Quantitative Cyber-Security

Colorado State University
Yashwant K Malaiya

CS559

L17: Coverage, Passwords

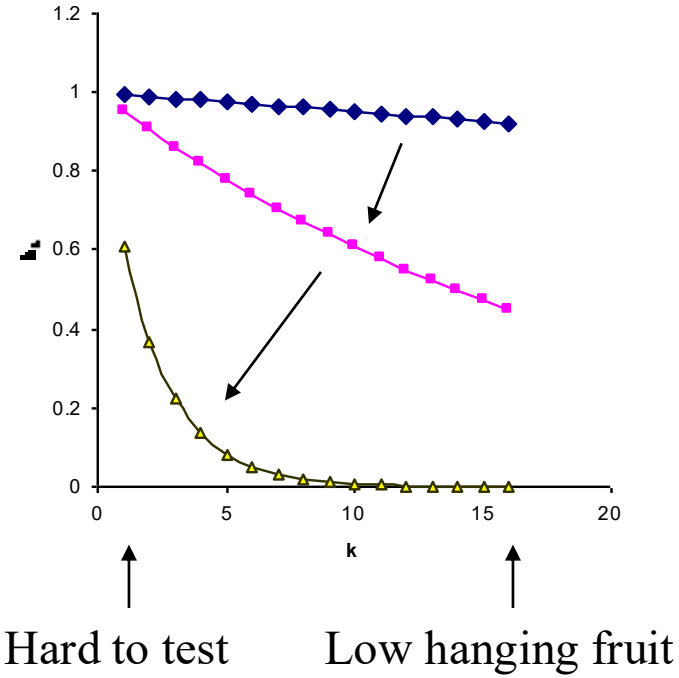


CSU Cybersecurity Center
Computer Science Dept

Detectability Profile: Software

- Software detectability profile is exponential
- Justification: Early testing will find & remove easy-to-test faults.
- Testing methods need to focus on hard-to-find faults.

As testing time progresses, more of the faults are clustered to the left.



Directed testing

Testing may be directed rather than random because

- Tester may wish to focus on functionality not adequately exercised by random testing (for example recovery code)
- Tester may wish to focus on more critical sections of the code.
- The probability of detecting a fault can be give by p_i, where p_i may be *greater or less than* k/N.

  P{a defect with dp p_i *not* detected by L vectors} = $(1 - p_i)^L$

- Where $p_i > \frac{k}{N}$ if the previous tests are not repeated, or the test has a good idea of where to look.
- When the *exhaustive set* (ES) of inputs are applied, then P{a defect with dp p_i not detected by ES} ≈ 0
  - Unlikely in most real situations.

Some common SRGM models

- Several models for ordinary bug finding process. Termed **Software Reliability Growth Models** (SRGMs).
- **Exponential SRGM**: assumes bug finding rate $\lambda(t)$ is proportional to remaining bugs at time N(t).

$$\lambda(t) = -\frac{dN(t)}{dt} = \beta_1 N(t)$$

- Which has the solution

$$\lambda(t) = \beta_0 \beta_1 e^{-\beta_1 t}$$

- Where β_0 and β_1 are parameters to be determined. B_0 represents the initial number of bugs and β_1 a measure of test effectiveness.

Defect Density

- Exponential defect finding model is
$$\lambda(t) = \beta_0 \beta_1 e^{-\beta_1 t}$$
- β_0 represents the initial number of bugs.
- If the initial *defect density* is D(0), and the software size (measured in 1000 lines of code, i.e. KLOC) is S, then
$$\beta_0 = D(0) \times S$$
- The initial defect density is a function of the software development process and the degree of prior defect removal.
- The defect finding rate gradually declines, it takes infinite time to find them all according to the exponential model.
- The final defect density is sometimes used as a release criterion.

SRGM : "Logarithmic Poisson"

- If testing combines random and directed testing, the Logarithmic Poisson arises.
- **Logarithmic Poisson** model, by Musa-Okumoto, has been found to have a good predictive capability

$$\mu(t) = \beta_o \ln(1 + \beta_1 t) \qquad \lambda(t) = \frac{\beta_o \beta_1}{1 + \beta_1 t}$$

- Applicable as long as $\mu(t) \leq N(0)$. Practically always satisfied.
- Parameters β_o and β_1 don't have a simple interpretation. An interpretation has been given by Malaiya and Denton (What Do the Software Reliability Growth Model Parameters Represent?).

References

- Y. K. Malaiya, S. Yang, “The Coverage Problem for Random Testing,” IEEE International Test Conference 1984, pp. 237-245.
- Y.K. Malaiya, A. von Mayrhauser and P. Srimani, “An Examination of Fault Exposure Ratio,” IEEE Trans. Software Engineering, Nov. 1993, pp. 1087-1094.
- S. C. Seth, V. D. Agrawal, H. Farhat, "A Statistical Theory of Digital Circuit Testability," IEEE Trans. Computers, 1990, pp. 582-586.
- K. Wagnor, C. Chin, and E. McCluskey, “Pseudorandom testing. IEEE Trans. Computer, Mar. 1987, pp. 332—343.
- E. N. Adams, "Optimizing Preventive Service of Software Products," in IBM Journal of Research and Development, vol. 28, no. 1, pp. 2-14, Jan. 1984.
- J R Dunham, "Experiments in software reliability: Life-critical applications," IEEE Tran. SE, January 1986, pp. 110 - 123
- H. Hashempour, F.J. Meyer, F. Lombardi,, "Analysis and measurement of fault coverage in a combined ATE and BIST environment," Instrumentation and Measurement, IEEE Transactions on , vol.53, no.2, pp.300,307, April 2004.
- Y. K. Malaiya, "Assessing Software Reliability Enhancement Achievable through Testing", Recent Advancements in Software Reliability Assurance 2019, pp. 107-138

Quantitative Security

Colorado State University

Yashwant K Malaiya

CS 559

Coverage based approaches



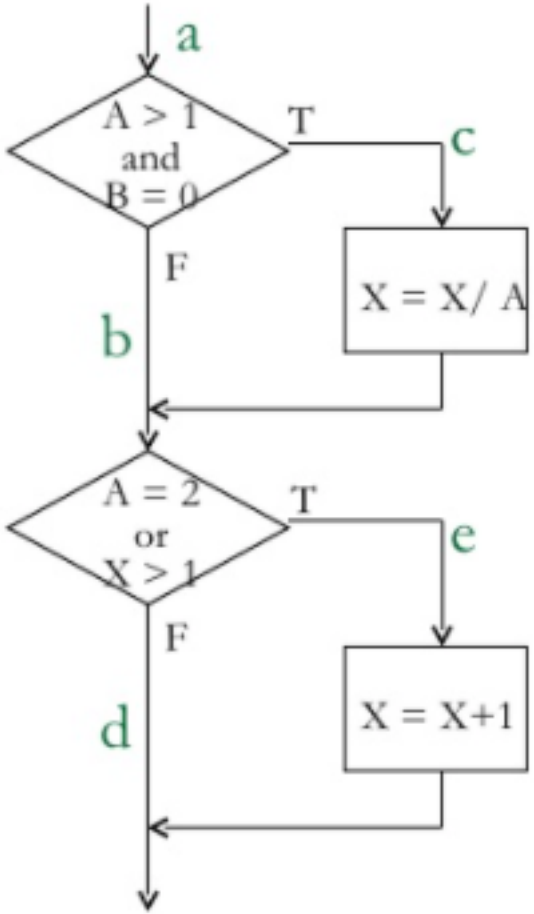
CSU Cybersecurity Center
Computer Science Dept

