Measuring Real-World Accuracies and Biases in Modeling Password Guessability

Blase Ur, Sean M. Segreti, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor,
Saranga Komanduri, Darya Kurilova, Michelle L. Mazurek[†], William Melicher, Richard Shay
Carnegie Mellon University, [†]University of Maryland

Abstract

Parameterized password guessability—how many guesses a particular cracking algorithm with particular training data would take to guess a password—has become a common metric of password security. Unlike statistical metrics, it aims to model real-world attackers and to provide per-password strength estimates. We investigate how cracking approaches often used by researchers compare to real-world cracking by professionals, as well as how the choice of approach biases research conclusions.

We find that semi-automated cracking by professionals outperforms popular fully automated approaches, but can be approximated by combining multiple such approaches. These approaches are only effective, however, with careful configuration and tuning; in commonly used default configurations, they underestimate the real-world guessability of passwords. We find that analyses of large password sets are often robust to the algorithm used for guessing as long as it is configured effectively. However, cracking algorithms differ systematically in their effectiveness guessing passwords with certain common features (e.g., character substitutions). This has important implications for analyzing the security of specific password characteristics or of individual passwords (e.g., in a password meter or security audit). Our results highlight the danger of relying only on a single cracking algorithm as a measure of password strength and constitute the first scientific evidence that automated guessing can often approximate guessing by professionals.

1 Introduction

Despite decades of research into alternative authentication schemes, text passwords have comparative advantages—familiarity, ease of implementation, nothing for users to carry—that make a world without text passwords unlikely in the near future [5]. Two-factor authentication, single-sign-on systems, password managers, and biometrics promise to obviate remembering a distinct password for each online account, but passwords will not disappear entirely.

Text passwords have been compromised with alarming regularity through both online and offline attacks. While online attacks are mitigated through rate-limiting password-entry attempts, faulty rate limiting contributed to the iCloud photo leak [39]. In offline attacks, including recent ones on LinkedIn [7], eHarmony [62], Gawker [2], and Adobe [48], an attacker steals a database of (usually) hashed passwords and tries to recover passwords through offline guessing. Because password reuse is common [14], recovered passwords can often be used to access accounts on other systems.

A key aspect of improving password security is making passwords more computationally expensive to guess during offline attacks. Cracking tools like the GPU-based oclHashcat [57] and distributed cracking botnets [13, 17] enable attackers to make 10^{14} guesses in hours if passwords are hashed using fast hash functions like MD5 or NTLM. These advances are offset by the development of hash functions like bcrypt [52] and scrypt [47], which make attacks more difficult by requiring many iterations or consuming lots of memory.

Unfortunately, users often create predictable passwords [7, 29], which attackers can guess quickly even if the passwords are protected by a computationally expensive hash function. In some cases, predictable passwords are a rational coping strategy [54, 60]; in other cases, users are simply unsure whether a password is secure [66]. System administrators encourage strong passwords through password-composition policies and password-strength meters. The design and effectiveness of such mechanisms hinges on robust metrics to measure how difficult passwords are to guess.

In recent years, traditional entropy metrics have fallen out of favor because they do not reflect how easily a password can be cracked in practice [3, 31, 69]. It has instead become common to measure password strength by running or simulating a particular cracking algorithm, parameterized by a set of training data [4, 31, 69]. This approach has two main advantages. First, it calculates the guessability of each password individually, enabling data-driven strength estimates during password creation [10, 33]. Second, it estimates real-world security against existing, rather than idealized, adversarial techniques. A disadvantage of this approach is that the (simulated) cracking algorithm may not be configured or trained as effectively as by a real attacker, leading to inaccurate estimates of password strength.

This paper reports on the first study of how various cracking approaches used by researchers compare to real-world cracking by professionals, as well as how the choice of approach biases research conclusions. We contracted a computer security firm specializing in password recovery to crack a set of passwords chosen for their diversity in password-composition policies. We then computed the guessability of these passwords using four popular approaches. We tested many configurations of two well-known password-cracking toolkits: John the Ripper [49] and oclHashcat [57]. We also tested two approaches popular in academia: Weir et al.'s probabilistic context-free grammar (PCFG) [70] and Ma et al.'s Markov models [40].

