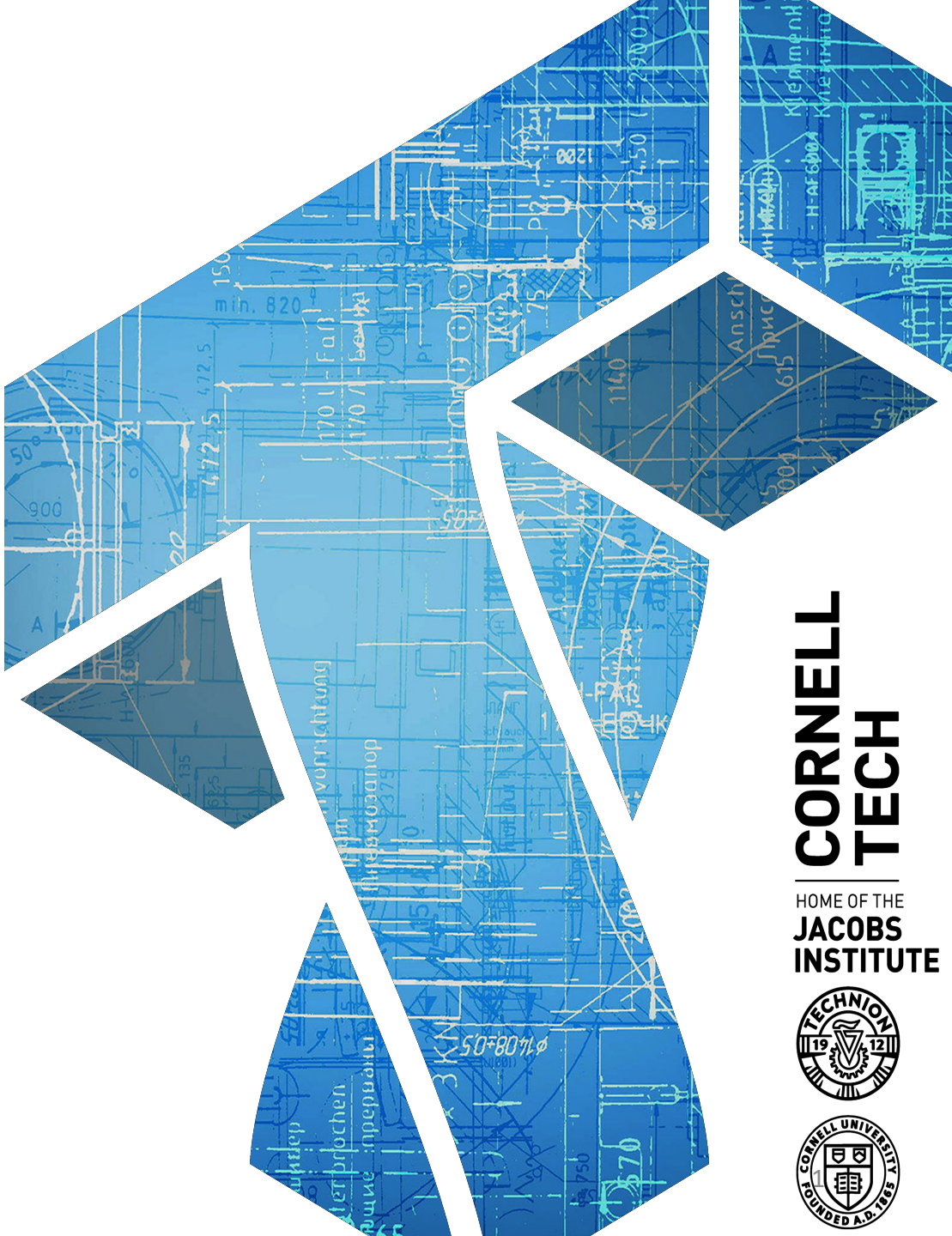
CS 6431: Credential stuffing & mitigations



Instructor: Tom Ristenpart

https://github.com/tomrist/cs6431-fall2021

CORNELL TECH

HOME OF THE JACOBS INSTITUTE

Summary from last time

- Older entropy measures, estimates inaccurate
- Needed new empirical approaches:
  - measurement studies, breach data
- Password models power:
  - Strength meters use guess rank estimate for best known guessing attacks
  - Simulation-based studies utilize these models as well
  - Improvements to cracking tools (CMU password guessing tool)
- What were attacks described in Microsoft blog post?

