Improving Real-world Password Guessing Attacks via Bi-directional Transformers

Ming Xu et al.
USENIX ‘23

Part 1

Introduction

Password Leaks are Prevalent

https://www.getsafeonline.org/personal/news-item/twitch-password-change-urged-following-massive-data-leak/

Part 1

Introduction

Password Leaks are Prevalent

We have a total of 15,347,816,022 Records from the following 971 Datasets, free for download once you unlock them.

[014,936,670] | 2015 - *(000webhost.com)* 000webhost Database ➞ Download Here!
[007,476,940] | 2021 - *(datpiff.com)* DatPiff Database ➞ Download Here!
[007,633,234] | 2018 - *(blankmediagames.com)* BlankMediaGames Database ➞ Download Here!
[015,003,961] | 2021 - *(epik.com)* Epik Database ➞ Download Here!
[000,819,478] | 2014 - *(warframe.com)* Warframe Database ➞ Download Here!
[004,105,095] | 2018 - *(avvo.com)* Avvo Database ➞ Download Here!
[000,018,131] | 2017 - *(crackingitaly.com)* CrackingItaly Database ➞ Download Here!
[010,604,307] | 2019 - *(armorgames.com)* Armor Games Database ➞ Download Here!
[017,551,044] | 2012 - *(disqus.com)* Disqus Database ➞ Download Here!
[000,040,308] | 2011 - *(eazysubs.com)* EazySubs Database ➞ Download Here!
[008,661,578] | 2020 - *(123rf.com)* 123RF Database ➞ Download Here!
[000,073,944] | 2020 - *(shockgore.com)* ShockGore Database ➞ Download Here!
[000,004,606] | 2019 - *(kiwifarms.net)* KiwiFarms Database ➞ Download Here!
[000,020,902] | 2014 - *(bell.ca)* Bell2014 Database ➞ Download Here!
[000,501,407] | 2015 - *(bitcointalk.org)* Bitcoin Talk Database ➞ Download Here!
[045,619,628] | 2020 - *(animaljam.com)* Animal Jam Database ➞ Download Here!
[000,134,047] | 2012 - *(whmcs.com)* WHMCS Database ➞ Download Here!
[000,352,120] | 2013 - *(torrent-invites.com)* Torrent Invites Database ➞ Download Here!
[000,645,786] | 2020 - *(filmai.in)* Filmai.in Database ➞ Download Here!
[005,024,908] | 2011 - *(zhenai.com)* Zhenai Database ➞ Download Here!
[000,179,030] | 2015 - *(thefappening.so)* The Fappening Database ➞ Download Here!
[000,751,700] | 2018 - *(sonicbids.com)* Sonicbids Database ➞ Download Here!
[000,181,871] | 2017 - *(thetvdb.com)* TheTVDB Database ➞ Download Here!
[000,561,991] | 2019 - *(xkcd.com)* XKCD Database ➞ Download Here!
[000,235,233] | 2020 - *(utahgunexchange.com)* Utah Gun Exchange Database ➞ Download Here!
[002,192,857] | 2019 - *(startribune.com)* StarTribune Database ➞ Download Here!
[004,775,203] | 2020 - *(vakinha.com.br)* Vakinha Database ➞ Download Here!
[007,097,178] | 2019 - *(forums.gamesprite.me)* GameSprite Forums Database ➞ Download Here!
[049,913,963] | 2020 - *(gravatar.com)* Gravatar Scrape ➞ Download Here!
[005,003,937] | 2021 - *(robinhood.com)* Robinhood Database ➞ Download Here!
[002,479,044] | 2020 - *(drizly.com)* Drizly Database ➞ Download Here!
[006,353,564] | 2019 - *(eatstreet.com)* EatStreet Database ➞ Download Here!
[001,022,884] | 2014 - *(binweevils.com)* Bin Weevils (2014) Database ➞ Download Here!
[001,079,970] | 2018 - *(artsy.net)* Artsy Database ➞ Download Here!
[000,478,824] | 2017 - *(coinmama.com)* Coinmama Database ➞ Download Here!
[000,422,959] | 2014 - *(avast.com)* Avast Database ➞ Download Here!
[002,064,274] | 2015 - *(game-tuts.com)* GameTuts Database ➞ Download Here!
[001,436,486] | 2015 - *(aternos.org)* Aternos Database ➞ Download Here!
[000,227,746] | 2014 - *(cannabis.com)* Cannabis.com Database ➞ Download Here!
[001,301,460] | 2020 - *(geniusu.com)* GeniusU Database ➞ Download Here!
[005,888,405] | 2020 - *(appen.com)* Appen Database ➞ Download Here!