Unsurprisingly, a professional attacker updating his strategy dynamically during cracking outperformed fully automated, "fire-and-forget" approaches (henceforth simply referred to as *automated*), yet often only once billions or trillions of guesses had been made. We found that relying on a single automated approach to calculate guessability underestimates a password's vulnerability to an experienced attacker, but using the earliest each password is guessed by any automated approach provides a realistic and conservative approximation.

We found that each approach was highly sensitive to its configuration. Using more sophisticated configurations than those traditionally used in academic research, our comparative analysis produced far more nuanced results than prior work. These prior studies found that Markov models substantially outperform the PCFG approach [18, 40], which in turn substantially outperforms tools like John the Ripper [16, 69, 72]. We found that while Markov was marginally more successful at first, it was eventually surpassed by PCFG for passwords created under typical requirements. Furthermore, the most effective configurations of John the Ripper and Hashcat were frequently comparable to, and sometimes even more effective than, the probabilistic approaches.

Both the differences across algorithms and the sensitivity to configuration choices are particularly notable because most researchers use only a single approach as a security metric [10, 12, 19, 42, 56, 65, 69]. In addition, many researchers use adversarial cracking tools in their default configuration [11, 14, 15, 20, 21, 28, 34, 71]. Such a decision is understandable since each algorithm is very resource- and time-intensive to configure and run. This raises the question of whether considering only a single approach biases research studies and security analyses. For instance, would substituting a different cracking algorithm change the conclusions of a study?

We investigate these concerns and find that for comparative analyses of large password sets (e.g., the effect of password-composition policies on guessability), choosing one cracking algorithm can reasonably be expected to yield similar results as choosing another.

However, more fine-grained analyses—e.g., examining what characteristics make a password easy to guess—prove very sensitive to the algorithm used. We find that per-password guessability results often vary by orders of magnitude, even when two approaches are similarly effective against large password sets as a whole. This has particular significance for efforts to help system administrators ban weak passwords or provide customized guidance during password creation [10, 33]. To facilitate the analysis of password guessability across many password-cracking approaches and to further systematize passwords research, we introduce a Password Guessability Service [9] for researchers.

In summary, this paper makes the following main contributions: We show that while running a single cracking algorithm or tool relatively out-of-the-box produces only a poor estimate of password guessability, using multiple well-configured algorithms or tools in parallel can approximate passwords' vulnerability to an expert, real-world attacker. Furthermore, while comparative analyses of large password sets may be able to rely on a single cracking approach, any analysis of the strength of individual passwords (e.g., a tool to reject weak ones) or the security impact of particular characteristics (e.g., the use of digits, multiple character classes, or character substitutions) must consider many approaches in parallel.

2 Related Work

In this section, we discuss commonly used metrics of password strength (Section 2.1) and describe popular categories of password-cracking attacks (Section 2.2).

2.1 Password Security Metrics

While estimated entropy was once a leading password strength metric [8], it does not reflect what portion of a set can be cracked easily [3, 31, 69]. Two main classes of metrics have emerged in its place: statistical metrics and parameterized metrics. Both classes focus on *guess-*

ability, the number of guesses needed by an adversary to guess a given password or a fraction of a set.

Statistical metrics are particularly valuable for examining password sets as a whole. For example, Bonneau introduced partial guessing metrics [3] for estimating the number of guesses required for an idealized attacker, who can perfectly order guesses, to guess a given fraction of a set. Since password distributions are heavy-tailed, very large samples are required to determine a set's guessability accurately.

Parameterized metrics instead investigate guessability under a cracking algorithm and training data [4, 31, 69]. These metrics thus model an adversary using existing tools, rather than an idealized attack, though the metric is only as good as the chosen algorithm and training data. Parameterized metrics can also be used to compare password sets without fully running the algorithm [40].

