PassGAN Implementation

Taylor Barmak, Rachel Benton,
Tanay Bapat, and Nithin Vijayakumar

December 6, 2020

Motivation

Paper: https://arxiv.org/pdf/1709.00440.pdf

Paper Motivation
- Makes the case for using stronger passwords
- To improve on HashCat and John the Ripper
- To discover more passwords by comparing hashes
- Novel application of GANs (applying to text instead of images)

Our Motivation
- Learn more about Generative Adversarial Networks
- Gain experience with Deep Learning applications
- Learn how to protect against password cracking

Background

- PassGAN is a tool that can generate passwords based on an input dataset.
- This tool uses a Generative Adversarial Network, which utilize Generator/Discriminator subnetworks that constantly evaluate the other, so each gets more adept at their task.
- The Generator therefore gets trained on how to generate passwords that are similar to the input distribution, to “trick” the Discriminator, which tries to distinguish between generated and real passwords.
- The end tool would just be the Generator Network, which can generate billions of passwords.

Related Work

- Two popular rule-based password guessing tools are John The Ripper and HashCat. They use multiple types of password guessing strategies.
- Markov models were used to generate password guesses by Narayanan. Markov models use manually defined password rules.
- This technique was improved on with Probabilistic Context-Free Grammars in a paper by Weir. These PCFGs demonstrated how to learn rules from password distributions.
- Ma and Durmuth have extended the work on PCFG.
- The first paper using neural networks on passwords is in 2006 by Ciaramella.
- Recently, Melicher et al. wrote a paper about using recurrent neural networks on passwords. This, however, was used to assess the strength of a password.

Claim / Target Task

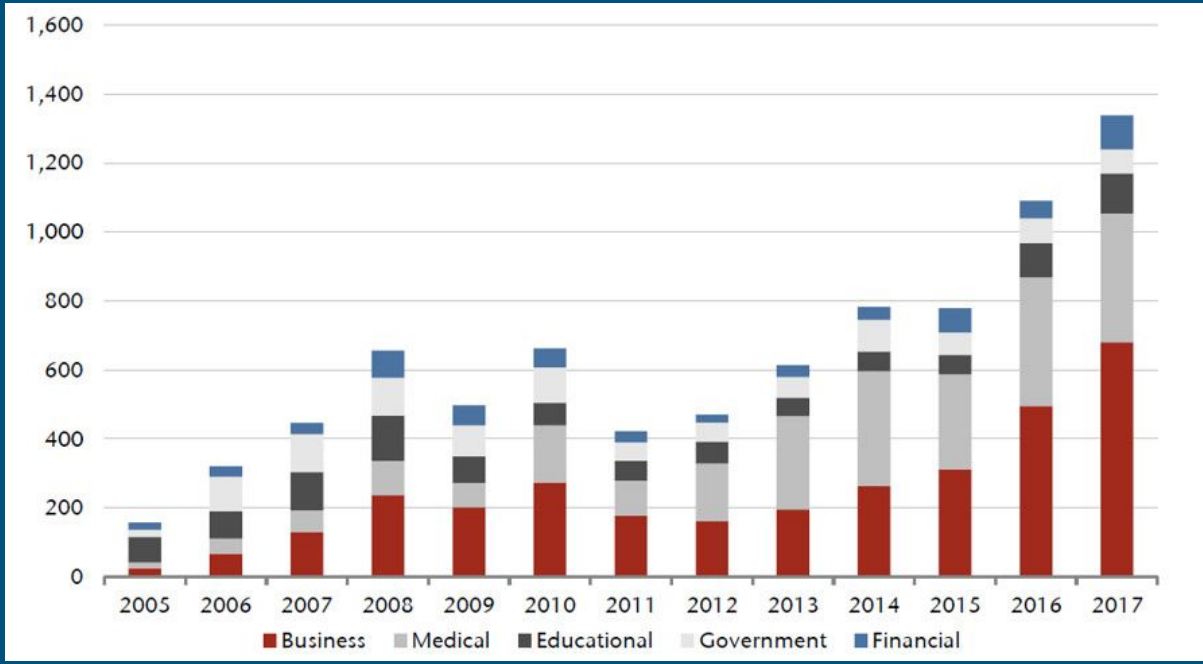
- Our target task is to reproduce this paper and implement PassGAN, and attempt to find areas for improvement.
- We will use PyTorch to build a Generative Adversarial Network to create passwords and test them against known hashes.
- We want to reproduce the novel results found in the paper on the RockYou and LinkedIn datasets.
- In addition, we would like to test our implementation on several other similar data sets that were leaked after the paper's initial release.
- We will show the importance of not using common passwords or the same password for multiple websites.