Test Coverage Measures

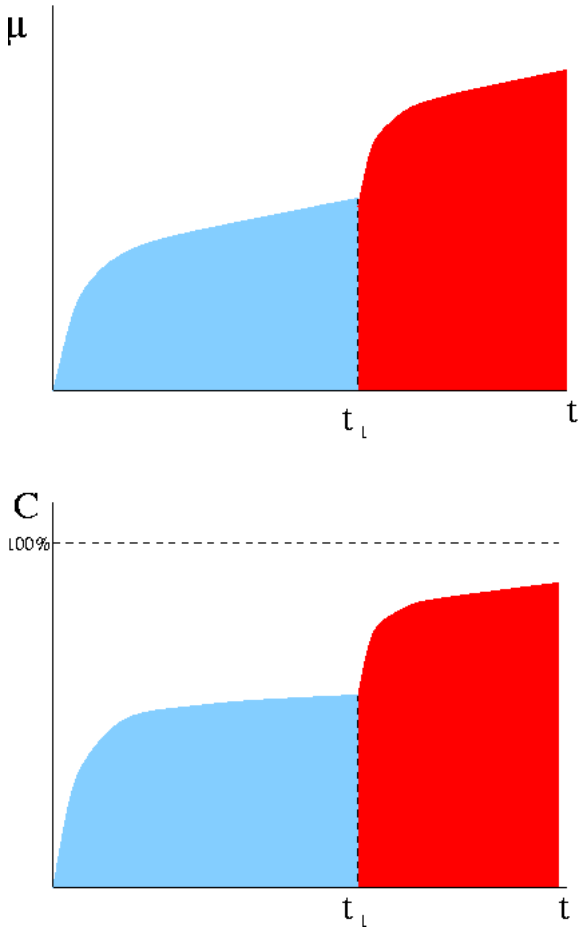
- Structiral:
  - Statement or Block coverage
  - Branch or decision coverage
- Data-flow:
  - P-use coverage: p-use pair: variable defined/modified - use as predicate
  - C-use coverage: similar -use for computation
- Subsumption hierarchy:
  - Covering *all branches* cover *all statements*
  - Covering *all p-uses* cover *all branches*

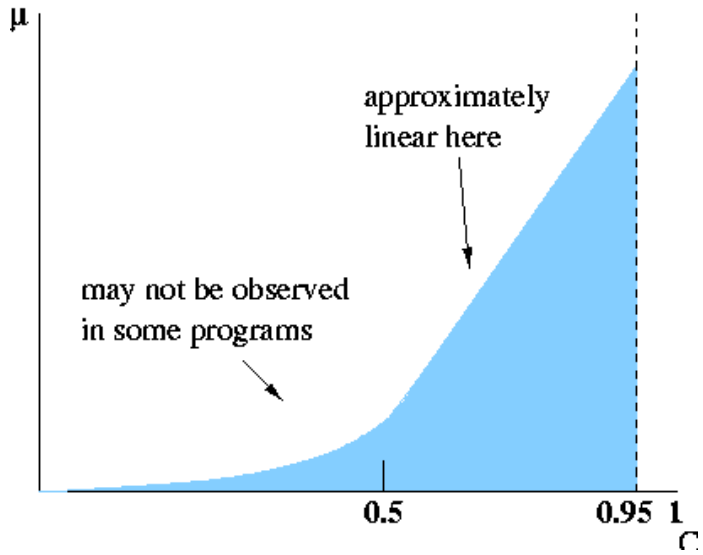
Test Coverage Measures

- Test case A =2, B=0, X=4
  - Covers branches a, c, e
  - Covers all the statements
- Test case A =1, B=1, X=1
  - Covers branches a, b, d
- Two test cases for 100% branch coverage.



Modeling : Defects, Time, & Coverage





Malaiya, Li, Bieman, Karcich, Skibbe, 1994
Li, Malaiya, Denton, 1998

Coverage Based Defect Estimation

- Coverage is an objective measure of testing
  - Directly related to test effectiveness
  - Independent of processor speed and testing efficiency
- Lower defect density requires higher coverage to find more faults
- Once we start finding faults, expect coverage vs. defect growth to be linear

Logarithmic-Exponential Coverage Model

- Hypothesis 1: defect coverage growth follows logarithmic model

$$C^0(t) = \frac{\beta_0^0}{N^0} \ln(1 + \beta_1^0 t), \quad C^0(t) \le 1$$

- Hypothesis 2: test coverage growth follows logarithmic model

$$C^i(t) = \frac{\beta_0^i}{N^i} \ln(1 + \beta_1^i t), \quad C^i(t) \le 1$$

Log-Expo Coverage Model (2)

- Eliminating t and rearranging,

$$C^0 = a_0^i \ln[1 + a_1^i(\exp(a_2^i C^i) - 1)], \quad C^0 \le 1$$

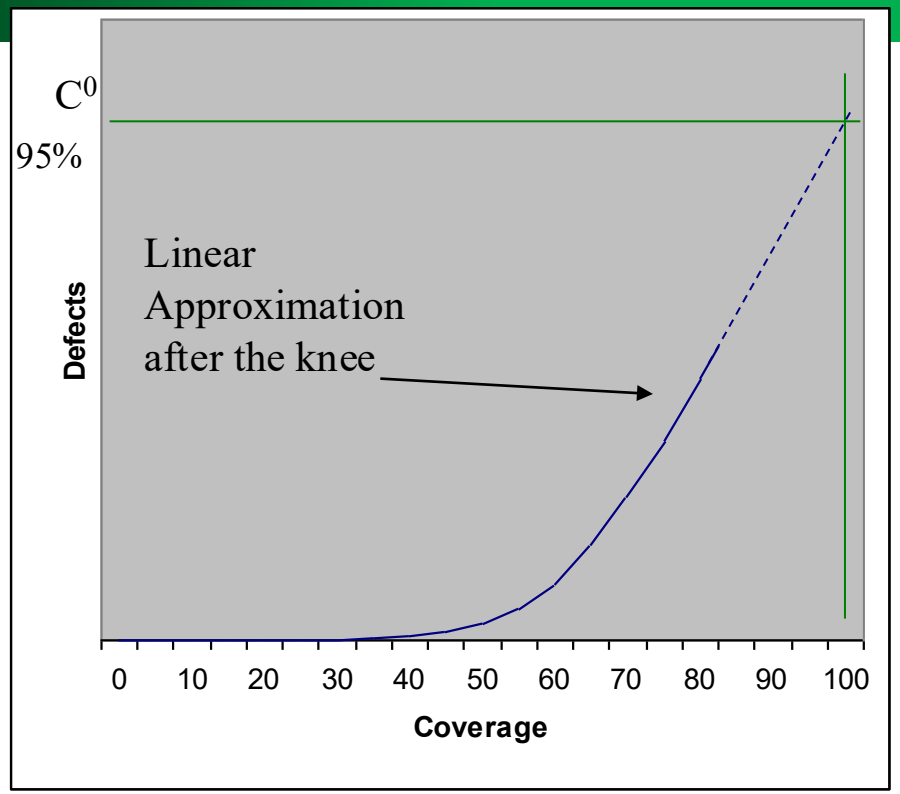
where C^0 : defect coverage, C^i : test coverage

a_0^i, a_1^i, a_2^i : parameters ; i : branch cov, p - use cov etc.

