p4ssw0rd s3cur1ty

byte size briefing

10110101 01101010 11010101

AGENDA

* big leaks
* access control
* password storage
* cryptographic hash functions
* attacks
* password strength factors
* passwords managers
* guidelines

TL/DR

TL/DR USER

* generate **random passwords**

* **do not reuse**

* use **password manager**

* backup your passwords (offline/offsite)

* use **memorable passphrases**

TL/DR PROGRAMMER

* use **bcrypt**, **scrypt** or **pbkdf2** hash functions

* use **dynamic salts**

* require **sufficient password complexity**

* **do not enforce** very **strict patterns**

* **do not implement** anything **yourself**

BIG LEAKS

BIG LEAKS

* **RockYou** (2009), 32mln, plain text

* **game changer** in password cracking

* breach revealed the way people think

BIG LEAKS

* **Gawker** (2010), 1.3mln, encrypted DES

* **Twitter accounts compromised** due to password reuse

* top 5 passwords

    * 123456, password, 12345678, **lifehack**, qwerty

BIG LEAKS

*** Sony** (2011), 1mln, plain text

* a **lot of repetition between Sony and Yahoo** dumps published next year

BIG LEAKS

* **LinkedIn** (2012), 8mln, unsalted SHA1

* SHA1 is **slightly slower** than MD5

BIG LEAKS

* **Adobe** (2013), 152mln, encrypted 3DES/ECB

* **hints in plain text**

* the greatest crossword puzzle in the history of the world (XKCD)

HAVE I BEEN PWNED?

haveibeenpwned.com

ROCKYOU ANALYSIS

* **passwords are short,** mostly 6-10 characters

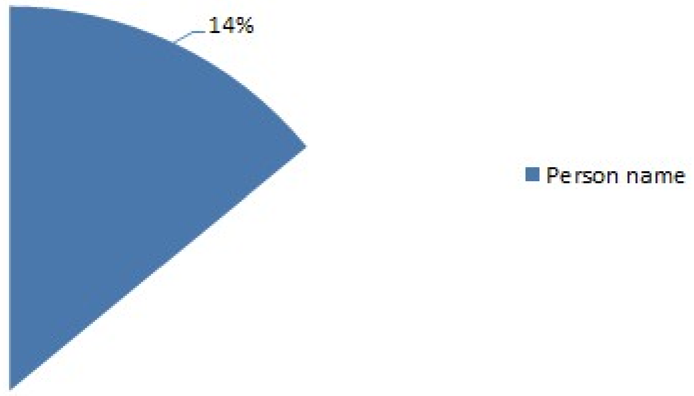
* **capital letters** mostly come **at the beginning** of a password