https://breachforums.st/Announcement-Database-Index

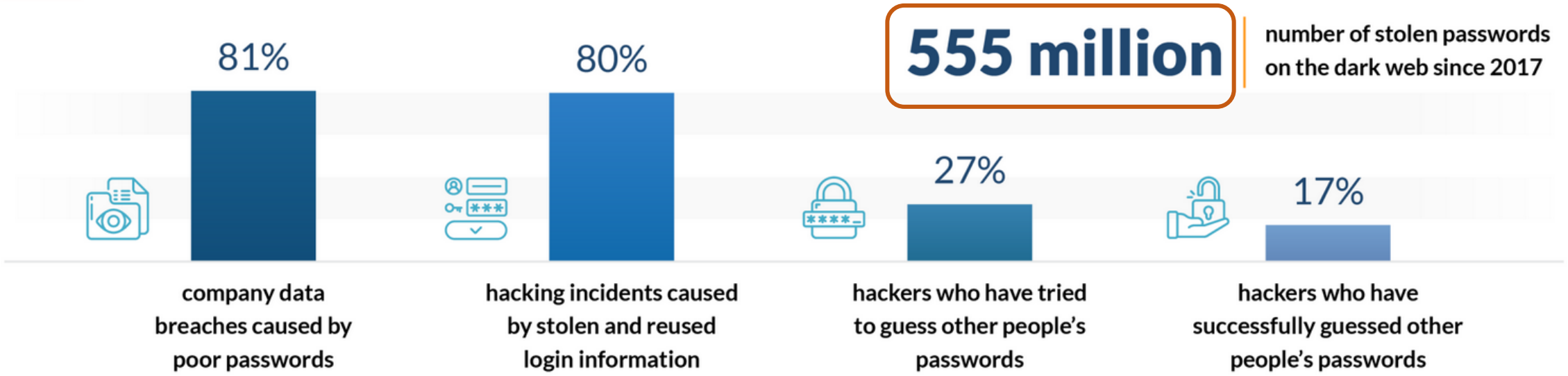
Part 1

Introduction

Password Leaks are Prevalent

2 Password Breach Statistics

Sources: CNET, Google, Verizon, TraceSecurity

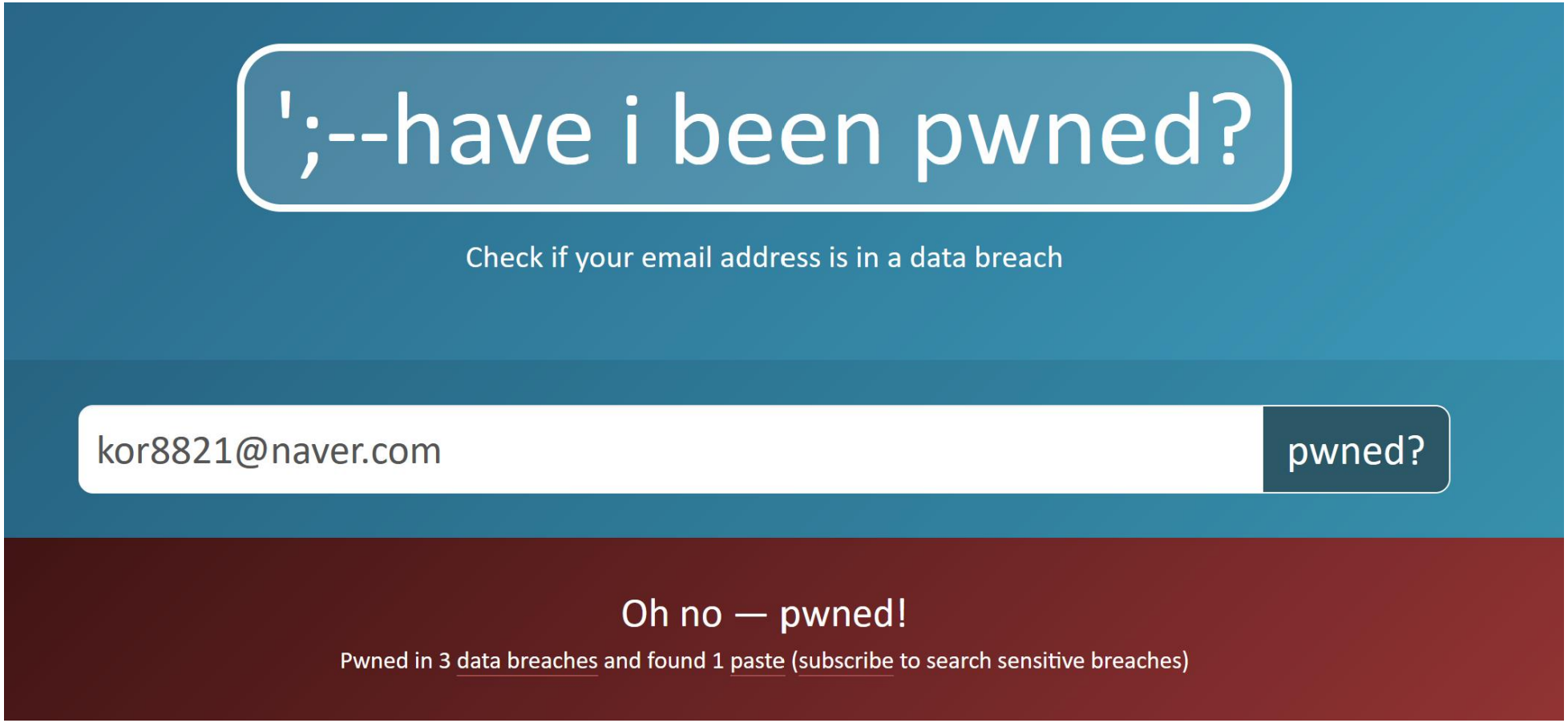


https://financesonline.com/password-statistics/

Part 1

Introduction

Password Leaks are Prevalent



https://haveibeenpwned.com/

Two types of Password Guessing Attacks

No info

General Guessing Attacks (Markov-based, Brute-Force)

Extra info

Real World Guessing Attacks

Two types of Password Guessing Attacks

No info

General Guessing Attacks (Markov-based, Brute-Force)

Extra info

Real World Guessing Attacks

Real World Guessing Scenarios

Conditional Password Guessing CPG[1]

*a*s*o*d → passWord

Targeted Password Guessing TPG[2]

Dan1999 → D@n19!9

Adaptive Rule-based Password Guessing ARPG[3]

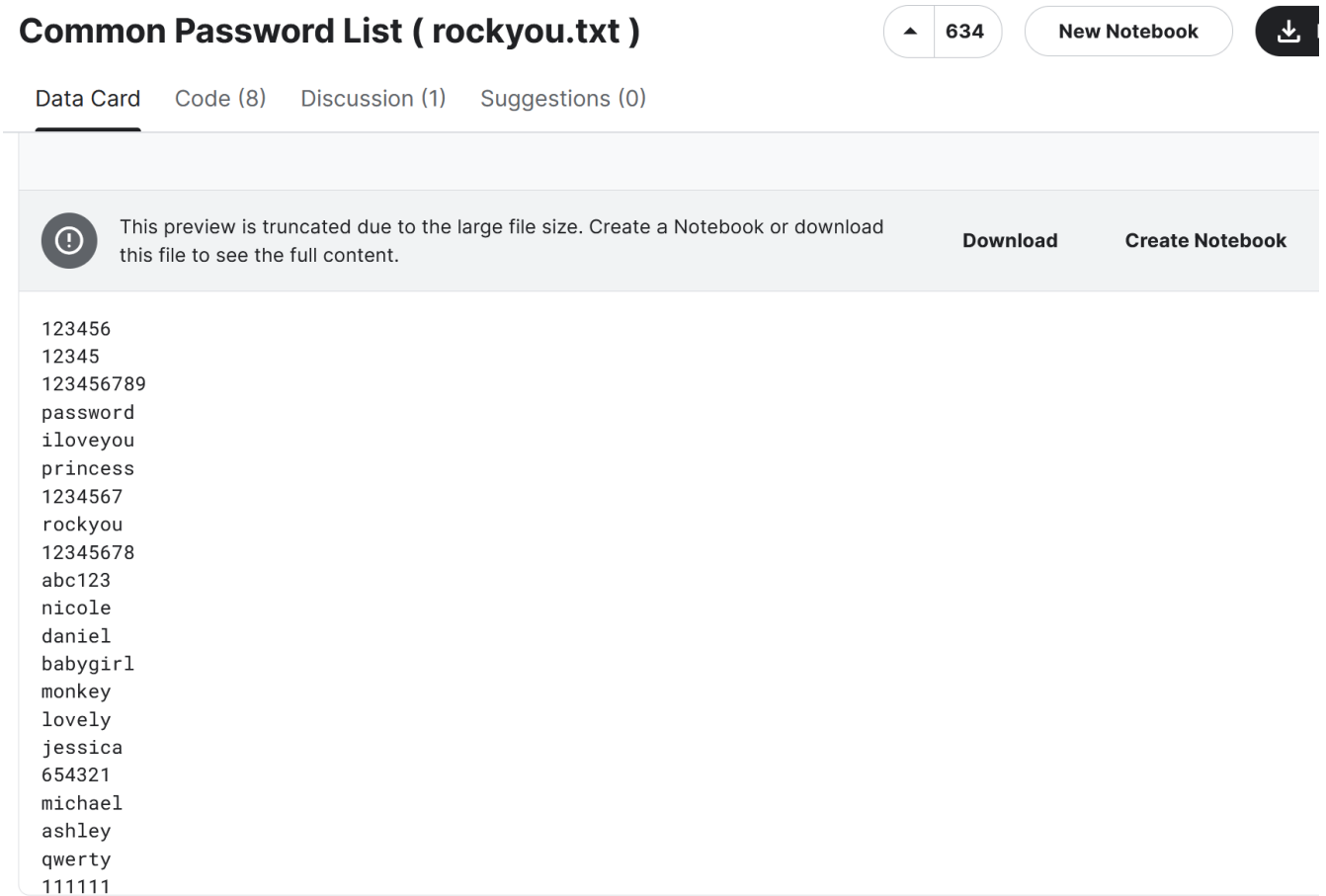
(a->@) → passWord → p@ssWord

[1] Pasquini et al., Improving password guessing via representation learning, SP '21
[2] Pal et al., Beyond Credential Stuffing: Password Similarity Models using Neural Networks, SP '19
[3] Pasquini et al., Reducing Bias in Modeling Real-world Password Strength via Deep Learning and Dynamic Dictionaries, USENIX '21

Part 2

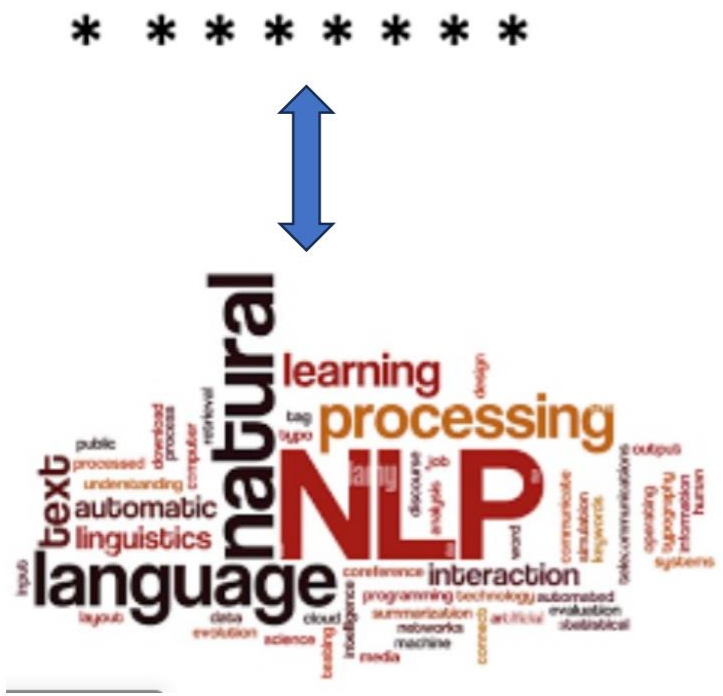
Introduction to Password Guessing

Password Guessing w/ LMs?



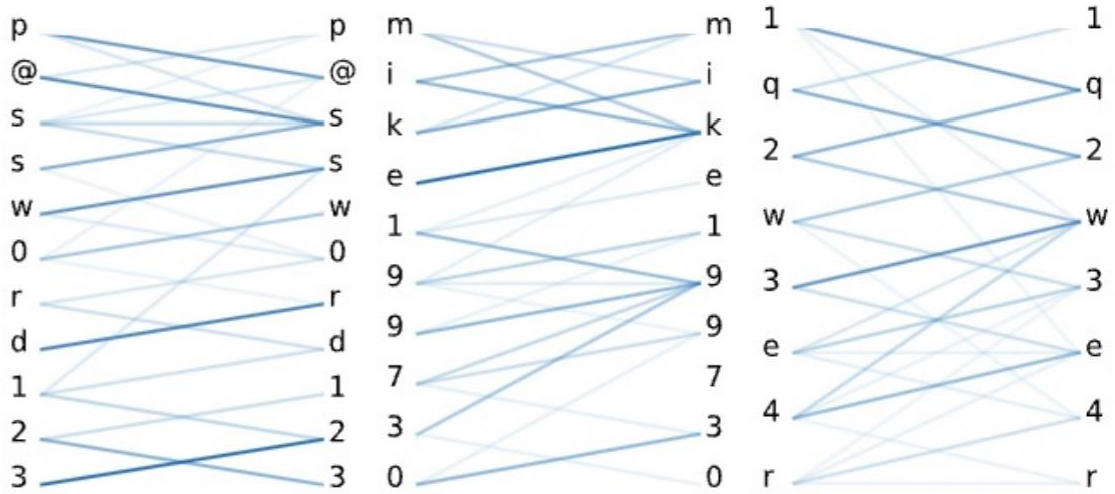
https://www.kaggle.com/datasets/wjburns/common-password-list-rockyoutxt

Password Guessing w/ LMs?



