via cryptographic hash, and the hash value is stored in /etc/shadow.
- Password encoding function must satisfy two properties:
  - Encoding is easy: any input password by the user will be encoded and verified with the stored value
  - Irreversible: stored value is hard to be reverted back to the original value
- There are many ways to implement the password encoding function

crypt() – keys and salts

- crypt() is the password encryption function
- based on a variant version of Data Encryption Standard (DES)
- Takes two arguments:
  - **keys** – user's passwords
  - **salts** - two-character string chosen from the set [a-zA-Z0-9./]. This string is used to perturb the algorithm in one of 4096 different ways
    - Make two identical passwords look different

crypt()

- You can call crypt() from C, or in Perl:

```
$ perl –e 'print crypt("mypass", "s1"), "\n"'
s1tR0evFyi.yQ
$ perl –e 'print crypt("mypass", "s2"), "\n"'
s2JQ85JElCMeU
```

- Note the salt is prefixed to the hash result
- By default, crypt() is based on DES
- DES is actually an encryption algorithm, but used as a hash function
  - Password as the key
  - Without the key, the encrypted string cannot be decrypted, so it's a one-way function

crypt()

- How crypt() works with DES:
  - Take the first 8 characters of the password
  - Concatenates the low 7-bits of each of these 8 characters into a 56-bit key
  - This 56-bit key is used to encrypt repeatedly a constant string (usually a string consisting of all zeros)
  - The encryption result is 64 bits, and is split into 11 6-bit ASCII characters
  - Add two characters at the front, the final result is a series of 13 printable ASCII characters

crypt()

- This means that even you provide a password with more than 8 bytes, only the first 8 bytes are considered

```
$ perl –e 'print crypt("mypass12", "s1"), "\n"'
s1mtUUG4w8zLw
$ perl –e 'print crypt("mypass12345", "s1"), "\n"'
s1mtUUG4w8zLw
```

crypt() – more variants

- The glibc2 version of crypt() supports more algorithms
- crypt() takes a string starting with "id“, followed by the salt terminated by "$":

  idsalt$key

- salt can have up to 16 characters
- id identifies the encryption method used

crypt() – more variants

- The following id values are supported:
  - 1: md5
  - 2a: Blowfish
  - 5: SHA-256 (since glibc2.7)
  - 6: SHA-512 (since glibc2.7)
- The ASCII string of the password has size:
  - MD5: 22 characters
  - SHA-256: 43 characters
  - SHA-512: 86 characters

crypt() – more variants

- Example:

```
$ perl -e 'print crypt("mypass", "\$1\$abcdef\$"), "\n"'
$1$abcdef$nRHvewzGZJoYskd0AlE9r/
```

- Find out more about crypt()
  - man 3 crypt

Call crypt() from C

- See mypass_sample.c
- How to compile:
  - **gcc –o mypass_sample mypass_sample.c -lcrypt**

Roadmap

- How passwords work in Linux
- Password cracking: John the Ripper
- How passwords work in Windows
- Password protection
- One-time password

Password Cracking

- Goal: Guess the password to gain access
- Brute-force:
  - try all possible combinations of password
  - if DES is used, key space is 2^{56}
- More common strategies:
  - dictionary attack: selects common words from a dictionary (i.e., a word list)
  - by common patterns used (e.g., testing123)

Password Cracking

- How to start cracking?
  - Approach 1: Pick a username, guess a password, and then try to log in a system
    - Provide evidence of cracking, as failed logins may be logged
  - Approach 2: Crack the hashed passwords directly
    - Need to obtain copies of **`/etc/passwd`** and **`/etc/shadow`**
    - Run the hash algorithm with guessed passwords
- Let's use Approach 2
  - may not be realistic, but a bad administrator who owns the root of a system may like this approach

John the Ripper

- Features:
  - A fast, powerful password cracker
  - cracks password encoding algorithms used by crypt()
  - users can provide a word list for dictionary attack
- Designed for password recovery by system admins, but could be used by attackers for malicious use
- Official site: http://www.openwall.com/john/
  - Download the patched source code from course website

Running John the Ripper

- First, unshadow the password from /etc/shadow and /etc/passwd

```
unshadow /etc/passwd /etc/shadow > pass.db
```

you need root to access this file

- Inside pass.db – just like old systems that don't use Shadow Suite

```
pclee:s1mtUUG4w8zLw:500:501::/home/pclee:/bin/bash
```