- For "large" Ci, we can approximate

$$C^0 = -A^i + B^i C^i$$

Coverage Model, Estimated Defects



$$C^0 = -A^i + B^i C^i, \quad C^i > C^i_{knee}$$

- Only applicable after the knee
- Assumptions : Stable Software

Location of the knee

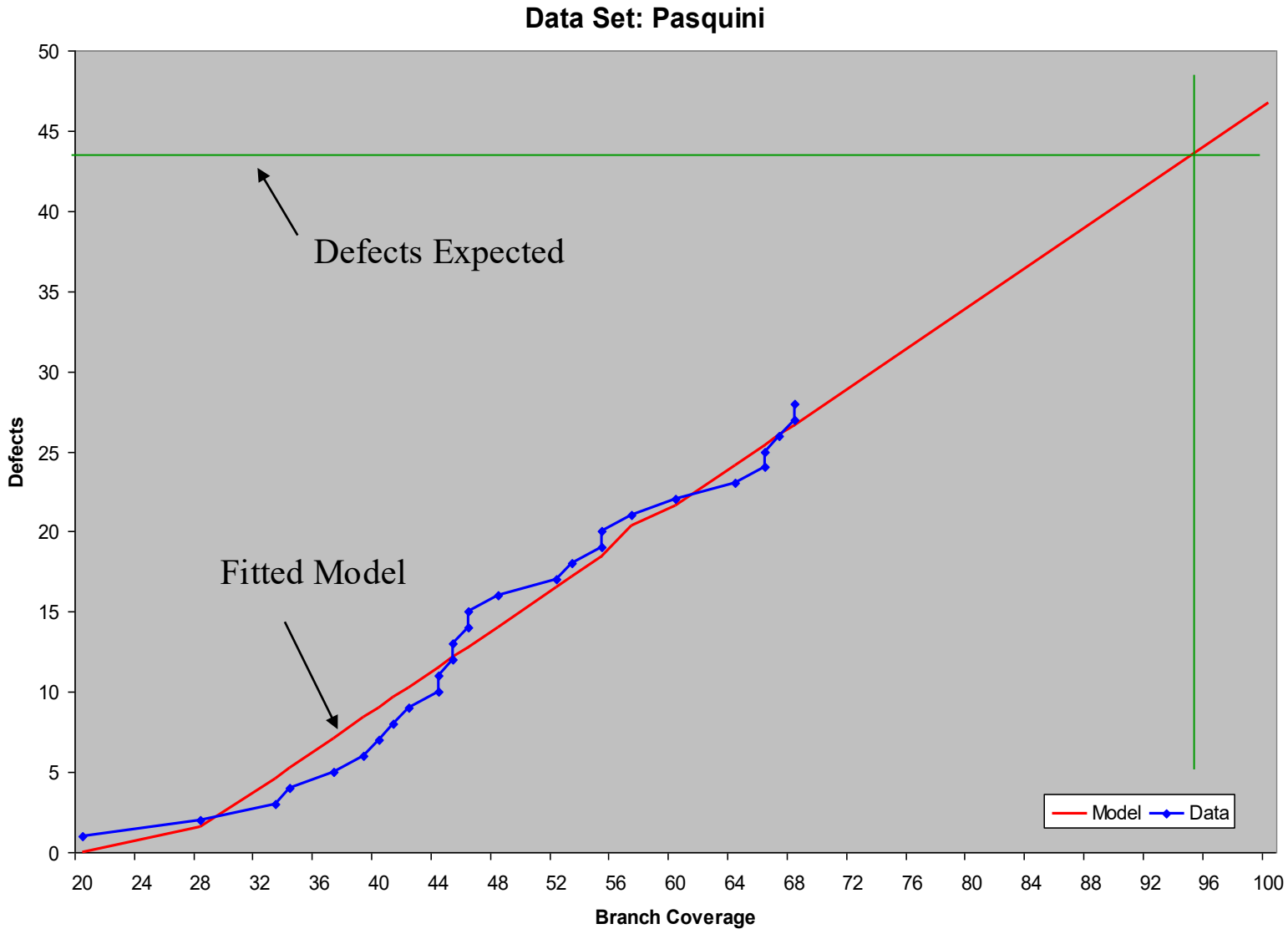
$$C_{knee} \quad 1 - \left(\frac{E_{min}}{D_{min} E_0} \right) \quad D_0$$

- Based on interpretation through logarithmic model
- Location of knee based on initial defect density
- Lower defect densities cause knee to occur at higher coverage
- Parameter estimation : Malaiya and Denton (HASE '98)

Data Sets Used: Vouk and Pasquini

- Vouk data
  - from N version programming project to create a flight controller
  - Three data sets, 6 to 9 errors each
- Pasquini data
  - Data from European Space Agency
  - C Program with 100,000 source lines
  - 29 of 33 known faults uncovered

Defects vs. Branch Coverage



Authentication

- Authentication: the process of verifying an actor's identity
- Needed for security of systems
  - Permissions, capabilities, and access control are all contingent upon knowing the identity of the actor
- Parameterized as a username and a user's proprietary information
  - The proprietary information attempts to limit unauthorized access

Types of proprietary info

- Actors provide their proprietary information to login to a system
- Three classes of proprietary info:
  1. Something you know
     - Example: a password
  2. Something you have
     - Examples: smart phone, a smart card or authentication fob
  3. Something you are
     - Examples: fingerprint, voice scan, iris scan, face
  4. Somewhere you are (& IP address, device type, OS being used)

Checking Passwords

- The system must validate passwords provided by users
- Thus, passwords must be stored somewhere
- Simple scheme: plain text (is this good?)

password.txt

cbw	p4ssw0rd
sandi	i heart doggies
amislove	93Gd9#jv*0x3N
bob	security

Problem: Password guessing

How easy it is to guess a password?

- If your keyboard has R= 95 unique characters,
- randomly constructing a password from that whole set, 12-character password, then L = 12.
- 95^{12}= 540,360,087,662,636,962,890,625 passwords

Entropy = $\log_2(R^L)$ = 78.9 bits assuming passwords are created randomly

Ascii is 8 bits. Thus about 2^{12x8}

- Non-randomness makes password guessing easier.
- Measures of password strength proposed and used

Password guessing at login? Can be defeated by

- Limited number of tries: 3-5
- Blocking attempts from unknown/suspected IP addresses

Problem: Password File Theft

- **Attackers often compromise systems**
- They are often able to steal the password file
  - Linux: /etc/shadow
  - Windows: c:\windows\system32\config\sam
- If the passwords are plain text, what happens?
  - The attacker can now log-in as any user, including root/administrator
- Thus, Passwords should never be stored in plain text, but using ..

