

Authentication

February 3, 2012

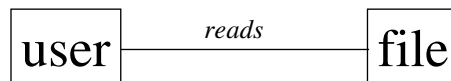
Administrative – submittal instructions

- answer the lab assignment's questions in written report form, as a text, pdf, or Word document file (no obscure formats please)
- email to csci530l@usc.edu
- exact subject title must be "authenticationlab"
- deadline is start of your lab session the following week
- reports not accepted (zero for lab) if
 - late
 - you did not attend the lab (except DEN or prior arrangement)
 - email subject title deviates

Authentication

- verifying identity of a user
 - example: logging into a system
 - example: GPG – digital signature is the authentication mechanism
- that user's ID gets “embedded” in his shell/interface process
- authentication != authorization
 - authorization establishes what user can do once authenticated

Bigger picture - how we think of it



Bigger picture - how it actually works

users don't read files, processes do



↓ program that copies one file to another

```
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
int main()
{
    char c;
    int in, out;

    in = open("file.in", O_RDONLY);
    out = open("file.out", O_WRONLY|O_CREAT, S_IRUSR|S_IWUSR);

    while(read(in,&c,1) == 1)
        write(out,&c,1);

    exit(0);
}
```

note system calls "open" "read" "write"
They do the file access
user? *isn't even mentioned* in the calls

Bigger picture - how it actually works

AUTHENTICATION HERE
up front, determines account
for first (shell) process

same account, carried forward by inheritance
from shell process to this spawned one



```
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
int main()
{
    char c;
    int in, out;

    in = open("file.in", O_RDONLY);
    out = open("file.out", O_WRONLY|O_CREAT, S_IRUSR|S_IWUSR);

    while(read(in,&c,1) == 1)
        write(out,&c,1);

    exit(0);
}
```

note system calls "open" "read" "write"
They do the file access
user? *isn't even mentioned* in the calls

Bases for authentication

- something people know
 - password
- something they have
 - smart card
 - sim (subscriber identity module) card
 - hardware token
- something about them
 - retina/iris
 - fingerprint
 - DNA
 - voice
 - ear
 - face
- somewhere they are
 - login only works at certain terminals

Extent of authentication

- one or a combination of methods may be used
 - depending on needed degree of protection
 - “single-factor” “multi-factor”
- examples
 - system login
 - for user account, the matching password (single-factor)
 - system with fingerprint reader (e.g. modern laptop)
 - for user account, the matching password *and* finger (2-factor)
 - ATM transaction
 - for bank account, the card *and* matching pin (2-factor)

How can authentication be broken?

- passwords
 - can be guessed/cracked
- smartcards
 - can be copied or stolen
- fingerprints
 - can be copied using scotch tape

Example: passwords

- something people know
- most common, familiar basis for authentication

What's makes bad ones?

- only letters or only numbers (hackme, 09112002)
- recognizable words (john1, R2D2)
- foreign language words (bonjour1, hastalavistababy)
- hacker terminology (H4XOR, 1337)
- personal info (names, birthdates, addresses)
- reverse words (nauj, esrever)

What's makes good ones?

- at least 8 characters (conventionally, but why 8??)
- mixed case
- mixed letters and numerals
- punctuation/non-alphanumeric symbols included
- something you can remember

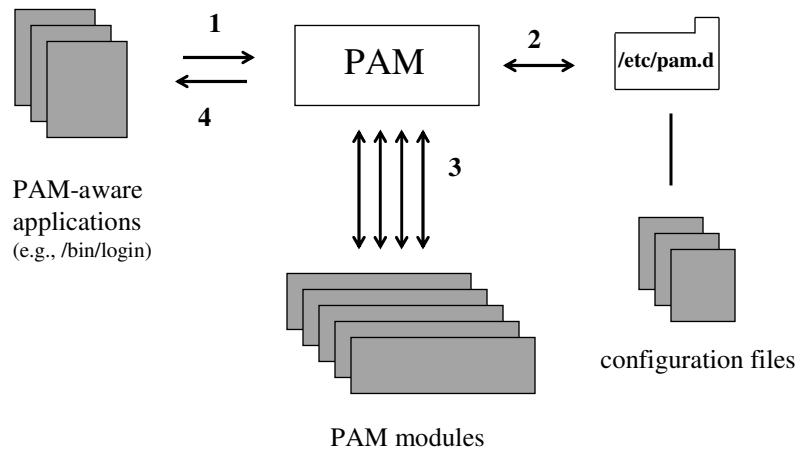
Making a good one

- think of a memorable phrase
 - “wasn’t that a dainty dish to set before the king”
 - “in the beginning god created the heavens and the earth”
- make it an acronym
 - wtaddtsbtk
 - itbgcthate
- substitute non-letters for letters
 - w7@dd7\$b7k
 - i7bgcth@te
- capitalize something
 - w7@DD7\$b7k
 - i7BGCTH@te

Forcing strong passwords

- create them for users, don’t let users choose them
- use PAM’s pam_cracklib module
- or other PAM password evaluation modules
 - pam_passwdqc (<http://www.openwall.com/passwdqc>)

PAM architecture



Operation sequence

- app calls PAM (1)
- PAM reads app's PAM config file (2)
- PAM calls PAM modules as listed in the file (3)
 - each succeeds or fails independently
- PAM itself succeeds or fails, depending on the modules' outcomes (4)
 - returns its overall outcome to app
- app proceeds (if success) or terminates (if failure)

Password crackers

- John the Ripper (<http://www.openwall.com/>)
- Cain and Abel (<http://www.oxid.it/cain.html>)
- Slurpie (<http://www.ussrback.com/distributed.htm>)
- Aircrack (WEP-specific) (<http://www.aircrack-ng.org/>)

Is guesswork easy or hard?

