Identification & Authentication

part 1

Stallings: *Chapter 3, 22*
Lidinsky: *Much Supplementary Material*

Identification

By what are you Identified?

What you **know**

Password, pass phrase, social security number, PIN, credit card number…

What you **have**

Physical key, credit card, driver's license, social security card…

What you **are**

Face, fingerprint, DNA, retinal scan…

Identification

Where does a physical signature fit into this classification?

Know? *Have?* *Are?*

How about a digital signature?

Authentication

Authentication

To positively verify the identity of a user, device, or other entity in a computer system, often as a prerequisite to allowing access to resources in a system.

Authentication can be:

Passwords

Message encryption that guarantees to the ultimate reader that the message is from the alleged sender and has not been compromised.

Related to non-repudiation and digital signatures

Authentication Overview

Passwords

- ***Overview, Strong secure passwords, password crackers***
- *S/Key*

PPP schemes

- *PAP, CHAP, EAP*

Third party authentication systems

- *Kerberos, TACACS+, RADIUS*

Mutual authentication

Digital certificates

Tokens

Biometric authentication

Multifactor authentication

Federated Identity Management

Authentication vs. Passwords

Some of the literature seems to equate *authentication* with *passwords & pass phrases*.

It’s not!

Passwords are one means of authentication.

Definitions
UserName and Password

Let’s focus now on user names and, especially, passwords

User Name

Unique character string used to identify an individual when logging onto a computer/network

Password

Secret combination of keystrokes that, when combined with a username, authenticates a user to a computer or network or web site or…

Passwords

Passwords

An age old problem

- *Users loose or forget passwords*
- *If passwords are too simple, they are easily determined by others*
- *If passwords are too complex, users can't remember them*
  - They end up on sticky notes on the monitor

Most Common Passwords

1. 123456
2. password
3. 12345678
4. qwerty
5. abc123
6. 123456789
7. 111111
8. 1234567
9. iloveyou
10. admin
11. 1234567890
12. letmein
13. photoshop
14. 1234
15. monkey
16. shadow
17. sunshine
18. 12345
19. password1
20. princess
21. azerty
22. trustno1
23. 000000

Passwords*
Other General Considerations

Length

Long ones are harder to crack and harder to remember

Types of characters (hopefully) allowed

Letters *Numbers* *Symbols*

Upper case *Lower case* *Case sensitive*

Non-displayable

Is change required

If so, how often and how repeatable

Required changes make remembering harder and result in the sticky notes on the monitors

Passwords
Length

Seven characters or more

- *Some password-cracking software products work in increments of seven characters*
  - We'll discuss this more later
- *While at first counter intuitive, seven characters may be better than nine characters*

Passwords

A Long Weak Password Is Still Weak

“1234abcde” is a weak password

A weak password is a weak password, no matter the length

e.g., *`qwertyuiopasdf`*   *`thisismypassword`*

But it’s still better not to push your luck with only a 5 or 6 character password

Passwords

Some Characteristics of Weak Passwords

All lower case letters

All upper case letters

All numbers

Yearly date in several standard formats

e.g.,	*111014*	*141011*	*10112014*	*410111*
	10Nov2014	*10nov14*	*14Nov10*	*41von01*
	11/10/14	*11-10-14*	*10-Nov. 2014*	*10-Nov 2014*

There are possibly 50 different date formats

50 x 365 = 18,250 different formats

Can exhaustively test all these in a few minutes

Passwords

Some Characteristics of Weak Passwords

Passwords that contain words, names, and phrases

e.g., mydogspot1 2billlid yksnidil3

tobeornottobe 2beornot2be Bill123

passworD1

Passwords

Some Characteristics of Weak Passwords

Passwords related to the login name

loginName	***Password***
lidinsky	*yksnidil24*
raymond	*ray123*

Mangled passwords

lidinsky	*L1d1nsky*
raymond	*raym0nd*
georgio	*ge0rg10*

Passwords

Some Characteristics of Strong Passwords

Case sensitive mix of letters, numbers, <u>and</u> <u>symbols</u>

No words, names, dates, or phrases

Seven or more characters with no hints

Examples

`1%RfEj&8<`

Nine mixed characters, but no hints

Possibly better yet: *`1%RfEj&ab`*

False hint in last two characters: all lower case letters

But these strong passwords are hard to remember

They end up on a Post-it note stuck to the monitor

Passwords

Easier to Remember Semi-strong Passwords

Phrases in mixed languages?