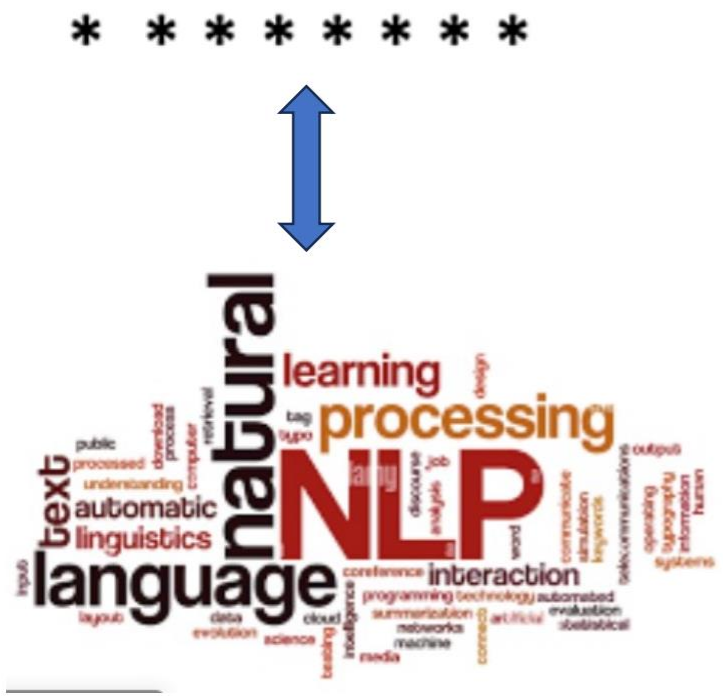
Bi-directional transformers

Pre-trained framework



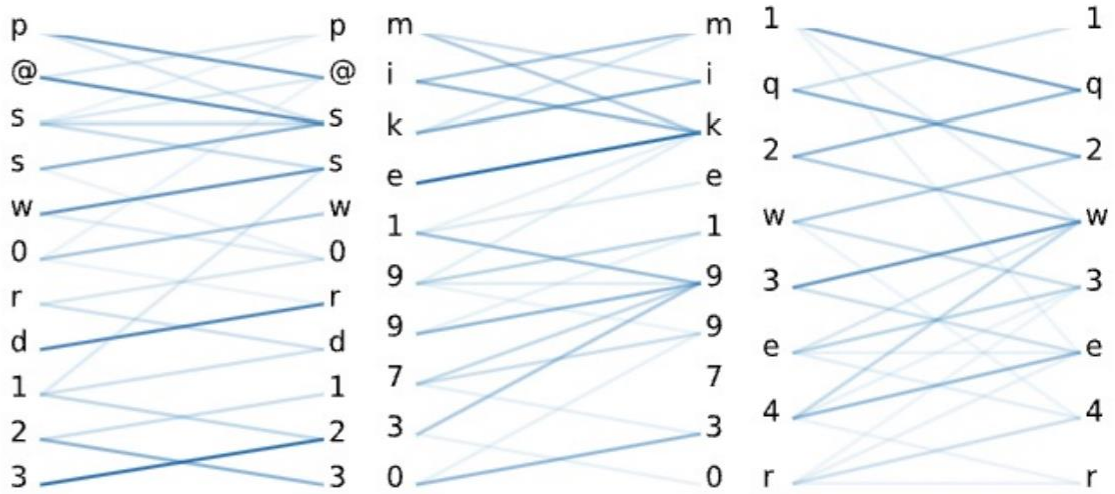
https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

Password Guessing w/ LMs?



Bi-directional transformers

Pre-trained framework



https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

Pre-Training: Dataset

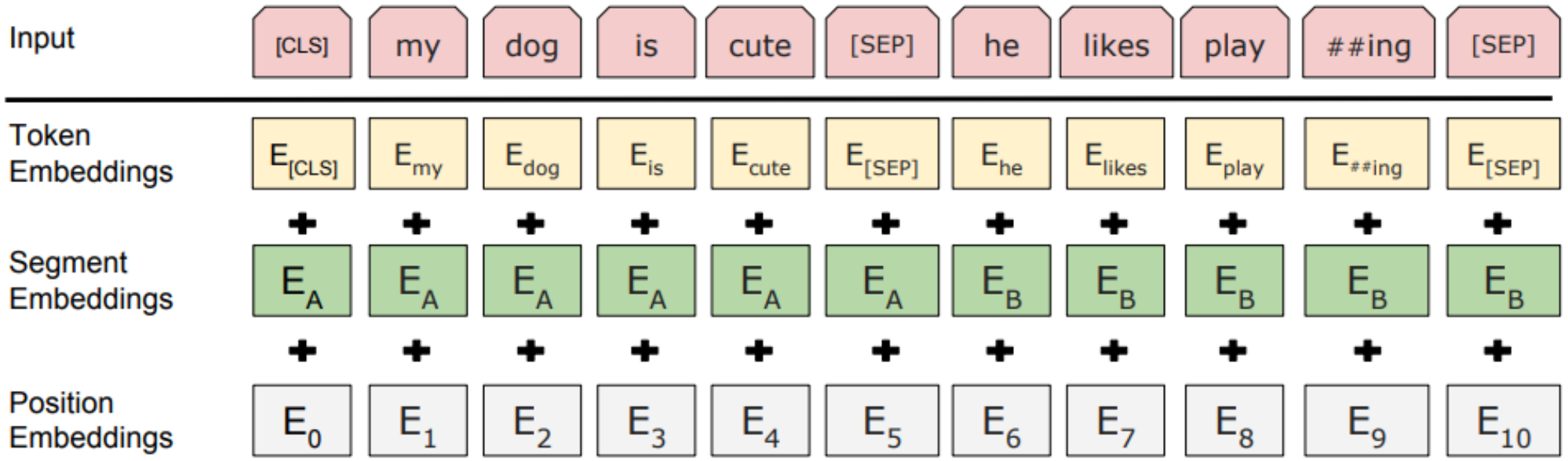
RockYou2021.txt

- Compilation of old password leaks, probable and commonly-used passwords, and wordlists