Credential stuffing attacks

tom, password1

Login server

tom	$salt_1$, $H(salt_1$, password1)
alice	$salt_2$, $H(salt_2$, 123456)
bob	$salt_3$, $H(salt_3$, p@ssword!)

tom	password1
sally	hell0w0rlD4
nick	cloudflareRul3z
…	…

- Simulations suggest works **~40%** of time (people reuse passwords!)
- Some evidence that ***majority*** of account compromises due to credential stuffing

December 18, 2017

4iQ Discovers 1.4 Billion Clear Text Credentials in a Single Database

Understanding attacks against password-based auth

- Variety of ways credentials leaked
  - Breaches, phishing, key loggers, …
- Underground ecosystem of criminals trying to exploit accounts to monetize compromised accounts
  - How can criminals monetize?

Let’s unpack the Thomas et al. study

- [Thomas et al. 2017]: First study of credential leak underground. Excellent measurement study paper
- Sources of data:
  - Crawl of public, private forums for password breaches
  - Phishing kits (PHP & HTML code)
  - Key loggers
  - GMail emails
  - Compromise detection mechanisms

Collecting password breaches

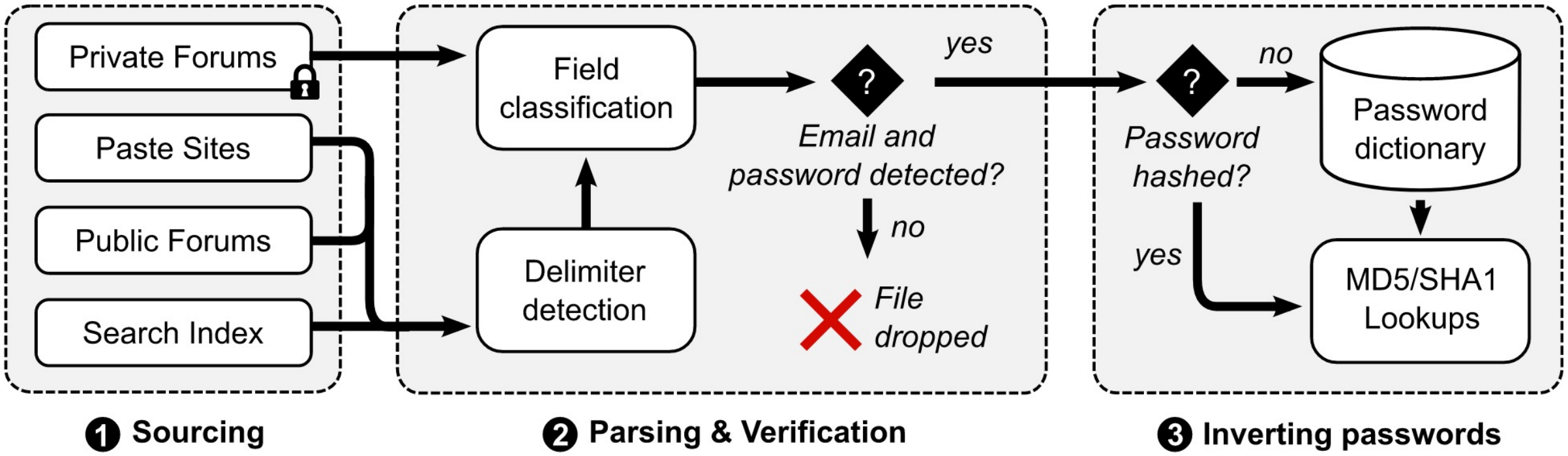


Figure 1: Collection framework for identifying credential leaks on public websites and private forums.