* **numbers and punctuation** mostly show up **at the end**

* strong tendency to use **first names followed by years**

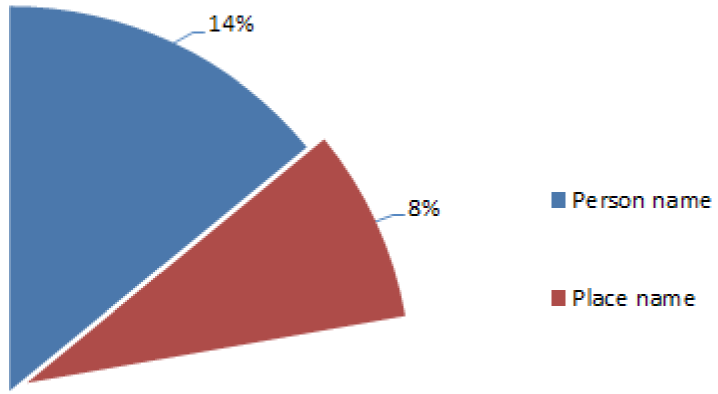
LULZSEC RELEASES 2011

* 14% personal name



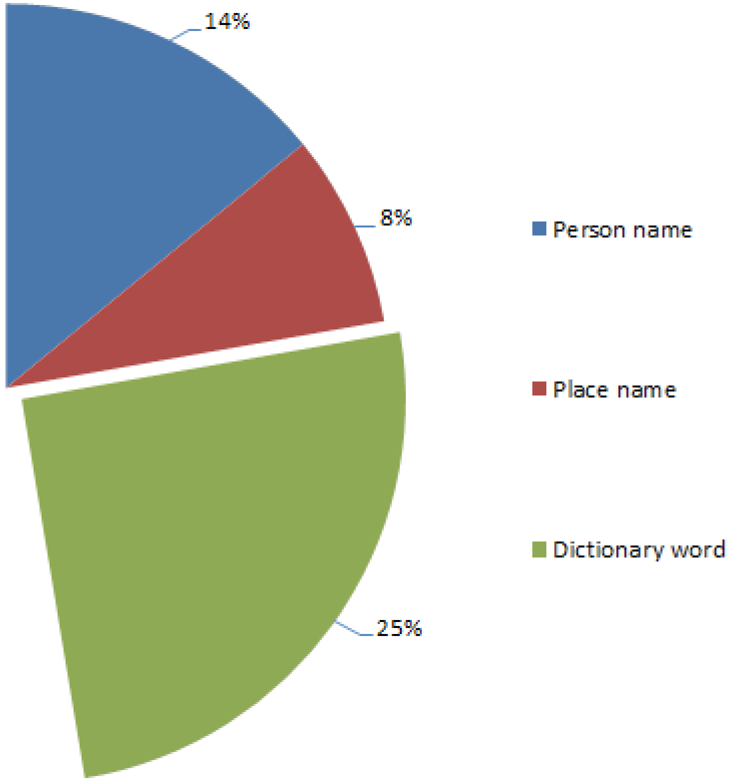
LULZSEC RELEASES 2011

* 8% place name



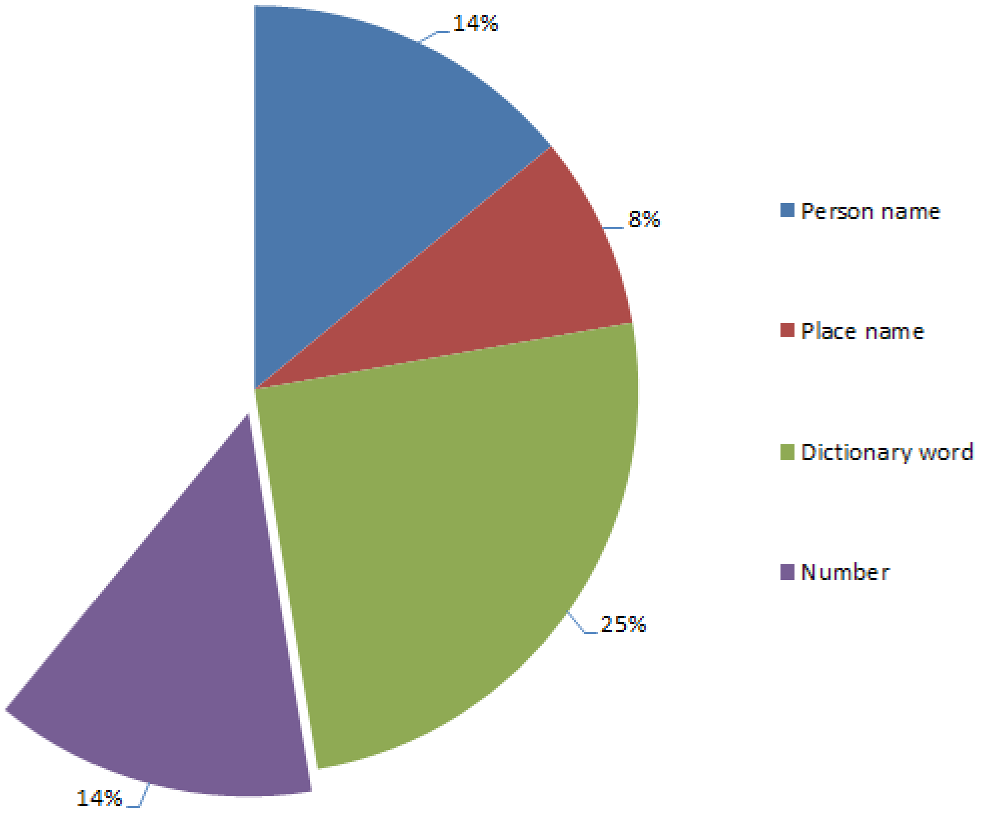
LULZSEC RELEASES 2011

* 25% dictionary word



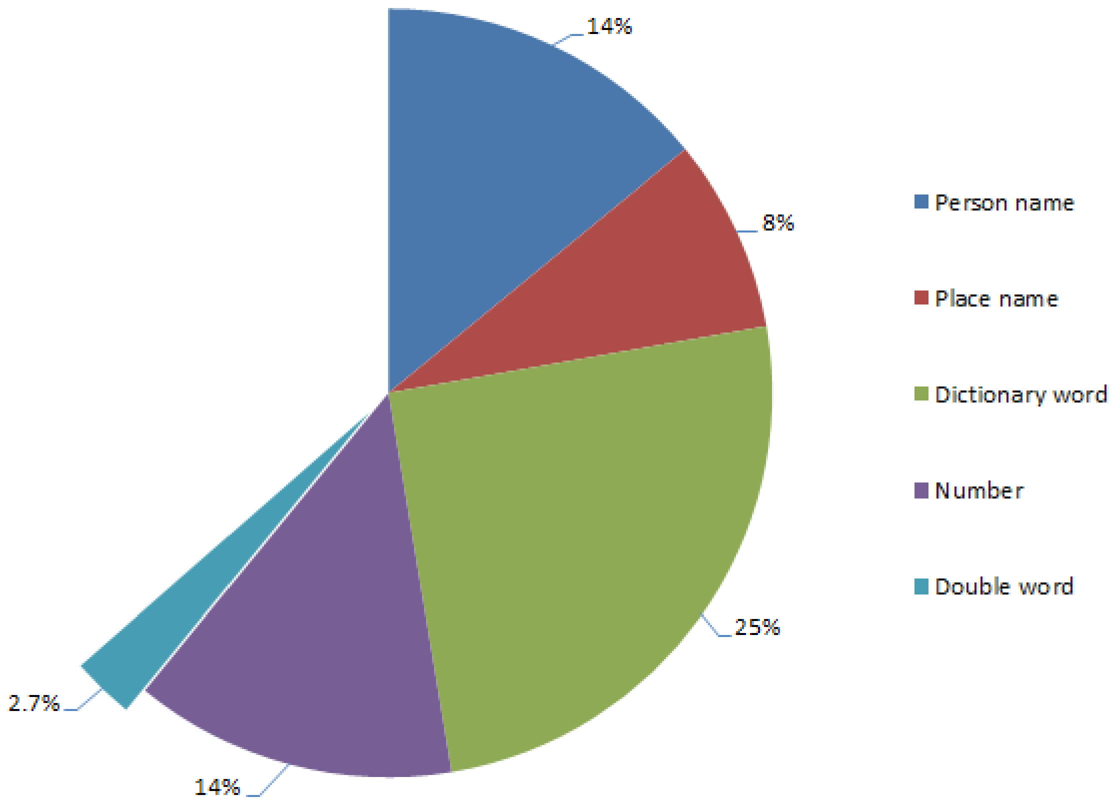
LULZSEC RELEASES 2011

* 14% purely numeric



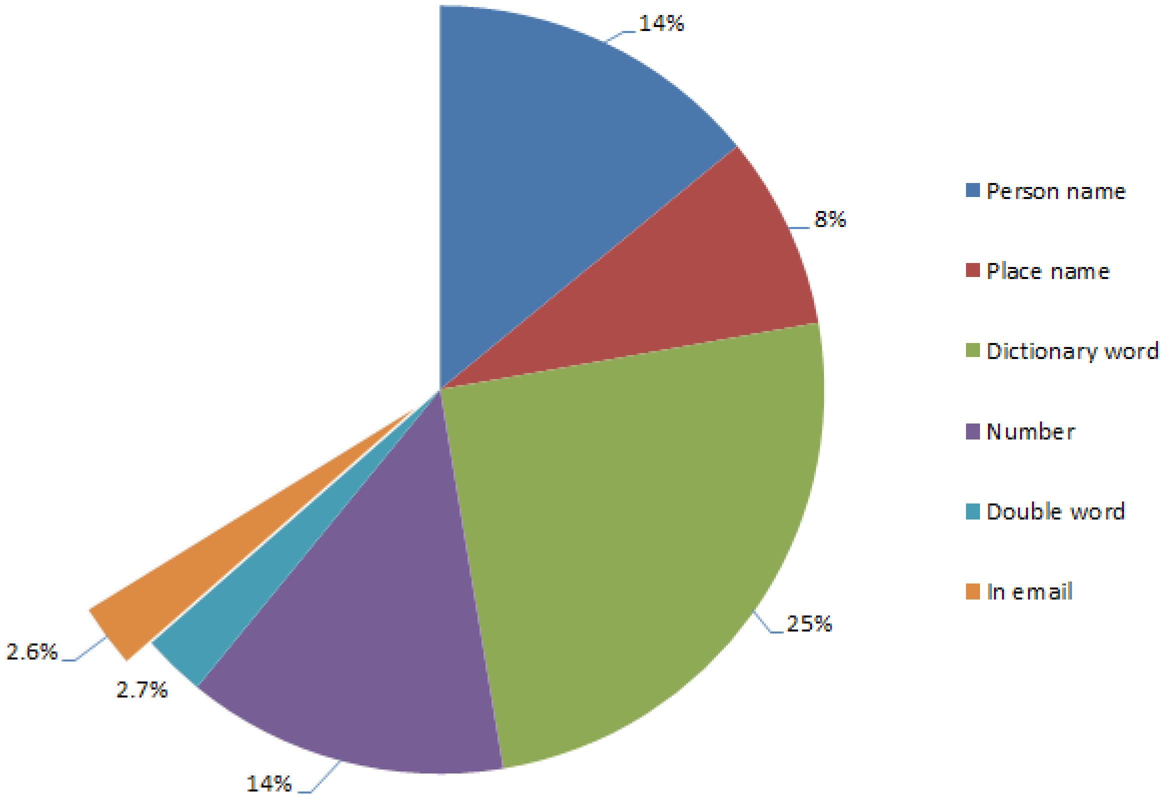
LULZSEC RELEASES 2011

* 2.7% double word



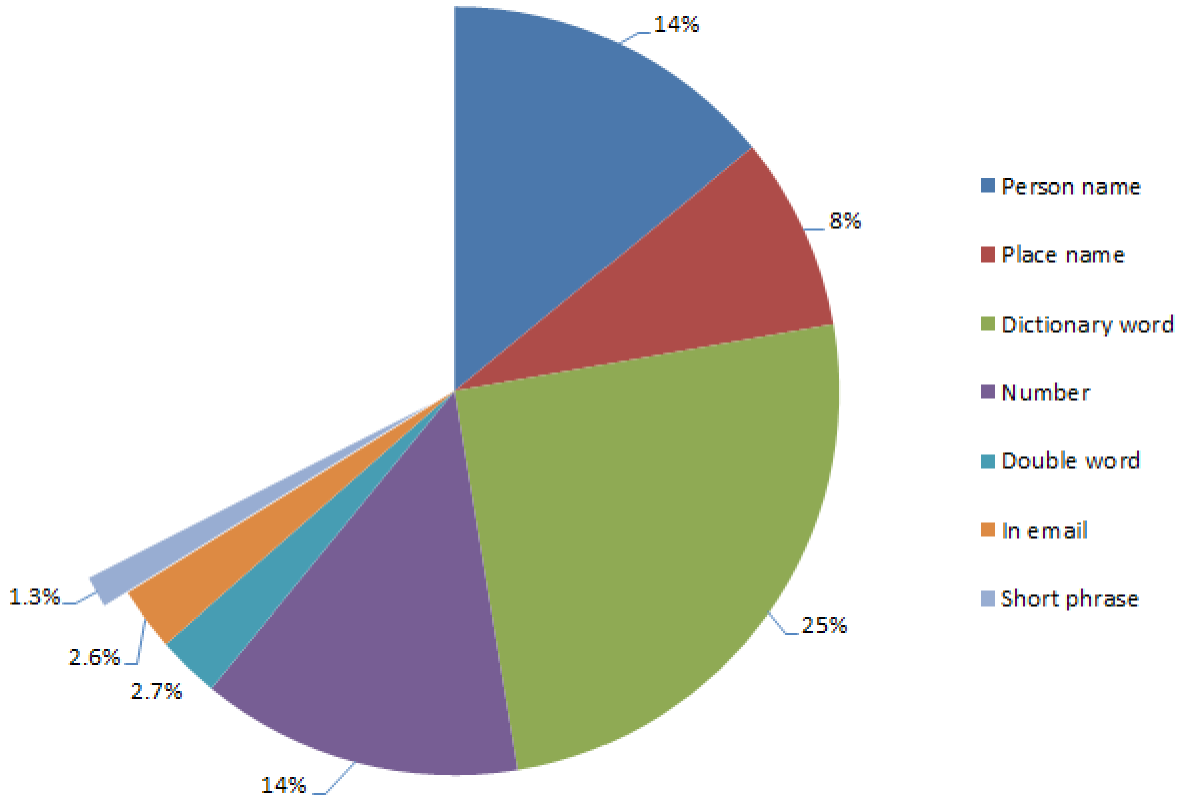
LULZSEC RELEASES 2011

* 2.6% in email



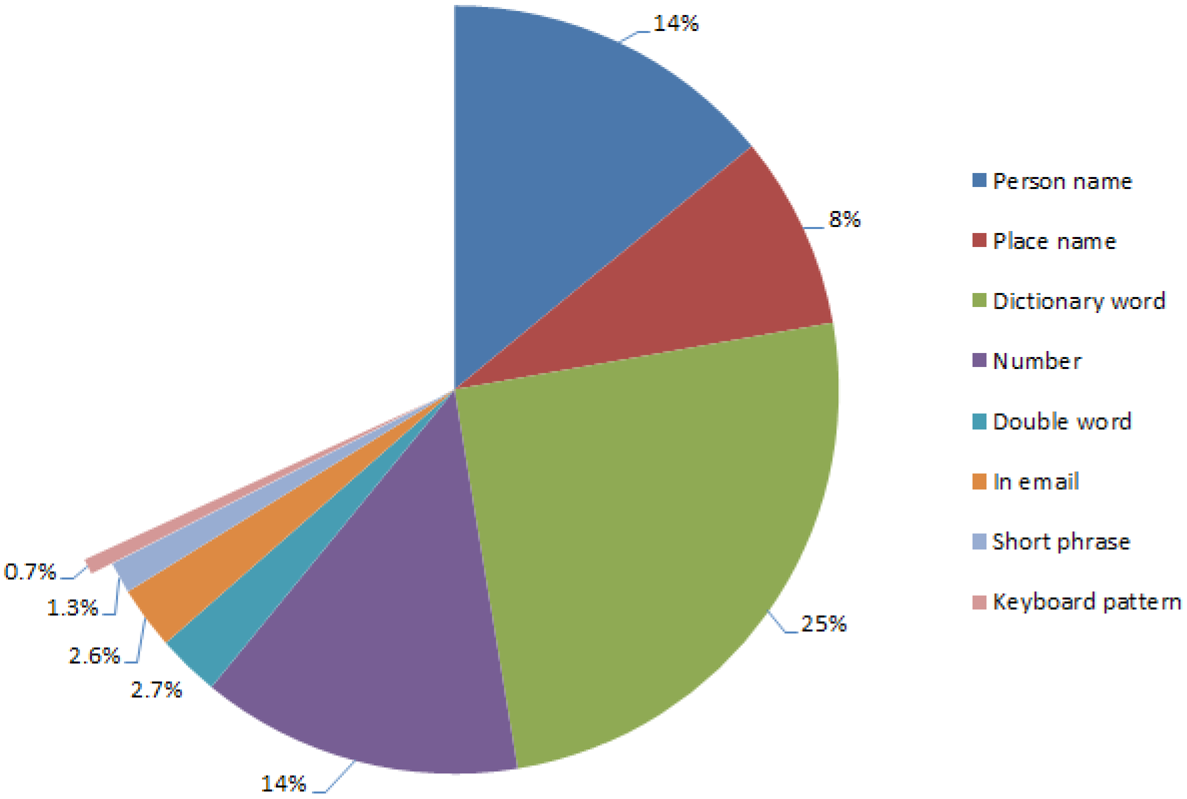
LULZSEC RELEASES 2011

* 1.3% short phrase



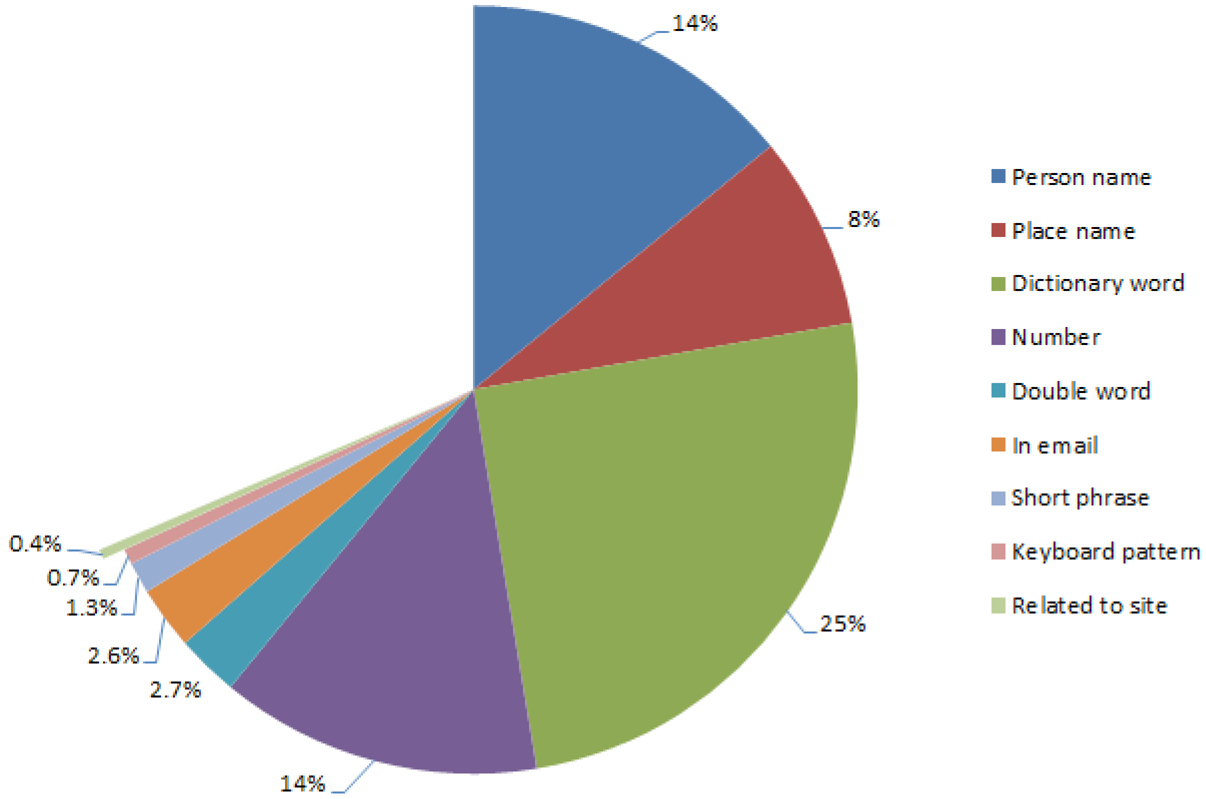
LULZSEC RELEASES 2011

* 0.7% keyboard pattern



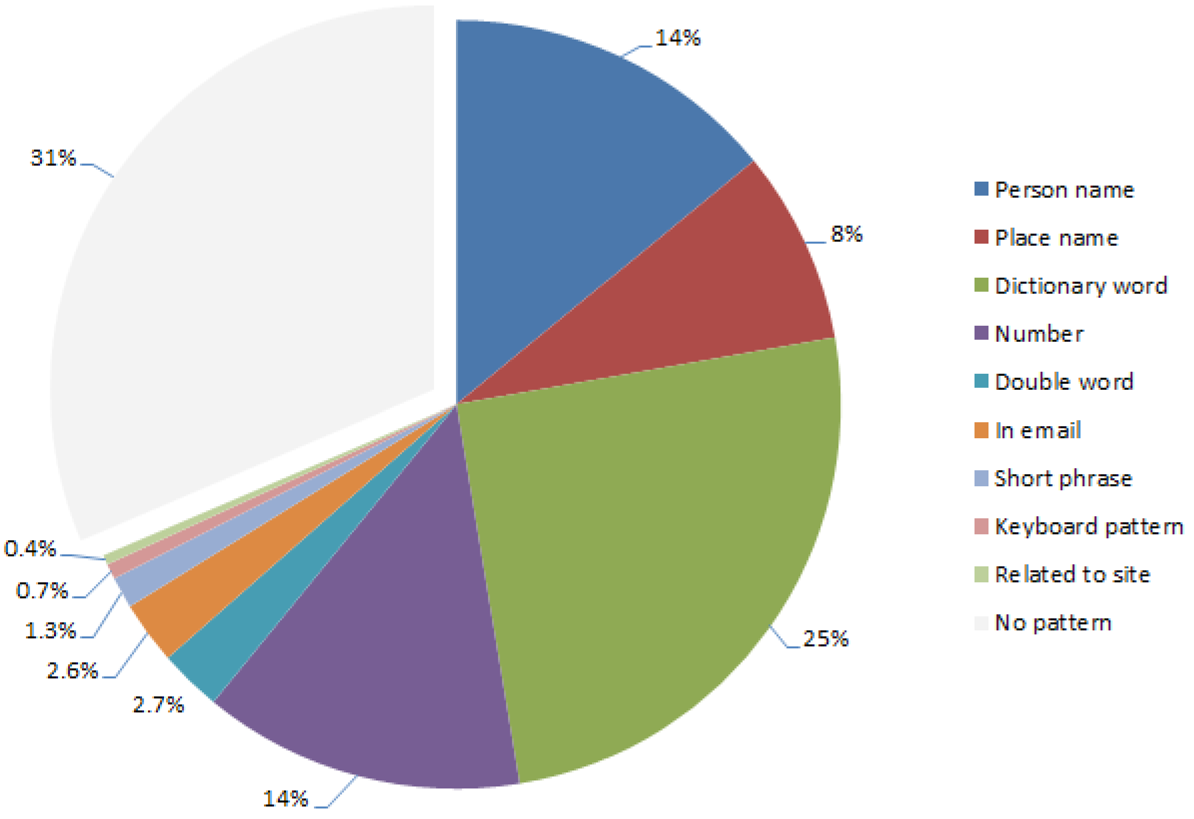
LULZSEC RELEASES 2011

* 0.4% site related



LULZSEC RELEASES 2011

* 31% unknown or no pattern



TOP 25 PASSWORDS 2013

123456

password

12345678

qwerty

abc123

123456789

111111

1234567

iloveyou

adobe123

123123

Admin

1234567890

letmein

photoshop

1234

monkey

shadow

sunshine

12345

password1

princess

azerty

trustno1

000000

ACCESS CONTROL

ACCESS CONTROL

* **identification** - who you are

* **authentication** - something you know/have/are

* **authorization** - what you can do

SOMETHING YOU KNOW

* static password

* PIN

* pattern (mobile phone)

SOMETHING YOU HAVE

* dynamic password list