- an alphabet is a set of symbols
- how many words of a certain length can you compose from an alphabet?
- depends on
 - the number of symbols in the alphabet
 - the particular wordlength

Password strength determinants

- the number of possible characters it contains
 - its character set
- its length
 - its character count

Is guesswork easy or hard?

- | | |
|---|--|
| ● a 2-symbol alphabet has <ul style="list-style-type: none">– 2 one-letter words– 4 two-letter words– 8 three-letter words | ● a 10-symbol alphabet has <ul style="list-style-type: none">– 10 one-letter words– 100 two-letter words– 1000 three-letter words |
| ● a 3-symbol alphabet has <ul style="list-style-type: none">– 3 one-letter words– 9 two-letter words– 27 three-letter words | ● an α -symbol alphabet has <ul style="list-style-type: none">– α one-letter words– α^2 two-letter words– α^3 three-letter words– α^p p-letter words
(alphabet length raised to password length) |

How many words are there?

- 26-letter alphabet, wordlength 3: $26^3 = 17576$
- 52-letter alphabet, wordlength 3: $52^3 = 140,608$
 - double alphabet length yields 8 times as many words
- 26-letter alphabet, wordlength 6: $26^6 = 308,915,776$
 - double word length yields 17576 times as many words
- 52-letter alphabet, wordlength 6: $52^6 = 19,770,609,660$
- a fish is in a pond – harder to catch in a bigger pond

How many words are there? *

Perfect Passwords
GRC's Ultra High Security Password Generator

3,127 sets of passwords generated per day
4,595,744 sets of passwords generated for our visitors

Generating long, high-quality random passwords is not simple. So here is some totally random raw material, generated just for YOU, to start with.

Every time this page is displayed, our server generates a unique set of custom, high quality, cryptographic-strength password strings which are safe for you to use:

Character Set	Length	Password Space
64 random hexadecimal characters (0-9 and A-F)	16-character alphabet	$16^{64} = 10^{77}$
63 random printable ASCII characters	94-character alphabet	$94^{63} = 10^{124}$
62 random alphanumeric characters (a-z, A-Z, 0-9)	62-character alphabet	$62^{63} = 10^{112}$

comparatively, IPv6 address space 10^{38}

<https://www.grc.com/passwords.htm>

* i.e. "What's the password space?"

Password strength determinants

- the number of possible characters it contains
- its length
- the randomness of character selection



a 3rd criterion!! human-dependent!

How you can help the cracker: -- shrink the pond!

- the cracker's task:
 - finding the password = eliminating all the other candidates
- how you can do part of the cracker's task for him
 - eliminate candidates, reduce the candidate-space
- how you can eliminate candidates:
 - use only letters, or only numbers (eliminates using both)
 - use recognizable words, foreign words (eliminates non-words)
 - use personal info ...
 - use hacker terminology ...
 - use reverse words ...
- better yet: send cracker your password on a postcard
 - above suggestions for use only if have no stamps!

Stated differently

- cracking is process of eliminating the non-passwords
- by confining passwords to known (predictable) patterns users pre-eliminate huge parts of the password space

“For instance, the cracking programs rely on the fact that a typical user will probably not start a password with a special character in the first position but will nearly always put it somewhere near the end of the space-- therefore you can shave enormous amounts of cracking time with a cracking program that is written to contemplate this - so that it will not start a brute force guessing attack on a password that assumes a special character is in the first position....

“In essence, these cracking programs go through a protocol of routines or hacking steps from greatest reward/least effort to finally greatest effort/least reward (i.e. purely random) hoping to get lucky and snag an answer before they have to go through the entire keyspace.”

BFTCalc.xls from <http://www.mandylionlabs.com/PRCCalc/BruteForceCalc.htm>

Password cracking methods

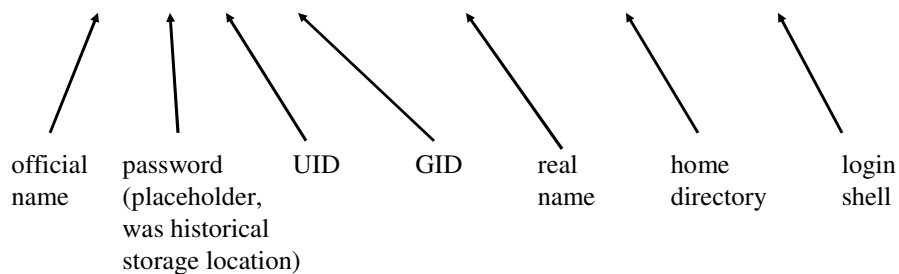
- dictionary attack
- hybrid attack
- brute force attack
- can either be applied against a live system in place, or a file containing the passwords offline