e.g., 1mykuttAspot2

1mycobakAspot2

But password dictionaries include multiple languages

Substitutions

e.g., 3beeOrKnot4bee

A strong version of *my dog spot*

my%c0baka>sp0t

Passwords

Frequent & Non-repeatable Changes

Frequently changed non-repeatable strong passwords are essentially an impossibility for the average person to remember

These, with certainty, will be written down somewhere

Password crackers such as *L0pht*, *LC5*, *RainbowCrack* and *John the Ripper* take advantage of the same things that make passwords easier to remember

Computer Passwords

Computer Passwords

Computer passwords are not stored on a computer

Instead tuples of *login names* and *hashes* of the passwords are stored

What happens when you log in using your *login name* and *password*?

- *The login function logs the name that you type and hashes the password that you type*
- *Compares your login name and your password hash with the tuple of your login name and password hash that it has stored*
- *If the two hashes agree, you are allowed access*

Unix and Linux Passwords

Unix Passwords

- Unix passwords are encrypted with the Unix *crypt()* function
  - *crypt() is a hash function based upon DES*
  - *It is <u>not</u> the Unix crypt utility*
    - Rather weak encryption
  - *crypt() really should have been called something else*
- *crypt()* hashes the plaintext password
  - *Uses one of several "salts"*
    - Salts are given as an argument

Old Unix Passwords

Old Unix (not Linux) passwords were stored in the ***/etc/passwd*** text file as part of the account information for each user

The account info format for each user looks like this:

loginName **:** passwordHash **:** uid **:** gid **:** userInfo **:** homeDirectory **:** loginProg

Examples

```
lidinsky:Qp47caKps8tN:123:123:Bill Lidinsky:/home/lidinsky:/bin/bash
student:$1$3MkY0$edjui.5DE7DYn/F8V25S:14076:0:99999:7:::
root:$l$w6jJr$7jsF0g7za9J//ul9Wul:14076:0:99999:7:::
```

Items are separated with a "**:**"

The password is hashed

A Security Problem

The ***/etc/passwd*** file must be readable globally because info in it is used by many Unix tools and applications

This presented a serious security problem

What to do?

Could have changed many many tools and applications

Better idea: Create an **/etc/shadow** *file*

Current Unix & Linux Passwords

Similar to old Unix but

The password entry in ***/etc/passwd*** is replaced with an "x"

The hashed password is put into ***/etc/shadow*** file

So in /etc/passwd we have

loginName **:** **x** **:** uid **:** gid **:** userInfo **:** homeDirectory **:** loginProg

Examples of */etc/passwd* accounts

```
student:x:1001:1001:student:/home/student:/bin/bash
root:x:0:0:root:/root:/bin/bash
```

Current Unix & Linux Passwords

In the ***/etc/shadow*** file there is for each login name the following line format:

loginName **:** passwordHash **:** *#days_since_last_changed* **:** *#days_before_may_be_changed* **:** *#days_after_which_must_be_changed* **:** *#days_to_warn_user_of_expiring* **:** *#days_after_expired_account-disabled* **:** *#days_since_account_disabled* **:** *reserved_filed*

/etc/shadow examples

```
lidinsky:Qp47caKps8tN:723:0:99999:7:::
student:$1$3MkY0$edjui.5DE7DYn/F8V25S:14076:0:99999:7:::
root:$l$w6jJr$7jsF0g7za9J//ul9Wul:14076:0:99999:7:::
```

Linux Passwords

Why put the hashed password in the */etc/shadow* file?