* RSA token

* proximity card

SOMETHING YOU ARE

* biometrix

    * fingerprint

    * retina scan

MULTI-FACTOR AUTH

* authentication **is critical**

* **improves safety**

* involves **various groups**

* **difficult to compromise both factors**

    * static password & PIN sent by text

MULTI VERIFICATION

* similar concept, but **significantly weaker**

* involves one group

    * **email accessed** via phone

    * **PIN sent by text** to the same device

PASSWORDS PROS

* **cheap**, text fields, keyboard, hashing

* **simple**, everybody knows how to use keyboard

PASSWORDS CONS

* short, **characteristic string**

* can be **intercepted** when sent over net (mitm, heartbleed)

* once **stolen and changed**, legitimate user **might loose access**

PASSWORD STORAGE

PASSWORD STORAGE

* **plain text**, flawless victory

* **encrypted**, plains will leak if private key is compromised, one to rule them all

* **salted hashes**, the only secure way

HASH FUNCTIONS

HASH FUNCTIONS

* **not every hash** function **is secure**

* cryptographic hash functions have to meet **strict requirements**

HASH FUNCTIONS

* **one way** function

* **collision resistant**

* provide **high variability**

ONE WAY FUNCTION

* **easy to compute** on every input, **hard to invert** output

* very **difficult to compute original message** (crack hash)

COLLISION RESISTANT

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

