Targeted Password Cracking with OSINT Data

Benjamin Rader
IUPUI
Purdue School of Engineering & Technology

Abstract—Password convenience stems from how humans choose passwords out of memorability. Moreover, passwords can closely relate to publicly available information connected to the person that creates them. Therefore, OSINT (Open Source Intelligence) techniques can be used to curate memorable and publicly available points of data around that entity. The substantive parts of the data and unique keywords can then be used to build wordlists for targeted password-guessing attacks against a particular user's password hash. Traditionally this involves combing keywords into massive combinations while appending other common words, patterns, or characters based on assumptions about password choice. However, many passwords are memorable because they form readable phrases, involve a human understanding of entities, or the password uses content words like pronouns in ways that are unique to human thinking. I propose that the security of these types of memorable passwords hinges on the ability of AI models to generate these sorts of targeted wordlists and on the availability of such models to mal-intent individuals. I conducted targeted wordlist generation using a traditional manual permutative approach with 2 public tools (Cordialie and Mentalist) along with an AI and machine learning approach using 2 more public tools (GPT3 and OMEN+). All methods utilize a custom wordlist composed of data that can be obtained through social media and sometimes compromised plaintext password datasets are used to train models. The author's social media page is used as an example, and generated wordlists are compared to an old password using string similarity algorithms such as Damerau-Levenshtein and the Jaccard Index. Results show that GPT-3 and OMEN+ wordlist generation underperforms when compared to even the more naive Cordialie program. I hypothesize that AI and ML-based implementations are currently more useful for cracking large compromised sets of password hashes and that manual permutative methods are currently more performant in targeted attacks. In general, targeted attacks require more sophistication and maturity before becoming a viable method for the average hacker.

1 Introduction

There are three types of factors of authentication. That is, there are three general categories to ponder when looking at how to validate an identity. One can authenticate someone's identity by looking at what they know, what they are, or what they have. [11] What one may have could be some form of a key, maybe an item that no one else knows about except the entity that is validating your identity, or even a literal key whether it utilizes digital or physical means. Locks authenticate people because they have a key that has notches which match the configuration of the pins in the lock core [32]. What you are could be hidden in your DNA, your fingerprints, your eyes, or even your voice. Although it could be argued these are a form of what you have, these generally refer to biometric-related keys. Lastly, what you know refers to information or data stored in your brain. The most common manifestation of "what you know" authentication are passwords.

Passwords are still the dominant method for authentication [27]. 80% of data breaches involve password attacks at some step of the attack chain [8]. Research also shows that using MFA (multi-factor authentication) can stop 99.9% of attacks [31]. However, it is unrealistic to think that every person and organization has the knowledge or the will to execute such a task. Therefore, we must look at the root issues with passwords, considering that most things are secured with them in some form.

1.1 Basics of Password Cracking

A login form that utilizes an email address and password input are self-explanatory. A complication arrives when one focuses on the comparison step where the password given is compared with the password stored by the authenticator. An insecure implementation will directly compare the given password input with the plaintext password that is associated with the email address. The attacker will immediately have the information needed to login as long as they can see this data in transit or in storage. This leaves for a much easier attack. In a secure implementation, the password is not stored in plaintext but rather as a "hash [25]."

A hash is simply the output of a hash function, and in this case, the input for the hash function is a password and the outputted hash is really just a set of pseudo-random characters. A hash function uses a "trapdoor" or one-way function. If given an input, say a string, then that string input will always generate the exact same outputted hash. In other words, it must be deterministic. Additionally, there are other characteristics that make up hash functions: it should be very unlikely that two different inputs produce the same output, should be easily computed, and accept all input lengths. Most importantly, the one-way function must be that - one way. Another word for this is preimage-resistance. If an attacker has the outputted hash, then it should be impossible to find the input of that hash without implementing a brute-force attack or by trying inputs [13]. There cannot be a guaranteed way to find the input of the hash which is faster than just trying inputs and comparing them to the known hash. This is how password cracking is done. The attacker will simply try inputs with the same known hash function till they obtain the same hash output. There could be collisions or chances of finding a different

input for the same output but this is unlikely [25].

1.2 Stats on Password-Related Attacks

First off, password reuse is a rampant issue especially for attack vectors like password sprays and the more targeted hacks. One effort was done by SpyCloud over a set of 1.5 billion compromised log-in combinations (email and password), and they found that 60% of the credentials were reused across multiple accounts [24]. A password manager is a quick fix, however it does not address the root problem. Password strength is an issue already. Data shows that most people still use short passwords that can be cracked in a fraction of a second and websites still allow this [7]. Some websites have forced a change or forced the use of MFA. However, even this cannot completely mitigate future issues. That is, once artificial intelligence is leveraged more for password cracking, then memorized passwords could become more obsolete.

1.3 Are Passwords Secure?

Passwords that are long enough and with enough entropy are indeed, at the moment, secure. This is especially true for the average person which is likely not the target of a nation-state level threat with the means to crack very long passwords. This doesn't take into account other attack vectors. However, there are cases where it is reasonable to assume that nation-state level threats are "brute-forcing" longer passwords. This could be the case for internal passwords such as one for root access or admin rights where MFA maybe isn't used. Such attacks are not well-known because of their nature, but the possibility is not unreasonable. However, these types of brute force attacks are easier to quantify and much harder to pull off, unless you have massive capital for computing infrastructure [4]. The real potential problem is AI-based attacks that can crack memorable passwords.

2 Literature Review

2.1 Passwords and Memory

Considering passwords are a form of "what you know" authentication, then the definition implies that you need to have the ability to remember it. One survey showed that 41% of people memorize their online passwords [30], and other surveys say that 53% of people memorize them [18]. This is the case, even for most password managers, where you need to know the password to the password manager. The generated passwords from those various tools themselves might not be feasibly memorable or at least not cost effective for using the password managing tool. Still, the main password is, and most people don't even use password managers for a number of reasons: suspicious of the security, inconvenient, or didn't know about them. Therefore, one can assume that most passwords are memorable ones.

Evidently, memorable passwords have quite of few inherent flaws. If someone knows you enough, then they can probably guess your password. Unfortunately or not so unfortunately, language is complex and people can remember all sorts of things or simply not share memorable things with others or publicly online. Additionally, you must have the intuition on how humans construct their passwords [16]. Nonetheless, the nature of social media and the lack of infrastructure on the Internet that cultivates privacy practices with personal data creates opportunities for cracking memorable passwords. Not to mention, most users never check their privacy settings so it is likely that this information can directly or indirectly be obtained [19].