- Around 82 Billion

https://github.com/ohmybahgosh/RockYou2021.txt

Framework – BERT

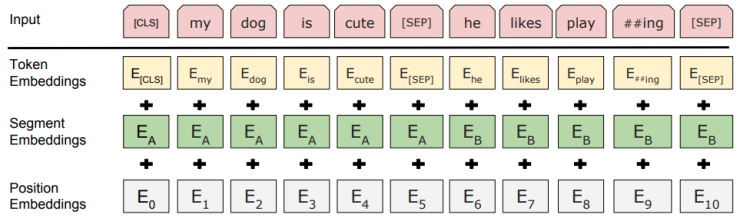


https://pub.towardsai.net/understanding-bert-b69ce7ad03c1

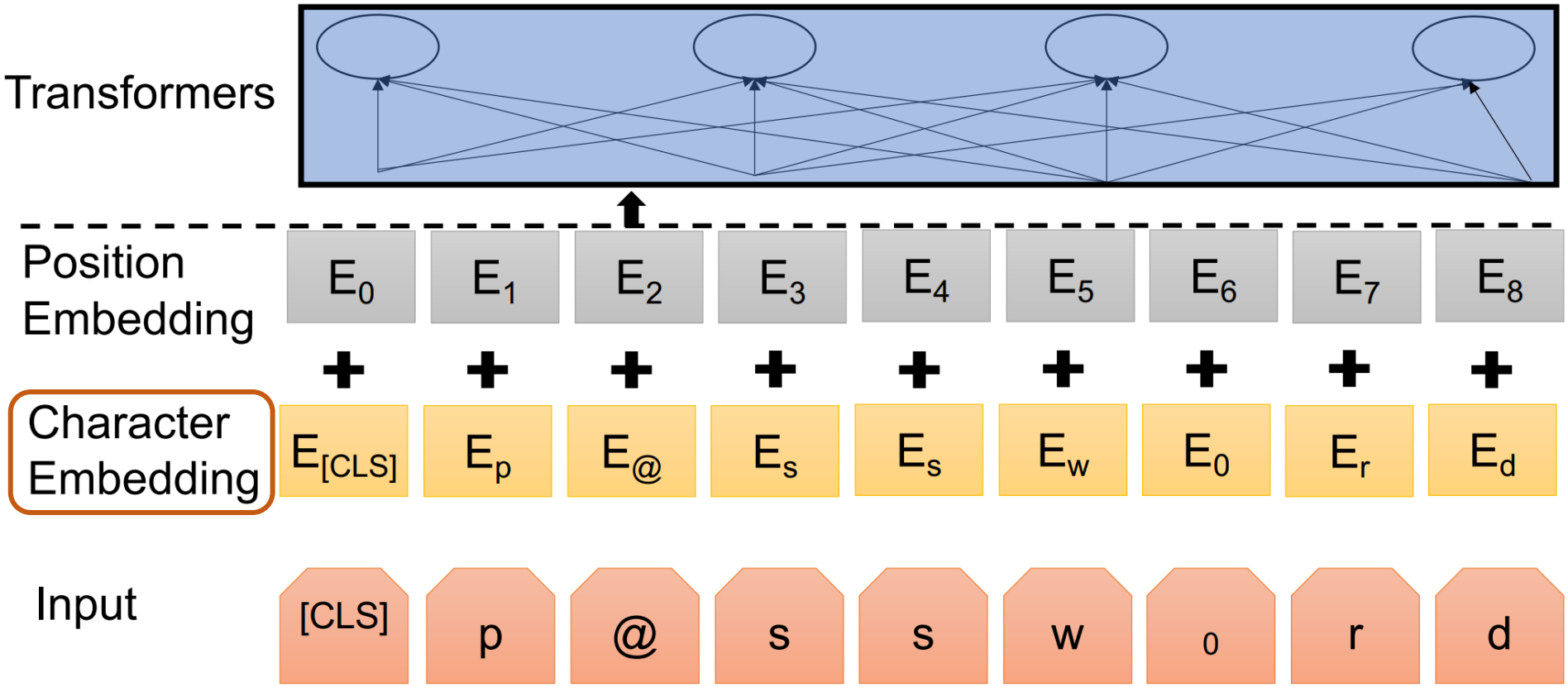
Part 3

PassBERT

Framework – Pre-Training BERT



Pre-train (Objective: **MLM**) ➡ Pre-trained password parameters (contextualized embedding) ➡ Specific attack models

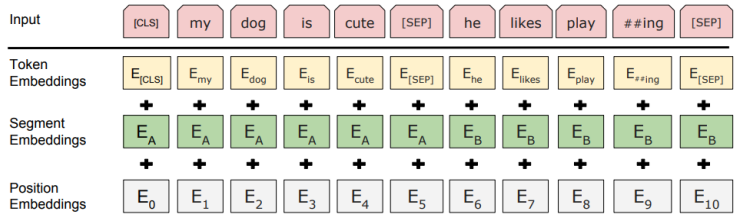


https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

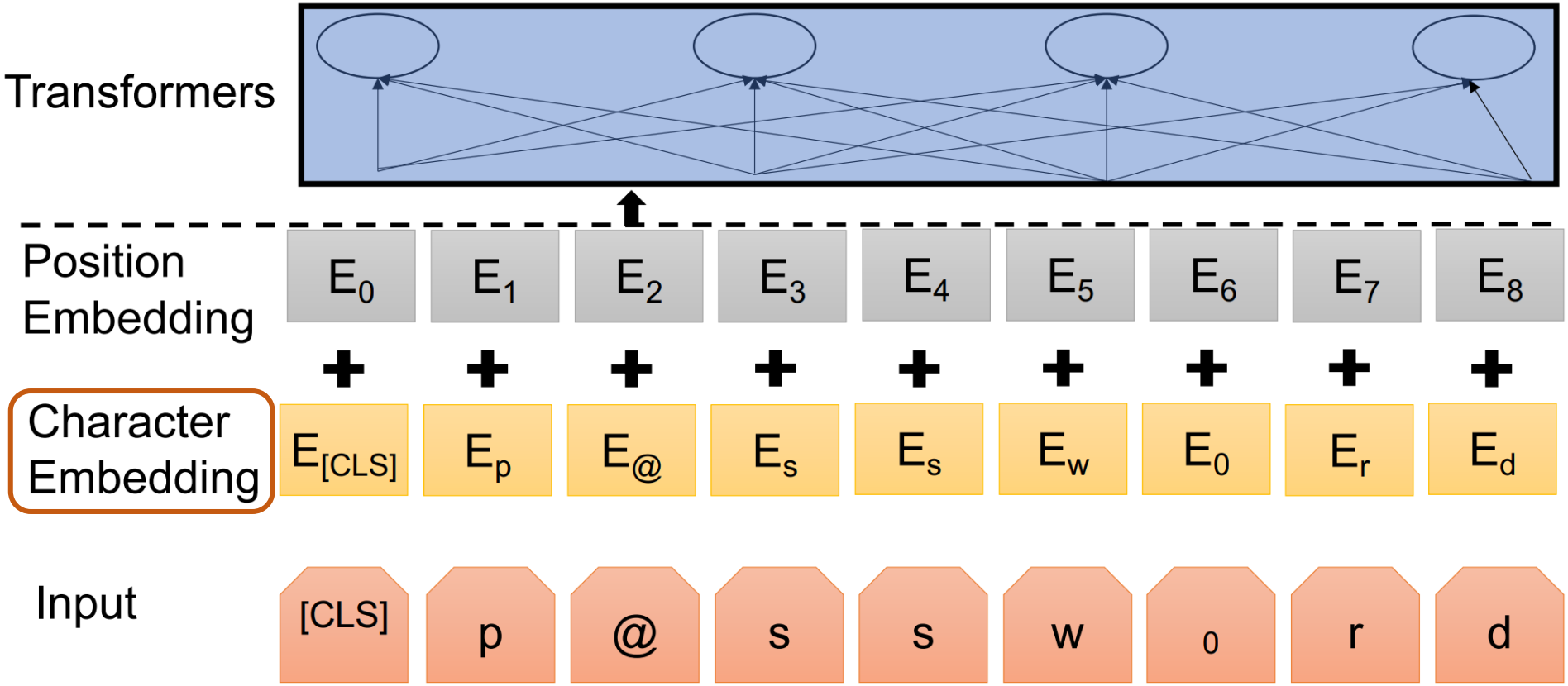
Part 3

PassBERT

Framework – Pre-Training BERT



Pre-train (Objective: **MLM**) ➡ Pre-trained password parameters (contextualized embedding) ➡ Specific attack models