Because, unlike */etc/passwd*, it has restricted access

Must be **su** *to access it*

Windows Passwords

Windows Passwords

Windows OSs may hash/encrypt a password twice using two different algorithms

- *LM or LMHash*
- *NT or NTHash*

In Vista & WinServer 2008 and beyond LM is disabled by default

- *But can be enabled*

LM password algorithm is very weak

LM **Windows Passwords**

LM allows alphanumeric characters and symbols (👍)

LM converts all password plaintext characters to upper case before hashing (👎)

Pads out plaintext passwords to 14 bytes (👎)

Divides modified plaintext passwords into two 7-byte parts (👎)

Each part is used to create a 64-bit DES key (👎)

Each DES key is used to encrypt the ASCII string ***KGS!@#$%*** resulting in two 8-byte cyphertext values (👎)

Concatenates the two values to form the LM hash (👎)

Notes

Use a fixed salt across all Windows OSs (👎)

Number of password characters must be ≤ 14

LM **Windows Passwords**

Passwords ≤ 14 characters or bytes are padded with nulls

Example:

Bill *becomes* **BILL000 00000000** *before hashing*

BILL000 *and* **00000000** *are then hashed separately*

To each 7-byte part an odd parity byte is added

This 8-byte value becomes a DES key

DES is used with a fixed salt to create each 8-byte hash

The two hashes are concatenated together create the 16-byte LM hash

NT Hash **Windows Passwords**

NT hash allows alphanumeric characters & symbols (👍)

Characters and symbols are represented as Unicode

Keeps both lower and upper case characters (👍)

Does not split up the plaintext password (👍)

But still uses a fixed salt (👎)

Creates a 16-byte hash using MD4 hashing algorithm

Windows Passwords

From Win2K forward the two hashes are both calculated and stored in the SAM (Security and Account Manager) part of the registry

There are ways to eliminate the LM hash in older OSs

But then backward compatibility is compromised

Windows stores encrypted password hashes in the Windows Registry

Unlike Linux, password files don't exist in Windows

Need special tools to extract passwords and other account information

e.g., pwdump, pwdump2, pwdump3, samdump…

No Salt Implications

The same plaintext password will yield the same hash on all Windows OSs

A password cracker can therefore create a large lookup table and break passwords on any Windows computer

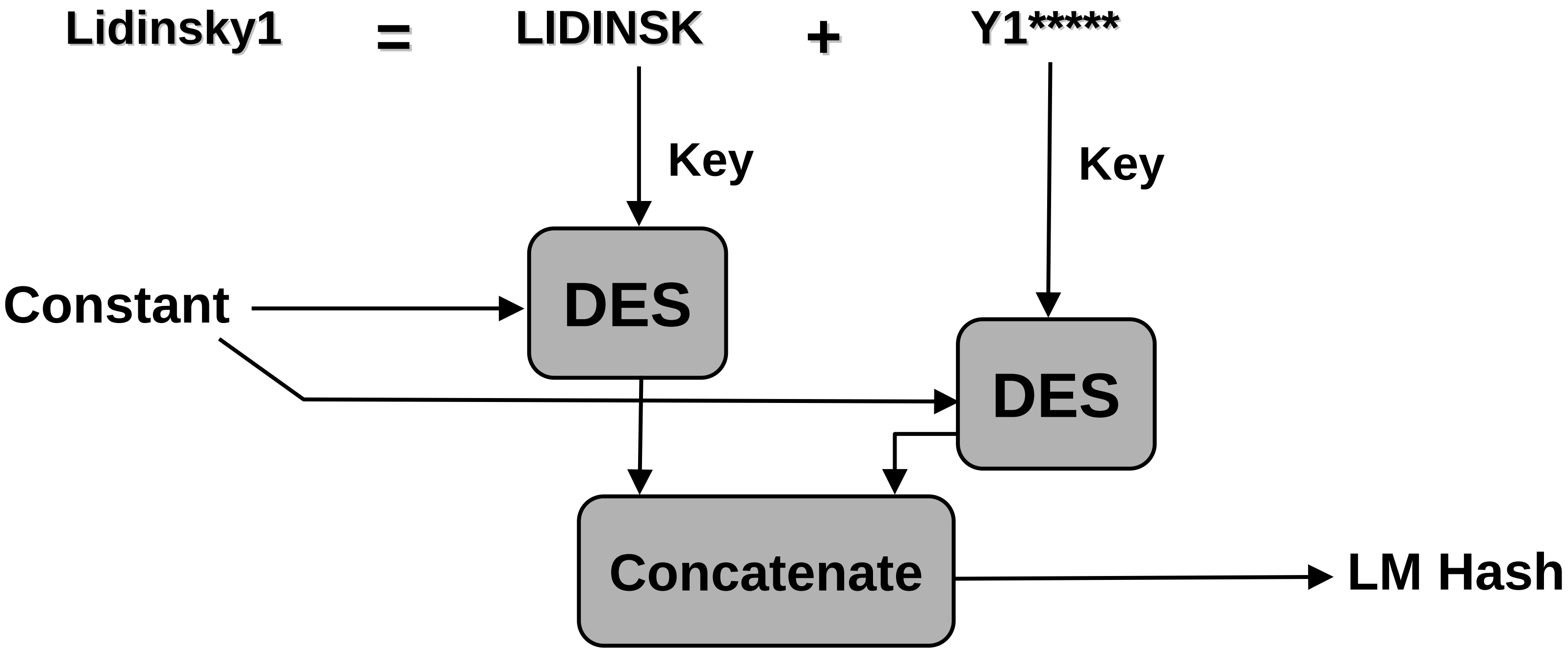
- *Lookup table consists of tuples for all possible character strings*
  - Character string Hash
- *Originally proposed by Ron Rivest in 1980s*
- *Improved dramatically by Philippe Oechslin in 2002*
  - Called "RainbowCrack" as a general term