However, it should be mentioned that such vulnerable memorable passwords only have weaknesses because the memorability is connected to personally-related information. Some password memorability characteristics rely on the memorability of patterns in cases of slightly more security-aware users. Some experiments showed that encouraging users to create a "password formula" is effective at creating memorable and strong passwords that are not loosely connected to the person in any way. By memorizing the formula or pattern, the user can expand the formula to limitless passwords while only needed to memorize the pattern or formula [35].

Some scholars such as Woods and Siponen, argued that this is merely a problem of users not having motivation to remember, confidence to remember, or a basic understanding of memory [33]. Another experiment they conducted, showed that having the user verify their password often or multiple times increased memorability from around 40% to 70% [34]. However, people still have, on average, around 40 passwords [18], and most people use password managers for this very reason [30]. In theory, it may seem like a good idea to simply improve your memory, but this in inconvenient.

2.2 Password Managers

Password Managers can directly combat any problem of password memorability because the passwords that are stored in the manager application do not have to be remembered. Additionally, most password managers have password generators built into them which can generate passwords or passphrases that can't possibly relate to the user in any way. Essentially, these passwords have lots of entropy and are practically random [31]. In Verizon's annual data breach report, they had recommendations for companies. The top three recommendations were to use two-factor authentication (not just a password), do not reuse passwords, and to use a password manager [8]. In another study from Yubico and the Ponemon Institute, it was found that most respondents, including ones in IT fields, did not improve credential management by using a password manager, but rather they simply used "stronger" passwords. This is problematic, because password strength will, at some point, mean that the password is inherently hard to remember. Not using a password manager can slowly turn into a problem [21]. Some conducted surveys show that people don't use password managers because they are not sure they need one, or they are not sure that the password managers are secure [30]. Even more problematic, a 2019 study from Carnegie Mellon University indicates that only a small portion of password manager users utilize

the password generator [1]. Using password managers still requires that users choose good passwords.

2.3 Untargeted vs Targeted Guessing

With password memorability in mind, there two ways that threat actors can take advantage of this weakness. Firstly, attackers can learn the general nature of memorable passwords and use the learnt patterns to guess other passwords. With these patterns, they can go for a "wide-nozzle" type of attack and attempt to crack a list of password hashes - hoping to exploit the low hanging fruit. Such an attack doesn't focus on the individual characteristics of the victims who created the passwords. The wide-nozzle or password spray relies on the general patterns. Considering that most people use weak passwords [18], this type of attack can be cost effective in saving time while still resulting in cracked hashes. On the contrary, a second extension of this attack can combine the macro-social characteristics of password memorability with the individual components: personal data, open source intelligence, and even personal acquaintance. In other words, we can combine peoples' password patterns with the data points of a specific person to pull off a targeted attack. However, this type of targeted attack assumes you understand the target well enough, and it also assumes that the password cracker targets patterns that the target happens to use. There is a high chance that once of these pieces are missed considering the sheer scale of language and personally-related keywords [17].

2.4 Wordlist Generation (Creating Dictionaries)

As explained in the introduction, password cracking generally refers to hashing inputs till the desired hash function output is obtained. In which case, this means that the password has been found. Password cracking and especially brute-force approaches, rely on computational efficiency when working with larger passwords. Lists are simply the most efficient way to try inputs for hash functions. Most password cracking software or scripts work with the GPU (graphics processing unit) or even specialized processors which can crack passwords at ludicrous speeds. These tools maximize the computational efficiency and leave the guessing methods up to the user who is knows the patterns. The name of the game isn't password cracking but rather wordlist generation. The best generated wordlist should include strings that are close in mathematical proximity, semantics, and structure to the actual password or passphrase.

I am defining two approaches to generating wordlists (targeted or untargeted):

1) Explitcly Defined Generation
2) AI Defined Generation

2.5 Explicitly Defined Generation

One can view the process of creating wordlists as always involving a neural network of sorts. For the case of explicitly defined generation, wordlist creation is manually guided and defined by the user or person. To an extent, the password cracker's mental model becomes a tool for generating wordlists. However, it is absurd to think that a person can output enough guesses to crack any passwords. Most passwords will take hundreds of thousands of guesses, even with smart guessing strategies. It is more pragmatic to take an automated password mangling approach which utilizes human chosen keywords that are related to the victim and how passwords are creating. Password mangling has two parts. First, various patterns and common password structures are assumed or defined. Second, these patterns and structures are combined with keywords or any data points that are applicable to the victim. The password mangler or wordlist generator uses patterns, assumptions, or keywords to create combinations of points of data and common password components to generate wordlists of likely passwords. There also exists ways of automatically training password cracking tools to generate mangling rules from datasets [6]. Mangling is a programmatic and manual approach to creating wordlists, but can still be very powerful and effective especially since deep learning and machine learning cannot easily make the same assumptions about patterns as humans.

2.6 AI Defined Generation

AI defined generation uses AI models instead of human thinking to structure, define, and create wordlists. AI models can guess those which password manglers can't. That is, unless AI or ML models are being use to generate mangling rulesets or define patterns to be used for wordlist generation. In those cases, the success rate for password cracking has shown to be dramatically better than a person defining the mangling rules themselves. This is because every person has their own assumptions about how individuals create their passwords [20] [3]. The difference is that AI models can, to an extent, learn these patterns and apply them. This can manifest itself in the form of supervised or unsupervised learning, but the main point is that the AI model is applied instead of human thinking to generate the wordlists or permutative and mangling rulesets.

Some of these AI models rely on phonetic patterns, some on keyboard path biases, some on semantic patterns [36], and even some on knowledge and ontology relations such as those that make up Wikipedia [12]. It seems that most of these AI models are not flexible in the types of password patterns which they exploit. Nonetheless, many of them are quite performant in cracking.

Password mangling can be directly applied to both targeted and untargeted attacks and so can AI models. However, in terms of targeted vs. untargeted approaches, the cost-effectiveness of various attack vectors and strategies will noticeably differ across password guessing AI models. For instance, most of the AI password cracking models focus on the characteristics of large datasets of compromised passwords, but do not focus on targeted password components such as date of birth, family member names, or other points of personal data. In other words, most existing models cannot be fed OSINT data to generate passwords

for profiled or investigated individuals. Evidently, these models would be more useful in cracking larger portions of compromised password datasets rather than targeted attacks on specific individuals. Fortunately for the public but not for this study, there are only a few publicly available data models which can utilize personal data with AI models to craft better password guesses. These include the Ordered Markov Enumerator "plus" (OMEN+) model [10] and a version of the Generative Pre-Trained Transformer 3 (GPT3) model [16]. Just as with any AI model, these targeted attacks require a lot of personal data. OMEN+ can be trained on a large password data set, and then use a file with "hints" or keywords in them to give the enumerator password hints. The targeted password guessing GPT3 model is trained on 10 thousand users, including usernames, phone numbers, and personal descriptions. The smallest "Ada" model from the OpenAI API models was used. The training data of the ten thousand users was then paired with the users' passwords. This allowed the GPT3 model to learn how to correlate users' personal data with passwords. However, this is a small dataset to train GPT3 on. The training data for GPT3 was crowdsourced from the password recovery website "Hashmob [16]." The research for both of these models showed that untargeted cracking attacks could be drastically improved, and targeted attacks only seemed to effectively work on distributions close to the datasets (overfitting).