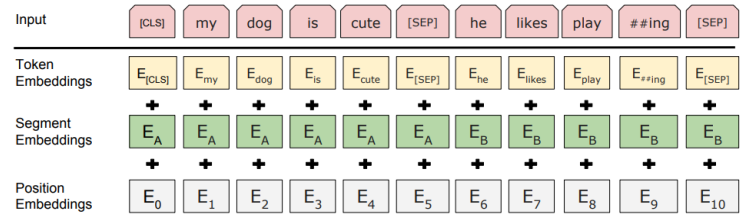


https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

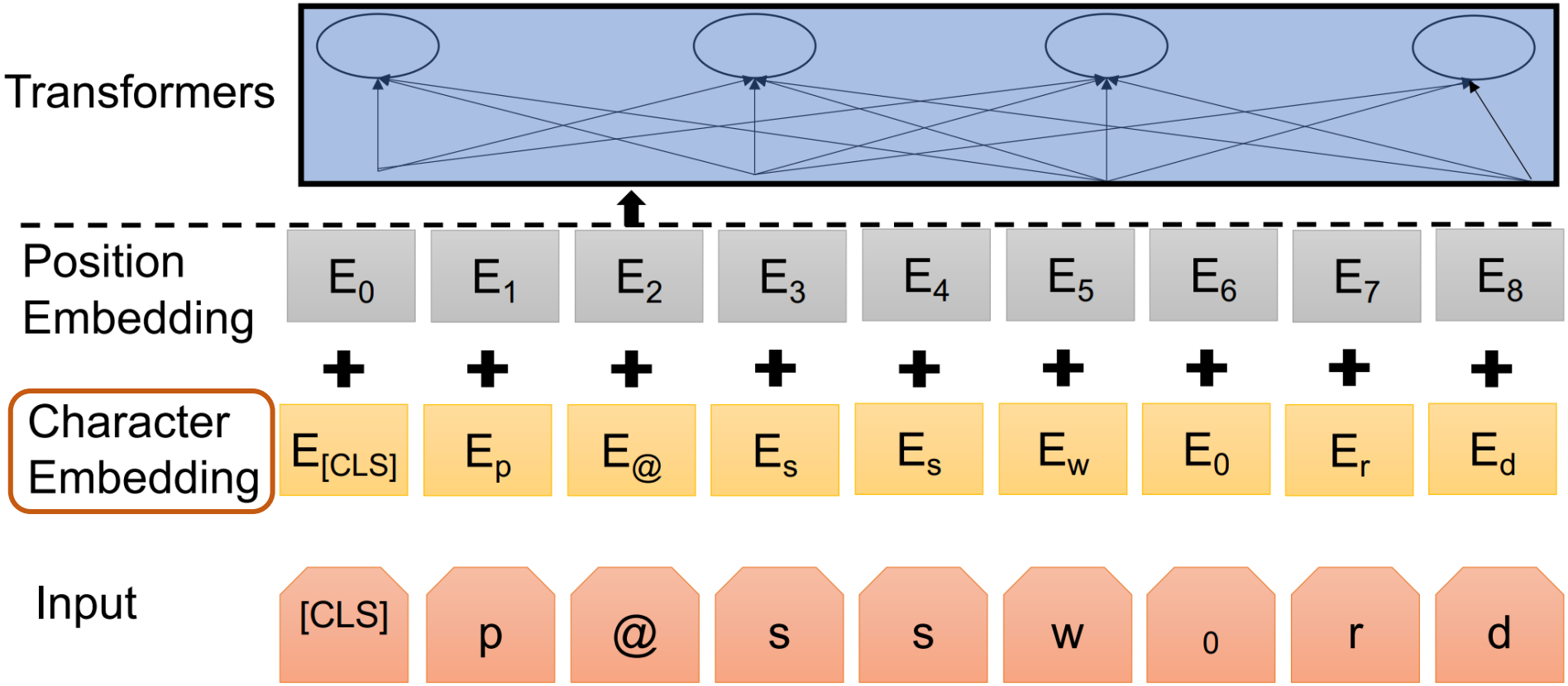
Part 3

PassBERT

Framework – Pre-Training BERT

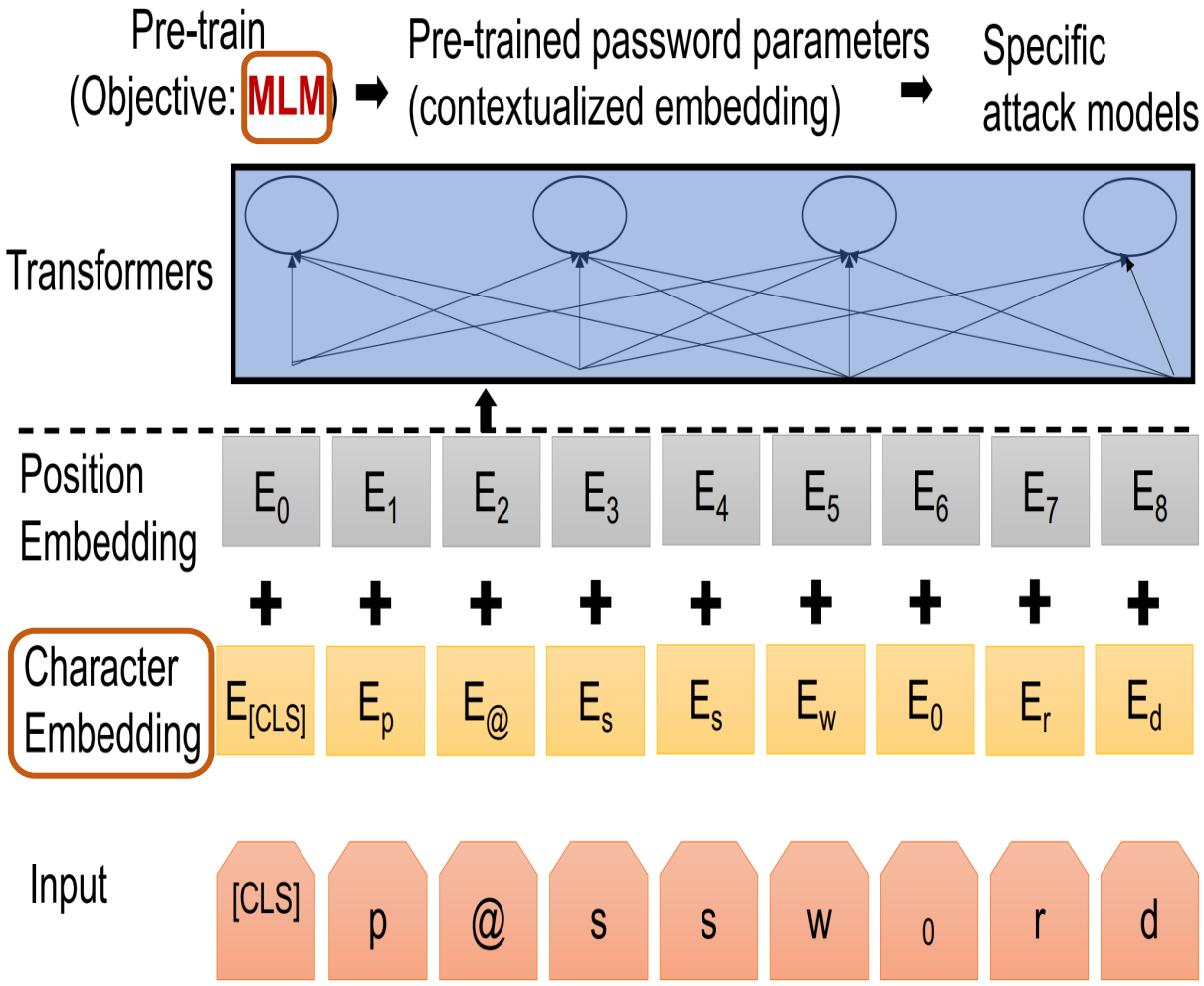


Pre-train (Objective: **MLM**) ➡ **Passw0rd123 != JoshBeckett4131!** Pre-trained password parameters (contextualized embedding) ➡ Specific attack models



https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

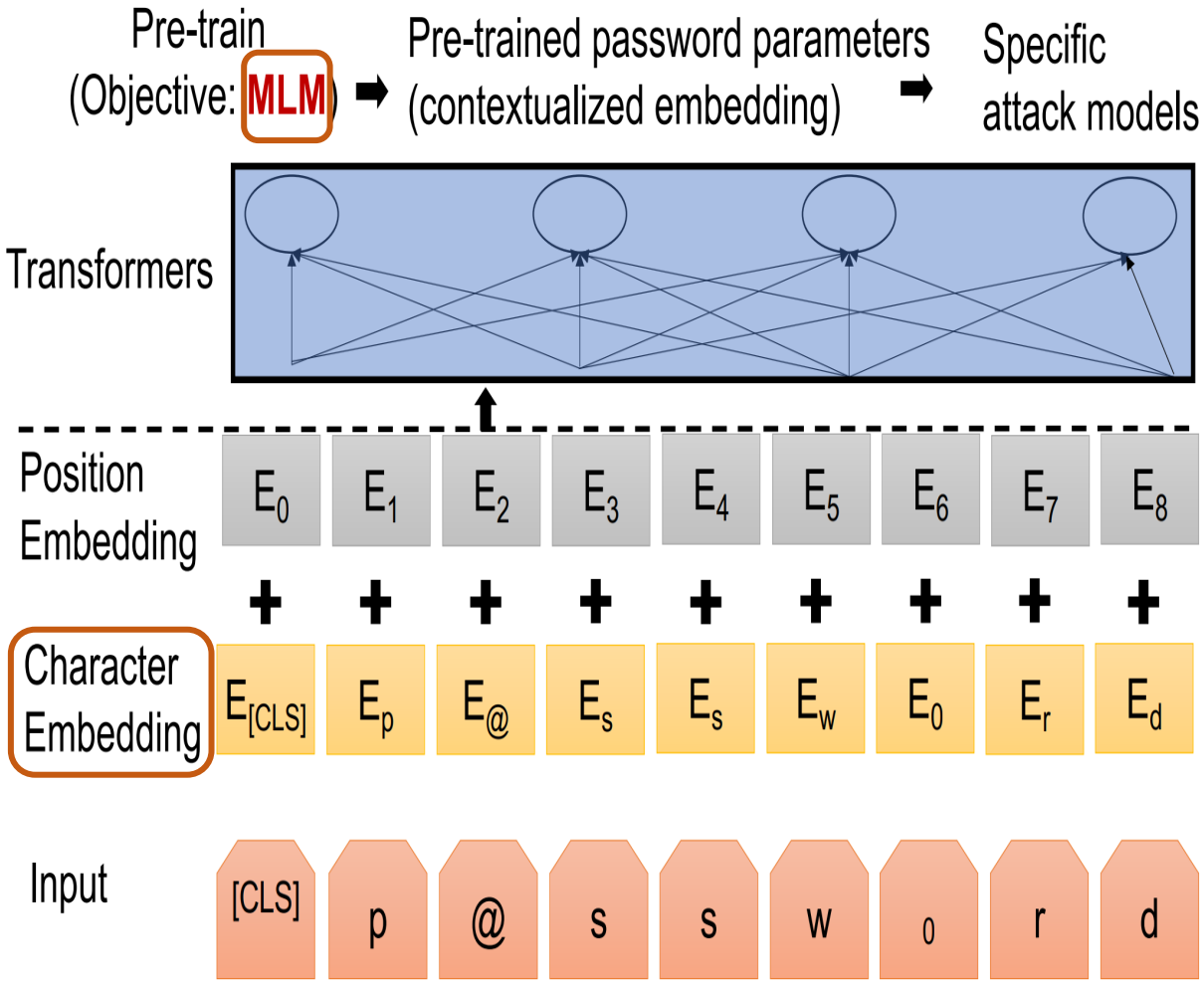
Framework – Pre-Training BERT



Transformer encoder (2,332,259)	
Layers	output shape
`Input layer`	[batch-size, seq-length]
`Embedding layer`	[batch-size, seq-length, 256]
`Transformer block`	[batch-size, seq-length, 256]
`Transformer block`	[batch-size, seq-length, 256]
`Transformer block`	[batch-size, seq-length, 256]
`Transformer block`	[batch-size, seq-length, 256]
`FullyConnected`	[batch-size, seq-length, 256]
`Output layer`	[batch-size, seq-length, 99]