Meta pays the price for storing hundreds of millions of passwords in plaintext, 9/27/2024 .. Ireland have fined Meta $101 million

Research is not homework

- Research represents continuous effort. If you can finish a report in a few hours on a single day, it is not research.
- Summarizing a few articles, without any interconnections, classification or trends is not research.
- Copying and pasting text and using AI generated text is plagiarism.

In class question

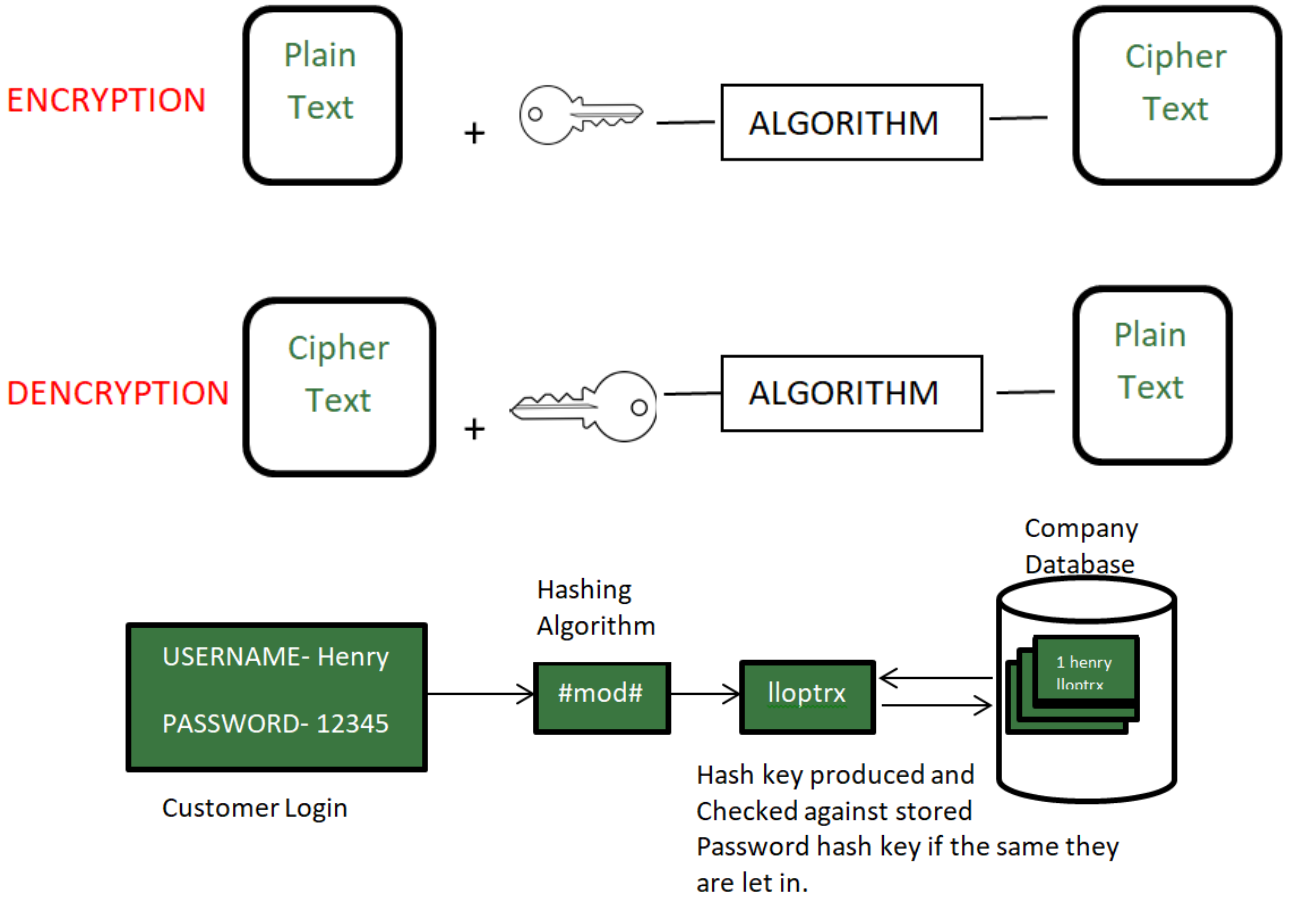
A system allows the following in the password

- Uppercase letters (A through Z)
- Lowercase letters (a through z)
- Numbers (0 through 9)
- The following non-alphabetic characters: ` ~ ! @ # $ % ^ & * ( ) _ + - = { } | [ ] \ : " ; ' < > ? , . / (32)

What is the total number of passwords if the password is 8 characters? What is the entropy ($\log_2(R^L)$)?

Answer:

Encryption vs Hashing



- Impossible to reconstruct password from hash but ..

Colorado State University

Hashed Passwords

- Key idea: store encrypted versions of passwords
  - Use one-way cryptographic hash functions
  - Examples: md5, sha1, sha256, sha512 etc (128, 160, 256, 512 bits)
- Cryptographic hash function transform input data into scrambled output data
  - Deterministic: hash(A) = hash(A)
  - High entropy: small change, significant effect
    - md5('security') = e91e6348157868de9dd8b25c81aebfb9
    - md5('security1') = 8632c375e9eba096df51844a5a43ae93
    - md5('Security') = 2fae32629d4ef4fc6341f1751b405e45
  - Collision resistant
    - Locating A' such that hash(A) = hash(A') takes a long time
    - Example: 2^{21} tries for md5

Ack: Northeastern U

Hashed Password Example



User: cbw



md5('p4ssw0rd') =
2a9d119df47ff993b662a8ef36f9ea20

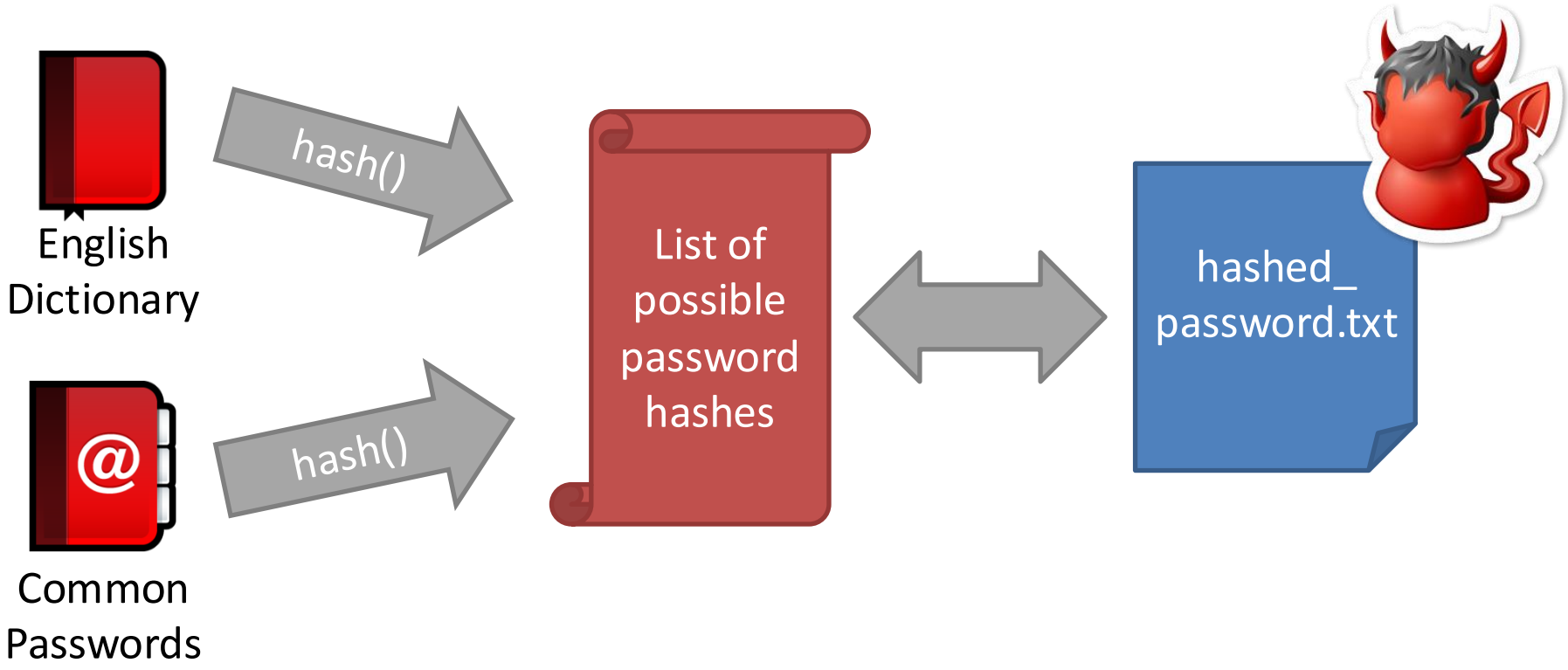
md5('2a9d119df47ff993b662a8ef36f9ea20')
= b35596ed3f0d5134739292faa04f7ca3