2.7 OSINT

Before one can utilize open source intelligence, they must know the scope of the task. Osint is defined as such: "OSINT is publicly produced and publicly available data that can be collected and shared without breaking corporate or public laws or policies, needing a warrant, or participating in what would be commonly considered shady practices [28]." It is vital to be explicit about the definition of OSINT, because other illegal methods could be more effective at obtaining keywords for password cracking. However, those options are not available in legal research or without express permission. Therefore, this research focuses on data obtained during OSINT investigations, simple search queries, or public knowledge about individuals and entities. A report from the Security Insiders states that "34% of OSINT practitioners reported that they had no prior experience with OSINT collection and 85% have received little or no training in OSINT techniques and risks [2]." It is likely, that most cybersecurity practitioners, OSINT, or Threat Intelligence investigators do not completely understand the scope of OSINT when it comes to obtaining somewhat-personal data on a person. In the case of social media, this would mean not impersonating actual people who are related to a target, not breaking terms of service, and especially not hacking individuals to obtain data.

There is not a lot of statistics or data that specifically talk about personal data exposure, how much data the typical person has publicly available, or studies that detail the ease of it. Generally, OSINT (open source intelligence) is straightforward and can be leveraged by the most amateur threat actors. OSINT refers to the procurement of publicly available data without violating laws or policies defined by data owners. OSINT curation on a target can be manual or automated, tool-based or native to the system they are scraping, and one can typically do so with little effort. Most users don't utilize privacy settings on social media to their fullest either. One study in 2018 showed data relevant to OSINT and specifically social media. In the study, 23% of participants shared personal information on social media, 46% used their real names, 45% used their real pictures for their profile, 54% did not attempt to read the privacy statement or terms, and 80% of the participants neither check the social media company practices nor know about the privacy settings of their own profile. [5]. In a separate 2018 study with about 1400 respondents, about half of the respondents never checked their privacy or security settings for apps [19]. There may not exist a lot of data to show exactly how much data is exposed on the internet that is leverageable in targeted attacks. However, we can be sure with this data there is a motive to take advantage of the lack of awareness and minimizing of publicly available personally identifiable information (PII). Overall, it is rather elementary to obtain data about someone that can be used for cybercrime.

3 System Design

3.1 Ideal Scenario for OSINT-Based Attacks

In an ideal scenario, we would have millions of points of data that include 1. Unstructured textual data scraped from public social media accounts and 2. Passwords of those social media accounts or other user-associated accounts. Problem being, that such datasets do not exist and cannot ethically or legally be created. As mentioned earlier, there are crowdsourced workflows that could procure substantial amounts of this data, but it would not be enough data to train naive AI models. If there existed such a dataset, then we would merely train an AI model (take your pick) to generate potential passwords or wordlists based on the unstructured textual data. There are numerous models that could utilize this including but not limited to knowledge graph based implementations, generative adversarial network (GAN) models, recurrent neural networks (RNN), generative pretrained transformers (GPT), and many more. In this ideal scenario, the model could be unsupervised or supervised and learn patterns of personal data compared to the passwords that are used. However, as stated before, such datasets do not exist or are not available to the public. Therefore, a more hypothetical and unorthodox approach must be taken which assumes a lot of variables.

3.2 Targeted Keyword Permutation & Word Mangling Design

Word mangling or what I would call "keyword permutation" is a partly manual and partly automated process. It was stated previously, word mangling requires an understanding of what makes passwords memorable or more likely to be used and also data about the target. This data can be obtained with a variety of OSINT techniques. In this case, I utilize a URL scraper that creates wordlists from the stopwords and data at the URL. This part can

be as manual or as automated as desired. However, more data will require larger wordlist creation during the word mangling process. Therefore, the attacker should choose keywords wisely.

The process which I have designed for this case will pull keywords from a social media page. I could look at the ”word2vec” distance or semantic similarities with something like ”spacy” to see if that social media textual data is similar to data that I will manually put down that pertains to my real password. However, my knowledge of semantic models and word embeddings is limited, so no semantic analysis will be done for either password mangling or the subsequent AI model tests. The keywords from the social media page will be used to generate wordlists by using selected word mangling tools from GitHub. Steps will then be duplicated using manually inputted keywords that I provide to the mangler tools. This will allow for a comparison between completely automated and partly-manual word mangling strategies.

3.3 Targeted AI-Based Wordlist Generation Design

Targeted implementations of password-applicable AI models are far and few. In fact, utilization of the available models is often limited in wordlist generation speed and volume. Therefore, a targeted approach for AI-based wordlist generation will only utilize a small number of password guesses. Guesses will be on the scale of thousands rather than millions, billions, or trillions.

3.4 Hardware Utilization & Constraints

This project focuses on what is readily available to the average attacker and not advanced state nation-level threats. Most GitHub implementations do not compute using the GPU (graphical processing unit) which can lend to great cracking speeds, and some of the code is in programming languages that are not optimized for rapid operations (Python). Therefore, speed will only be recorded if it lends to the cost effectiveness of the implementation. Runtime for particular algorithms and wordlist generation will be recorded when the datasets or wordlists are large and the computation speed is noticeably affected.

3.5 Dataset Use

Plaintext password datasets are only used for the OMEN+ model. Training the other models manually was not within the scope of this project.