Where unix stores passwords

- historical
 - /etc/passwd –list of users & their hashed passwords
 - was readable by any user (rw-r--r--)
- current
 - /etc/passwd –list of users only (no passwords)
 - is readable by any user (rw-r--r--)
 - /etc/shadow –their hashed passwords
 - is readable by root user only (rw-----)

/etc/passwd entries hold user information

craig:x:507:507:Craig Smith:/home/craig:/bin/bash



/etc/shadow entries hold ancillary user information

craig:\$1\$2YL52jhL\$:11992:60:75:3:14:12417:134550548

↑ reserved
↑ user name
↑ hashed password
↑ other, unrelated items

Newer distributions, longer hashes

```
fedora7 VMware Remote Console - Devices -
[root@localhost ~]# cat /etc/shadow | tail -3
user1:$1$uDHmvoDF$ijrh0EGdUosPsdutIaJjh/:14869:0:99999:7:::
user2:$1$uIZb5mLw$1jcpZ5CNAKpCRUqCx45lL1:14869:0:99999:7:::
user3:$1$7oqt7kYc$kiE1duCa6I0RRXR51Rp7// 14869:0:99999:7:::
[root@localhost ~]# _

fedora13 VMware Remote Console - Devices -
[root@localhost etc]# cat /etc/shadow | tail -3
user1:$6$UroqpB2d$iKFEHmk0Pp0Y0diegLvZGBb26nYDF6phi50I0vUwvWM1R0tkgJbWqyZFzZwo
lMw/CQnnapT60uZZR0rB/9L/:14869:0:99999:7:::
user2:$6$anFc7SRQ$LH5DuZ9irQmTrsrFCqpQFC2gvbQSYkB/UMmwLSW4iqbUX.Cv..UB3B2igDy7uk
0PBTYC14GCaX6ou2ih46wB20:14869:0:99999:7:::
user3:$6$YQ4B9XTU$MkZ_9qeN6_juP0W056UhaL6Wwv7JTUD3RwfSHwnt0p/NV91zYK38qm.RUzNzEHZ
6_1vH6DoUo3UP1HbRmqRadW1:14869:0:99999:7:::
[root@localhost etc]# _
```

Dictionary attack

- *generated/dynamic* dictionary attack
 - list of dictionary words that are tried (i.e., hashed and compared to hash of actual password)
 - very quick
- *file-based/static* dictionary attack
 - *pre*-compute hashes of all the words in the dictionary
 - re-sort it, on the hashes instead of the words
 - password crack becomes a computationless hash lookup
- if the password is not an exact match to a word on the list, crack will fail

Hybrid attack

- uses a dictionary list but can detect slight variations to words, or combinations of words.
- example: if the word *hello* is in the database, but the password is *Hello*, a dictionary attack will not break the password, but a hybrid attack will
- generally finds many more words than a dictionary attack
- not as quick as dictionary attack

Bruteforce attack

- will try every character combination until it finds the password
- slow in proportion to password space
- will always find the password, given time

Bruteforce attack time estimator

The screenshot shows an Excel spreadsheet with the following data and formulas:

Character Set	Size	Entropy or Keyspace of password
Upper Case Letters	26	1
Lower Case Letters	26	308,915,776
Numbers	10	1
Special Characters	32	1
or Purely Random Combo of Alpha/Numeric	62	1
or PURELY Random Combo of Alpha/Numeric/Special	94	1
PHRASE or WORD SUBJECT TO A DICTIONARY ATTACK	5	1
password length in Characters	8	308,915,776 or 309,915,776

Other key values from the spreadsheet:

- Website: <http://www.mandylionlabs.com/index15.htm>
- 309 million combinations
- Total Workload in Floating Point Processes: 154,457,888
- Number of KV's a Desktop Computer Can Try efficiently in an Hour (2"2"33): 17,073,883,184
- Estimated Gross Number of hours to Crack: 0.01 hours
- On Distributed Level: 0.00 days

<http://www.mandylionlabs.com/PRCCalc/BruteForceCalc.htm>

Recent dynamic user/pass guessing attack on my system

- noted in /var/log/btmp and /var/log/secure
- About 700000 login attempts
- from about 2100 remote IPs
- from several to 53000 login attempts each
- using about 26000 user name guesses
- with unknown password guesses