In contrast to statistical metrics, parameterized metrics have two important properties. First, they estimate the guessability of each password individually. Estimating guessability per-password is important for security audits (e.g., identifying weak passwords) and to provide feedback to a user about a password she has created. This latter promises to become more widespread as proactive feedback tools move from length-and-character-class heuristics [15] to data-driven feedback [10, 33]. Second, parameterized metrics aim to estimate security against real-world, rather than idealized, attacks. Researchers previously assumed automated techniques approximate real-world attackers [31, 69]; we are the first to test this assumption against attacks by professionals.

Parameterized metrics have been used to measure password strength in a number of previous studies [10, 14, 16, 20, 21, 31, 34, 40, 42, 53, 56, 65, 68, 69, 72]. While there are many different methods for cracking passwords, as we detail in Section 2.2, time and resource constraints lead many researchers to run only a single algorithm per study. However, it remains an open question whether this strategy accurately models real-world attackers, or whether choosing a different algorithm would change a study's results. We address this issue.

Throughout the paper, we refer to the *guess number* of a password, or how many guesses a particular parameterized algorithm took to arrive at that password. Because the algorithm must be run or simulated, there is necessarily a *guess cutoff*, or maximum guess after which remaining passwords are denoted "not guessed."

2.2 Types of Guessing Attacks

Researchers have long investigated how to guess passwords. A handful of studies [12, 16, 53] have compared the aggregate results of running different cracking approaches. Other studies have compared results of running different cracking approaches based on guess numbers [11, 18, 40]. We are the first to examine in detail the magnitude and causes of differences in these approaches' effectiveness at guessing specific passwords; we also compare approaches from academia and adversarial tools to a professional attacker. In this section, we highlight four major types of attacks.

Brute-force and mask attacks Brute-force attacks are conceptually the simplest. They are also inefficient and therefore used in practice only when targeting very short or randomly generated, system-assigned passwords.

Mask attacks are directed brute-force attacks in which password character-class structures, such as "seven lowercase letters followed by one digit" are exhausted in an attacker-defined order [58]. While this strategy may make many guesses without success, mask attacks can be effective for short passwords, as many users craft passwords matching popular structures [37, 63]. Real-world attackers also turn to mask attacks after more efficient methods exhaust their guesses. We evaluated mask attacks in our initial tests. Unsurprisingly, we found them significantly less efficient than other attacks we analyzed.

Probabilistic context-free grammar In 2009, Weir et al. proposed using a probabilistic context-free grammar (PCFG) with a large training set of passwords from major password breaches [67] to model passwords and generate guesses [70]. They use training data to create a context-free grammar in which non-terminals represent contiguous strings of a single character class. From the passwords observed in its training data, PCFG assigns probabilities to both the structure of a password (e.g., *monkey99* has the structure {*six letters*}{*two digits*}) and the component strings (e.g., "99" will be added to the list of two-digit strings it has seen). A number of research studies [11, 16, 19, 31, 40, 42, 56, 65, 69, 72] have used PCFG or a close variant to compute guessability.

Kelley et al. proposed other improvements to Weir et al.'s PCFG algorithm, like treating uppercase and lowercase letters separately and training with structures and component strings from separate sources [31]. Because they found these modifications improved guessing effectiveness, we incorporate their improvements in our tests. In addition, multiple groups of researchers have proposed using grammatical structures and semantic tokens as PCFG non-terminals [53, 68]. More recently, Komanduri proposed a series of PCFG improvements, including supporting hybrid structures and assigning probabilities to unseen terminals [32]. We incorporate his insights, which he found improves guessing efficiency.

Markov models Narayanan and Shmatikov first proposed using a Markov model of letters in natural language with finite automata representing password structures [45]. Castelluccia et al. used a similar algorithm for password meters [10]. John the Ripper and Hashcat offer simple Markov modes in their cracking toolkits as well.

Recently, Duermuth et al. [18] and Ma et al. [40] independently evaluated many variations of Markov models and types of smoothing in cracking passwords, using large sets of leaked passwords for training. Both groups compared their model with other probabilistic attacks, including Weir et al.'s original PCFG code, finding particular configurations of a Markov model to be more efficient at guessing passwords for some datasets. We use Ma et al.'s recommended model in our tests [40].