OMEN+ was trained using numerous available datasets:

1) rockyou.txt
2) 3 explicitly-named website’s breach data (I won’t name these in an academic paper)
3) hak5.txt
4) honeynet2.txt
5) myspace.txt
6) NordVPN.txt
7) 10-million-password-list-top-1000000.txt
8) 100k-most-used-passwords-NCSC.txt
9) probable-v2-vpa-top4800.txt
10) Lizard-Squad.txt
11) phpbb.txt
12) tuscl.txt
13) Top304Thousand-probable-v2.txt
14) Top204Thousand-probable-v2.txt
15) Appearances-Top304Thousand-probable-v2.txt

4 Methods & Analysis

All models and implementations will utilize one or both of two potential inputs: a custom wordlist composed of data that can be obtained through social media or compromised plaintext password datasets. The model or algorithm will then be used to generate wordlists. Wordlists may have varying lengths. However, they will all be given the same amount of effort as this experiment focuses on what the average attacker can do a benefit from. Lastly, the generated wordlists will be compared to several potential passwords chosen by the tester. Several string similarity functions will be used to quantitatively determine the performance of each wordlist in guessing or approximately guessing the actual password. Due to lack of instances for social media OSINT data compared to actual passwords and absence of such a dataset, a combined histogram for each similarity metric will be produced to show the similarity or distance distributions of the guess vs. the actual password. This will represent the effectiveness of each algorithm for targeted password guessing attempts.

4.1 General Architecture

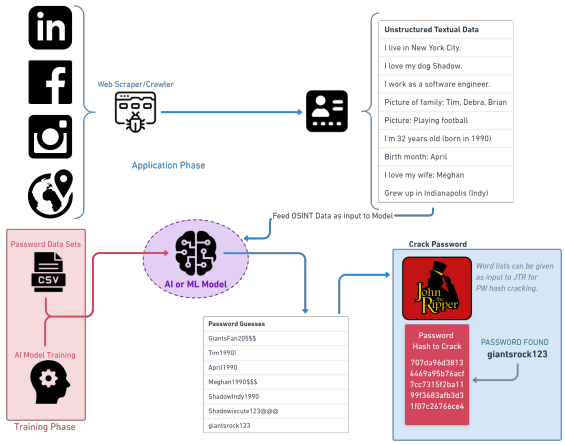

Fig. 1. Example architecture

Figure 1 exemplifies a potential architecture that could be used with AI or ML-based models. In the top-left data is scraped from various social media websites, public websites, location data, Wikipedia crawls, and any other form of OSINT. This data is then prepared, cleaned, and curated into unstructured textual data. This data can bed fed into various models where stopwords may be removed or other forms of data transformation may take place. Moreover, keyword expansion could also take place where the extracted keywords or sentences are expanded into other potentially useful points of data. AI and ML models are then trained on large datasets of compromised and realistic passwords.

These datasets can be distributions or they can be deduplicated. This depends on the model being used. After models are trained on the password data, they are also fed the data obtained in the OSINT and application phase. The model is then used to generate potential wordlists. These wordlists can be used in a password cracking tool which utilizes the GPU and is optimized for hashing and comparing hashes.

4.2 Keyword List Creation

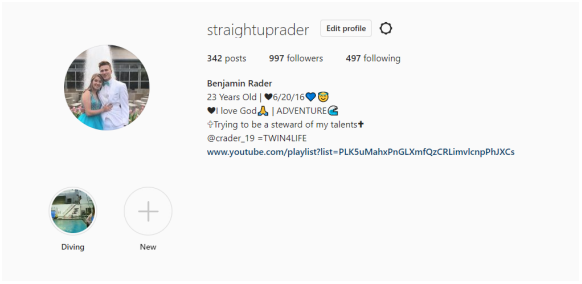

Fig. 2. My old Instagram page and biography.

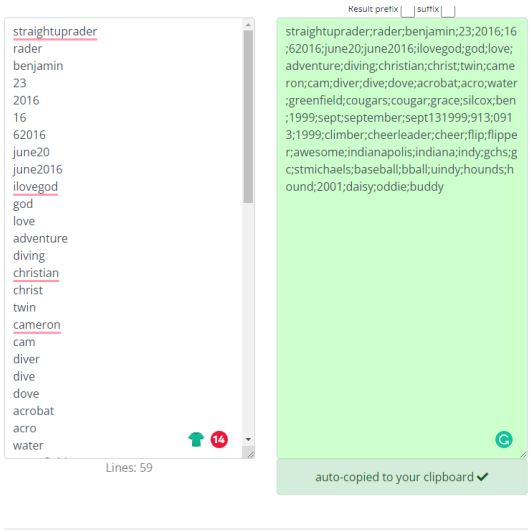

Fig. 3. Used a tool online to convert txt file to ";" delimited version for use with other tools.

As seen in figure 2, my account does not have very much information. However, it does have points of information that I am likely to value the most including family information (brother), girlfriend's name and anniversary, religion, and other potential keyword from posts that are not shown. This data is aggregated into a ".txt" file to be used with other programs. I even convert this later on into a tab delimited version as shown in Figure 3.

4.3 Dataset Aggregation

Plaintext password datasets will only be used for the OMEN+ ML model due to it being the only available targeted model that can utilize password datasets.

4.3.1 Dataset Sources

Datasets are obtained from multiple sources:

1) https://haveibeenpwned.com/Passwords
2) https://figshare.com/articles/dataset/ Yahoo_Password_Frequency_Corpus/2057937
3) https://hashes.org/leaks.php
4) https://github.com/rndinfosecguy/pastePasswordLists
5) https://github.com/berzerk0/Probable-Wordlists/tree/ master/Real-Passwords
6) https://github.com/kennyn510/wpa2-wordlists/tree/ master/Wordlists
7) https://haveibeenpwned.com/Passwords
8) https://github.com/danielmiessler/SecLists/tree/ master/Passwords

4.3.2 Aggregating Text Files

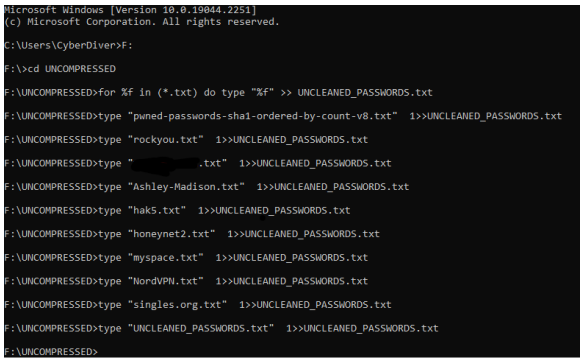

Fig. 4. Windows CMD script to combine text files.

I used multiple tools to try combing the text files (Windows OS):

- Dymerge
- Text_Merge.exe
- A basic Windows CMD script.

The HIBP (HaveIBeenPwned) dataset was about 50 gigabytes. Combining very large text files was not straightforward. Turns out, the CMD script was the most reliable (Fig. 4).

4.3.3 Accidentally Used Hashes For Most of the Data

Turns out, the HIBP dataset consisted completely of about 50 GB of SHA-1 hashes. Although these can be cracked easier than other hashes, this was not what I was looking for. This was only found out after using the tool "Klogg" (https://github.com/variar/klogg) to open the rather large text files and seeing nothing but hashes.