Recent dynamic attack

```
root@unexgate:~/temp
Jan 23 04:06:43 unexgate sshd[26850]: Failed password for illegal user support from ::ffff:1.234.20.68 port 59697 ssh2
Jan 23 04:06:46 unexgate sshd[26852]: Failed password for illegal user support from ::ffff:1.234.20.68 port 60895 ssh2
Jan 23 04:06:50 unexgate sshd[26853]: Failed password for illegal user support from ::ffff:1.234.20.68 port 33733 ssh2
Jan 23 04:06:54 unexgate sshd[26857]: Failed password for illegal user support from ::ffff:1.234.20.68 port 34911 ssh2
Jan 23 04:06:57 unexgate sshd[26859]: Failed password for illegal user support from ::ffff:1.234.20.68 port 36043 ssh2
Jan 23 04:07:01 unexgate sshd[26861]: Failed password for illegal user support from ::ffff:1.234.20.68 port 37078 ssh2
Jan 23 04:07:04 unexgate sshd[26864]: Failed password for illegal user support from ::ffff:1.234.20.68 port 38273 ssh2
Jan 23 04:07:08 unexgate sshd[26866]: Failed password for illegal user patrick from ::ffff:1.234.20.68 port 39296 ssh2
Jan 23 04:07:11 unexgate sshd[26868]: Failed password for illegal user patrick from ::ffff:1.234.20.68 port 40378 ssh2
Jan 23 04:07:15 unexgate sshd[26870]: Failed password for illegal user patrick from ::ffff:1.234.20.68 port 41351 ssh2
Jan 23 04:07:18 unexgate sshd[26872]: Failed password for illegal user patrick from ::ffff:1.234.20.68 port 42597 ssh2
Jan 23 04:07:22 unexgate sshd[26875]: Failed password for illegal user patrick from ::ffff:1.234.20.68 port 43683 ssh2
Jan 23 04:07:25 unexgate sshd[26877]: Failed password for illegal user patrick from ::ffff:1.234.20.68 port 44635 ssh2
Jan 23 04:07:29 unexgate sshd[26879]: Failed password for illegal user patrick from ::ffff:1.234.20.68 port 45826 ssh2
Jan 23 04:07:32 unexgate sshd[26881]: Failed password for illegal user patrick from ::ffff:1.234.20.68 port 46998 ssh2
Jan 23 04:07:36 unexgate sshd[26884]: Failed password for illegal user patrick from ::ffff:1.234.20.68 port 47998 ssh2
Jan 23 04:07:40 unexgate sshd[26886]: Failed password for illegal user patrick from ::ffff:1.234.20.68 port 49251 ssh2
Jan 23 04:07:44 unexgate sshd[26888]: Failed password for illegal user patrick from ::ffff:1.234.20.68 port 50376 ssh2
Jan 23 04:07:48 unexgate sshd[26890]: Failed password for illegal user patrick from ::ffff:1.234.20.68 port 51622 ssh2
Jan 23 04:07:51 unexgate sshd[26893]: Failed password for illegal user patrick from ::ffff:1.234.20.68 port 52846 ssh2
Jan 23 04:07:55 unexgate sshd[26895]: Failed password for illegal user patrick from ::ffff:1.234.20.68 port 53796 ssh2
Jan 23 04:07:58 unexgate sshd[26897]: Failed password for illegal user mtke from ::ffff:1.234.20.68 port 55002 ssh2
Jan 23 04:08:02 unexgate sshd[26899]: Failed password for illegal user mtke from ::ffff:1.234.20.68 port 56081 ssh2
Jan 23 04:08:06 unexgate sshd[26902]: Failed password for illegal user mtke from ::ffff:1.234.20.68 port 57375 ssh2
Jan 23 04:08:09 unexgate sshd[26904]: Failed password for illegal user mtke from ::ffff:1.234.20.68 port 58537 ssh2
Jan 23 04:08:14 unexgate sshd[26906]: Failed password for illegal user mtke from ::ffff:1.234.20.68 port 59618 ssh2
Jan 23 04:08:17 unexgate sshd[26908]: Failed password for illegal user mtke from ::ffff:1.234.20.68 port 60886 ssh2
Jan 23 04:08:21 unexgate sshd[26911]: Failed password for illegal user mtke from ::ffff:1.234.20.68 port 33686 ssh2
Jan 23 04:08:25 unexgate sshd[26913]: Failed password for illegal user mtke from ::ffff:1.234.20.68 port 34951 ssh2
Jan 23 04:08:28 unexgate sshd[26915]: Failed password for illegal user mtke from ::ffff:1.234.20.68 port 36105 ssh2
Jan 23 04:08:32 unexgate sshd[26917]: Failed password for illegal user mtke from ::ffff:1.234.20.68 port 37121 ssh2
Jan 23 04:08:35 unexgate sshd[26920]: Failed password for illegal user mtke from ::ffff:1.234.20.68 port 38313 ssh2
Jan 23 04:08:39 unexgate sshd[26922]: Failed password for illegal user mtke from ::ffff:1.234.20.68 port 39404 ssh2
Jan 23 04:08:43 unexgate sshd[26924]: Failed password for illegal user mtke from ::ffff:1.234.20.68 port 40607 ssh2
Jan 23 04:08:47 unexgate sshd[26926]: Failed password for illegal user mtke from ::ffff:1.234.20.68 port 41863 ssh2
Jan 23 04:08:50 unexgate sshd[26929]: Failed password for illegal user richard from ::ffff:1.234.20.68 port 43014 ssh2
Jan 23 04:08:54 unexgate sshd[26931]: Failed password for illegal user richard from ::ffff:1.234.20.68 port 44199 ssh2
Jan 23 14:03:54 unexgate sshd[29771]: Failed password for selim from ::ffff:76.169.60.103 port 51103 ssh2
Jan 23 14:04:08 unexgate sshd[29771]: Failed password for selim from ::ffff:76.169.60.103 port 51103 ssh2
Jan 23 19:15:13 unexgate sshd[31447]: Failed password for cracJun from ::ffff:10.47.90.251 port 33288 ssh2
Jan 23 19:20:51 unexgate sshd[31565]: Failed password for minevich from ::ffff:10.47.90.246 port 33595 ssh2
Jan 23 19:47:47 unexgate sshd[32724]: Failed password for washsht from ::ffff:10.47.90.230 port 41897 ssh2
Jan 24 07:10:37 unexgate sshd[27271]: Failed password for root from ::ffff:88.190.34.3 port 41621 ssh2
Jan 24 07:10:40 unexgate sshd[27291]: Failed password for root from ::ffff:88.190.34.3 port 42875 ssh2
Jan 24 07:10:44 unexgate sshd[27311]: Failed password for root from ::ffff:88.190.34.3 port 44239 ssh2
Jan 24 07:10:47 unexgate sshd[27331]: Failed password for bin from ::ffff:88.190.34.3 port 45672 ssh2
Jan 24 07:10:51 unexgate sshd[27351]: Failed password for bin from ::ffff:88.190.34.3 port 46007 ssh2
Jan 24 09:23:24 unexgate sshd[32711]: Failed password for root from ::ffff:218.108.0.68 port 25697 ssh2
Jan 24 09:23:32 unexgate sshd[32741]: Failed password for root from ::ffff:218.108.0.68 port 26014 ssh2
Jan 24 09:23:38 unexgate sshd[32761]: Failed password for root from ::ffff:218.108.0.68 port 26458 ssh2
Jan 24 09:23:44 unexgate sshd[32781]: Failed password for root from ::ffff:218.108.0.68 port 26821 ssh2
Jan 24 09:23:51 unexgate sshd[32811]: Failed password for root from ::ffff:218.108.0.68 port 27195 ssh2
Jan 24 09:23:57 unexgate sshd[32841]: Failed password for root from ::ffff:218.108.0.68 port 27564 ssh2
```