Mangled wordlist attacks Perhaps the most popular strategy in real-world password cracking is the dictionary attack. First proposed by Morris and Thompson in 1979 [43], modern-day dictionary attacks often combine *wordlists* with *mangling rules*, string transformations that modify wordlist entries to create additional guesses. Wordlists usually contain both natural language dictionaries and stolen password sets. Typical mangling rules perform transformations like appending digits and substituting characters [50, 59].

Many modern cracking tools, including John the Ripper [49], Hashcat [57], and PasswordsPro [30], support these attacks, which we term *mangled wordlist attacks*. The popularity of this category of attack is evident from these tools' wide use and success in password-cracking competitions [36,51]. Furthermore, a number of research papers have used John the Ripper, often with the default mangling rules [11, 14, 15, 20, 21, 28, 34, 71] or additional mangling rules [16, 19, 72].

Expert password crackers, such as those offering forensic password-recovery services, frequently perform a variant of the mangled wordlist attack in which humans manually write, prioritize, and dynamically update rules [23]. We term these manual updates to mangling rules *freestyle rules*. As we discuss in Section 3, we evaluate guessability using off-the-shelf tools relying on publicly available wordlists and mangling rules. We also contract a password recovery industry leader to do the same using their proprietary wordlists and freestyle rules.

3 Methodology

We analyze four automated guessing algorithms and one manual cracking approach (together, our five *cracking approaches*). We first describe the password sets for which we calculated guessability, then explain the training data we used. Afterwards, we discuss our five cracking approaches. Finally, we discuss computational limitations of our analyses.

3.1 Datasets

We examine 13,345 passwords from four sets created under composition policies ranging from the typical to the currently less common to understand the success of password-guessing approaches against passwords of different characteristics. Since no major password leaks contain passwords created under strict composition policies, we leverage passwords that our group collected for prior studies of password-composition policies [31, 42, 56]. This choice of data also enables us to contract with a professional computer security firm to crack these unfamiliar passwords. Had we used any major password leak, their analysts would have already been familiar with most or all of the passwords contained in the leak, biasing results.

The passwords in these sets were collected using Amazon's Mechanical Turk crowdsourcing service. Two recent studies have demonstrated that passwords collected for research studies, while not perfect proxies for real data, are in many ways very representative of real passwords from high-value accounts [20, 42].

Despite these claims, we were also curious how real passwords would differ in our analyses from those collected on Mechanical Turk. Therefore, we repeated our analyses of Basic passwords (see below) with 15,000 plaintext passwords sampled from the RockYou gaming site leak [67] and another 15,000 sampled from a Yahoo! Voices leak [22]. As we detail in Appendix A.4, our Basic passwords and comparable passwords from these two real leaks yielded approximately the same results.

Next, we detail our datasets, summarized in Table 1. The **Basic** set comprises 3,062 passwords collected for a research study requiring a minimum length of 8 characters [31]. As we discuss in Section 4, the vast majority of 8-character passwords can be guessed using off-the-shelf, automated approaches. Hence, we give particular attention to longer and more complex passwords, which will likely represent best practices moving forward.

System administrators commonly require passwords to contain multiple character classes (lowercase letters, uppercase letters, digits, and symbols). The **Complex** set comprises passwords required to contain 8+ characters, include all 4 character classes, and not be in a cracking wordlist [46] after removing digits and symbols. They were also collected for research [42].

Recent increases in hashing speeds have made passwords of length 8 or less increasingly susceptible to offline guessing [24, 31]. We therefore examine 2,054 **LongBasic** passwords collected for research [31] that required a a minimum length of 16 characters. Finally, we

Table 1: Characteristics of passwords per set, including the percentage of characters that were lowercase (LC) or uppercase (UC) letters, digits, or symbols (Sym).