4.3.4 Dataset Cleaning and Preparation - Deduplication & ASCII Encoding

Most of the plaintext password dataset preparation involves getting rid of duplicates. Although OMEN could maybe work with better with a realistic distribution, it is not worth trying to do that considering OMEN+ works better with certain use cases or hashses such as bcrypt, PBKDF2, scrypt, or Argon2 [15]. Therefore, duplicates can be removed. Tools from the Unified List Manager for Windows are used (https://unifiedlm.com/Home). Specifically,

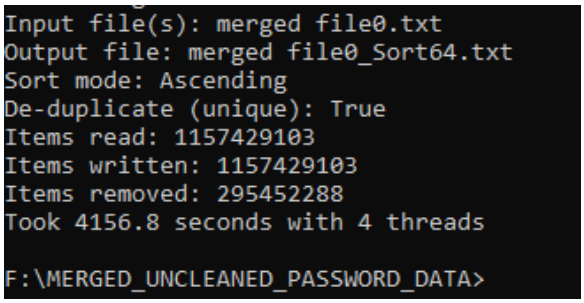

Fig. 5. Unified List Manager - sort 64 low memory to get rid of duplicate passwords from the combined dataset of plaintext passwords.

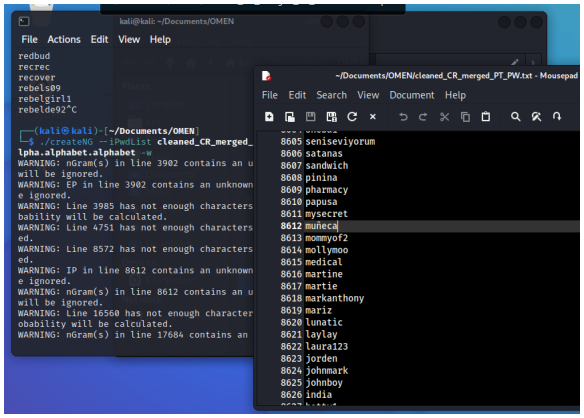

Fig. 6. Example of line in password dataset that breaks most wordlist generators.

"sort64lm" is used to go through the aggregated file, get rid of duplicates, and sort the file alphabetically. Figure 6 shows an error that occurs with OMEN+. In order to run any password datasets through OMEN+ only ASCII characters can be used. This could be problematic especially on realistic datasets that are much larger. This goes to show that tools are not good enough yet for realistic use cases with targeted password guessing. However, a simple fix is to just remove the character using the **"enconv"** command in Linux. An error about not having enough characters for lines in the text file also occurred. There were lines ranging from one chracter to eight characters long. This error is quite helpful in creating wordlists, because using passwords that are too short will be useless to realistic password cracking. After all, passwords shorter than eight characters are practically cracked instantly. Linux also has commands for deleting these lines.

4.4 Wordlist Generation

4.4.1 Training AI & ML Models - only OMEN+

Creating a wordlist in OMEN+ is quite straightforward as outlined in their GitHub documentation. A few commands are run: a command to generate an alphabet from the dataset file, a command to create the "enumerator" model, and then a command to generate a wordlist of some user-defined length. Training could not be done with the GP3 model, because the access costed money or more effort than was necessary for this project.

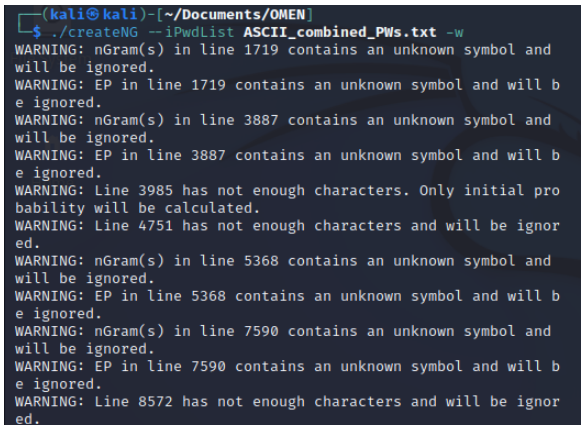

Fig. 7. Kali Linux screenshot of OMEN creating an enumerator.

4.4.2 Scraping Data from GPT3 Password Generator

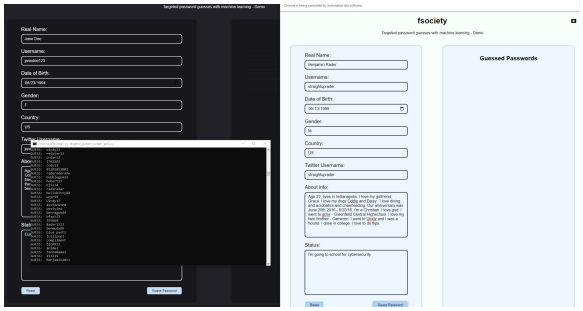

Fig. 8. Scraping Data from the Targeted Password Guessing Heroku App found in the Automated Targeted Password Guessing GitHub page - https://github.com/ACM-Research/targeted-password-guesses

Learning how to create my own GPT3 model through OpenAI was not feasible in one semester. Therefore, I utilize a live Heroku App that was linked from the GitHub page for "targeted-password-guesses." Heroku apps are sort of like little servers that can be run at a very low price while still having good enough computation power for little API-based projects. In this case, the Heroku app sends personally-related data (username, social media bio, name, birthday) to the OpenAI API and utilizes a trained GPT3 model to generate password guesses based on the data inputted in the Heroku app. This data is sent back to the Heroku app to be displayed. In this case, I wrote a 200-line Python script using Selenium to scrape the data and automatically keep sending requests. These requests were stored line-by-line in a text file [16].

4.4.3 A failed AI-based attempt - PassHemorrhage

After running this tool and sending a few issues into GitHub, it was apparent that this project was discontinued. This project utilized a GAN-based implementation that fed any type of personal data into the mix. This was the only implementation of a targeted model that was GAN-based. Sadly, this did not work out as there were a lot of errors in the actual code. It even used an outdated version of Python [23].

4.4.4 "Naive Permutation" - Cordialie

Cordialie is a naive attempt at creating permutations of inputted keywords. It does not actually create permutations though, despite the claims on the GitHub page. However, cordiale does mangle the inputted keywords to look like potential clever versions of the keywords. It also may append commond suffixes or prefixes to the keywords, transform certain characters to "l33t" versions of the characters, and it changes capitalization [14].