Running John the Ripper

- How to use:

```
john pass.db
```

  - Run the default cracking mode

```
john --restore
```

  - Resume an interrupted session after CTRL-C

```
john --show pass.db
```

  - Display the cracked passwords

John's Modes

- **Single crack mode**: uses login/GECOS information as passwords
  - GECOS field means a field in /etc/passwd (e.g., full name)

```
$ john --single pass.db
…
alice            (alice)
guesses: 1  time: 0:00:00:00 100%  c/s: 4820  trying: bob1901 - bob1900
```

  - fast, but produces only limited results

John's Modes

- **Wordlist mode**: specify a word list for cracking

```
john --wordlist=words.lst pass.db
```

  - Find exact match in words.lst

```
john --wordlist=words.lst --rules pass.db
```

  - enable word mangling rules (e.g., given abc, I'll search for abc1)

John's Modes

- Incremental mode: tries all possible character combinations

```
john --incremental=alpha pass.db
   (letters only)
john --incremental=digits pass.db
   (numbers only)
john --incremental=alnum pass.db
   (letters and numbers)
john --incremental=lanman pass.db
   (letters, numbers, and some special characters)
```

More on John

- Just type **`john`**, and see the usage

```
[pclee@localhost run]$ john
John the Ripper password cracker, version 1.7.6
Copyright (c) 1996-2010 by Solar Designer and others
Homepage: http://www.openwall.com/john/

Usage: john [OPTIONS] [PASSWORD-FILES]
--single                  "single crack" mode
--wordlist=FILE --stdin   wordlist mode, read words from FILE or stdin
--rules                   enable word mangling rules for wordlist mode
--incremental[=MODE]      "incremental" mode [using section MODE]
--external=MODE           external mode or word filter
--stdout[=LENGTH]         just output candidate passwords [cut at LENGTH]
--restore[=NAME]          restore an interrupted session [called NAME]
--session=NAME            give a new session the NAME
--status[=NAME]           print status of a session [called NAME]
--make-charset=FILE       make a charset, FILE will be overwritten
--show                    show cracked passwords
--test[=TIME]             run tests and benchmarks for TIME seconds each
--users=[-]LOGIN|UID[,..] [do not] load this (these) user(s) only
--groups=[-]GID[,..]      load users [not] of this (these) group(s) only
--shells=[-]SHELL[,..]    load users with[out] this (these) shell(s) only
--salts=[-]COUNT          load salts with[out] at least COUNT passwords only
--format=NAME             force hash type NAME: DES/BSDI/MD5/BF/AFS/LM/crypt
--save-memory=LEVEL       enable memory saving, at LEVEL 1..3
```

Roadmap

- How passwords work in Linux
- Password cracking: John the Ripper
- How passwords work in Windows
- Password protection
- One-time password

Passwords in Windows NT/2000/XP

- Windows NT line of OSes stores two password hashes: Lan Manager (LM) Hash and NT Hash
  - LM Hash: primary hash function for old Windows system (e.g., Windows 95/98)
  - LM hash is stored in latest Windows OSes for backward compatibility
  - As of Vista, storing LM hash is disabled by default

LM Hash

- How it works?
  - user's ASCII password converted to uppercase
  - password null-padded to 14 bytes
  - split into 2 7-byte halves, each being a secret DES key to encrypt a fixed string
  - resulting hashes are concatenated
- Password is case insensitive

LM Hash

- Weaknesses:
  - Passwords only limited to ASCII characters
    - only 95 printable characters are available
  - Lowercase characters converted to uppercase
    - only 69 are remaining
  - Each half is hashed independently
    - Try to apply brute-force on each half = $69^7 \sim 2^{43}$
- Yet, for backward compatibility, some Window systems may still store LM hash

NT Hash

- NT hash is based on MD4 algorithm
  - Password is case sensitive
  - Max to 14 characters on Windows NT, no limit on Windows 2000 or XP
- NT hash now used in NTLM, the Windows authentication suite
  - NTLMv1: based on LM Hash and NT Hash
  - NTLMv2: based on NT hash and HMAC-MD5
  - Use a challenge-response protocol for authentication

Roadmap

- How passwords work in Linux
- Password cracking: John the Ripper
- How passwords work in Windows
- Password protection
- One-time password

Password Protection

- Avoid bad passwords
  - bad passwords based on names, words, or numbers, e.g., joe102367, fido2000, testing123, 12345678, nc1701-d
  - avoid using your birthday, your mom's birthday, non-English words (which are part of dictionary attacks)

Password Protection

- Create good passwords
  - Use at least one character from each of these character classes:
    - a-z,
    - A-Z,
    - Punctuation, such as ! (* $ 0-9
  - If DES passwords are used, choose 6-8 characters; if MD5 passwords are used, choose any number of characters (e.g., more than 15)

Password Protection

- A simple way to create effective passwords:
  - Think of a phrase that is relatively obscure, but easy to remember
  - Could be a line from a song, book, or a move
  - Create an acronym from it, with capitalized words and punctuations
    - **E**verything **i**s **n**ow **u**nder **m**y **c**ontrol!
      - 3!nuMc!