Collecting drop points

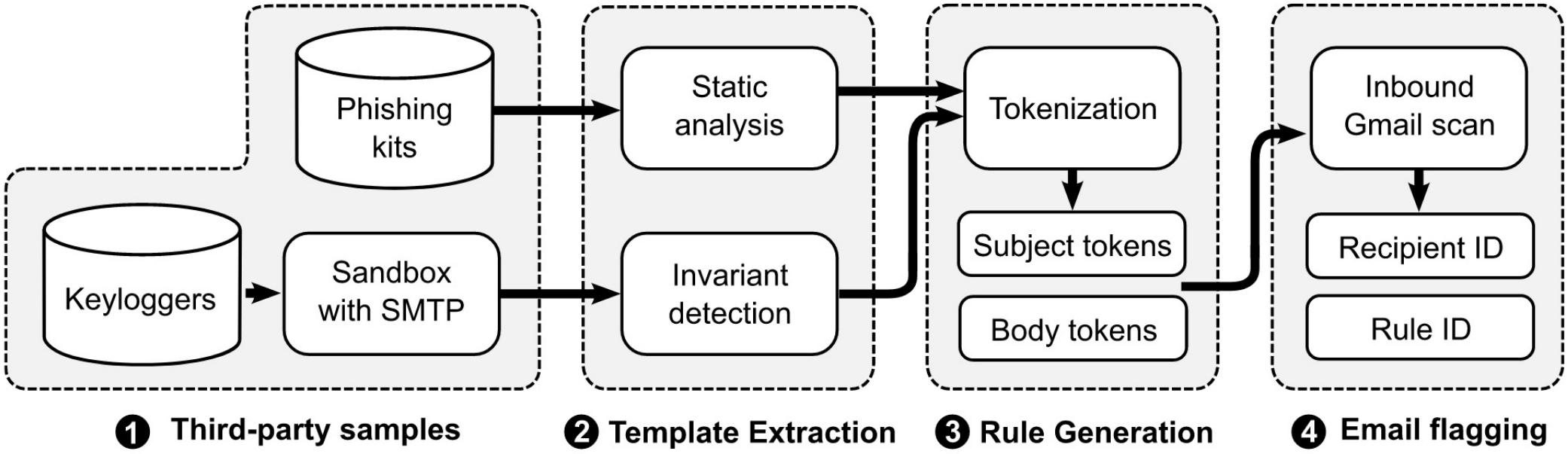


Figure 2: Framework for identifying inbound messages that contain credentials stolen by phishing kits and keyloggers.

Resulting datasets

- Private leak forums provide bulk of leaks
- Most credentials in data set exposed by leaks
  - What is obvious take-away?
  - Why is that potentially wrong interpretation?
- Password reuse rate: 12-19%
  - Less than 40% estimated by Das et al., others

Table 1: Summary of datasets from our collection pipelines.

Dataset	Samples	Time Frame
Credential leaks	3,785	06/2016–03/2017
Phishing kits	10,037	03/2016–03/2017
Keyloggers	15,579	03/2016–03/2017
Credential leak victims	1,922,609,265	06/2016–03/2017
Phishing kit victims	3,779,664	03/2016–03/2017
Keylogger victims	2,992	03/2016–03/2017
Phishing victim reports	12,449,036	03/2016–03/2017
Keylogger victim reports	788,606	03/2016–03/2017

Match rate and hijack odds

- Check if password matches accounts' current one
- Hijack detection independent of type exposure (?)
  - Can use to figure out risk of hijack associated with each exposure

Table 9: Risk associated with passwords stolen through leaks, phishing kits, or keyloggers.

Password source	Password match rate	Hijacking odds ratio	Failed login odds ratio
Credential leak	6.9%	11.6x	1.4x
Phishing kit	24.8%	463.4x	1.7x
Keylogger	11.9%	38.5x	1.5x

	Hijacked	Not hijacked
Credential leaked	H_{leak}	N_{leak}
Not leaked	H_{unleak}	N_{unleak}

Odds ratio = $$\frac{H_{\text{leak}}/N_{\text{leak}}}{H_{\text{leak}}/N_{\text{unleak}}}$$

Understanding attackers

- Same tooling as early 2000s
  - Not much pressure on adversaries to improve
- Handful of actors w/ outsized impact
  - Campaign behavior observed
- High rate of theft:
  - Up to 69,000 creds per week via phishing

Table 12: Top 10 geolocations associated with the last sign-in to email accounts receiving stolen credentials.

Phishing Kit Users		Keylogger Users	
Geolocation	**Popularity**	**Geolocation**	**Popularity**
Nigeria	41.5%	Nigeria	11.2%
United States	11.4%	Brazil	7.8%
Morocco	7.6%	Senegal	7.3%
South Africa	6.4%	United States	6.4%
United Kingdom	3.3%	Malaysia	5.8%
Malaysia	3.2%	India	5.7%
Indonesia	3.1%	Philippines	4.6%
Tunisia	2.0%	Turkey	3.2%
Egypt	1.6%	Thailand	2.8%
Algeria	1.3%	Egypt	2.7%
Other	18.6%	Other	42.7%

High level takeaways

- Password leaks highest volume of credential exposure
  - Less likely to lead to hijacks. Hypothesized due to anti-hijack mechanisms preventing exploitation
- Phishing more likely to lead to hijacks
  - Steal more information useful for evading anti-hijack mechanisms
- Attack tools haven’t had to change for long time

In-class quick exercise

- Could you do this study at a university?
- Think about it for a few minutes, feel free to discuss among yourselves
- We'll come back and discuss

How to improve defenses in light of study?