4.4.5 "Manual/Permutative/Mangler" - Mentalist

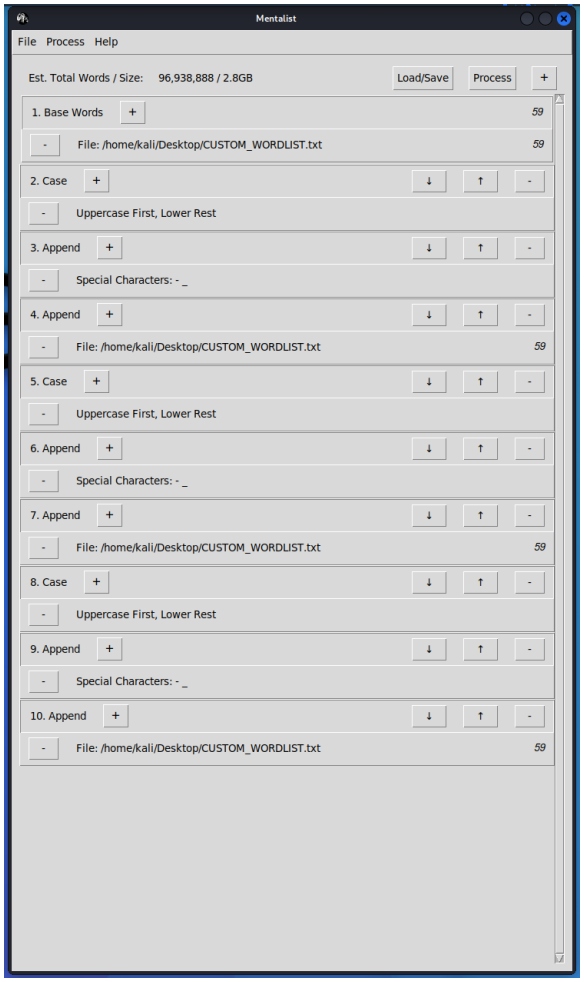

Fig. 9. The Mentalist GUI running in a Kali Linux virtual machine.

Mentalist is the epitome of keyword permutations and mangling flexibility. The keywords from the OSINT phase are fed into the program as "Base Words." Then, the user can define how to create permutations between "case", substitution, append, or prepend. Between these actions, the user can generate limitless types of patterns or passwords. In this case, the keywords from OSINT are combined in every possible order, the first letter of each keyword is made upper and lowercase, and characters are appended to the end in some permutations. Just to be clear, each of these actions do or do not take place in some permutations so that every combinations of keywords and actions in Mentalist is taken. This results in text files that are on the scale of gigabytes [26].

4.5 Evaluation using String Similarity Metrics

Fig. 10. Custom string similarity code running on Kali Linux VM.

A python program was created to compare each password guess from the generated wordlists to the hypothetical password. The hypothetical password uses the same structure as an old and known password that was being used with the account. In this case, we are testing each password guess with the password - "Ilovegrace2016." Comparison is done in python using the "python-string-similarity" library [22]. Two string similarity algorithms are used including Damerau-Levenshtein and the Jaccard index. Damerau-Levenshtein is a string-distance algorithm where the distance is "is the minimum number of operations needed to transform one string into the other, where an operation is defined as an insertion, deletion, or substitution of a single character, or a transposition of two adjacent characters." Jaccard index is a similarity computation that is formally defined as the size of the intersection divided by the size of the union of the sample sets.

5 Findings

5.1 Output of String Similarity Stats

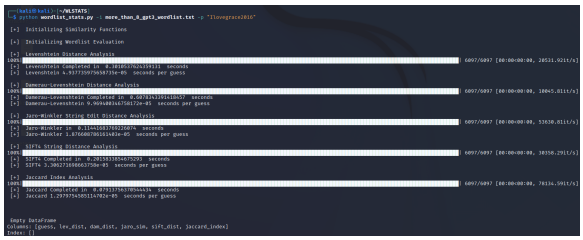

Fig. 11. Wordlist_stats.py (created by me) running in Kali Linux.

5.1.1 Runtime Over GPT3 (Ada) Wordlist

- Guesses: 6097
- Damerau-Levenshtein runtime: 0.007 seconds
- Jaccard runtime: 0.08 seconds

5.1.2 Runtime Over OMEN+ Wordlist

- Guesses: 503,593
- Damerau-Levenshtein runtime: 40 seconds
- Jaccard runtime: 6 seconds

5.1.3 Runtime Over Cordialie Wordlist

- Guesses: 4,897,170
- Damerau-Levenshtein runtime: 10 minutes 11 seconds
- Jaccard runtime: 1 minutes 36 seconds

5.1.4 Runtime Over Mentalist Wordlist

- Guesses: 2,637,754
- Damerau-Levenshtein runtime: 5 minutes 35 seconds
- Jaccard runtime: 37 seconds