https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

Framework – Pre-Training BERT



Transformer encoder (2,332,259)	
Layers	output shape
Input layer	[batch-size, seq-length]
Embedding layer	[batch-size, seq-length, 256]
Transformer block	[batch-size, seq-length, 256]
Transformer block	[batch-size, seq-length, 256]
Transformer block	[batch-size, seq-length, 256]
Transformer block	[batch-size, seq-length, 256]
FullyConnected	[batch-size, seq-length, 256]
Output layer	[batch-size, seq-length, 99]

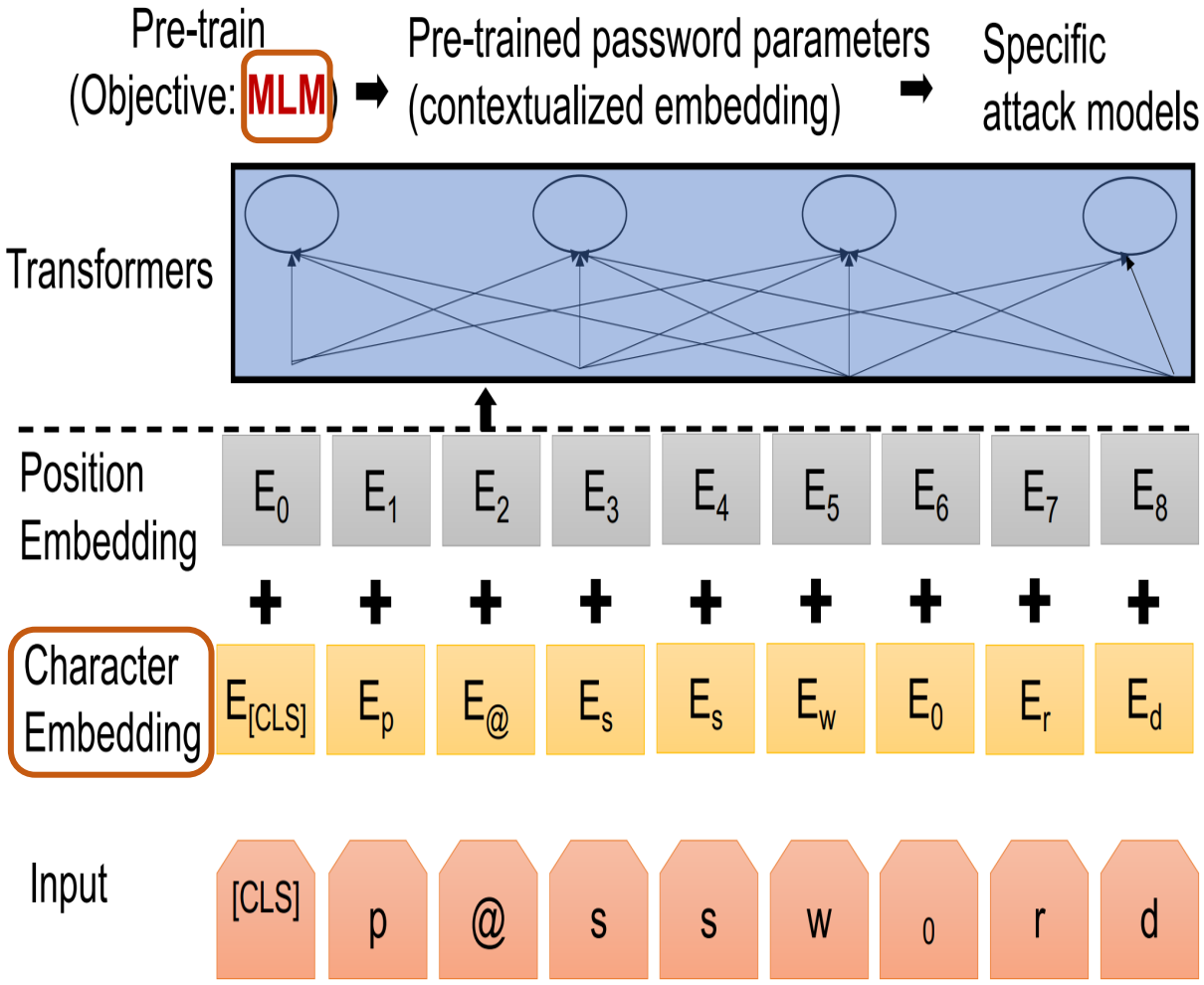
- Corpus: 95 ASCII characters + EOS, BOS, PAD, UNK
- Length Constraint: 32
- Masking: Same as BERT, Character-Level
- Same Architecture with BERT-Mini

https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

Part 3

PassBERT

Framework – Pre-Training BERT



Transformer encoder (2,332,259)	
Layers	output shape
Input layer	[batch-size, seq-length]
Embedding layer	[batch-size, seq-length, 256]
Transformer block	[batch-size, seq-length, 256]
Transformer block	[batch-size, seq-length, 256]
Transformer block	[batch-size, seq-length, 256]
Transformer block	[batch-size, seq-length, 256]
FullyConnected	[batch-size, seq-length, 256]
Output layer	[batch-size, seq-length, 99]

$$\underset{\theta}{argmax} \quad \frac{1}{|D_{training}|} \sum_{(pivot,pwd)\in D_{training}} \log P(pivot \rightarrow pwd|\theta)$$

https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

Real World Guessing Scenarios

Conditional Password Guessing CPG[1]

*a*s*o*d → passWord

Targeted Password Guessing TPG[2]

Dan1999 → D@n19!9

Adaptive Rule-based Password Guessing ARPG[3]

(a->@) → passWord → p@ssWord

[1] Pasquini et al., Improving password guessing via representation learning, SP '21
[2] Pal et al., Beyond Credential Stuffing: Password Similarity Models using Neural Networks, SP '19
[3] Pasquini et al., Reducing Bias in Modeling Real-world Password Strength via Deep Learning and Dynamic Dictionaries, USENIX '21

Part 3

PassBERT

Fine-Tuning BERT: CPG

CPG model (2,332,259)	
Layers	output shape
`Pre-trained layers`	[batch-size, seq-length, 256]
`FullyConnected` `Output layer`	[batch-size, seq-length, 256] [batch-size, seq-length, 99]

- Data: Rockyou 2009, 000Webhost, Neopets, Cit0day

- Keep Architecture, with same masking mechanisms.

$$P(pwd \mid pivot) = \prod_{c_i \in pwd,\ mask_i \in pivot} P(c_i \mid mask_i, pivot)$$

https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

Part 3

PassBERT

CPG – Evaluation

pivots	Neopets (%)				CitOday (%)			
	CE	**PT*	*VT*	*PT*	*CE*	**PT*	*VT*	*PT*
common	68.62	74.04	77.25	**80.02**	67.65	75.66	79.90	**83.23**
uncommon	77.35	73.88	79.40	**83.51**	69.30	72.80	76.18	**80.06**
rare	70.62	75.52	76.07	**79.72**	63.70	70.08	71.83	**76.48**
super-rare	69.86	59.51	62.25	**73.41**	45.90	46.11	47.86	**52.50**
average	71.61	70.73	73.74	**79.16**	61.64	66.16	68.94	**73.06**

- CE: CWAE(SoTA), *PT: PassBERT w.o. pre-training, initialized with random variable
  VT: Vanilla BERT, PT: PassBERT
- Four classes of pivots: When pivot(i.e. p*ssw*r**) is fed into model, model's # of response which fits into each classes. Suggested by CWAE model.
  Common(1k~15k), uncommon(50~150), rare(10~15), super-rare(1~5)
- Pivots are at least consisted of four un-masked characters and five masked one.

https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

Real World Guessing Scenarios

Conditional Password Guessing CPG[1]

*a*s*o*d → passWord

Targeted Password Guessing TPG[2]

Dan1999 → D@n19!9

Adaptive Rule-based Password Guessing ARPG[3]