Roadmap

- How passwords work in Linux
- Password cracking: John the Ripper
- How passwords work in Windows
- Password protection
- One-time password

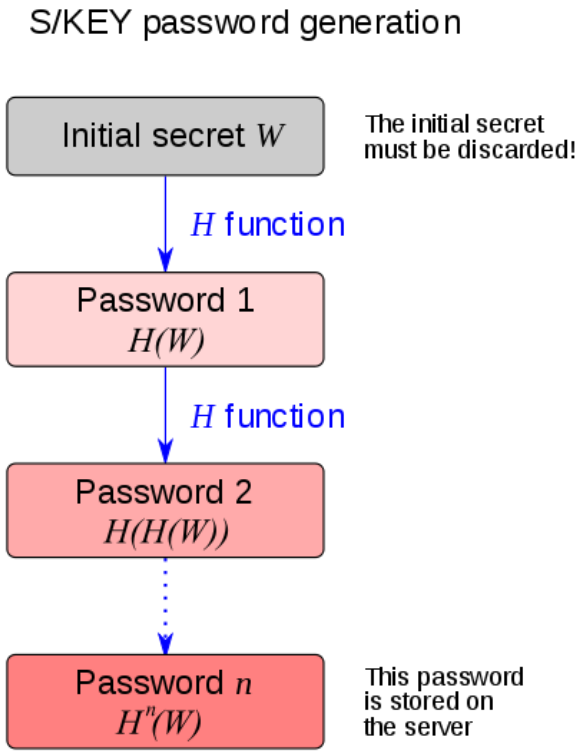
One-Time Password

- A one-time password (OTP) is a password that is valid for one login session or transaction
  - Advantage: robust against replay attacks as in static password
  - Disadvantage: human cannot memorize the password

See: http://en.wikipedia.org/wiki/One-time_password

S/KEY

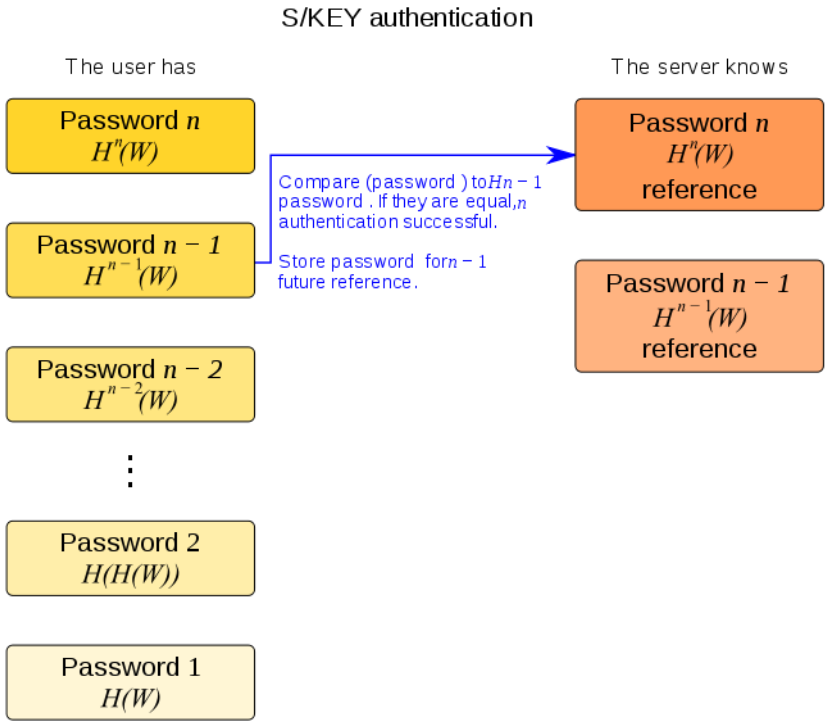
- Password generation:
  - Client generates n passwords, starting with an initial secret W
  - Client discards W
  - Stores $H^n(W)$ on the server