- Credential leaks for credential stuffing
- Phishing
- Keyloggers

Credential stuffing attacks

tom, password1

Login server

tom	$salt_1$, $H(salt_1$, password1)
alice	$salt_2$, $H(salt_2$, 123456)
bob	$salt_3$, $H(salt_3$, p@ssword!)

tom	password1
sally	hell0w0rlD4
nick	cloudflareRul3z
…	…

- Simulations suggest works **~40%** of time (people reuse passwords!)
- Some evidence that ***majority*** of account compromises due to credential stuffing

';--have i been pwned?

Password Checkup

Offered by: google.com

Protocols for checking if password is in breach with third-party service

[Li, Pal, Ali, Sullivan, Chatterjee, R. – 2019] [Thomas et al. – 2019]

Private set membership

Client

username,*pw*

username,*pw* ∈ DB

C3 service

Breach DB

tom	password1
sally	hell0w0rlD4
nick	cloudflareRul3z
…	…

Reveals both username and password to service. Big problem!

Any ideas for ad hoc improvements?

Long literature in cryptographic theory on private set intersection (PSI) starting with [Meadows 1986], dozens of papers

Oblivious Pseudorandom Functions (OPRF)

- PRF is a keyed function F_K: MsgSp -> $\{0,1\}^n$
  - Computationally indistinguishable from random function
- Oblivious PRF [Freedman et al. 2005]:
  - Client holds message M
  - Server holds PRF key K
  - Interactive protocol so that client obtains $F_K(M)$ while:
    - Server learns nothing
    - Client learns just $F_K(M)$

2HashDH OPRF

[Jarecki, Kiayias, Krawczyk 2014]

- Fix elliptic curve group G (written multiplicatively)
- H : {0,1}* -> G is hash function that maps into G
  - Must be careful building H for G elliptic curves
- PRF is $F_K(M) = H(M)^K$
- This allows simple oblivious protocol:
  - Client computes $Z = H(M)^X$ for random X, sends to server
  - Server computes Z^K and sends it back
  - Client computes $Y = Z^{1/X} = (H(M)^{XK})^{1/X} = H(M)^K = F_K(M)$
- Security definitions and analysis non-trivial

Google & Li et al. credential checking protocol

Client

$\text{Trunc}_{16}(username)$, $H(username,pw)^X$

$H(username,pw)^{X*K}$, bucket

C3 service

Breach DB

tom	password1
sally	hell0w0rlD4
nick	cloudflareRul3z
…	…

Pick X at random
("blind" the hash)

Holds secret key K for OPRF

$\text{bucket} = \{ H(username',pw')^K \mid \text{Trunc}_{16}(username') = \text{Trunc}_{16}(username) \}$

Compute $Y = H(username,pw)^K$
by removing blinding

If $Y \in$ bucket then
(*username*,*pw*) is in a breach

- Bucketization leaks some info but only about username
  - Reduces size of bucket (bandwidth cost)
- OPRF requires just two client-side exponentiation (ECC multiplication), one server side
  - Google uses slow hash for abuse resistance

First version of Google protocol

Client

$\text{Trunc}_{16}(username,pw)$, $H(username,pw)^X$

$H(username,pw)^{X*K}$, bucket

C3 service

Breach DB

tom	password1
sally	hell0w0rlD4
nick	cloudflareRul3z
…	…

Holds secret key K for OPRF

What is potential problem with this version?

This leaks information about password to C3 service

Hash prefix gives 2^{16} x increase in guessing budget $\neq$ 2^{16} x improvement in guessing attack success rate

[Li et al. 2019] used simulations to show ~10x improvement in success rate for guessing attacks with budget 1,000 (assuming attacker knows username)

Deployment as browser extension

- Password Checkup
- Looked at login data, ran protocol to check if in a breach
- Anonymous analytics about user actions

Metric	Value
Extension users	667,716
Logins analyzed	21,177,237
Domains covered	746,853
Breached credentials found	316,531
Warnings ignored	81,368 (26%)
Passwords reset	82,761 (26%)

Table 3: Summary of the anonymous telemetry data reported over the course of our analysis window from February 5–March 4, 2019.

Credential *tweaking* attacks

tom, password1

tom, Password1

tom, password123

Login server

tom	password1
sally	hell0w0rlD4
thyla	MozillaRulez
…	…

Breach notification services

tom	$salt_1$, $G_K(salt_1$, password123)
alice	$salt_2$, $G_K(salt_2$, 123456)
bob	$salt_3$, $G_K(salt_3$, p@ssword!)

Unclear if attackers using in wild yet

Early academic work suggests efficacy:

[Das et al. 2014]

[Wang et al. 2016]

} Ad hoc mangling rules

Password similarity models

[Pal et al. 2019]

User	Password Lists
tom	password1, cornelltechYay!
sally	hell0w0rlD1 helloworld123
nick	cloudflareRulez, Rugby123, rugby1
...	...