(a->@) → passWord → p@ssWord

[1] Pasquini et al., Improving password guessing via representation learning, SP '21
[2] Pal et al., Beyond Credential Stuffing: Password Similarity Models using Neural Networks, SP '19
[3] Pasquini et al., Reducing Bias in Modeling Real-world Password Strength via Deep Learning and Dynamic Dictionaries, USENIX '21

Fine-Tuning BERT: TPG

TPG model (7,077,026)	
Layers	output shape
`Pre-trained layers`	[batch-size, seq-length, 256]
`FullyConnected` `Output layer`	[batch-size, seq-length, 256] [batch-size, seq-length, 9122]

- Data: BreachCompilation, Collection#1
  [User Email, [password#1, #2 …]

- With four operations – replace1, replace2, keep, delete
  Maximum Edit Dist. = 4, # of operations: 1(keep)+1(delete)+95(rep.1)+95^2(rep.2) = 9122

$$P(pwd \mid pivot) = \prod_{c_i \in pwd,\ mask_i \in pivot} P(c_i \mid mask_i, pivot)$$

https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

Part 3

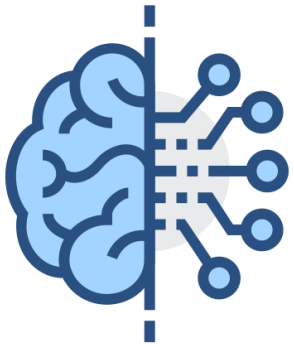
PassBERT

Fine-Tuning BERT: TPG

TPG model (7,077,026)	
Layers	output shape
`Pre-trained layers`	[batch-size, seq-length, 256]
`FullyConnected` `Output layer`	[batch-size, seq-length, 256] [batch-size, seq-length, 9122]

Dan1999

Leaks



Generates

D@n19!9

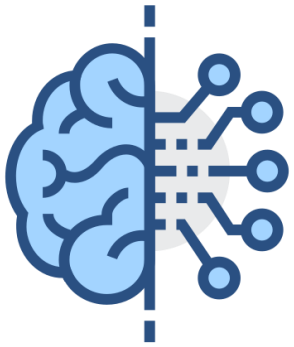
Part 3

PassBERT

Fine-Tuning BERT: TPG

TPG model (7,077,026)	
Layers	output shape
`Pre-trained layers`	[batch-size, seq-length, 256]
`FullyConnected` `Output layer`	[batch-size, seq-length, 256] [batch-size, seq-length, 9122]

Leaks



Generates

D@n19!9

Dan1999

Old PW	D	a	n	1	9	9	9
Operation	keep	Rep.1	keep	keep	keep	Rep.1	keep
Generated	D	@	n	1	9	!	9

TPG – Evaluation

Attack model	BreachCompilation (%)			Collection#1 (%)		
	10	100	1,000	10	100	1,000
Pass2path	6.42	11.52	14.71	4.37	10.84	14.98
*PassBERT	12.63	15.67	17.94	11.21	15.42	18.22
Vanilla BERT	**12.72**	**15.79**	**18.01**	**11.35**	15.45	**18.23**
PassBERT	12.68	15.71	17.96	11.24	**15.47**	18.21

- Pass2path: SoTA model. Used same evaluation principle.
  * Given ⟨username, ⟨pw_ 1, pw_ 2⟩⟩ list, picks one password, and guesses other password within 1k attempts. Metric: Number of cracked / Number of Accounts
  * Randomly sample 100k data which fits user has two passwords.
- 10, 100, 1000 indicates number of guesses.

https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

Part 3

PassBERT

Real World Guessing Scenarios

Conditional Password Guessing CPG[1]

*a*s*o*d → passWord

Targeted Password Guessing TPG[2]

Dan1999 → D@n19!9

Adaptive Rule-based Password Guessing ARPG[3]

(a->@) → passWord → p@ssWord

[1] Pasquini et al., Improving password guessing via representation learning, SP '21
[2] Pal et al., Beyond Credential Stuffing: Password Similarity Models using Neural Networks, SP '19
[3] Pasquini et al., Reducing Bias in Modeling Real-world Password Strength via Deep Learning and Dynamic Dictionaries, USENIX '21

Part 3

PassBERT

Fine-Tuning BERT: ARPG

Table 12: Architecture of the ARPG attack model. We train two networks for two rules-set of *PasswordPro* and *Generated*, whose size of neural networks is 3,998,000 and 9,952,904.

ARPG model (3,998,000; 9,952,904)	
Layers	output shape
`Pre-trained layers`	[batch-size, seq-length, 256]
`FullyConnected` `Output Layer`	[batch-size, seq-length, 256] [batch-size, rules-set-size]

- Data: BreachCompilation(Target PWs), 000Webhost(WordList)
  [User Email, [password#1, #2 ···] / [password, 123456, qwerty ···]

- Rule set : PasswordPro(3,120) and Generated(14,278)
  [delete last char, append_ 1, prepend_ 3···]

- Given word from WordList, checks whether Target PW can be generated within Rule set.

https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

Part 3

PassBERT

Fine-Tuning BERT: ARPG

Table 12: Architecture of the ARPG attack model. We train two networks for two rules-set of *PasswordPro* and *Generated*, whose size of neural networks is 3,998,000 and 9,952,904.

ARPG model (3,998,000; 9,952,904)	
Layers	output shape
`Pre-trained layers`	[batch-size, seq-length, 256]
`FullyConnected` `Output Layer`	[batch-size, seq-length, 256] [batch-size, rules-set-size]

- Data: BreachCompilation(Target PWs), 000Webhost(WordList)
  [User Email, [password#1, #2 ···] / [password, 123456, qwerty ···]

- Rule set : PasswordPro(3,120) and Generated(14,278)
  [delete last char, append_ 1, prepend_ 3···]

- Given word from WordList, checks whether Target PW can be generated within Rule set.

https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

Fine-Tuning BERT: ARPG

- WordList: [“password, “123456”, “qwerty”]
- Rule set : [“delete last char”, “append_1”, “prepend_3”]
- Target PWs : [“password1”, “1234561”, “123qwerty”]

Fine-Tuning BERT: ARPG

- WordList: [“password”, “123456”, “qwerty”]
- Rule set : [“delete last char”, “append_1”, “prepend_3”]
- Target PWs : [“password1”, “1234561”, “123qwerty”]

```
train_data = [
    {"input": "password", "labels": [0, 1, 0]},
    {"input": "123456", "labels": [0, 1, 0]},
    {"input": "qwerty", "labels": [0, 0, 1]},
]
```

Rule Classification!

ARPG– Evaluation

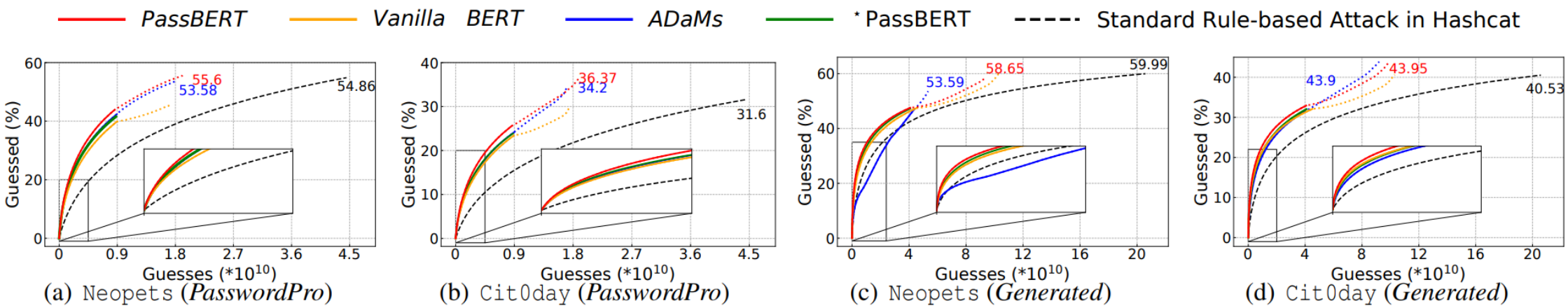


Figure 4: Cracking rates of ARPG, where we show the final cracking rates of dynamic models. The dotted line is the result with the dynamic strategies in respective attack models.

https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

ARPG– Evaluation

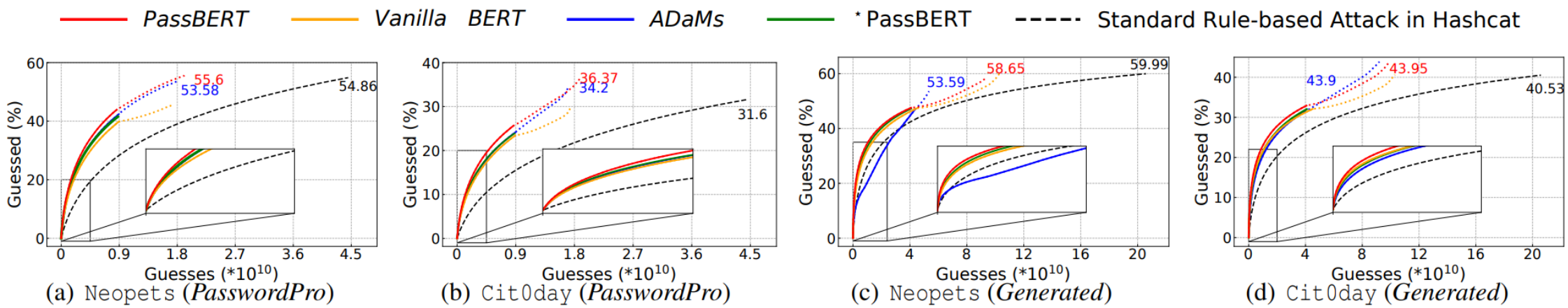


Figure 4: Cracking rates of ARPG, where we show the final cracking rates of dynamic models. The dotted line is the result with the dynamic strategies in respective attack models.