hashed_passw…xt	
cbw	2a9d119df47ff993b662a8ef36f9ea20
sandi	23eb06699da16a3ee5003e5f4636e79f
amislove	98bd0ebb3c3ec3fbe21269a8d840127c
bob	e91e6348157868de9dd8b25c81aebfb9

Attacking Password Hashes

- Problem: users choose poor passwords
  - Most common passwords: 123456, password
  - Username: cbw, Password: cbw
  - Common password patterns: 111111, qwerty, ..
- Default passwords (*password*, *default*, *admin*, *guest etc*) if not changed can be a security hazard.
- Weak passwords enable dictionary attacks using Rainbow tables
  - Pre-computed tables of common passwords, dictionary entries etc and their hashes
  - Hundreds of gigabytes or terabytes of data

Dictionary Attacks



- Common for 60-70% of hashed passwords to be cracked in <24 hours [Free Password Hash Cracker]

Pwned Passwords

- Pwned Passwords are **501,636,842** real world passwords previously exposed in data breaches. They're searchable online:
- https://haveibeenpwned.com/Passwords
- The searches are anonymized using a mathematical property called *k*-anonymity

Hardening Password Hashes

- Key problem: cryptographic hashes are deterministic
  - hash('p4ssw0rd') = hash('p4ssw0rd')
  - This enables attackers to build lists of hashes
- Solution: make each password hash unique
  - Add a **salt** to each password before hashing
  - hash(salt + password) = password hash
  - Each user has a *unique, random* salt
  - Salts can be stores in plain text

Example Salted Hashes

hashed_password.txt

cbw	2a9d119df47ff993b662a8ef36f9ea20
sandi	23eb06699da16a3ee5003e5f4636e79f
amislove	98bd0ebb3c3ec3fbe21269a8d840127c
bob	e91e6348157868de9dd8b25c81aebfb9

hashed_and_salted_password.txt

cbw	a8	af19c842f0c781ad726de7aba439b033
sandi	0X	67710c2c2797441efb8501f063d42fb6
amislove	hz	9d03e1f28d39ab373c59c7bb338d0095
bob	K@	479a6d9e59707af4bb2c618fed89c245

Attacking Salted Passwords



hash()

List of possible password hashes

No matches



hashed_ and_salted_ password.txt

cbw	a8
sandi	0X
amislove	hz
bob	K@



hash('0X' + word)

List of possible password hashes w/ salt *0X*

c

sandi YYYY

Breaking Hashed Passwords

- **Stored password hashes should always be salted**
  - Forces the attacker to brute-force each password individually
- Problem: it is now possible to compute cryptographic hashes very quickly
  - GPU computing: hundreds of small CPU cores
  - nVidia GeForce GTX Titan Z: 5,760 cores
  - GPUs can be rented from the cloud very cheaply
    - 2x GPUs for $0.65 per hour (2014 prices)

Examples of Hashing Speed

- A modern x86 server can hash all possible 6 character long passwords in 3.5 hours
  - Upper and lowercase letters, numbers, symbols
  - $(26+26+10+32)^6$ = 690 billion combinations
- A modern GPU can do the same thing in 16 minutes
- Most users use (slightly permuted) dictionary words, no symbols
  - Predictability makes cracking much faster
  - Lowercase + numbers → $(26+10)^6$ = 2B combinations

Hardening Salted Passwords

- Problem: typical hashing algorithms are too fast
  - Enables GPUs to brute-force passwords
- Solution: use hash functions that are designed to be **slow**
  - Examples: bcrypt, scrypt, PBKDF2
  - These algorithms include a work factor that increases the time complexity of the calculation
  - scrypt also requires a large amount of memory to compute, further complicating brute-force attacks

bcrypt Example

- Python example; install the *bcrypt* package

```
[cbw@ativ9 ~] python
>>> import bcrypt
>>> password = "my super secret password"
>>> fast_hashed = bcrypt.hashpw(password, bcrypt.gensalt(0))
>>> slow_hashed = bcrypt.hashpw(password, bcrypt.gensalt(12))
>>> pw_from_user = raw_input("Enter your password:")
>>> if bcrypt.hashpw(pw_from_user, slow_hashed) == slow_hashed:
...         print "It matches! You may enter the system"
...   else:
...         print "No match. You may not proceed"
```

Work factor

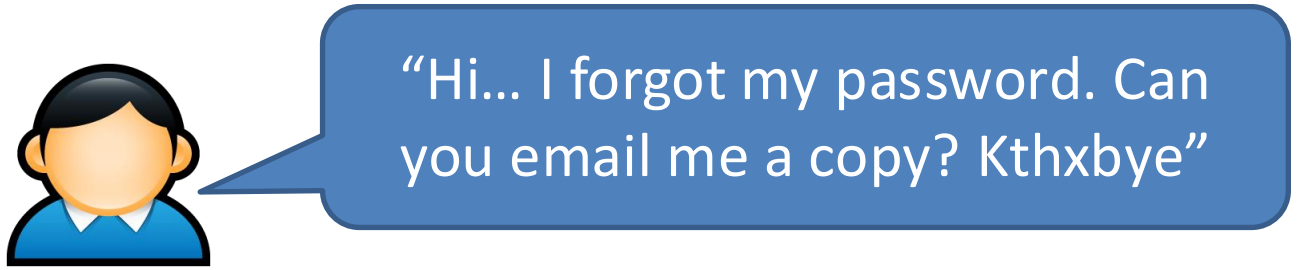
Password Storage Summary

1. **Never store passwords in plain text**
2. **Always salt and hash passwords before storing them**
3. **Use hash functions with a high work factor**

- These rules apply to any system that needs to authenticate users
  - Operating systems, websites, etc.