LM Hash Generation

Converted to upper case

Padded with NULL to 14 characters

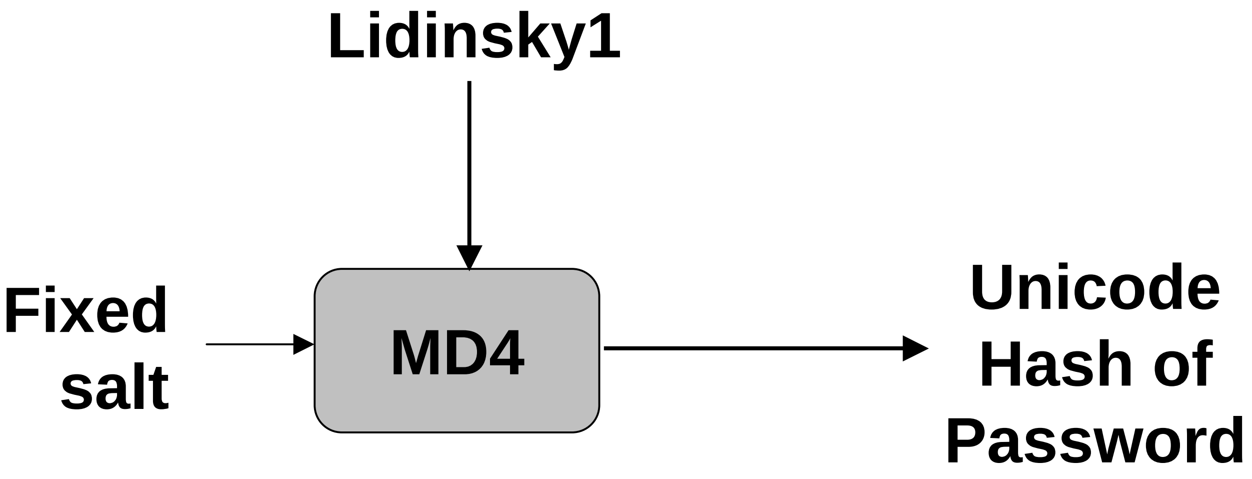
Separated into two 7 character strings



NT Hash Generation

Just hash the password

Then store it



Password Cracking

Taxonomy of Password Cracking Schemes

Intelligent guessing based upon side information

- *Conventional*
- *Mangled*

Dictionary

- *Conventional*
- *Mangled dictionary*

Precalculated

- *Exhaustive (e.g., Rivest)*
- *Rainbow (e.g., Oechslin)*

Brute force

Intelligent Guessing

Conventional

- *Use plaintext account information*
- *e.g., login name, full user name*

Mangled

- *Modified versions of names*
- *e.g., Combinations of upper and lower case characters*
  - *Combinations of names and numbers*
  - *Combinations of names, numbers and other symbols*