Defending against someone trying to break into a system

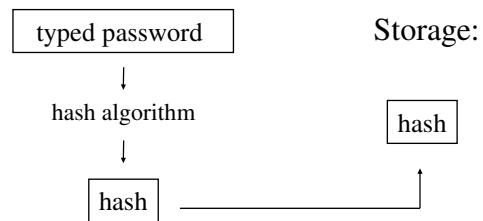
- auto-logout
 - if the user enters the wrong password n times, disable their account for a certain period
- protect the password list on your system
 - make sure the administrator has access and no one else, so a normal user cannot copy it onto another system
- enforce periodic changes (e.g. 90 days)
 - puts narrow time window requirement on the crack

Deliberate obstacles to cracking

- extra hash algorithm code to slow them down
 - diminish brute force attempts/time
- password salting
 - render static dictionary attack infeasible

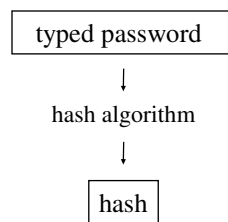
unsalted password: generating

Generate time

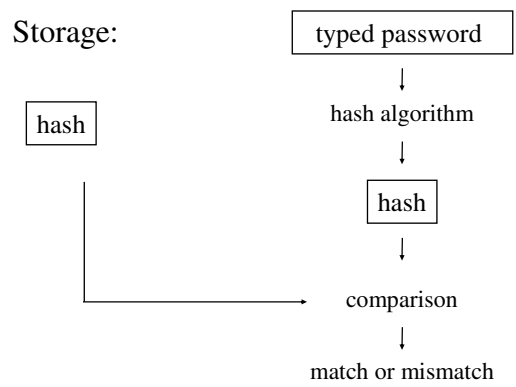


unsalted password: authenticating

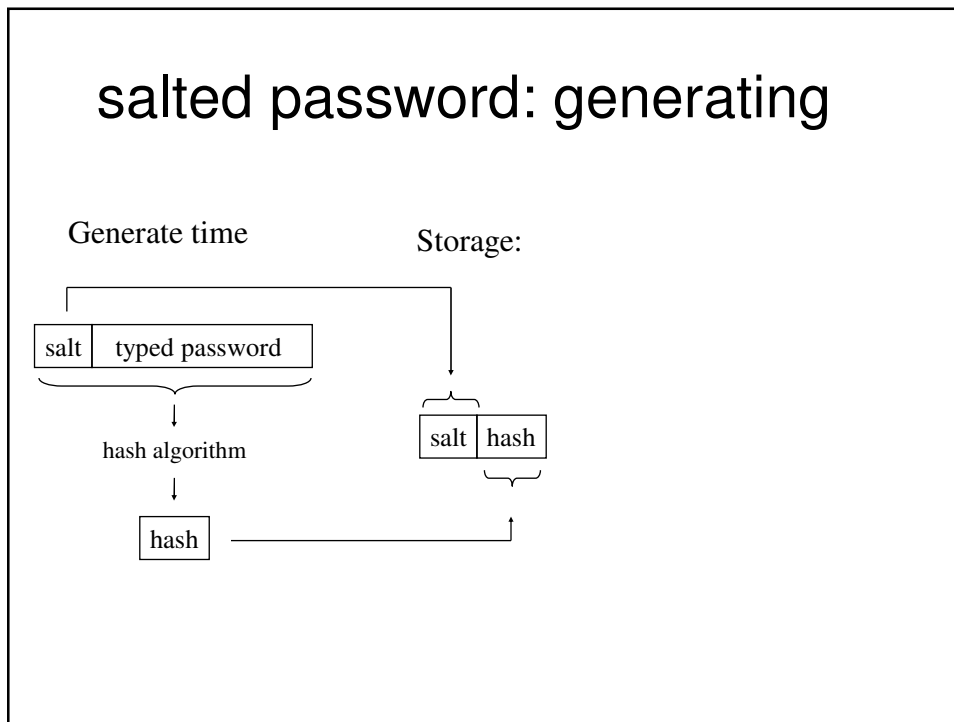
Generate time



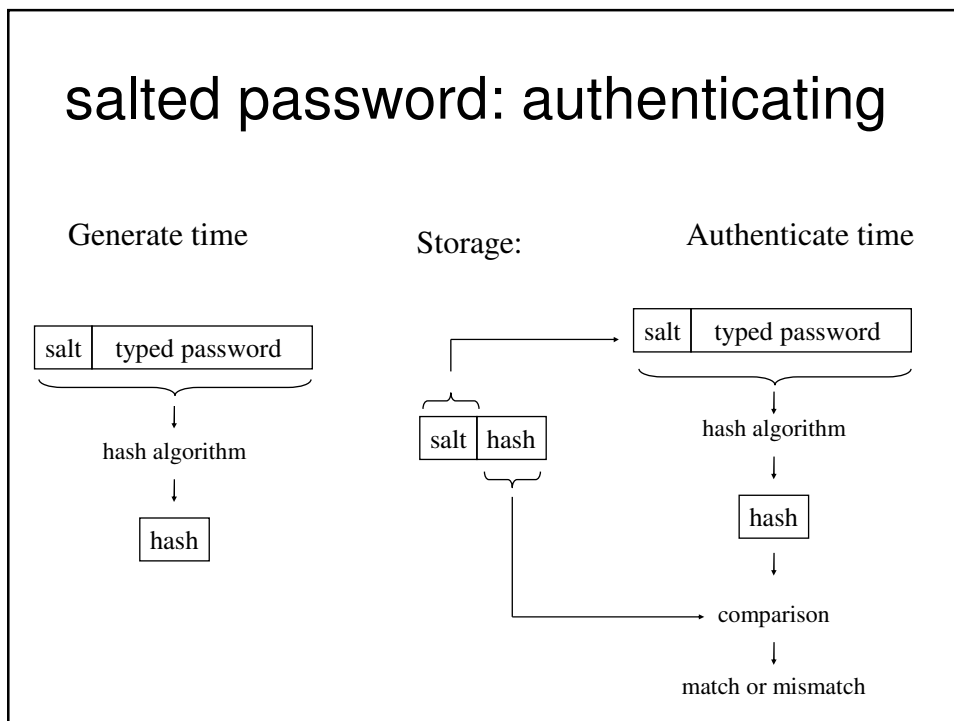
Authenticate time



salted password: generating



salted password: authenticating



Benefit of salt