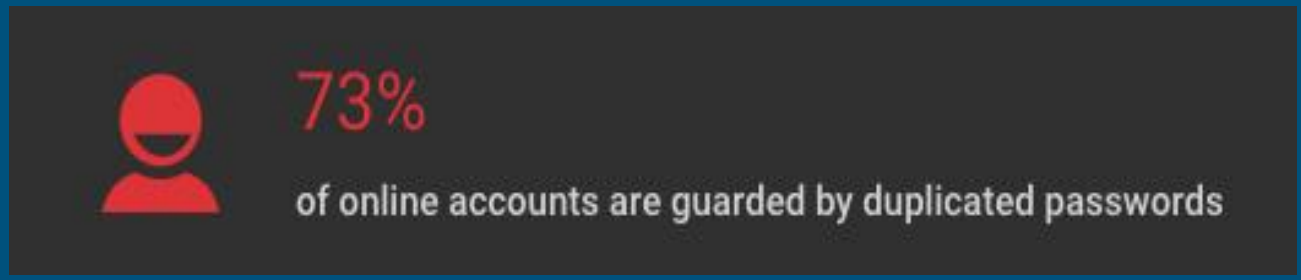
Why needed (I)

- Current bleeding-edge password-guessing tools:
  - HashCat & John the Ripper
    - Check billions of passwords per second against password hashes
    - Utilizes dictionary-generation rules
  - Issues
    - limited success due to *specialized expertise needed*
    - Very *laborious* algorithms
    - Needs to be tuned on new data to work effectively
- Figures demonstrating *need for password security research*:
  - (See Next Slide)

Why needed (II)

Data Breaches Over Time





Proposed Solution

- Recreate PassGAN tool exactly according to the Paper
- Use a series of Residual blocks composed of 1-dimensional convolutional layers for the Generator and Discriminator
- Will create passwords using a Generative Adversarial Network (GAN) on PyTorch
- Planning to use LinkedIn and RockYou datasets (from paper)
  - to test tool and two of the paper's main claims
- Additionally, test tool on Dubsmash dataset
  - a leak of 160M+ passwords
  - so far, roughly 20M have been cracked publicly