* difficult to **access one's account** with **different password**

* collisions happen because inputs are **compressed** to **short bit sequence**

HIGH VARIABILITY

* **very different outputs** for **similar inputs**

* difficult to **reveal any characteristics** of password

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

ISSUE: GUESSING

* **key stretching**

  * **slow down** computations (guesses/second)

  * **increase memory** requirements

ISSUE: RAINBOW TABLES

* **key stretching**

    * multiple level RT required

* **dynamic salts**

    * required separate RT for each salt

ISSUE: COLLISIONS

* **longer digest** means **smaller risk of collision**

* normally this is not big issue

SALTS

* **random data** used as an **additional input**

* make **pre-computed attacks ineffective**

* prevent from **Google hacking**

* **same passwords** produce **different hashes**

* cracker will have to attack **one hash at the time**

* does not need to be secret

KEY STRETCHING

* **improves security**

* **adds** time and/or memory **complexity**

    * **additional iterations** of hash function

    * **additional memory** requirements

* makes **RT ineffective** and **slows down BF**

* **secure hash functions** with key stretching

HASH FUNCTIONS

* MD5

* SHA family

* BCrypt

* PBKDF2

* SCrypt

MD5

* **very fast!**

* general purpose function designed to hash gigabytes of data

* **not designed to hash passwords**

* known collisions attacks (binary data)