Lots of info about password similarity

First discovered by 4iQ on the “Dark Web”

1.4 billion email, password pairs
1.1 billion unique emails
463 million unique passwords

More than **150 million** users with **2 or more passwords**

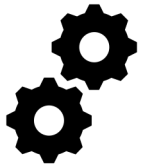
Around **10%** of distinct password pairs of same user are within **1 edit distance**

Password similarity models

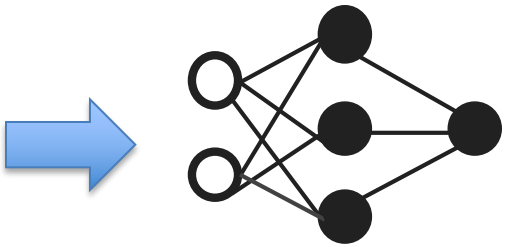
[Pal et al. 2019]

User	Password Lists
tom	password1, cornelltechYay!
sally	hell0w0rlD1, helloworld123
nick	cloudflareRulez, Rugby123, rugby1
…	…

Lots of info about password similarity



Machine learning



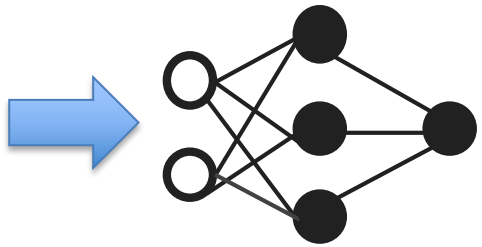
Similarity model:

$$P(w' \mid w)$$

Models probability user selects w' given old password w

Good similarity model = good credential tweaking attack

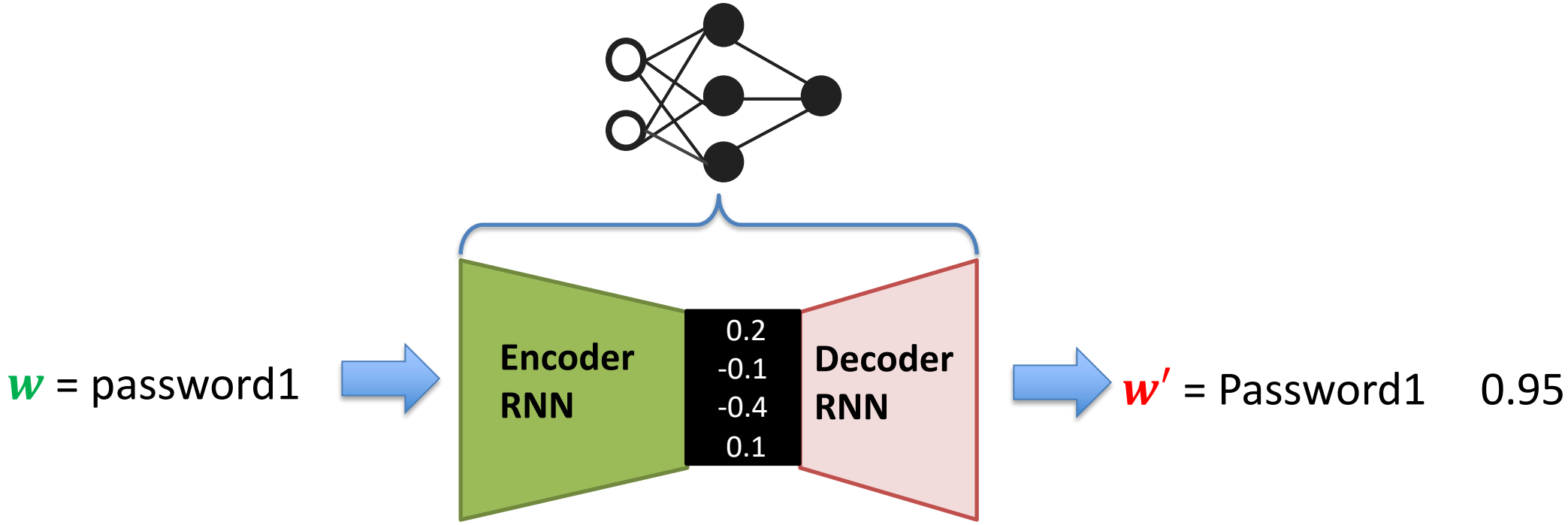
w = password1



Requires algorithm to enumerate
Use beam search

w'	$P(w' \mid w)$
Password1	0.95
password123	0.80
pASSWORD1	0.73
…	…

Password similarity models via deep learning

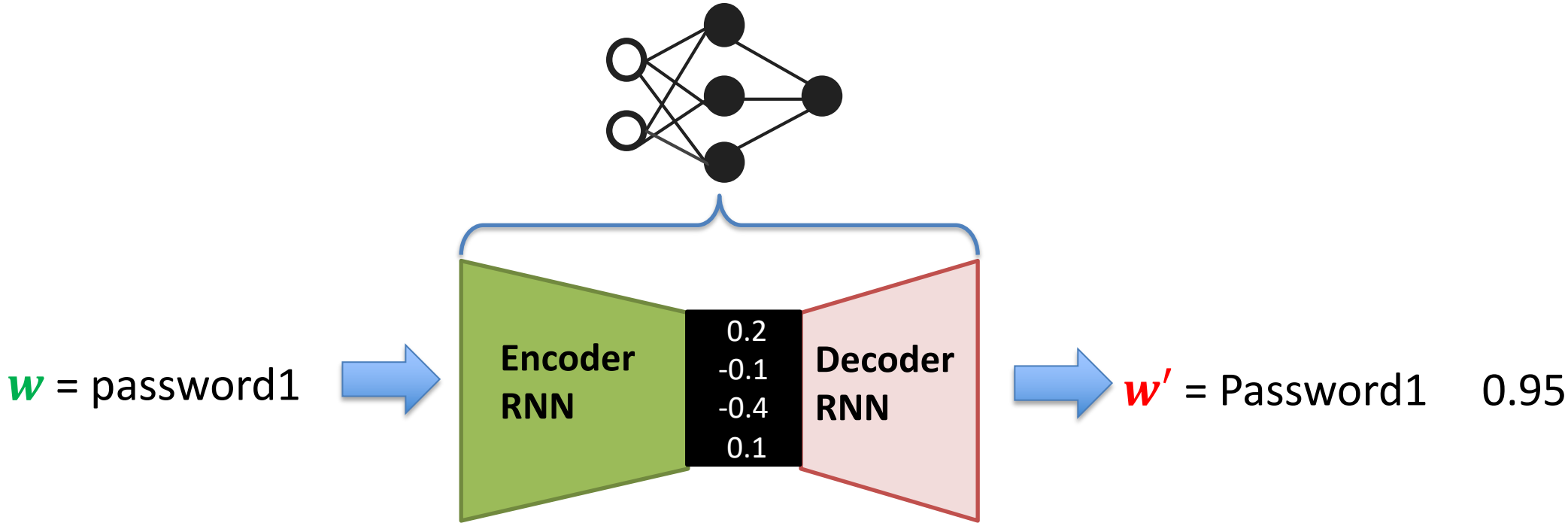


Recurrent neural network (RNN) encoder-decoder architectures

pass2pass: obvious approach like seq2seq [Sutskever et al. 2012] worse than prior work

pass2path: decode to a path (sequence of changes to w that yields w')
better focuses networks on learning similarities (?)

Password similarity models via deep learning



pass2path:

- Trained on **144 million** password pairs
- Took 2 days on Nvidia GTX 1080 GPU and Intel Core i9 processor
- Model has 2.4 million parameters, 60 MB on disk
- ~100 milliseconds to generate 1,000 guesses

Evaluating danger of credential tweaking

Audit of Cornell accounts

- ~500,000 accounts at Cornell
- Uses breach notification service
- Requires 8 character passwords with 3 different character classes

IT@Cornell

15,776 active accounts in leak dataset

pass2path cracked 8.4% in ≤1,000 guesses

Put 1,374 vulnerable accounts on watch list

First real-world evaluation of credential tweaking!

Defending against credential tweaking

- Unclear if attackers exploiting in practice
  - Thomas et al. study suggests not against Google accounts
- Still be nice to guide users away from weak variants of breached passwords
  - Personalized password strength meter (PPSM) [Pal et al. 2019]
  - Extending C3 services to warn about

