PASSWORD SECURITY PART 2

<<< PREVIOUSLY

PREVIOUSLY

* big leaks
* password strength
* weak & strong passwords
* passphrases
* guidelines

CRACKING PASSWDS >>>

THIS WAY >>>

AGENDA

* cryptographic hash functions

* hardware and software

* hashcat, offline attack

HASH FUNCTIONS

HASH FUNCTIONS

* not every hash function is secure
* one way function
* collision resistant
* provide high variability

PRINCIPLES
(SKIP)

ONE WAY FUNCTION

* easy to compute on every input

* hard to invert to get original message

COLLISION RESISTANT

* difficult to find two distinct inputs resulting in the same hash

* difficult to access account with different password

* collisions happen because inputs are compressed

HIGH VARIABILITY

* very different outputs for similar inputs

* difficult to reveal characteristics of password

```
"password"  | 286755fad04869ca523320acce0dc6a4
"password1" | 10b222970537b97919db36ec757370d2
```

ISSUES
(SKIP)

ISSUE: GUESSING

* solution: key stretching

* slow down computations

* increase memory requirements

ISSUE: RAINBOW TABLES

* solution: key stretching,
  (multiple level RT required)

* solution: dynamic salts
  (separate RT for each salt)

ISSUE: COLLISIONS

* longer digest means
  smaller risk of collision

* normally this is not big issue
  in case of passwords

ANSWER: KEY STRETCHING

* extra iterations of hash function
  make RT ineffective and
  slow down attacks

* memory complexity makes
  more difficult to implement
  cracker in GPU

ANSWER: SALTS

* random data as an additional input

* effective against:
  pre-computed attacks (RT)
  Google hacking

* same passwords produce different hashes

* guessing one hash at the time

SECURE HASH FUNCTIONS

SECURE HASH FUNCTIONS

* fast, general purpose hashes:
  ~~MD5~~, ~~SHA family~~

* slow, safe password storage:
  BCRYPT, PBKDF2, SCRYPT

MD5/SHA/SHA2

* general purpose functions,
  designed to hash gigabytes of data,
  very fast

* not for hashing passwords

BCRYPT

* designed to store static passwords

* salt built into algorithm

* configurable work factor,
  rounds: 2^{WF} (2^{12}, 4096 rounds)

PBKDF2

* similar level of security as BCRYPT

* secures WiFi (WPA2-PSK),
  4096 rounds of SHA1,
  SSID is salt (pre-computed RT!)

* change default SSID, don't use:
  c3com, NETGEAR, ZyXEL, linksys

SCRYPT

* the latest, secure hash function

* works on binary data as well

* advanced key stretching:
  computation & memory complexity

CRACKING: HARDWARE

HARDWARE

* hardware is fast and cheap
  (doesn't apply to EC2 ;)

* hashing runs in parallel

* GPUs scale almost linearly

* GPUs are much faster than CPUs,
  loads of specialized cores

CPU PERFORMANCE

* number of cores
* clock frequency
* 32/64 bits
* SSE2, few arithmetic instructions per clock
* optimizations made by compiler

GPU PERFORMANCE

* number of ALUs,
  Arithmetic Logic Units

* number of stream processors

* optimizations

RADEON CITY DEC-2012

RADEON CITY DEC-2012

* 25-GPU cluster

* output

```
  180 bln  MD5     h/sec
   63 bln  SHA1    h/sec
   71 kilo BCRYPT  h/sec
```

* all your password
  are belong to us (niche joke ;)

CRACKING: SOFTWARE

SOFTWARE

* GitDigger

* CUPP

* Hashcat

GIT DIGGER

* scraped entire github to build dictionary

* similar tools used to scrap Wikipedia and code.google

GIT DIGGER: RESULTS

* passwords, usernames, emails, common files and directories

* OCR on images for stored secrets

* static code analysis for vulns

* .gitignore and .svn folders, SSH keys, tokens and static salts

CUPP

* common user passwords profiler

* creates personalized dictionaries
  used against specific person

* effective against mortals

DICTIONARIES (SKIP)

PRIMARY DICTIONARY

* compact and related to target
* all leaks on the Interwebz
* recovered plains
* common word lists
* quick tests, rule based, hybrid, combinator attacks

EXTENDED DICTIONARIES

* can be very large

* content of primary dictionary

* multiple languages

* scrapped wikipedia, github, code.google


HASHCAT

HASHCAT

* advanced, fast and free cracker
* many algorithms implemented
* supports CPU and GPU(OPENCL, CUDA)
* works on Linux, MacOS and Windows

HASHCAT: STRAIGHT

* straight attack

* checks one by one all strings in dictionary without modifications

* example:
  password, adobe123, linkedin, admin, letmein, qwerty, trustno1, 123456

HASHCAT: MASK/BF

* bruteforce/mask attack

* exhaustive, but time consuming,
  specify mask to reduce key space

* example:
  Password1 (mask: ?u?l?l?l?l?l?l?d)
  adobe123! (mask: adobe?d?d?d?s)
  123456789 (mask: ?d?d?d?d?d?d?d?d?d)

HASHCAT: COMBINATOR

* combinator attack

* combines words from list into pairs

* example:
  correct battery horse staple,
  awful password, awful-password1

HASHCAT: HYBRID

* hybrid attack

* combines bf and dictionary attack, words from list with prefix/suffix

* example:
  password1, password123, password(), passwordabc1

HASHCAT: RULE BASED

* rule based attack

* very fast regexp, runs in GPUs

* modifies words:
  add/replace chars, digits, symbols,
  leetspeak rule

* example:
  p4ssw0rd1, P4ssw0rd1, passwd123,
  PaSwOrD123

HASHCAT: MARKOV

* bruteforce++ attack

* hashcat builds markov chains, identifies patterns and probability

* will find password quicker

* if u cn rd ths u cn gt gd jb

SAMPLE SCENARIO

REFERENCES: WEB

* sekurak.pl
* arstechica.com
* troyhunt.com
* haveibeenpwned.com
* splashdata.blogspot.com
* mytrickytricks.blogspot.com
* entima.net/diceware/

REFERENCES: BOOKS

* Take Control of Your Passwords

* Perfect Password