SHA/SHA2/SHA3

* **very fast!**

* general purpose function designed to hash gigabytes of data

* **not designed to hash passwords**

* known collisions attacks (binary data)

BCRYPT

* **designed to store static passwords**

* **requires salt** which is built into algorithm

* configurable **key stretching** (work factor)

* work factor **determines** computation **complexity**

* WF +1 **doubles time** required to **compute hash**

PBKDF2

* provides **similar level of security** as **BCrypt**

* used to **secure WiFi** (WPA2 standard)

  * **SSID is salt**, pre-computed RT exists

  * change default SSID if not randomized

  * sample: 3com, NETGEAR, ZyXEL, linksys

SCRYPT

* the latest and **most secure hash function**

* **works on binary data** as well

* **advanced key stretching** with configurable

    * computation complexity

    * memory requirements

ATTACKS

EASIER THAN EVER

* attacking hashes is **easier than ever**

* **cheap and fast hardware** (GPUs), cloud computing

* modern and **advanced tools** are available

* **massive leaks revealed** real life passwords and allow to **tune rules** and understand people

TWO KINDS OF ATTACK

* **online attack** against **live system** and infrastructure

* **offline, cryptographic attack** against hashed passwords once database was compromised

ONLINE ATTACK

ONLINE ATTACK

* require **careful planning and precision**

* **targeted attacks** against selected accounts: root, admin, administrator

* **effective** against **poorly secured accounts**

    * password == login/email/blank/default

    * compact and/or personalized dictionaries

PHASES OF ATTACK

* collecting information

* **discovering limitations**

* preparing dictionaries and tools

DISCOVER LIMITATIONS

* **password complexity requirements**

* username policy

* account lockout

* captchas

* WAF, IPS/IDS, filtering, throttling

OFFLINE ATTACK

OFFLINE ATTACK

* **cryptographic attack against hashes**

* phases of attack

    * **search** existing **hash databases**

    * prepare **dictionaries**, rainbow tables and tools

    * perform **various attacks** (many rounds)

MAIN TYPES OF ATTACK

* bruteforce

* dictionary

* hybrid

* rule based

* rainbow tables

BRUTEFORCE

* exhaustive, tests **every combination**

* it **will find every password** in theory

* in practice it is **used in very narrow ranges**

* **time consuming**

BRUTEFORCE

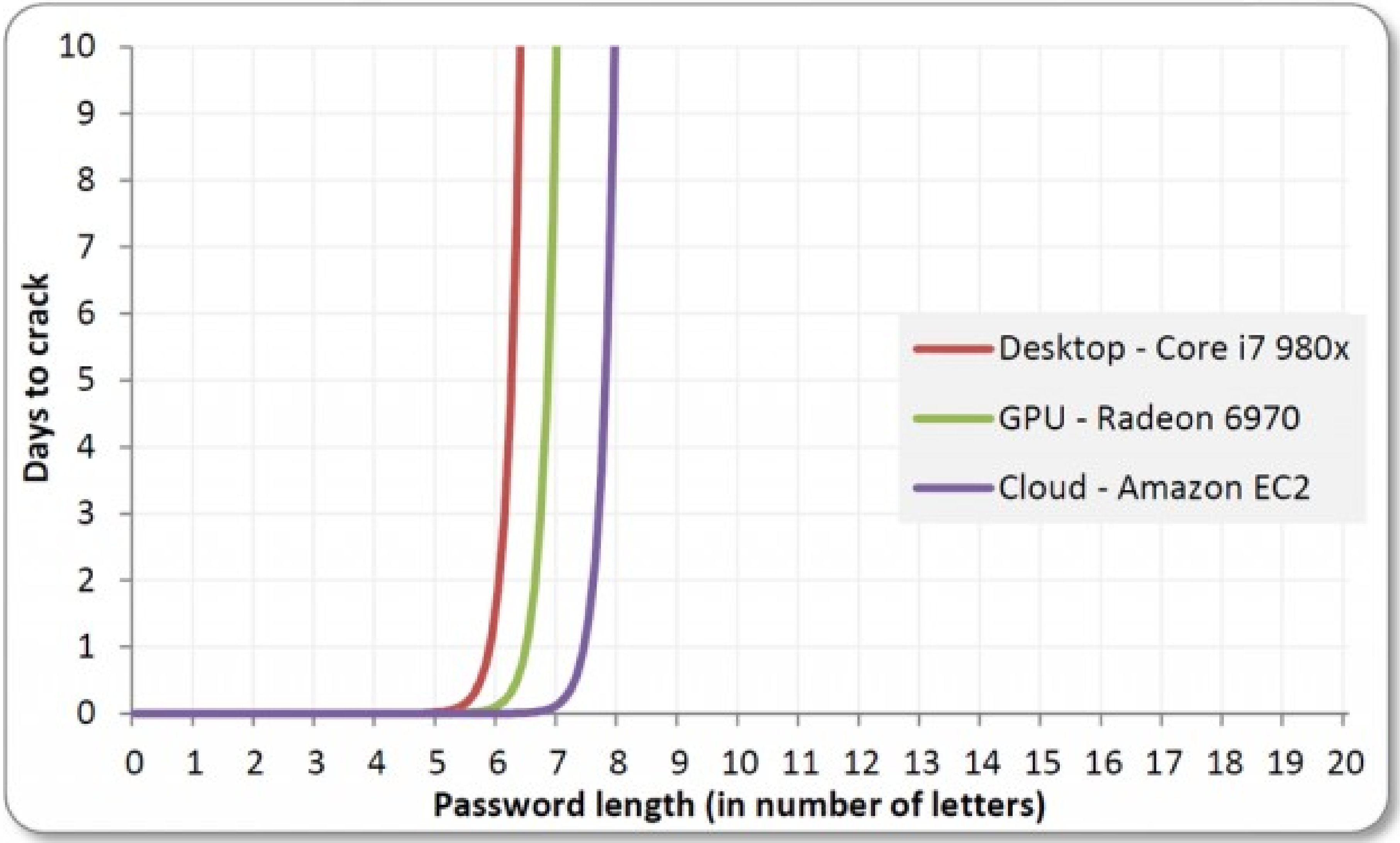
* effective against **short passwords** (up to 10 characters)

* **ineffective against long passwords**

* example

    * a, aa, ab, ac... zzzzzz

EXPOTENTIAL WALL OF BF



DICTIONARY

* tests all **words from list** called a **dictionary**

* **does not guarantee success**

* **dictionary is a key**, common words, names, places, dates...