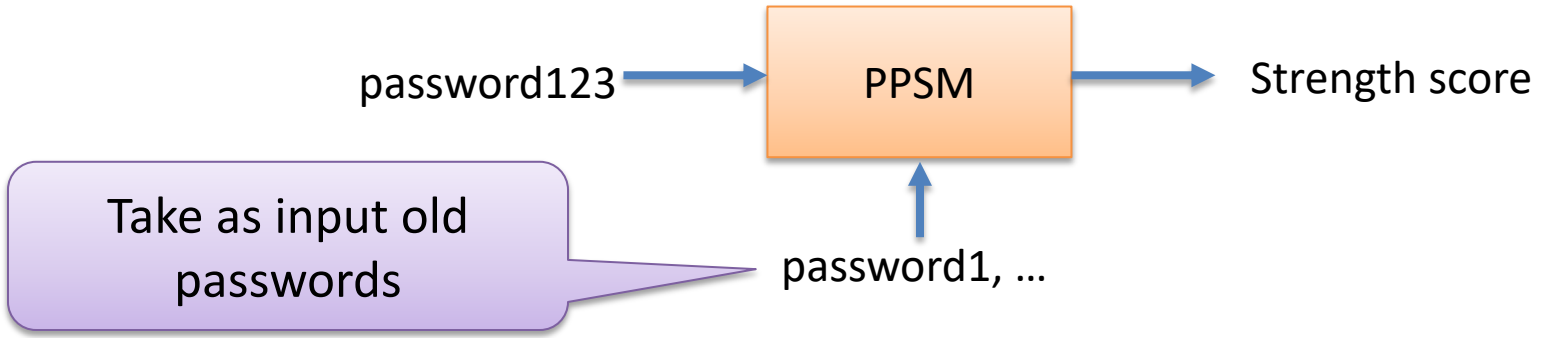
Personalized password strength meters

Starting point is standard password strength meters (PSMs)

4 = strong
3 = sort of guessable
2 = guessable
1 = really guessable
0 = trivially weak

Personalized password strength meters

Starting point is standard password strength meters (PSMs)



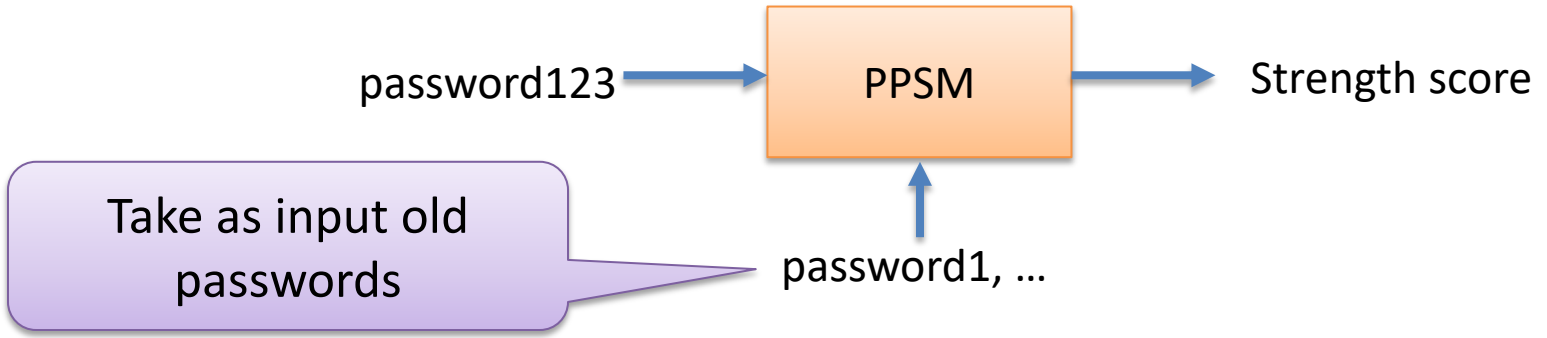
4 = strong
3 = sort of guessable
2 = guessable
1 = really guessable
0 = trivially weak

Old password(s) can be taken from:

- Breach dataset
- User inputting old password during change
- Other user passwords (e.g., password manager)

Personalized password strength meters

Starting point is standard password strength meters (PSMs)



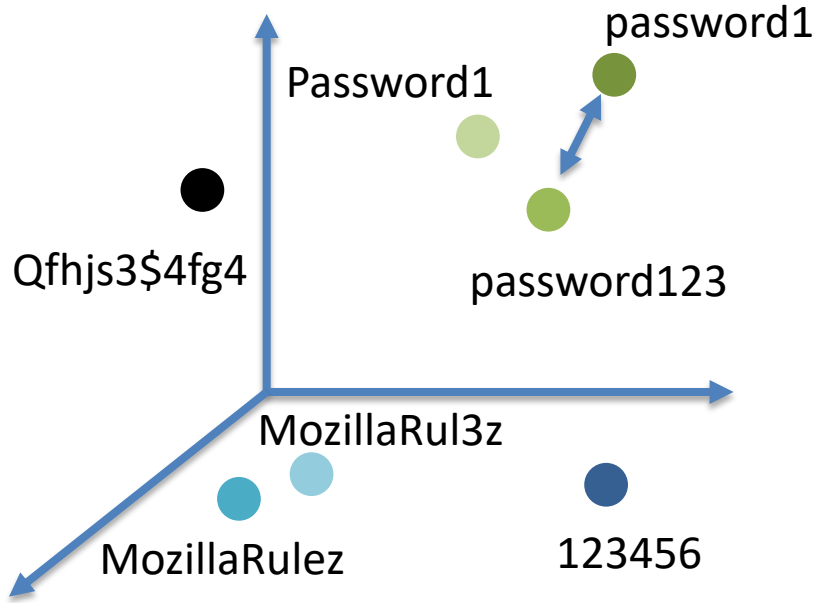
4 = strong
3 = sort of guessable
2 = guessable
1 = really guessable
0 = trivially weak

