Password authentication

David Morgan

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
Bases for authentication

- something people know
  - password
- something about them
  - retina/iris
  - fingerprint
  - DNA
  - voice
  - ear
  - face
- something they have
  - smart card
  - SIM (subscriber identity module) card
  - hardware token
- 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)
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?? is 8 still “enough”?)
- 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
  - itbgcethate
- substitute non-letters for letters
  - w7@dd7$b7k
  - i7bgceth@te
- capitalize something
  - w7@DD7$b7k
  - i7BGCTH@te
Forcing strong passwords

- create them for users, don’t let users change 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
- PAM itself succeeds or fails, depending on the modules’ outcomes
  - returns its overall outcome to app (4)
- 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)
- hashcat (http://hashcat.net/oclhashcat-plus)
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
  - 2 one-letter words
  - 4 two-letter words
  - 8 three-letter words
- a 3-symbol alphabet has
  - 3 one-letter words
  - 9 two-letter words
  - 27 three-letter words
- a 10-symbol alphabet has
  - 10 one-letter words
  - 100 two-letter words
  - 1000 three-letter words
- an n-symbol alphabet has
  - \( n \) one-letter words
  - \( n^2 \) two-letter words
  - \( n^3 \) three-letter words
  - \( n^m \) \( m \)-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? *

Password strength determinants

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

a 3rd criterion!! human-dependent!
(arginally, the most important factor these days)
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.”

Human factor discovery

From passwords captured in large volume in recent publicized leaks, more commonly than ever expected, people choose 9-character passwords whose first 5 characters are letters, last 4 are numbers, first character is sometimes capitalized, others are not.

Random:

<table>
<thead>
<tr>
<th>any</th>
<th>any</th>
<th>any</th>
<th>any</th>
<th>any</th>
<th>any</th>
<th>any</th>
<th>any</th>
<th>any</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>96</td>
<td>96</td>
<td>96</td>
<td>96</td>
<td>96</td>
<td>96</td>
<td>96</td>
<td>96</td>
</tr>
</tbody>
</table>

wordspace \(96^9 = 7 \times 10^{17}\)

Observed:

<table>
<thead>
<tr>
<th>alpha</th>
<th>lower</th>
<th>lower</th>
<th>lower</th>
<th>lower</th>
<th>num</th>
<th>num</th>
<th>num</th>
<th>num</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

wordspace \(52 \times 26 \times 26 \times 26 \times 10 \times 10 \times 10 \times 10 = 2 \times 10^{11}\) 1 million times bigger 1 million times smaller

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
    * 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

- official name
- password (placeholder, was historical storage location)
- UID
- GID
- real name
- home directory
- login shell
/etc/shadow entries hold ancillary user information

craig:$1$2YL52jh$L$:11992:60:75:3:14:12417:134550548

Newer distributions, longer hashes
Hash algorithm selectability

```
[red]localhost -> # useradd testuser
[red]localhost -> # authconfig --passalgo/md5 --update
[red]localhost -> # cat /etc/sysconfig/authconfig | grep PASSWDALGORITHM
PASSWDALGORITHM=md5
[red]localhost -> # echo "testuser:abracadabra" | chpasswd
[red]localhost -> # tail -l /etc/shadow
testuser:*5YVSF7tZD05Com1U0SwN2v5EBj8bh/0:15597:0:99999:7::
[red]localhost -> # authconfig --passalgo=sha256 --update
[red]localhost -> # cat /etc/sysconfig/authconfig | grep PASSWDALGORITHM
PASSWDALGORITHM=sha256
```

Dictionary attack

- **generated/dynamic** dictionary attack
  - list of dictionary words that are tried (i.e., hashed and compared to hash of actual password) one after another
  - 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

Brute force 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

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

© David Morgan 2011-2014

Next-gen cracking platform

“This $12,000 computer, dubbed Project Erebus v2.5 by creator d3ad0ne, contains eight AMD Radeon HD7970 GPU cards. Running version 0.10 of oclHashcat-lite, it requires just 12 hours to brute force the entire keyspace for any eight-character password containing upper- or lower-case letters, digits or symbols.”

http://arstechnica.com/security/2012/08/passwords-under-assault/

© David Morgan 2011-2014
Raw speed estimate advances

- 2004 PC: $2^{34}$/hr (per Mandylion spreadsheet, from 2004)
- 2012 PC, single HD7970 GPU: 8.2 billion/sec
- Factor of 1718

- 2012 dedicated cracker, 8 GPUs: 8x? 64 billion/sec?

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

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

```
typed password
  ↓
hash algorithm
  ↓
hash

Storage:
```

hash
unsalted password: authenticating

Generate time

| typed password | hash algorithm | hash |

Authenticate time

| typed password | hash algorithm | hash | comparison | match or mismatch |

salted password: generating

Generate time

| salt | typed password | hash algorithm | hash |

Storage:

| salt | hash | hash | hash |

© David Morgan 2011-2014
salted password: authenticating

Generate time:

- salt
- typed password

Storage:

- salt
- hash
- typed password

Authenticate time:

- hash
- comparison
- match or mismatch

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