* good dictionary can **break 50-60%** of the weakest passwords **straight away**

DICTIONARY

* attackers often use at least two dictionaries

    * **primary** and **extended**

* **personalized dictionaries** are **extremely effective** against mortals

DICTIONARY

* **ineffective against random passwords**

* example

    monkey, tree, car, cat, password1

PRIMARY

* **compact** and related to target

* **all leaks available on the Internet**

* already **cracked passwords**

* **common word lists**

* **quick tests**, **rule based** and **hybrid attacks**

EXTENDED

* content of **primary dictionary**

* **every word you can imagine**

* **multiple languages**

* programming languages syntax

* combination of **many smaller dictionaries**

* scrapped **wikipedia, github, code.google**

GIT DIGGER

* used to **scrap entire github** to build dictionary

* similar tools used to scrap

    * wikipedia

    * code.google

PRIMARY OBJECTIVES

* passwords, usernames, emails

* **common files** and **directories** (forced browsing)

* **images to perform OCR** (some people store secrets in images)

* **static code analysis** for **vulnerabilities**

EXTRAS

* a lot of **static salts**

* SSH **auth keys**

* .gitignore and .svn

* **various** hardcoded **tokens** (ex. csrf)

CUPP

* common **user passwords profiler**

* creates **personalized dictionaries** used against specific person

* **extremely effective** against careless mortals

CUPP INPUT

* name, surname, nicknames

* friends, family members

* dates, phone numbers

* keywords related to hobby, work, site

* it can generate **enormous lists of passwords**

HYBRID

* combines

    * **effectiveness of bruteforce**

    * **precision of dictionary attack**

* tests only **variations of common passwords**

HYBRID

- **ineffective against random passwords**
- example
  - johnabc, john1, john2, john123, john2000

RULE BASED

* similar to **regexp** rules

* **fast**, can be executed by **GPUs**

* adding characters, digits, replacing letters, leetspeak rule

RULE BASED

* **ineffective against random passwords**

* example

    * p4ssw0rd1, P4ssw0rd1, passabcd, password!

RAINBOW TABLES

* **speed up cracking process**

* **time-memory trade-off**

* precomputed table of hash chains

    * **first hash** & **last plain**

* which resolves into millions of other pairs

* **no need to compute hashes**

RAINBOW TABLES

* do not contain all plains/hashes

* **success rate is >96%**

* ineffective against

    * **hashes with dynamic salts**

    * **simple key stretching** (hashing many times)

RAINBOW TABLES

* regular (**hash->plain**) lookup table

4 chars	35,153,041	**913 MB**
5 chars	2,706,784,157	**70 GB**
6 chars	208,422,380,089	**5.4 TB**
7 chars	16,048,523,266,853	**417 TB**
8 chars	1,235,736,291,547,681	**32 PB**

RAINBOW TABLES

* MD5/SHA1 **rainbow tables**

  * **mixed-alpha-numeric**

8 chars	221,919,451,578,090	**160 GB**
9 chars	13,759,005,997,841,642	**864 GB**

  * **lower-alpha-numeric**

9 chars	104,461,669,716,084	**80 GB**
10 chars	3,760,620,109,779,060	**396 GB**

HARDWARE

* hardware is **fast** and **cheap**

* cracking can **run in parallel**

* GPUs **scale** almost **linearly**

* **GPUs are much faster** due to large quantities specialized processors

CPU PERFORMANCE

* number of cores

* clock

* 64 bits, yes/no?

* SSE2 (few arithmetic instructions per clock)

* optimizations performed by compiler

GPU PERFORMANCE

* number of ALUs (Arithmetic Logic Unit)

* number of stream processors

* RADEON, yes/no?

RADEON CITY 2012

RADEON CITY 2012

* all your passwords **are belong to us** ;-)

* 25-GPU cluster

* **180 billion** attempts/sec against **MD5**

* **63 billion** attempts/sec against **SHA1**

* **71 thousands** attempts/sec against **BCrypt**

SOFTWARE

* GitDigger

* CUPP

* Hashcat

HASHCAT

HASHCAT

* **advanced**, **fast** and **free** hash cracker

* **many algorithms** implemented

* efficient, **supports CPU/GPU**

* Linux, Windows, MacOS builds

HASHCAT: STRAIGHT

* straight attack

    * **checks one by one all strings** in dict without any modifications

* example

    * alfa, beta, gamma, web, experience

HASHCAT: BRUTEFORCE

* bruteforce attack
    * **useful in narrow ranges**
* example
    * a, aa, ab, ac... zzzzzz

HASHCAT: MASK

* mask attack

    * allows to specify **mask to reduce entropy**

* example

    * |mixed-case|**ass**|4-letters|3-digits|

HASHCAT: COMBINATOR

* combinator attack

    * combines words from list into pairs

    * **useful against longer passwords** made of shorter words

* example

    * correct battery horse staple (0 bits of entropy)

HASHCAT: HYBRID

* hybrid attack

    * combined bruteforce and dictionary attack

    * words from list with bruteforce prefix/suffix

* example

    * johnabc, john1, john2, john123, john2000

HASHCAT: TOGGLE CASE

* toggle case attack

    * words from list with upper/lower case variations

* example

    * passWORD1

HASHCAT: PERMUTATION

* permutation attack

    * checks permutations of each word from list

* example

    * asswordp1

HASHCAT: RULE BASED

* rule based attack

    * something like very fast **regexp**, **run in GPUs**

    * adding characters, digits, replacing letters, leetspeak rule

* example

    * p4ssw0rd1, P4ssw0rd1, passabcd, password!

HASHCAT: TABLE LOOKUP

* table lookup attack

* required map of characters, for example:

* a = 4, o = 0, i = 1

* effective against leet (1337) or similar

* variant of rule based attack

HASHCAT: BRUTEFORCE++

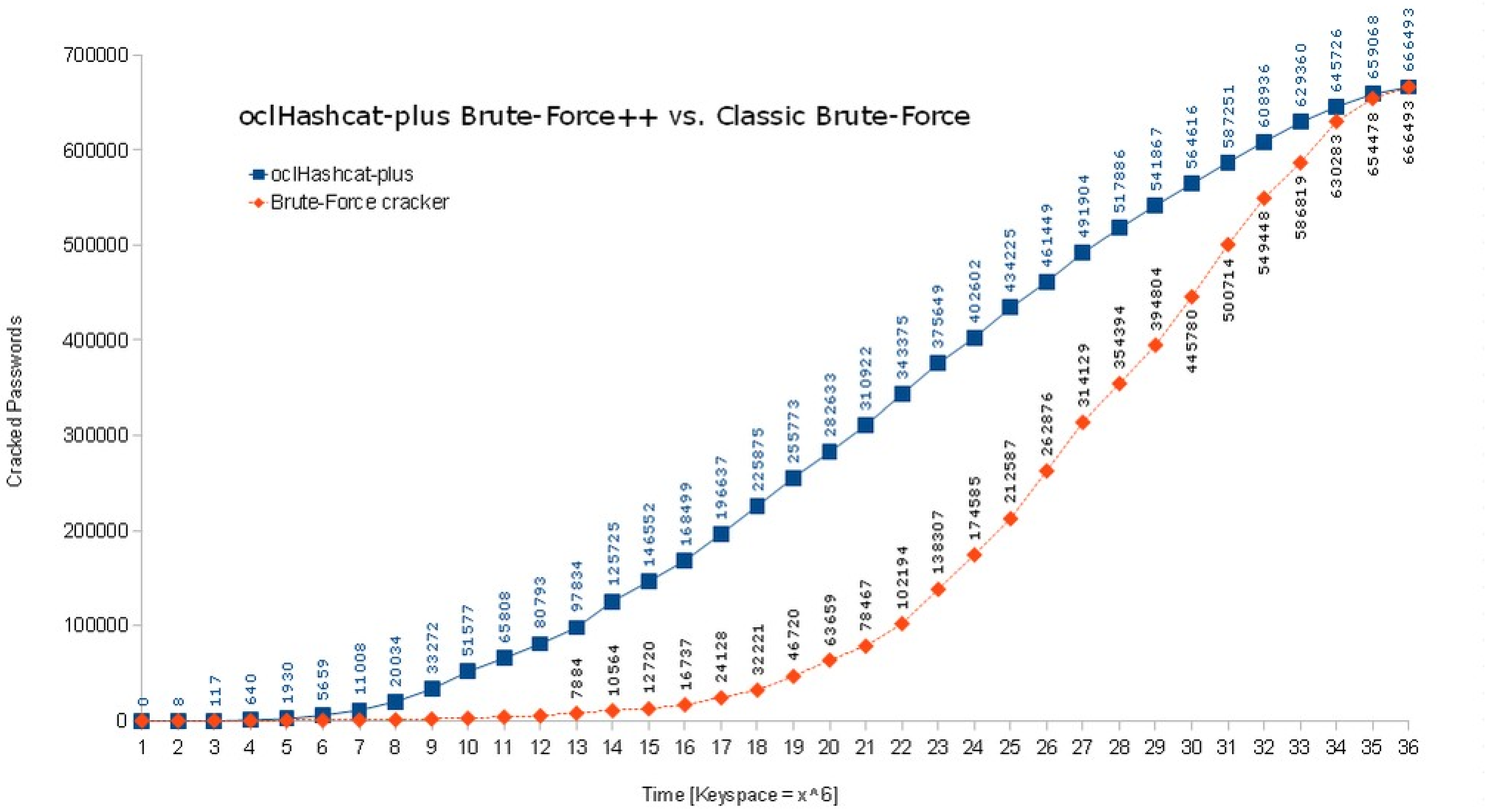
* markov attack

    * hashcat builds **markov chain**

    * advanced bruteforce attack using **patterns and probability**

    * statistically **will find password quicker** than bruteforce

HASHCAT: MARKOV



SAMPLE ATTACK

SAMPLE ATTACK: 1st ROUND

* BF **1-6 characters** (entire character space)

* BF **7-8 characters** (lower letters only)

* BF **7-8 characters** (uppercase letters only)

* BF **1-12 characters** (digits only)

SAMPLE ATTACK: 1st ROUND

* straight **dictionary attack**

* **rule based attack**

    * capitalize first letter

    * add digits at the end

    * l33t

    * reverse

SAMPLE ATTACK: 2nd ROUND

* various **hybrid attacks**

* word from list + all possible 3-character long (digit or symbol) strings

* word from list + all possible 4-character long (digit) strings

* word from list + all possible 4-character long (lowercase letters) strings

SAMPLE ATTACK: 3rd ROUND

* use statistic to build markov chain

* **extended word lists**

* custom rules and masks (**some patterns are already visible**)

* **combinator attack**

RESULTS

k1araj0hns0n
Sh1a-labe0uf
Apr!l221973
Qbesancon321
DG091101%
@Yourmom69
ilovetofunot
windermere2313
tmdmmj17

BandGeek2014
all of the lights
i hate hackers
allineedislove
ilovemySister31
iloveyousomuch
Philippians4:13
Philippians4:6-7
qeadzcwrsfxv1331

PASSWORD STRENGTH

PASSWORD STRENGTH

* **measured in bits**

* strength of **random passwords can be estimated**

* strength of human-generated password is **very difficult to estimate**

* **penetration testing** is useful, real life tests

FACTORS TO CONSIDER

* two factors to consider in determining strength

    * the average **number of guesses to exhaust**

    * how many **guesses per second**

FACTORS

* depends on user

* depends on developer

* depends on attacker

DEPENDS ON USER

* **complexity of password**

    * **length & randomness**

    * character classes and patterns

* can be controlled up to certain level by password policy

* **reuse yes/no**

DEPENDS ON DEVELOPER

* password storage
    * **hash** function **fast/slow**
    * **key stretching on/off**

DEPENDS ON ATTACKER

* **hardware involved**

* **identified** system **limitations**

* quality of **dictionaries**

* identified **other system vulnerabilities** (old backups under web root?)