	PasswordPro		*Generated*	
	Neopets	Cit0day	Neopets	Cit0day
1	]	]	**$1**	**$1**
2	**$1$2**	]]	$0Z1	’8
3	Z1	Z1	$2	c$1
4	]]	D2	@0Z1	c’8
5	$1$2$3	[	$2Z1	’7$1

Permutation functions	Descriptions	Example	Output
]	Delete the last character	]	passwor
$x	Append the character x to the end	$1	password1
Zn	Duplicate last character n times	Z2	passworddd
@x	Purge all instances of x	@s	paword
’n	Truncate word at position n	’6	passwo

https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

Part 3

PassBERT

Hybrid Password Strength Meter

- Hybrid approach using CPG, TPG, ARPG for comprehensive password security assessment.

- CPG: Assesses the strength of individual characters in a password.

- TPG: Estimates the number of guesses required to crack a password based on leaked data.

- ARPG: Identifies base words from complex passwords to assess vulnerability to adaptive attacks.

https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

Part 3

PassBERT

Hybrid Password Strength Meter

CPG

TPG

ARPG

character strength level:	
potential risks from target guessing attacks:	The input of "p@ssw0rd123" can be cracked when trying 825 guesses given the leaked "p@ssw0rd", make it more complex!

https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

Discussion and Future work

Discussion

- Pre-training works, works way better than plain BERTs, even compared with SoTAs.
- According to [4], high correlation between natural language and password, which is key for the success.
- Achieved SoTA for three guessing scenarios.

Future work

- Dubious about pre-training with Rockyou-2021,
  - Data consisted of passwords and word dictionaries. Author manually sampled 60M data.
  - However, the ratio of password should not be high. BERT's born capability is key?
  - Performance is mainly due to fine-tuning stage, fine-tuned with breached PWs.
- Changing to LLMs, i.e. GPT-4, Llama can do this job better, if [4] is valid.
- Author explored semantic relationship between characters, but what if tokens?

[4] D. Wang, H. Cheng, P. Wang, X. Huang, and G. Jian. Zipf's law in passwords. IEEE Trans. Information Forensics and Security, 12(11):2776–2791, 2017.
[5] Xu et al., Chunk-level password guessing: Towards modeling refined password composition representations. ACM SIGSAC '21

Thank You! Any Questions?

Part 5

Appendix

CPG vs. TPG vs. ARPG: Deeper Dive

Table 1: Summary of labeled datasets used in three real-world attacks, where all attack models' objective is to predict its labels (Y) given the input (X) as accurately as possible.

Attacks	**Labeled sets** a set of data X with its set of *labels* Y	**Example** $(x \longrightarrow y)$ with $(x \in X, y \in Y)$	**Explanations of labels Y**
CPG	partial passwords with *complete passwords*	p***w0rd*** $\longrightarrow$ *p@ssw0rd123*	complete passwords are those matching partial contexts
TPG	passwords with *minimal edit paths*	p@ssw0rd123 $\longrightarrow$ [(*delete, 8*), (*delete, 9*), (*delete, 10*)]; We can change it to "p@ssw0rd", where the [(*delete,8*)] refers to delete the character 1 in the eight position.	the minimal edit path (i.e., the shortest sequence of edit operations) refers to the shortest path to change the current password to another password from the same user.
ARPG	words with *hit rules*	p@ssw0rd123 $\longrightarrow$ [*delete last three characters*, *duplicate the last character once*]	hit rules are a subset of rules-set that is pre-processed the word and rule mappings upon two hypothetical datasets

https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

Part 5

Appendix

TPG performance

Table 6: Statistics of edit distance distribution by PassBERT within 1,000 guesses. For example, in `BreachCompilation`, we can crack 12,102 (98.5%) passwords among the total passwords with one edit operation.

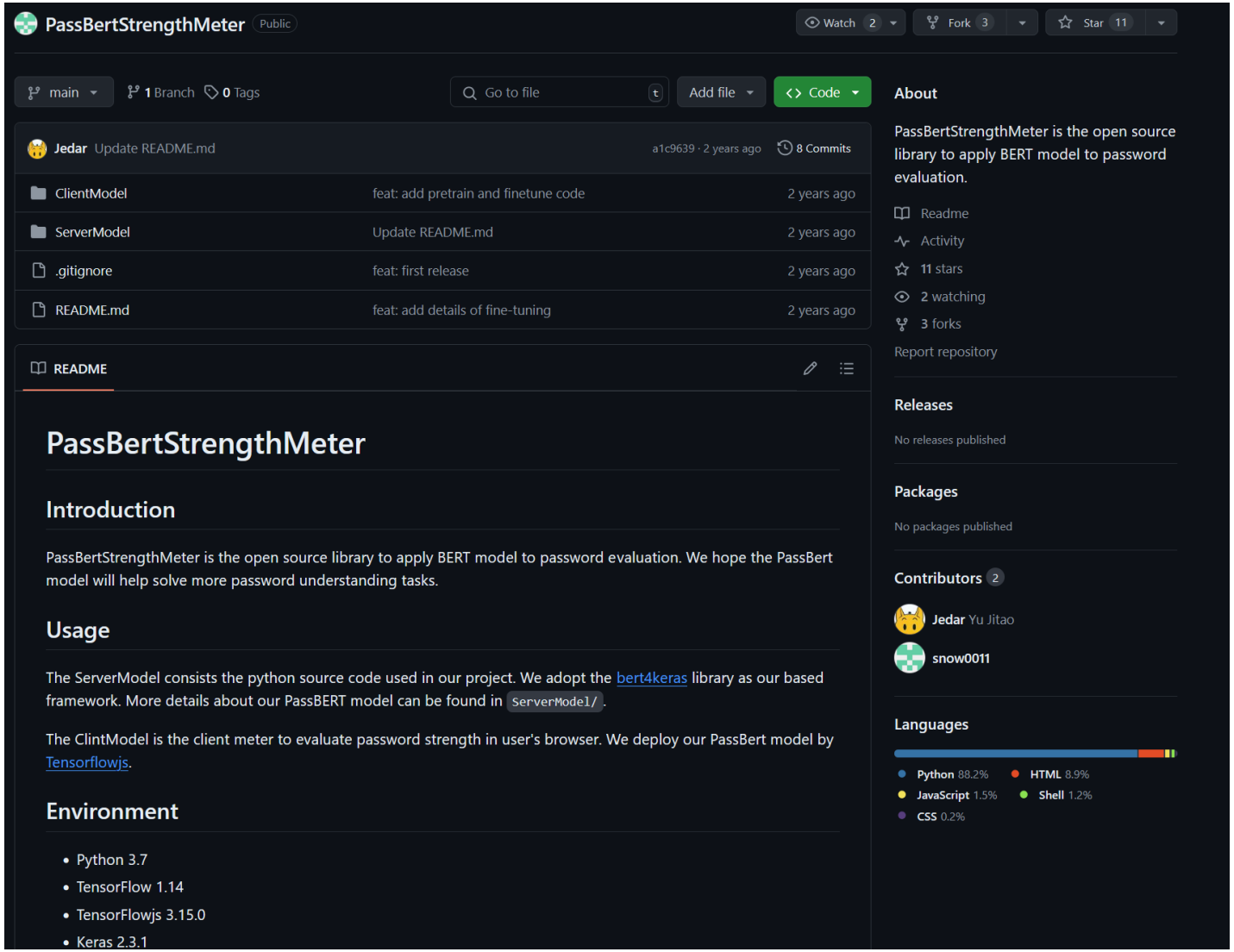
Edit distance	`BreachCompilation (4iQ)`	`Collection#1`
1	**12,102 (98.5%)**	**122,17 (96.7%)**
2	16,316 (91.7%)	16,257 (88.6%)
3	17,552 (84.8%)	17,792 (81.5%)
4	17,923 (75.4%)	18,175 (72.8%)
>4	39 (0.05%)	39 (0.05%)

https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

Part 5

Appendix

PassBERTStrengthMeter Github



https://github.com/snow0011/PassBertStrengthMeter