5.1.5 GPT3 Results

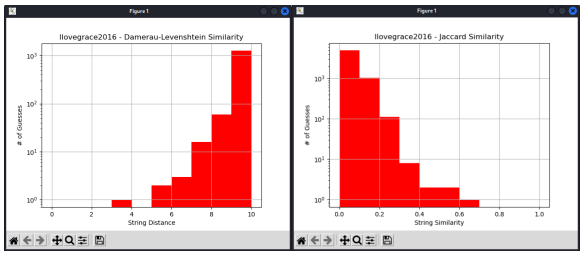

Fig. 12. Similarity & Distance Distributions - GPT3

5.1.6 OMEN+ Results

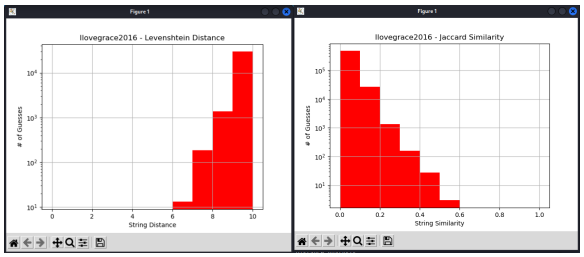

Fig. 13. Similarity & Distance Distributions - OMEN+

5.1.7 Cordialie Results

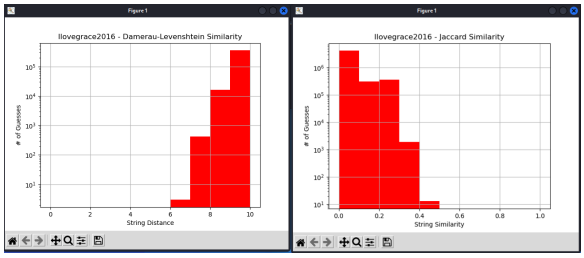

Fig. 14. Similarity & Distance Distributions - Cordialie

5.1.8 Mentalist Results

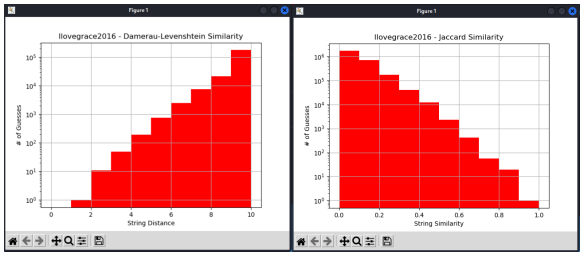

Fig. 15. Similarity & Distance Distributions - Mentalist

5.2 Final Findings

5.2.1 Notes on Similarity and Distance Metrics

It is important to note that the distributions are displayed using a logarithmic scale instead of a linear one. This is because the amount of guesses that are similar to the actual password will be small compared to the number of guesses that fall into more dissimilar scores in the distribution. Additionally, for the Damerau-Levenshtein distance metric, many results are not shown on the graph because the distance is more than 10. This explains why the total amount of guesses on the Damerau-Levenshtein distribution will not add up to the total number of guesses. On the contrary, the Jaccard chart guesses should add up to the total number of generated guesses.

5.2.2 AI vs. Manual

There is no question that word mangling and permutation are the winner for targeted password cracking. The problem with both GPT3 and OMEN+ seems to be that the tools are more affected by their training dataset than the OSINT data which is given during wordlist generation. However, Cordialie performed the worst despite it supposedly being able to ”print hundreds to millions of possible combinations for a specific target easily [14].” When diving into the Cordialie wordlist, one will find that it rarely generates smart combinations of the keywords, but rather appends, preprends, and transforms keywords at seemingly at random. This may be good for keyword expansion and creating new keywords for permutation. However, it does not perform well and results are dissimilar to the actual password. For OMEN+ (machine learning model), the results are better than Cordialie, but still far from ideal for password cracking. In fact, they even state that OMEN is better for cracking large datasets of certain types of hashes, and they recommend using dictionaries and mangling rules for other targeted attacks or attacks on ”very fast hashes [15].” It is important to note the shape of the distribution is important. The charts are displayed in a logarithmic scale and cases where the distribution has a ”straight” slant means that combination and permutation are more likely and that human-like or smart generation is probably not being done. This would explain why the GPT3 distribution looks the way that it does. OMEN+ does not perform terribly, because it had results that were only 6 distance away from the actual password. The string similarity also didn’t have terrible metrics with 4 or 5 results with 0.6 similarity. OMEN+ is likely to perform better on larger datasets, and this conclusion comes from analysis of the generated words, the guess distribution, and writings from its creator.

The performance of the GPT3 model does not seem great on the surface. However, one could argue that is performs beautifully. A straight or gradual slant for the results in the string similarity metric will mean that the password appears somewhere in the guess or partly in the guess. This would especially be true for password manglers where the keywords can be combined in a way to make an exact match for the true password. In other words, word mangling and combination are naive ways of finding the

password. Due to the nature of the similarity algorithm and permutative methods, a straight slant means that the user who is conducting OSINT must have obtained the exact or almost-exact keywords that make up the target's real password, and they must have also had knowledge of the target's password structure around those keywords. GPT3 is promising because the curve is not a gradual slope, but rather more of an "exponential" one. This means that it is not combining keyword in some sort of dumb approach, but rather taking the OSINT data combined with knowledge about passwords and creating wordlists that are innovative. Nonetheless, it still currently performs terribly for targeted password guessing attacks. Not to mention, the rate of generation for GPT3-based implementations is extremely slow. Manual and ML methods generated wordlists 10,000 times faster than GPT3 because it uses an API and must make requests to an external server. Despite the small number of requests, the similarity still yielded results that are promising. If a better model could be trained or utilize a different approach, then smart AI-based text generation such as with GPT3 could be very good realistic targeted attacks.

It's obvious when looking at the results, that Mentalist performed the best. Mentalist is a modular approach to word mangling and permutation of wordlists or dictionaries (same thing). There are results that also perfectly match the real password. The shape of the distribution is a slant precisely because of what was previously explained. Mentalist was used to do combinations of keywords while appending, prepending, and transforming the keywords and permutations based on what the user configured. The distribution looks good, but the straight slant comes as a result of the combinatory nature of Mentalist. For example, combing "I","love", and "2016" would be close to the real password, and the result woulf be a high metric for similarity, but this only happens because the user explicitly defined the correct keywords. Mentalist definitely outperforms every other approach in every way, but that fact only stands for the time being and until better text generation approaches are utilized with AI for wordlist generation [26].

5.2.3 Password Choice Caveats, Bias, and Explanations of Results

There is quite a lot of problems with the approach. This stems from the limits of finding datasets where a password and OSINT data are paired together. Such datasets do no exist. Therefore, an approach was taken which assumed that the "guesser" had the OSINT data of the user, and they also knew the general pattern or structure of their password. Essentially, the guesser knew the exact keywords and the relative structure of the password, but they did not know the order of the keywords within that structure. To say that this has an effect on the results is an understatement. In terms of keywords choice from OSINT, there are words that could be considered stopwords, but are actually a part of the password such as the word "love." In this case, "love" was included with the keywords because the guesser knew that the word "love" could be used between nouns like with a girlfriend's name or a dog's name. These are lucky assumptions and it is not likely these assumptions would be made in a realistic situation. Moreover, there are numerous generated guesses that make no sense in terms of human-understood phrases or password structures. For example, the name "mike" might be used and Mentalist generates the guess "mike-mike-mike-mike-mike." This would never appear as someone's password, but something like "mike-likespancakes" may appear. This is vindicative of where a model that understands passwords like humans do would come in handy. In reality, the problem of targeted password guessing is quite difficult, and technology has simply not reached the point where it is viable.

5.3 Relevant Threats & Attack Assumptions

5.3.1 Nation States & High Profile Attacks

Realistically, these attacks are likely not being utilized against the average citizen. There are numerous other attack vectors that are "low-hanging" or easier to leverage than a complicated guessing attack. However, a targeted attack such as the ones outlined and the ones hypothesized could theoretically be a current reality for state-sponsored and state-supported attacks. For example, their may be a target for some country where everything around the target is relatively secure except for an obtained password hash. The APT (advanced persistent threat) may utilize smart targeted password attacks to figure out the password. Such attacks are likely far and few as there is no evidence of such an attack which is being talked about. Such attacks are probably only conducted on the domestic and low-level hacker front for fun or for proof-of-concept. Hardware dilemmas with such attacks also support this assumption because wordlists can become very large and require extensive computational power to utilize.