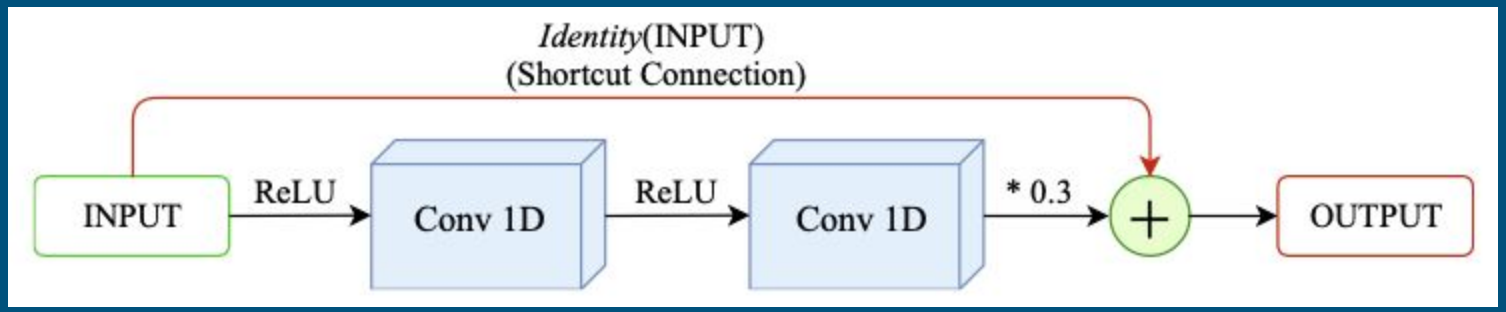
Implementation

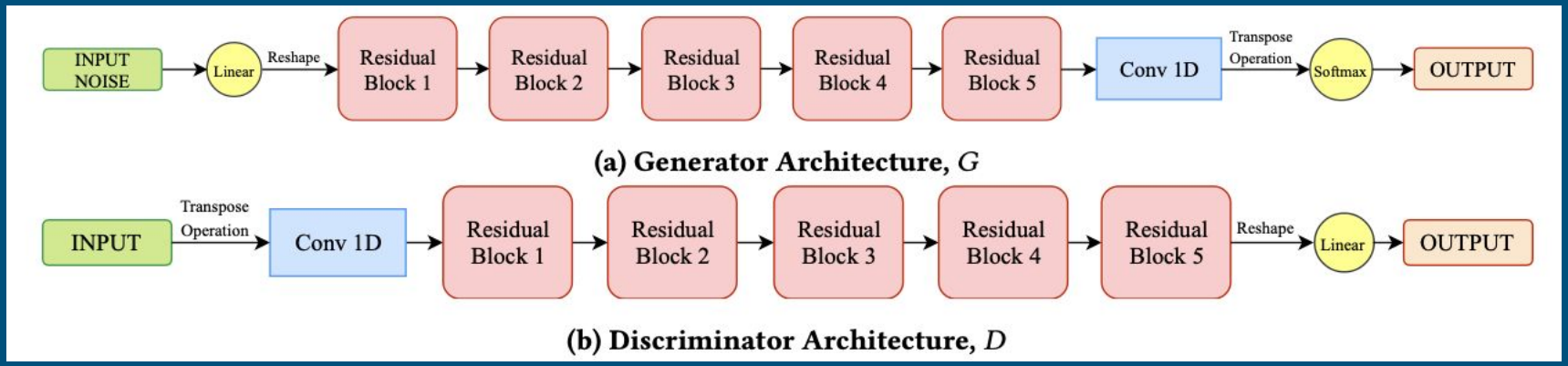
- Built using Pytorch following the exact architecture and techniques used in the paper
- Relied on the WGAN-GP paper, on which PassGAN built off of

while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

Implementation

- Generator and Discriminator composed of Residual Blocks
- Residual Blocks have shortcut connections which help prevent vanishing/exploding gradients





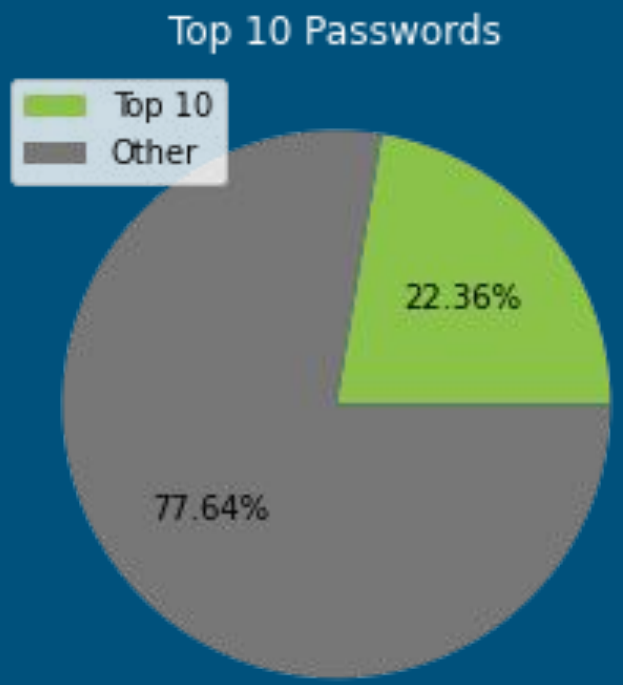
Data Summary

- Dubsmash dataset contains 22,119,460 cracked passwords
- Each line in the dataset had a hashed version of the password separated from the plaintext version by '=:'.
  - `pbkdf2_sha256$12000$00Lj5R6V38Kz$+dnX5a0ILWN/7vomuw1KCtebUP3jEXVeOE1pIJgKpW8=:purple11`