We use *password embeddings*:
Learn mapping f between password and vector space

f(password1) = [0.11, 0.82, … , .45]

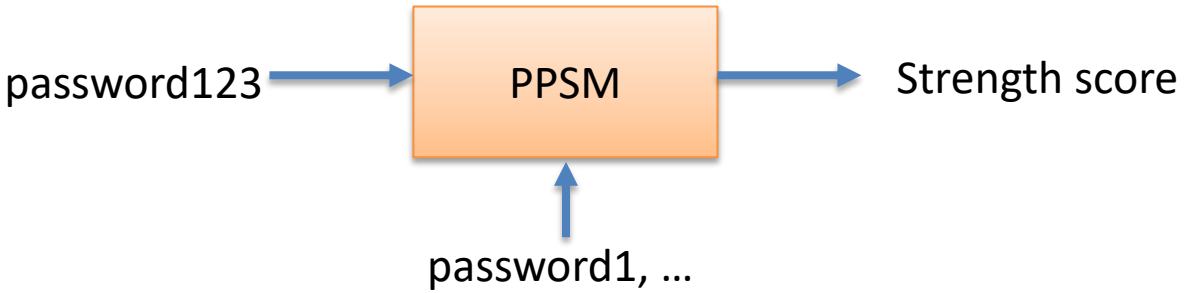
f(password123) = [0.08, 0.60, … , .90]

Dot product f(password) * f(password123) small
Based on FastText [Bojanowski et al. 2016]



Personalized password strength meters

Starting point is standard password strength meters (PSMs)



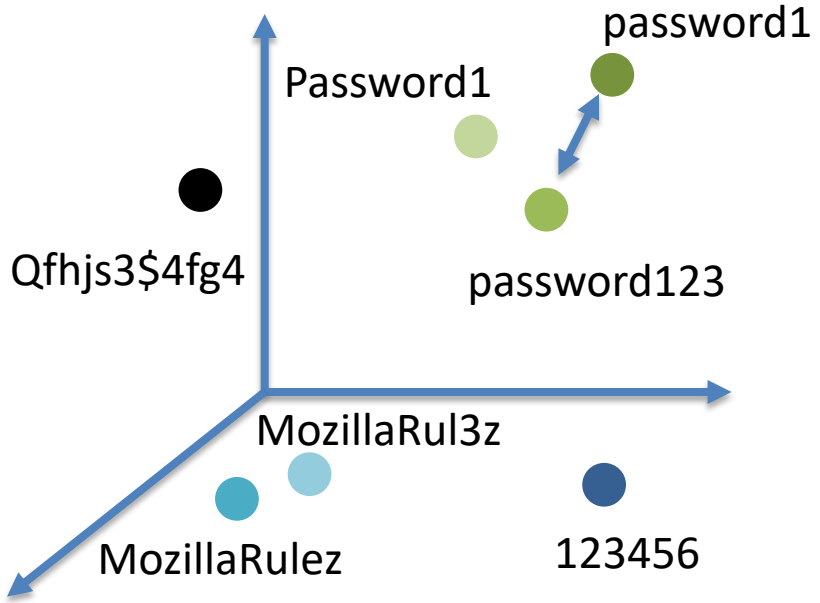
4 = strong
3 = sort of guessable
2 = guessable
1 = really guessable
0 = trivially weak

Our PPSM combines password embedding + zxcvbn

- 3 MB table + table lookups (very fast)

Marks all of pass2path-vulnerable Cornell passwords as trivially weak

Have not yet done proper usability study



Credential-tweaking aware C3

- PPSM only works when old password available
  - Use C3 service to learn if password in breach, then use PPSM on client side to rule out similar passwords
  - Use PPSM during password change flow
- C3 services so far only reveal exact match
  - If query with (*tom*,*password1*), no match even if (*tom*,*password*) is in a breach
  - Might I Get Pwned (MIGP) [Pal et al. 2021] extends Li et al. / Google approach to include similar passwords

Summary

- Account hijack serious problem
- Passwords are ubiquitous, relatively easy to use, but can be stolen and cracked
- 2FA best protection (particularly ones that are phish-proof) against account compromise
- Measurement studies, cryptography, machine learning, …