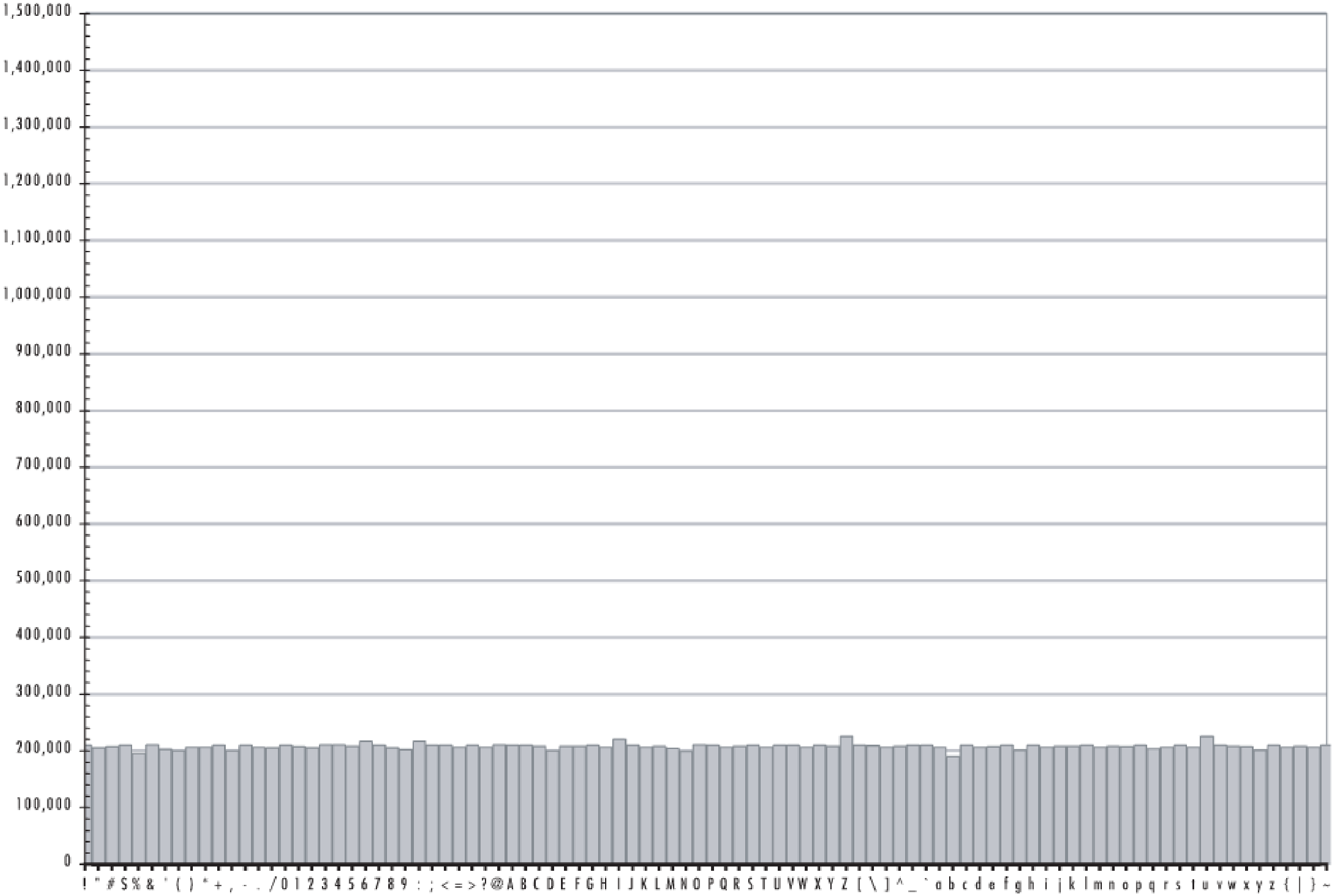
RANDOMNESS

* humans tend to **follow patterns**

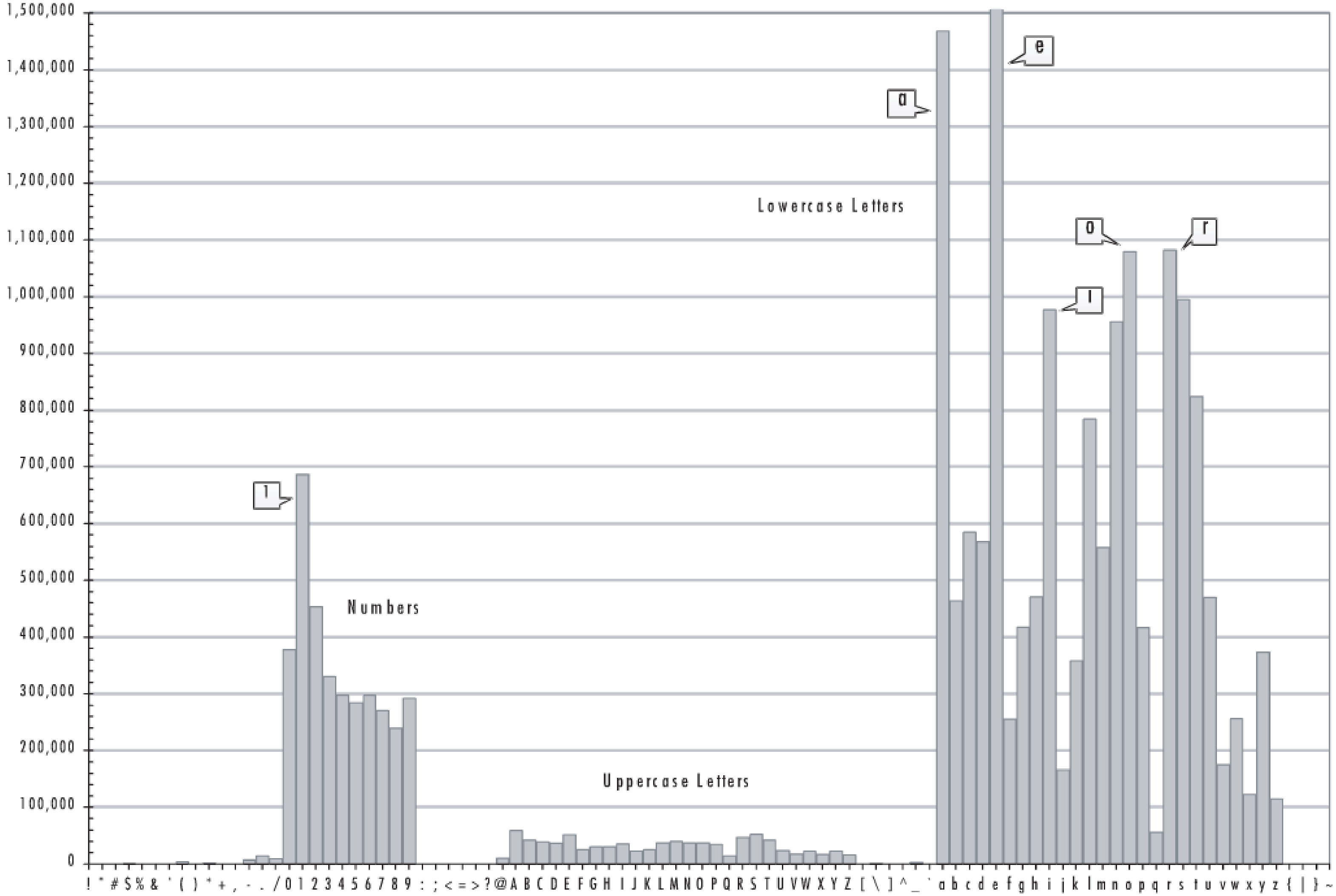
* are **unable to achieve sufficient entropy**

* users **rarely make full use of key space**

RANDOM DISTRIBIUTION



HUMAN PASSWORDS



WEAK PASSWORDS

* single dictionary words

* words with numbers appended

* obfuscated words (adds very little entropy)

* doubled words

* common keyboard sequences: qwerty, 123456

WEAK PASSWORDS

* any purely numeric passwords

* identifiers, usernames, emails

* anything personally related to an individual

    * license plate, phone numbers, addresses

    * dates, birthdays

    * names, nicknames, initials

WEAK PASSWORDS

kitty	1Kitty	1Ki77y
susan	Susan53	.Susan53.
jellyfish	jelly22fish	J3lly22Fish
smellycat	sm3llycat	$m3llycat.
allblacks	AllBlacks!	A11B1ack$!
jackbauer	jAckBauer	jA(kBauer
doctorhouse	Doct0rH0use	.Doct0rH0use.
adamsandler	adamSandler	#adamS@ndler
ilovemypiano	ILoveMyPiano	ILov3MyPi@no

STRONG PASSWORDS

```
$pwgen -n 15 -sy

*?/#C"*:8lq1:jV
.n'rUXJ+jcZ\%D9
7qvmh*O.q$:P$\M
oO8seLzCUbN}h#p
#-5L=UBd6!%vH4G
```

```
$pwgen -n 15 -s

cn9KgidMrOD0zjh
Xc4dXxuZpImQFOp
NvC0xBPt60VRMmk
FgUwSOsJl5Prw8V
VE2zQMO2gQaoiQL
```

PASSPHRASES

* **strong** and **memorable** secrets

* **short phrases will be cracked**

* **avoid popular phrases**, quotes, lyrics

* **introduce some variation** (mixed case, digits, specials)

* **dice word method**

PASSWORD MANAGERS

PASSWORD MANAGERS

* **too many accounts**

* people **can not remember** so many **good passwords**

* secure way of storing passwords

* provide tools to **generate strong passwords**

KEEPASS

* **database encryption** (AES 256 bits, CBC)

* SHA-256 key derivation method (256 bit key)

* **key stretching** (6000 SHA-256 iterations)

* **secure random number generation**

KEEPASS

* **process memory protection**, passwords are encrypted in RAM

* overwrites data before releasing security-critical memory cells

* locking database, **integrity tests**

* **2-factor authentication**

KEEPASS

* **2-channel auto-type obfuscation**

    * it sends **simulated key presses** to other applications

    * part of the sensitive information is **transferred via clipboard**, the rest by **sending keypresses**

GUIDELINES

4 USERS

* generate **random passwords**

* **do not reuse**

* use **password manager**

* backup your passwords (offline/offsite)

* use **memorable passphrases**

4 USERS

* **avoid** known **patterns**

* **change default passwords**

* **change password if compromised**

* **turn on 2-factor authentication** if feasible

4 PROGRAMMERS

* use **bcrypt**, **scrypt** or **pbkdf2** hash functions

* use **dynamic salts**

* require **sufficient password complexity**

* **do not enforce** very **strict patterns**

* **do not implement** anything **yourself**

CHECKLIST

* check hashes, **passwords can not be stored encrypted or in plain text**

* check if **hash function** is **still safe** (MD5/SHA1, big no no)

* check if **hashes are salted** with dynamic salt

* check if **key stretching is strong** (work factor is still sufficient)

CHECKLIST

* check if required **password complexity is sufficient**

* check if system allows obvious passwords (empty, equal to login, name or email)

* check if account is **locked down after 5 failed attempts**

REFERENCES

REFERENCES: WEB

* sekurak.pl

* arstechica.com

* troyhunt.com

* haveibeenpwned.com

* keepass.info

* wikipedia.org

REFERENCES: BOOKS

* Take Control of Your Passwords

* Perfect Password