Password Recovery/Reset

- Problem: hashed passwords cannot be recovered



- This is why systems typically implement password reset
  - Use out-of-band info to authenticate the user
  - Overwrite hash(old_pw) with hash(new_pw)
- Be careful: its possible to crack password reset

Password crackers

Techniques:

- **Random Guesses**
- **Dictionary Attacks**
- **Brute Force:** programmatic method to try all possible combinations
- **Credential Stuffing (stolen passwords)**
- **Password Spraying:** attempts a single, commonly used password against many accounts before proceeding to attempt a second password, thus avoiding account lockouts.
- **Phishing & Vishing** (voice calls)
- **Forced, Automated Password Changes and Resets**
- **Eavesdropping/Shoulder Surfing**

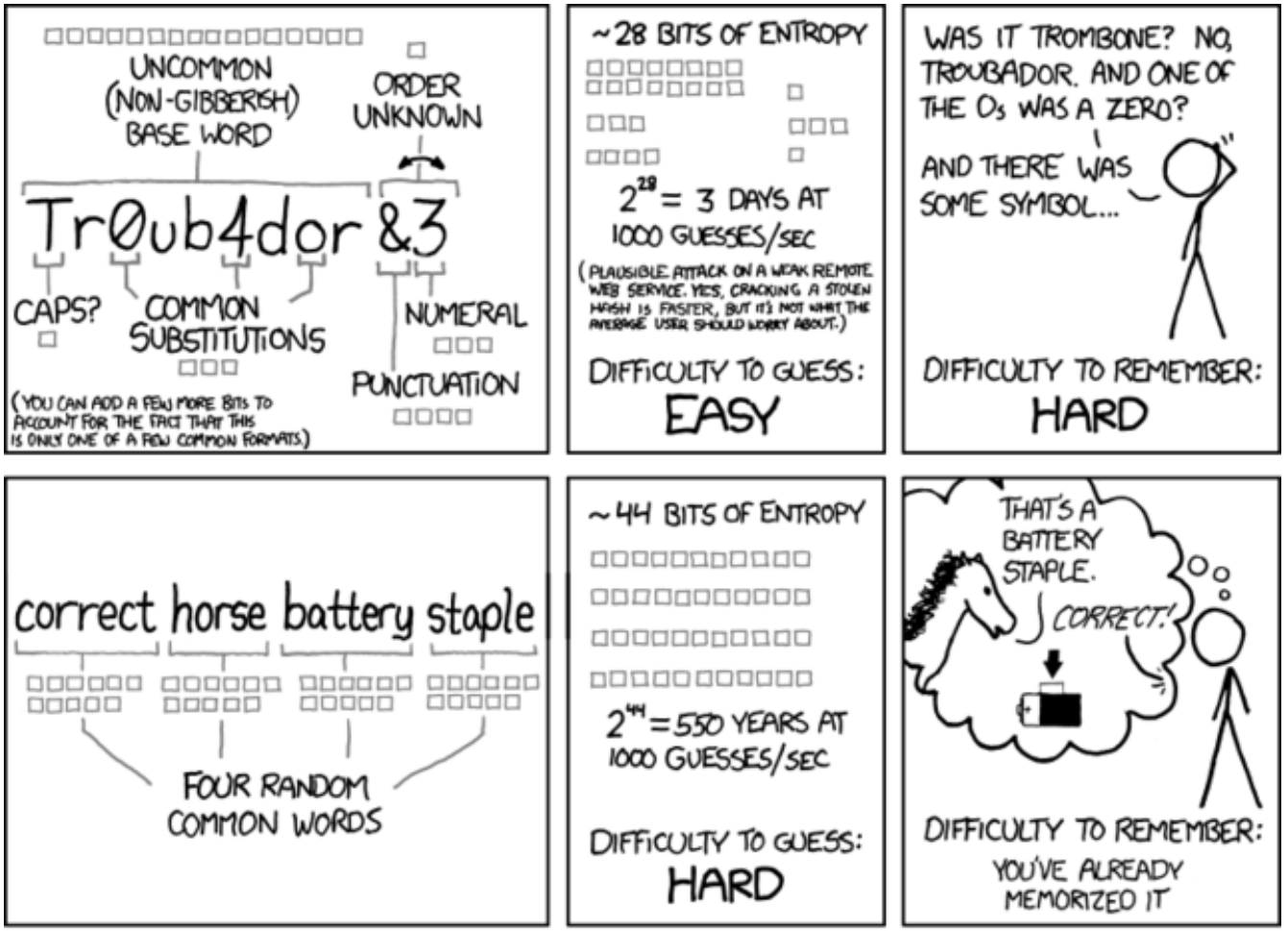
Common tools: Cain and Abel, John the Ripper, Hydra, Hashcat, Ophcrack

https://www.beyondtrust.com/blog/entry/password-cracking-101-attacks-defenses-explained

Forgotten passwords

- Too many passwords to remember
- “Strong” passwords can be hard to remember
- Traditional approach: user physically requests password reset
  - Using phone numbers or email addresses on record
  - Showing IDs
- Danger: fraudulently obtaining password using social engineering. May represent the weakest link in the password system.

Good passwords are bad



Security Questions

- Security Questions are used to authenticate when
  - Suspicious attempts
  - Forgotten passwords
- 33-39% may be guessed by family members, friends or those with access to personal information
- 20% of users could not remember their own answers.
- Possible solution: Multiple questions with a minimum threshold of right answers

Multifactor Authentication

- Smartphone with number xxx-xxx-xxxx: one in 10^{10}
  - About 33 bits of entropy
- Fingerprints might be unique. However, information may be lost when 25-80 minutiae are used for comparison. Uniqueness still being researched.
- Face recognition: 97.25% accuracy?

Quantitative Cyber-Security

Colorado State University
Yashwant K Malaiya

CS559

L18b: Password Managers



CSU Cybersecurity Center
Computer Science Dept

Password managers

- Can record username, password, form information etc.

for automatic filling.

  - Locally on a device
  - On the web
- Can generate good passwords
- The master password may be kept locally. If you forget it, you may have to extract it yourself.
- Some browsers may include password management capabilities
- Can protect against keyloggers