From: http://en.wikipedia.org/wiki/S/KEY

S/KEY

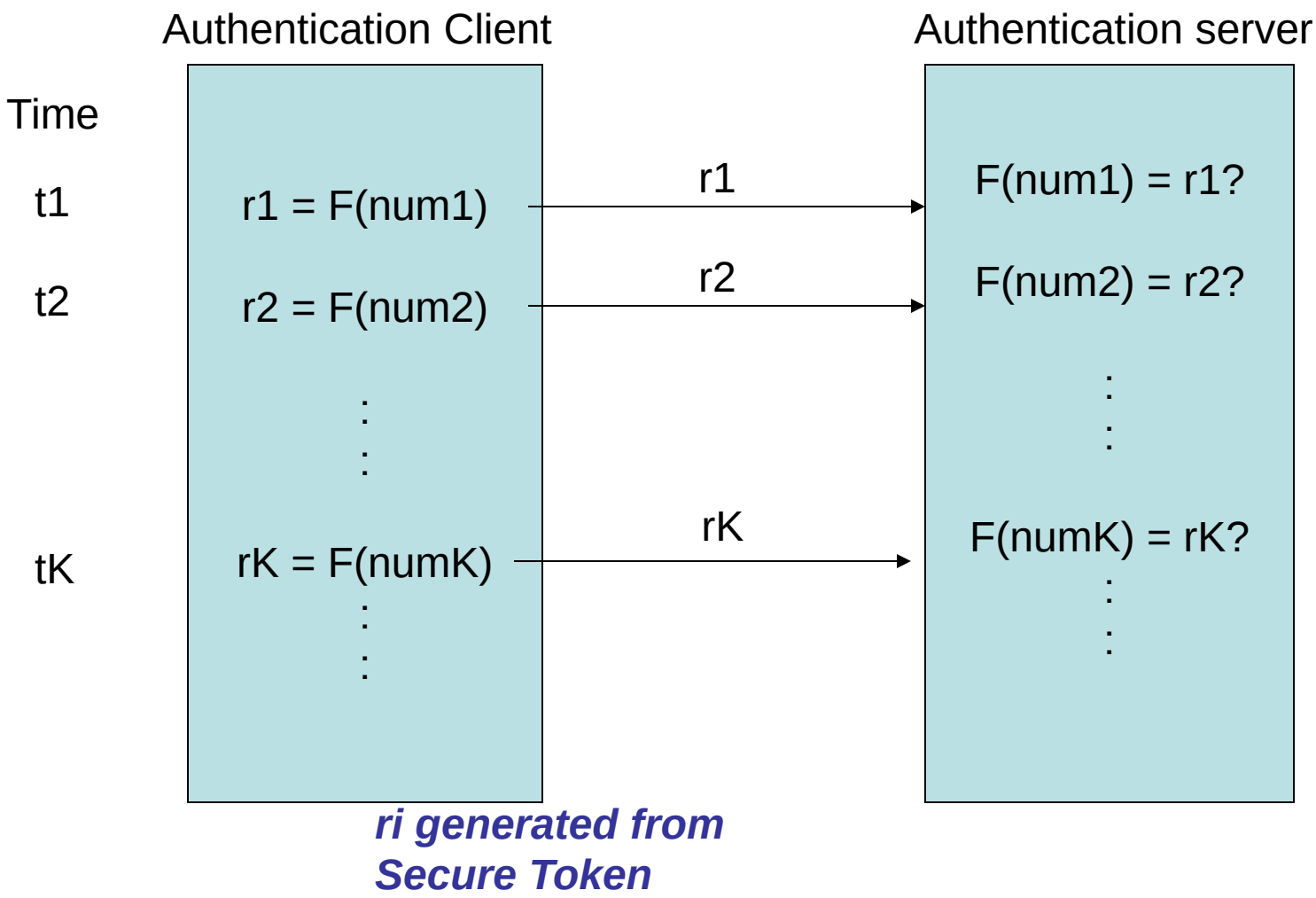
- Password authentication
  - user provides h = $H^{n-1}(W)$
  - server verifies if H(h) = $H^{n}(W)$
  - server stores $H^{n-1}(W)$
  - In general, user provides Hi(W), server verifies if $H(H^{i}(W)) = H^{i+1}(W)$
- Drawback: only limited passwords can be stored



RSA SecureID

- Use a secure token that generates a random number periodically (e.g., refreshed every 60 seconds)
- The clock inside the token is synchronized with the clock of the authentication server
  - Both the token and the authentication server are synchronized to generate the same random number at any point of time
- The battery of the token can last for years

RSA Secure ID



ri generated from Secure Token

References

- Linux Shadow Password HOWTO'
  - http://tldp.org/HOWTO/Shadow-Password-HOWTO.html
- Hacking Linux Exposed, 2nd edition, 2003, Chapter 9 "Linux Authentication"
  - You can access the electronic version of the book via CUHK library
- Steven Alexander, "Password Protection for Modern Operating Systems", The USENIX Magazine, 29(3), June 2004.
  - http://usenix.org/publications/login/2004-06/pdfs/alexander.pdf