- a given password is no longer deterministically mapped to a particular hash
 - greatly raises cost of file-based/static dictionary attack
 - identical passwords not detectable by their hashes

Hardware authentication tokens

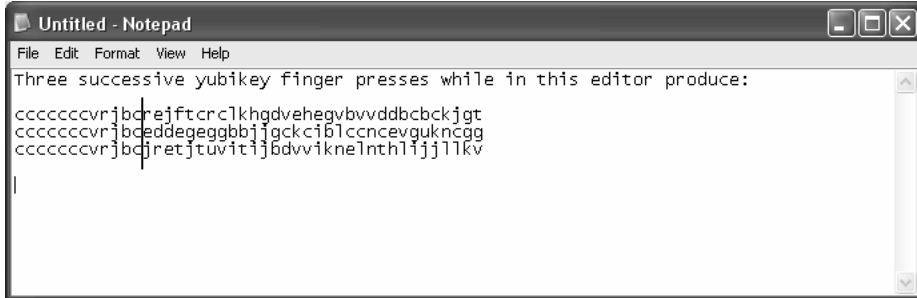
Yubico YubiKey



RSA SecurID



Pressing yubikey, in Notepad



yubikey is a USB keyboard device – it types 44 letters whenever pressed

right 32 letters one-time password, generated *this* time

left 12 letters invariant public ID of this yubikey, generated *every* time

Good play, bad replay

```
[root@localhost ~]# date +%H:%M:%S;ykclient id ccccccvrjbcitndbjrlrljdfjtrnbhngrbegkglv
07:50:44
Input:
client id: 16
token: ccccccvrjbcitndbjrlrljdfjtrnbhngrbegkglv
Verification output (0): Success
[root@localhost ~]#
[root@localhost ~]# date +%H:%M:%S;ykclient id ccccccvrjbcitndbjrlrljdfjtrnbhngrbegkglv
07:50:46
Input:
client id: 16
token: ccccccvrjbcitndbjrlrljdfjtrnbhngrbegkglv
Verification output (2): Yubikey OTP was replayed (REPLAYED_OTP)
[root@localhost ~]#
```