Intelligent Guessing approach is often tried first

- *Can be fast for poor passwords*
- *But is non-deterministic*

Intelligent Guessing
Conventional

Use plaintext account information

login name, full user name

Examples

Bill Lidinsky	*Guess: bill123 or wpl123*
lidinsky	*Guess: yksnidil*

Intelligent Guessing
Mangled

Modified versions of names

- *Combinations of upper and lower case characters*
- *Combinations of names and numbers*
- *Combinations of names, numbers and other symbols*

Examples

Bill Lidinsky	*Guess: Bill123 or B1ll123 or B!ll123* *Bill_123 or Bill 123*
lidinsky	*Guess: yksn1d1L*
yalline	*Guess: ya!!1n3 or zbmmj0f*

Dictionary

Conventional

- *Use dictionary table directly*
- *e.g., 100s of first names, dog names*
  - *All dates of a year in 20-40 different formats*

Mangled

- *Modified versions of dictionary entries*
- *e.g., first names with varying locations of upper and lower case letters*
- *e.g., first names with numbers added or inserted*
- *e.g., combinations of names, dates, numbers & symbols*

Dictionary

Conventional dictionary attacks can be reasonably fast if the password is in the dictionary

- *Non-deterministic*

Mangled dictionary attacks can take a lot of time

- *Days*
- *Non-deterministic*

Exhaustive Precalculation
Rivest Style Approach

Precalculates all the plaintext/hash tuples

Stores the tuples in a ordered way in a table

Using the target hash, find the one that matches in the table

Reduced the time needed to crack LM-type passwords over mangled dictionary approach

But can still take days -- just fewer days

Deterministic

Rainbow Precalculation
Oechslin Style Approach

Based on time/memory tradeoffs

- *Don't include all the precalculated tuples*
- *Let computational algorithm interpolate*

Stores selected tuples in a special way in a ***rainbow table***

Reduces substantially

- *The size of the table*
- *The time to search (because the table is smaller)*

Increases the computation needed for each hash try

Net effect:

- *Greater than factor of 2 improvement over exhaustive precalculation*

Rainbow Precalculation
Oechslin Style Approach

Deterministic

Claim: Can crack over 99.9% of all LM alphanumeric passwords in about 15 seconds using 1.4 GByte rainbow table

NT hash-type hashes

Maybe an hour or several hours

Unix/Linux hashes (salted)

Perhaps hours

Brute Force

Exhaustively try all passwords, given a specific character set

Takes a long time

Days to a week

Minimal memory required

Computationally the most intensive

Deterministic

Password Cracking Tools

Some Password Crackers

Today there are many password crackers available

- *Some are free*
- *Some work well for applications but not for OS passwords*

John the Ripper (free)

- *Intelligent guess, dictionary, mangling and brute force*
- *One of the few that can be used directly on Linux passwords*
- *Runs on Linux or Windows*
- *Needs additional word lists to be really effective against alphanumeric+symbol and multiple language passwords*

Some Password Crackers

EPRB (Elcomsoft Password Recovery Bundle) ($$$)

Used primarily for passwords on applications

RainbowCrack (free)

Based upon rainbow tables

Needs additional word lists to be really effective against alphanumeric+symbol and multiple language passwords

LC5 ($$$, @stake)

Intelligent guessing + extensive dictionary

Acquired by Symantec. Not sure of status

Some Password Crackers

OphCrack (free)

- *Live-CD with small rainbow tables and algorithm built in*
- *Also versions that can be installed in Windows and Linux*

PRTK (Password Recovery ToolKit) (AccessData, $$$)

- *Intelligent guessing + Multi-language dictionaries*
- *Mangling*
- *Used primarily for passwords on applications*
- *Part of software bundle for forensic analysis*

Some Password Crackers

Cain and Able (free)

- *Really a multipurpose system of tools*
- *Microsoft OSs*
- *Password cracking: dictionary & brute force*
- *Network sniffing for passwords*
- *Recording VoIP*
- *Wireless network key discovery*
- *Routing protocol analysis*
- *Some other stuff*
- *Additional word lists are needed*

Assign10a

None!

But there is an assignment at the end of the related lab.