5.3.2 Risk Model & Recommendations

In order for this attack to take place, the attacker needs to know information about the target (anything memorable) and what the structure of their password likely is. Additionally, certain verbs and stopwords may be included between keywords in passphrases. Due to the complexity of language, the sheer number of stopwords in the English dictionary, and the limitations of proposed GPT3 models it can be concluded that targeted password attacks are a low risk to the average individual.

However, if the user does want to be safe from other password attacks, then I would give 3 recommendations. Firstly, use a password manager to store and generate passwords for all of your services. Make sure to have a long passphrase that does not utilize keywords that are connected to publicly available information about you. Use inside jokes or secrets about you that are easy to remember. Secondly, use multi-factor authentication for accounts with high-risk data or functionality. Thirdly, check your security and privacy settings for applications that you use.

6 Conclusion

Targeted password cracking that utilizes OSINT data about an individual has not matured enough to be viable for the

average or even advanced attackers. Only state-sponsored attacks could utilize such attacks in a cost-effective manner. AI and machine learning models are great for cracking large datasets of passwords, but they are not sophisticated enough to understand language as humans do and create realistic password guesses. Therefore, AI models such as ”GPT-3” (trained on 10,000 examples) and ”OMEN+” perform poorly in targeted attacks. This is true even for the case where the attacker knows the exact keywords and the password’s structure. Mangling and explicitly-defined methods such as with ”Cordialie” and ”Mentalist” are very effective for targeted dictionary attacks or password guessing. However, this is only true if the attacker knows the keywords that make up the target’s password and the structure of the password. These findings are proven to be true through string similarity comparisons between the guesses and the actual password, and work is also done to understand the significance of the results.

7 Related Topics & Hypotheses

If I were to create a targeting password guessing framework then I would need several components. The method would need to have the ability to learn password patterns and structures based on large datasets. Secondly, the approach must involve the use of ontology (problem domain) and keyword expansion by using various ontology of knowledge graph models. For example, if given the word ”Indianapolis”, then the system would generate other keywords related to Indianapolis such as ”IUPUI”, ”Jaguars”, ”Pacers”, ”White River”, and other words that relate to Indianapolis. These keywords would then be fed into a smart AI-based model that could understand knowledge graph models and password patterns. This model would generate passwords that are very close to the real password by generating realistic, memorable, and ”smart” guesses. Such methods have been outlined using Wikipedia data, but the methods still need some refining [12].

References

[1] *Why people (don’t) use password managers effectively*, 2019.

[2] 2020 cyber threat intelligence report, 2020.

[3] *An implementation and evaluation of PDF password cracking using john the ripper and crunch*, 2021.

[4] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin Vandersloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. Imperfect forward secrecy: How diffie-hellman fails in practice. *Communications of the ACM*, 62, 10 2015.

[5] Shaukat Ali, Naveed Islam, Azhar Rauf, Ikram Ud Din, Mohsen Guizani, and Joel . Privacy and security issues in online social networks. *Future Internet*, 10(12), 2018.

[6] Vijayalakshmi Atluri, Di Pietro, Christian D Jensen, and Weizhi Meng, editors. *The Revenge of Password Crackers: Automated Training of Password Cracking Tools*. Computer Security – ESORICS 2022, Springer Nature Switzerland, 2022.

[7] Avast. 83

[8] Gabriel Bassett, C. David Hylender, Philippe Langlois, Alex Pinto, and Suzanne Widup. Dbir data breach investigations report, 2022.

[9] Daniel Brecht. Password security: Complexity vs. length [updated 2019], 01 2021.

[10] Claude Castelluccia, Chaabane Abdelberi, Markus Dürmuth, and Daniele Perito. When privacy meets security: Leveraging personal information for password cracking. 04 2013.

[11] Federal Trade Commission. Use two-factor authentication to protect your accounts, 09 2022.

[12] Sein Coray. *Óinn: A framework for large-scale wordlist analysis and struc-ture-based password guessing*. PhD thesis, 2019.

[13] Casey Crane. What is a hash function in cryptography? a beginner’s guide, 01 2021.

[14] Victor Viccenzo Franchesco. cordialie, 11 2022.

[15] Maximilian Golla. Omen: Ordered markov enumerator, 11 2022.

[16] Roman Hauksson and Brad Johnson. Automating targeted password guessing, 03 2022.

[17] Aikaterini Kanta, Iwen Coisel, and Mark Scanlon. A survey exploring open source intelligence for smarter password cracking. *Forensic Science International: Digital Investigation*, 35:301075, 2020.

[18] Branko Krstic. Impressive password statistics to know in 2022, 04 2022.

[19] Masaaki Kurosu, editor. *Why users ignore privacy policies – a survey and intention model for explaining user privacy behavior*. Springer International Publishing, 2018.

[20] S Li, Z Wang, R Zhang, C Wu, and H Luo. Mangling rules generation with densitybased clustering for password guessing. *IEEE Transactions on Dependable and Secure Computing*, pages 1–13, 2022.

[21] Ponemon Institute LLC. The 2020 state of password and authentication security behaviors report, 2020.

[22] luozhouyang. python-string-similarity, 12 2022.

[23] Mandex. Hollow667/passhemorrhage, 11 2022.

[24] Phil Muncaster. Password reuse at 6003 2021.

[25] OWASP. Password storage · owasp cheat sheet series, 2022.

[26] Henry Prince. Mentalist, 12 2022.

[27] Fintechnews Singapore. Passwords still the most-used authentication method which is proving to be costly, 10 2022.

[28] Media Sonar. Is osint legal?, 2020.

[29] Security.org Team. Password manager and vault 2021 annual report: Usage, awareness, and market size, 12 2022.

[30] Aliza Vigderman. Password manager and vault 2022 annual report: Usage, awareness, and market size, 12 2022.

[31] Alex Weinert. Your pa$$word doesn’t matter, 07 2019.

[32] Chris Woodford. How do locks and padlocks work?, 07 2008.

[33] Naomi Woods and Mikko Siponen. Too many passwords? how understanding our memory can increase password memorability. *International Journal of Human-Computer Studies*, 111:36–48, 2018.

[34] Naomi Woods and Mikko Siponen. Improving password memorability, while not inconveniencing the user. *International Journal of HumanComputer Studies*, 128:61–71, 2019.

[35] M Yıldırım and I Mackie. Encouraging users to improve password security and memorability. *International Journal of Information Security*, 18:741–759, 2019.

[36] Huanguo Zhang, Bo Zhao, and Fei Yan, editors. *Password Guessing Based on Semantic Analysis and Neural Networks*. Trusted Computing and Information Security, Springer Singapore, 2019.