↑ ↑ ↑
from pressing keyboard from pressing yubikey


↑
from command history recall buffer 2 seconds later
(one-time key, used 2nd time, is stale)

ykclient utility queries validation server with key

File Edit View Go Capture Analyze Statistics Telephony Tools Help

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.110	66.51.206.100	DNS	Standard query A api.yubico.com
2	0.000055	192.168.1.110	66.51.206.100	DNS	Standard query AAAA api.yubico.com
3	0.076977	66.51.206.100	192.168.1.110	DNS	Standard query response A 67.23.37.171
4	0.085743	66.51.206.100	192.168.1.110	DNS	Standard query response
5	0.086199	192.168.1.110	67.23.37.171	TCP	55434 > 80 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSV=2191992 TSEQ=
6	0.188641	67.23.37.171	192.168.1.110	TCP	80 > 55434 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSV=4801
7	0.188692	192.168.1.110	67.23.37.171	TCP	55434 > 80 [ACK] Seq=1 Ack=1 Win=14656 Len=0 TSV=2192094 TSEQ=480165380
8	0.188825	192.168.1.110	67.23.37.171	HTTP	GET /wsapi/verify?id=1660tpcccccvrjbcitndbjrlrljdfjtrnbhngrbegkglv HTTP
9	0.293670	67.23.37.171	192.168.1.110	TCP	80 > 55434 [ACK] Seq=1 Ack=147 Win=6912 Len=0 TSV=480165391 TSEQ=2192094
10	0.621146	67.23.37.171	192.168.1.110	HTTP	HTTP/1.1 200 OK [text/plain]
11	0.621213	192.168.1.110	67.23.37.171	TCP	55434 > 80 [ACK] Seq=147 Ack=359 Win=15680 Len=0 TSV=2192527 TSEQ=480165422
12	0.621382	192.168.1.110	67.23.37.171	TCP	55434 > 80 [FIN, ACK] Seq=147 Ack=359 Win=15680 Len=0 TSV=2192527 TSEQ=480165
13	0.733714	67.23.37.171	192.168.1.110	TCP	80 > 55434 [FIN, ACK] Seq=359 Ack=148 Win=6912 Len=0 TSV=480165435 TSEQ=21925
14	0.733779	192.168.1.110	67.23.37.171	TCP	55434 > 80 [ACK] Seq=148 Ack=360 Win=15680 Len=0 TSV=2192639 TSEQ=480165435

play:



Follow TCP Stream

Stream Content

```

GET /wsapi/verify?id=16&otp=ccccccvrjbcitndbjrlrljdfjjtrnbhngbrbegkglv HTTP/1.1
User-Agent: ykclient/2.3
Host: api.yubico.com
Accept: */*

HTTP/1.1 200 OK
Date: Wed, 01 Feb 2012 20:50:41 GMT
Server: Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny13 with Suhosin-Patch mod_ssl/2.2.9 OpenSSL/0.9.8g
X-Powered-By: PHP/5.2.6-1+lenny13
Vary: Accept-Encoding
Content-Length: 73
Content-Type: text/plain
X-Pad: avoid browser bug


h=zVrceDCCZMt3Lmes5wkIVat8Hg=
t=2012-02-01T20:50:42Z0170
status=OK

```

red - client to server

blue - server to client

replay:



Follow TCP Stream

Stream Content

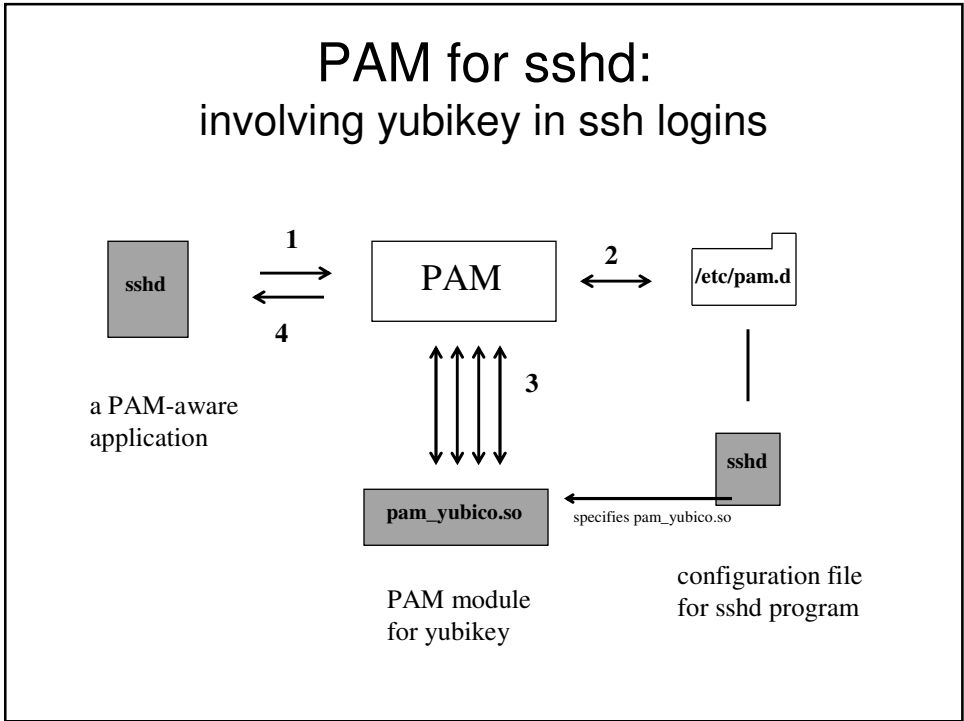
```

GET /wsapi/verify?id=16&otp=ccccccvrjbcitndbjrlrljdfjjtrnbhngbrbegkglv HTTP/1.1
User-Agent: ykclient/2.3
Host: api.yubico.com
Accept: */*

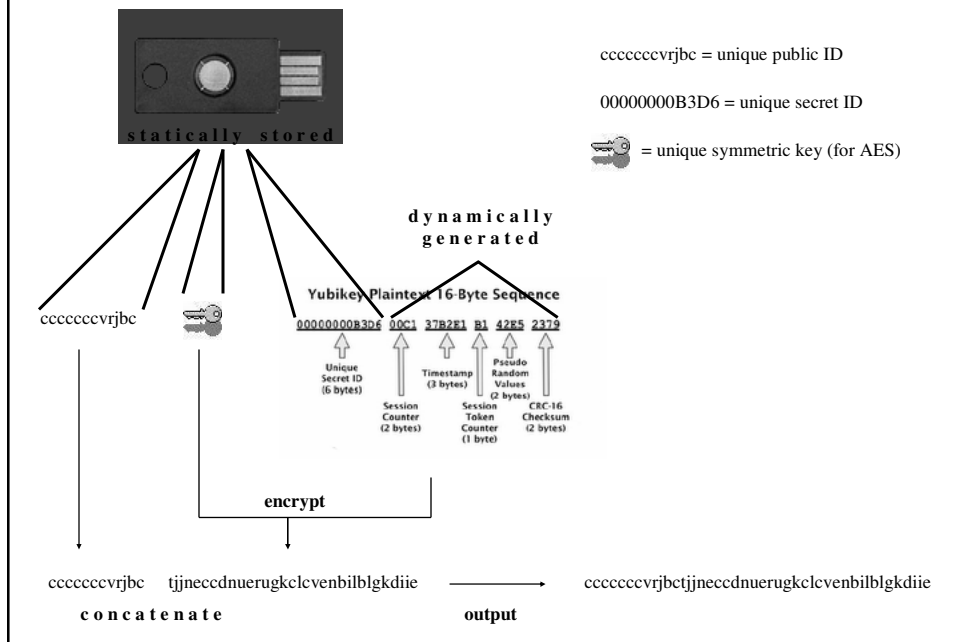
HTTP/1.1 200 OK
Date: Wed, 01 Feb 2012 20:50:44 GMT
Server: Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny13 with Suhosin-Patch mod_ssl/2.2.9 OpenSSL/0.9.8g
X-Powered-By: PHP/5.2.6-1+lenny13
Vary: Accept-Encoding
Content-Length: 83
Content-Type: text/plain
X-Pad: avoid browser bug

h=d/S1VQ8RwqfTnqenIH6b6wrj7g=
t=2012-02-01T20:50:44Z0580
status=REPLAYED_OTP

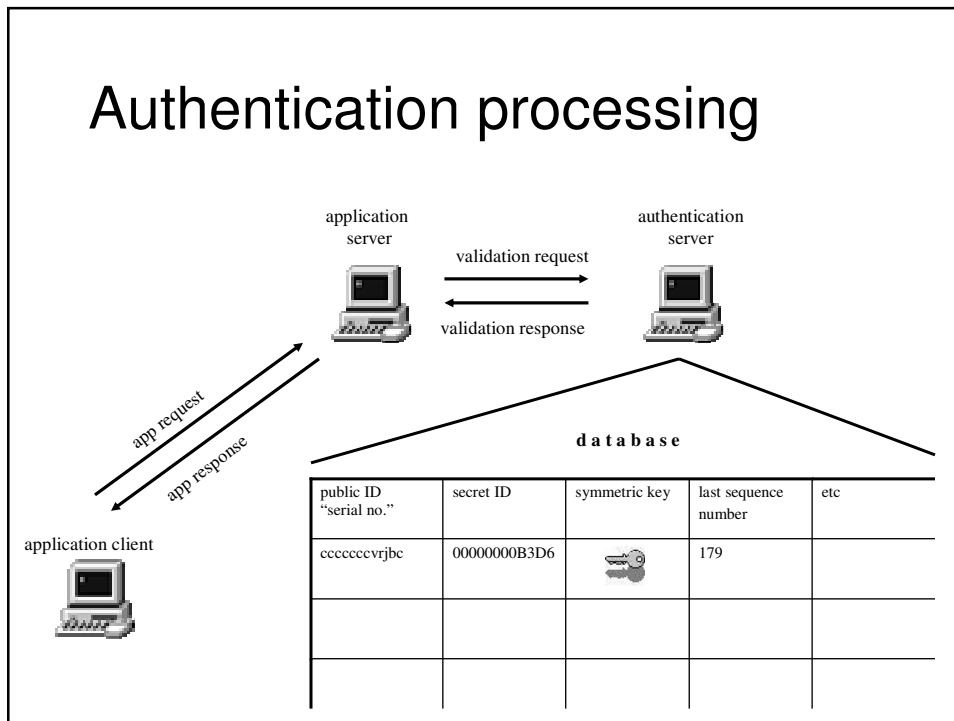
```



Construction of output



Authentication processing



This week's lab exercise

- primary form – supplied file
 - use *John the Ripper* (on Windows) to break the passwords in a supplied linux password file (historical style) holding 50 usernames with hashed passwords
- secondary form – your generated file
 - use the fedora virtual machine
 - login as root, create users and apply passwords
 - transfer the shadow file with new users' hashed passwords to the Windows system and repeat