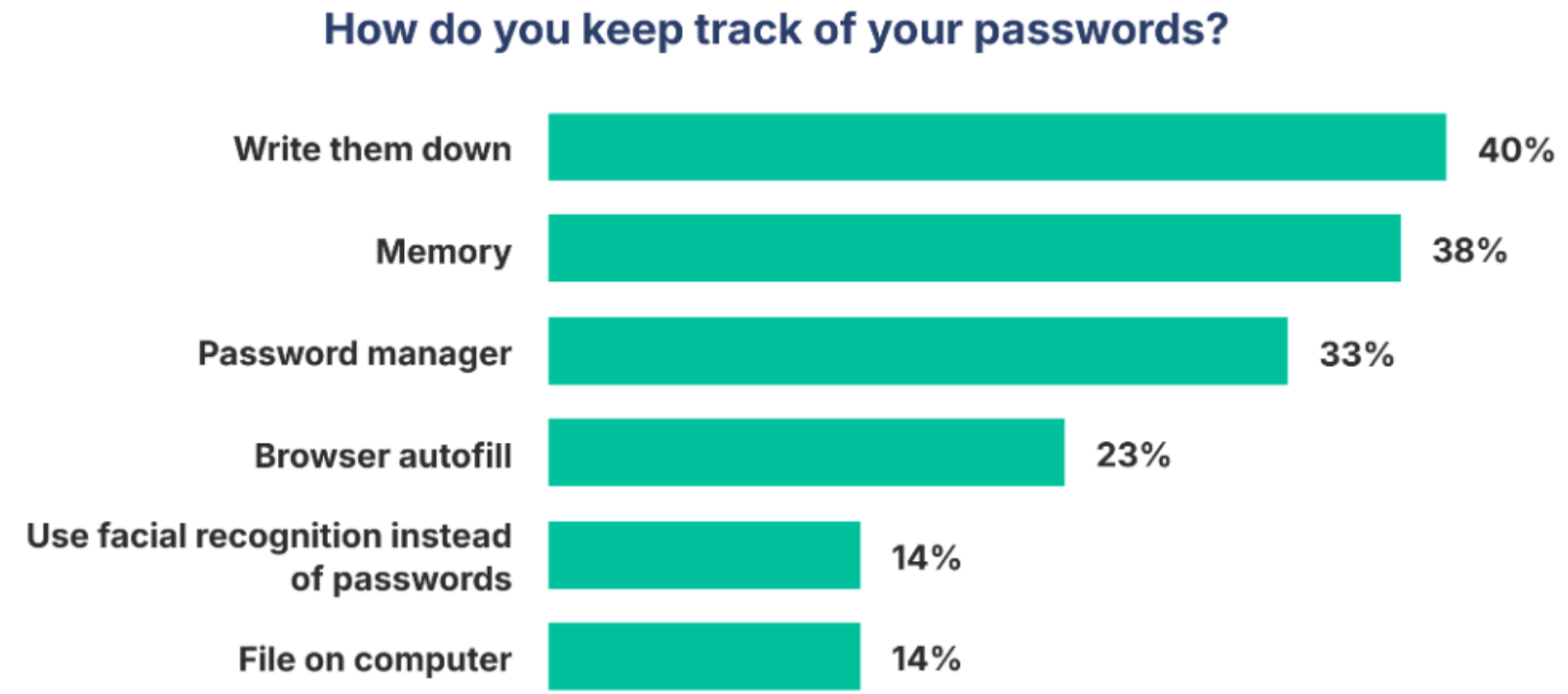
Disadvantages:

  - May have vulnerabilities
  - May be blocked by some websites

Password guidelines

- **Don’t reuse your passwords.** If you reuse the same password and it is discovered by a hacker, then your other accounts are at risk. Choose a unique password for each account.
- **Store your passwords in a secure password manager.**
- **Don’t share your passwords.** If you need to share your password, change it as soon as possible.
- **Change your password frequently.** For sensitive accounts, change your password at least once a year. For all other accounts, change your password anytime you notice suspicious activity.
- **Enable multi-factor authentication.**
- Set up mechanism to allow to reset the password.
- Use a password manager.