- We processed the data by extracting the plaintext password from each row.
- We only used passwords that were 10 characters in length or less.
  - 20,482,352 passwords (92.5%)
- '|' was added to pad passwords fewer than 10 characters.
  - `password|| hello||||| 12345|||||`

Data Summary

Password	Frequency
12345	2497511
123456	600495
123456789	504007
12345678	279592
1234567890	173441
dubsmash	144141
1234567	110885
password	95766
qwerty	95478
00000	77946



Code Walkthrough

Experimental Results

- Later iterations learned more natural sequences of characters
- Every 15,000 iterations, the dataset was shuffled and repeated

Iteration 1	o0['qJsx3P	M63VK}],hZ	{|wY*Zn9MK
Iteration 1,000	Sacdmama||	1utn1|34||	cdbdb123||
Iteration 5,000	lucexOQOQ@	12315|||||	chschos122
Iteration 10,000	Veyminop9|	morebbet||	gasmort|||
Iteration 25,000	pugharda||	almarty8ip	1234567893
Iteration 75,000	madia|||||	brace2013|	pania|||||
Iteration 158,000	gameett!||	asheelee18	12345678on

Experimental Analysis

- We generated 1,000,000,000 passwords
  - 16.3% were unique → 163,341,589
- The passwords captured 47.68% of the passwords of the unseen test set
- Sample of matched passwords
  - `mollyy angles99 sahara123 melinda 28082003`
- Sample of unmatched test set passwords
  - `joey1021 1607681 9946805264 purple01 07272001`
- Sample of unmatched generated passwords
  - `miay1333 to412515 Porthe608 Perkla98 Aohstriey`

Contributions

- We have built an implementation of the PassGAN paper

Why ours is new/significant:

- Ours is at a higher level of Python and PyTorch
  - ‘Most updated’ version of PassGAN
- Previous papers used deprecated software and libraries
  - Our code makes this process more accessible for people to experiment with
- Applying it to a new dataset
  - Demonstrates ability of tool to be generalized
- Reemphasize the importance of having different passwords
  - Show one of the ways passwords can be found

Constraints and Limitation

- Time Constraints:
  - Running the code took a significant amount of time
  - Didn’t allow for as much adjustment/finetuning throughout the project period
  - *Datasets*: Limit on how many different datasets could be used - each takes a significant amount of time
  - *Iterations*: Limited number of iterations can be done, limited our results, which limits the demonstrable capability of our tool
- Availability of password data to use
  - Limited selection options for password datasets
  - Limits how we can demonstrate ability of tool
- Couldn’t have worked with multiple datasets efficiently
  - Difficult to parallelize - related to time constraints

Conclusion and Future Work

- Using GANs is an effective way to generate passwords from a set of known passwords
- It can be generalized to datasets outside of the datasets used in the original paper
- These concepts are still SOTA in higher levels of Python
  - Reasserts efficacy of the PassGAN method
- We would like to do further research with
  - Training and testing splits within a single dataset vs training on multiple websites
  - Training with datasets on sites that have passwords with greater requirements
  - Trying passwords with different length requirements

References

Hitaj, Briland, et al. “PassGAN: A Deep Learning Approach for Password Guessing.” *Applied Cryptography and Network Security Lecture Notes in Computer Science*, 14 Feb. 2019, pp. 217–237., doi:10.1007/978-3-030-21568-2_11.

“More than 96% of Consumers Are Sharing up to Six Passwords with Other People.” SaferPass Blog, 14 July 2016, blog.saferpass.net/more-than-96-of-consumers-are-sharing-up-to-six-passwords-with-other-people/.

Reklaitis, Victor. “How the Number of Data Breaches Is Soaring - in One Chart.” *MarketWatch*, MarketWatch, 25 May 2018, www.marketwatch.com/story/how-the-number-of-data-breaches-is-soaring-in-one-chart-2018-02-26.

s3in!c. “Leak 'Dubsmash.com'.” Hashes.org - Leak 'Dubsmash.com', hashes.org/leaks.php?id=1793.