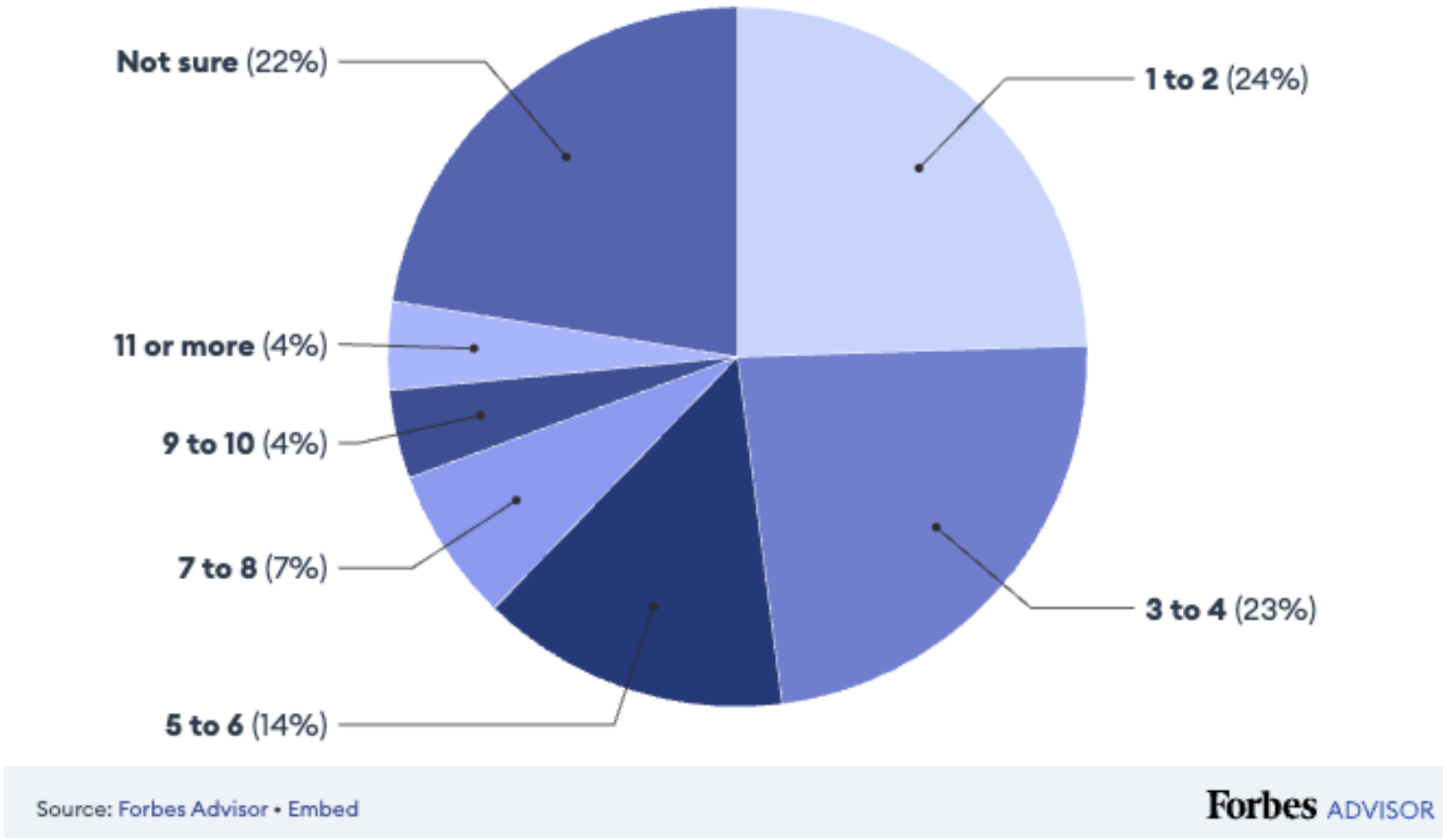
Where people keep their passwords



https://allaboutcookies.org/password-users-behavior-survey

Colorado State University

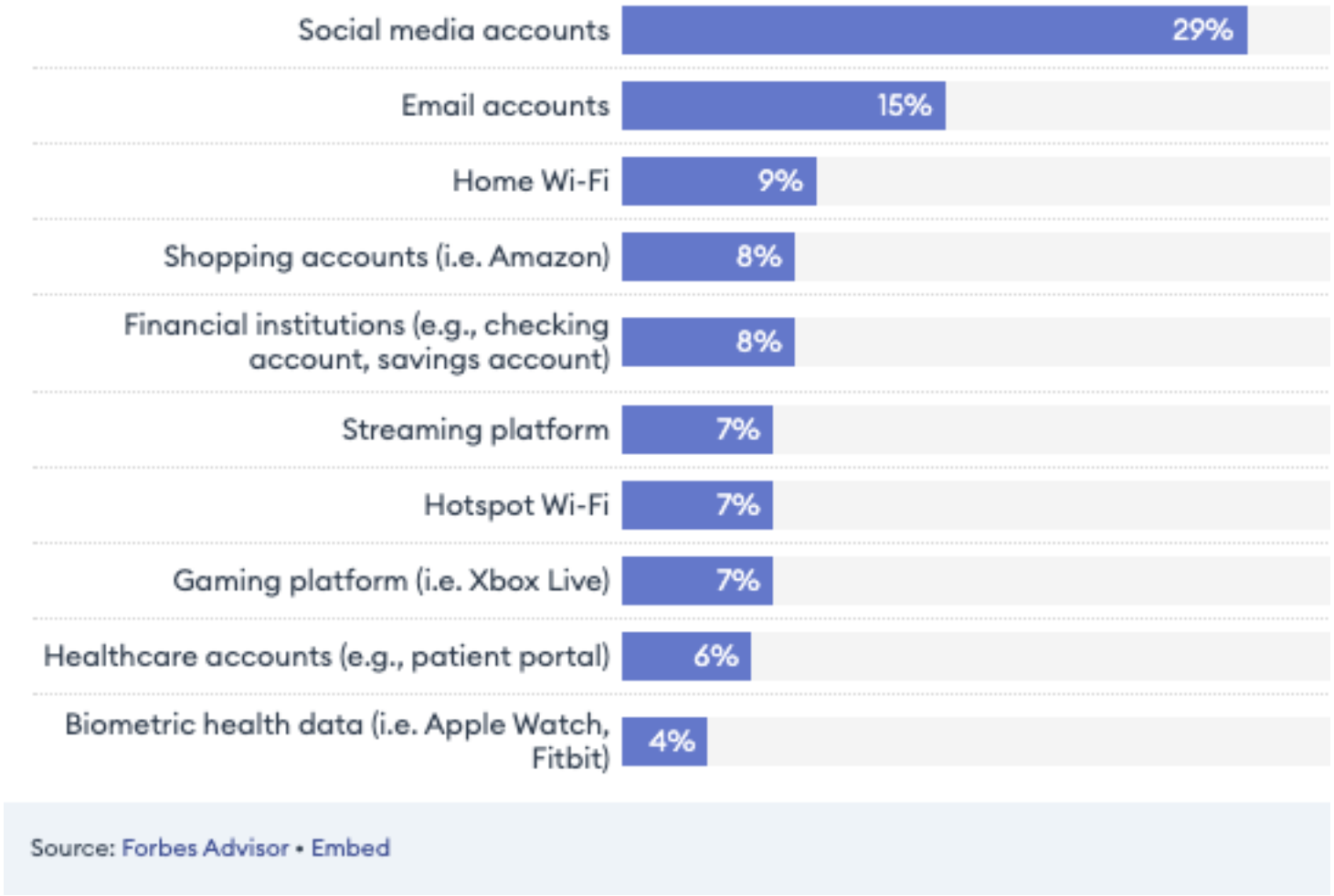
The Number of Accounts Where People Reuse the Same Password



https://www.forbes.com/advisor/business/software/american-password-habits/

Colorado State University

Passwords Are Most Likely to be Stolen



Need for Password Managers

- Average person needs about 168 passwords. NordPass
- Passwords can be easy to guess/crack. Common strings, reused passwords (52-78% of people) , large password collections
- Passwords are easily forgotten. 78% of the people need a resent within 3 month. Recovery can be time consuming, or prone to social engineering-based discovery.
- Password managers can save time. Password Manager Savings Calculator.

Common Password Managers

Name	Introduced	Million users	Free/Paid	Cost/year Ind/business
LastPass	2008	13.12 23.3%	both	36/p, 72/u b
1Password	2006	6.19. 4.5%	14-d trial	36/p, 96/u b
Roboform	1999	4.87 3.3%	paid	24/p, 35/u b
Keeper		3.4%		
Bitwarden	2016	4.59	both	10/p, 60/u b
Dashlane	2012	3.03 4%	both	78/p, 96/u b
Chrome	2015			
Mac				

What password manager do you primarily use?

What password manager do you primarily use?	2021	2024
Google Password Manager	8%	32%
Apple (iCloud Keychain or Passwords app)	7%	23%
LastPass	21%	11%
Bitwarden	8%	10%
1Password	7%	5%
Norton	3%	3%
Keeper	10%	3%
Dashlane	7%	2%
NordPass	3%	1%

https://www.security.org/digital-safety/password-manager-annual-report/

Cloud vs Local Password Storage

Password vaults is stored either

- Locally on the device (or on premise), may be synched with other devices
  - Will be lost with the device
  - May be hacked by malware
  - More secure?
- On the cloud, log in from any device
  - Recoverable if device lost, can be changed
  - May be stolen with a cloud breach
  - May be encrypted/decrypted at the device level (zero-knowledge architecture)

Master Password

Needed to login into the Password Vault

- Passwords in the vault may be lost if forgotten
- Might be possible to reset the Master Password if the recovery code is available.
- Guidelines:
  - Is unique to your password manager
  - Has not been used in other places online
  - Something you can remember, or write down to remember and put in a secure place
  - Strong: long, random, hard to guess
  - Enable two factor authentication
- Not needed to change frequently, unless evidence of a compromise

Password Generation

- Password Managers generate strong passwords with desirable attributes
  - Random combination of characters with symbols. It will be stored and applied thus no need to store
  - 16 characters better than 12
  - `kdH/3yBqXa*4n7r}`
  - `randomescapeemountoptimal`
  - Can also generate answers to questions

    MOTHER'S MAIDEN NAME? `Smith1980`

Password manager security

Lastpass breach August 2022:

- limited to the service's development environment that isn't connected to customers' data and encrypted vaults.
- LastPass has no access to users' master passwords, which are needed to decrypt their vaults.

Lastpass breach on June 14, 2015

- user email addresses, authentication hashes, password reminders and server per user salts were compromised
- no evidence that user vault data (a user's stored passwords) was taken or that accounts were accessed

https://www.engadget.com/lastpass-hacked-no-user-data-was-compromised-064640557.html

Research questions

- A password manages solves some key password related issues.
- What risks arise because of the use of password manager?
- What are the potential security holes in software, systems and human procedures?
- Any cases of unauthorized extraction of passwords from the password vault?

Peer Interaction

- Your interaction with other student's research is a part of the class.
- You will need to review semi-final report of two fellow students
  - Identify main contributions
  - Strength/weaknesses
  - Suggestions for improvements
  - Suggested additional references
- Presentations: reviews and comments
- Your interaction will be evaluated