		Length	**% of Characters**			
Set	#	Mean (σ)	LC	UC	Digit	Sym
Basic	3,062	9.6 (2.2)	68	4	26	1
Complex	3,000	10.7 (3.2)	51	14	25	11
LongBasic	2,054	18.1 (3.1)	73	4	20	2
LongComplex	990	13.8 (2.6)	57	12	22	8

examine 990 **LongComplex** passwords, also collected for research [56], that needed to contain 12+ characters, including characters from 3 or more character classes.

3.2 Training Data

To compare cracking approaches as directly as possible, we used the same training data for each. That said, each algorithm uses training data differently, making perfectly equivalent comparisons impossible.

Our training data comprised leaked passwords and dictionaries. The passwords were from breaches of MySpace, RockYou, and Yahoo! (excluding the aforementioned 30,000 passwords analyzed in Appendix A.4). Using leaked passwords raises ethical concerns. We believe our use of such sets in this research is justifiable because the password sets are already available publicly and we exclude personally identifiable information, such as usernames. Furthermore, malicious agents use these sets in attacks [23]; failure to consider them in our analyses may give attackers an advantage over those who work in defensive security.

Prior research has found including natural-language dictionaries to work better than using just passwords [31, 69]. We used the dictionaries previously found most effective: all single words in the Google Web corpus [26], the UNIX dictionary [1], and a 250,000-word inflection dictionary [55]. The combined set of passwords and dictionaries contained 19.4 million unique entries. For cracking approaches that take only a wordlist, without frequency information, we ordered the wordlist by descending frequency and removed duplicates. We included frequency information for the other approaches.

3.3 Simulating Password Cracking

To investigate the degree to which research results can be biased by the choice of cracking algorithm, as well as how automated approaches compare to real attacks, we investigated two cracking tools and two probabilistic algorithms. We selected approaches based on their popularity in the academic literature or the password-cracking community, as well as their conceptual distinctness. We also contracted a computer security firm specializing in password cracking for the real-world attack.

Most cracking approaches do not natively provide guess numbers, and instrumenting them to calculate guessability was typically far from trivial. Because this instrumentation enabled the comparisons in this paper and can similarly support future research, we include many details in this section about this instrumentation. Furthermore, in Section 5, we introduce a Password Guessability Service so that other researchers can leverage our instrumentation and computational resources.

For each approach, we analyze as many guesses as computationally feasible, making 100 trillion (10^{14}) guesses for some approaches and ten billion (10^{10}) guesses for the most resource-intensive approach. With the exception of Hashcat, as explained below, we filter out guesses that do not comply with a password set's composition policy. For example, a LongComplex password's guess number excludes guesses with under 12 characters or fewer than 3 character classes.

We define **Min**$_{\text{auto}}$ as the minimum guess number (and therefore the most conservative security result) for a given password across our automated cracking approaches. This number approximates the best researchers can expect with well-configured automation.

In the following subsections, we detail the configuration (and terminology) of the five approaches we tested. We ran CPU-based approaches (JTR, PCFG, Markov) on a 64-core server. Each processor on this server was an AMD Opteron 6274 running at 1.4Ghz. The machine had 256 GB of RAM and 15 TB of disk. Its market value is over $10,000, yet we still faced steep resource limitations generating Markov guesses. We ran Hashcat (more precisely, oclHashcat) on a machine with six AMD R9 270 GPUs, 2 GB of RAM, and a dual-core processor.

Probabilistic context-free grammar Weir et al.'s probabilistic context-free grammar (termed **PCFG**) [70] has been widely discussed in recent years. We use Komanduri's implementation of PCFG [32], which improves upon the guessing efficiency of Weir et al.'s work [70] by assigning letter strings probabilities based on their frequency in the training data and assigning unseen strings a non-zero probability. This implementation is a newer version of Kelley et al.'s implementation of PCFG as a lookup table for quickly computing guess numbers, rather than enumerating guesses [31].

Based on our initial testing, discussed further in Section 4.1, we prepend our training data, ordered by frequency, before PCFG's first guess to improve performance. As a result, we do not use Komanduri's hybrid structures [32], which serve a similar purpose. We weight passwords $10\times$ as heavily as dictionary entries.

We were able to simulate 10^{12} guesses for Complex passwords and 10^{14} guesses for the other three sets.

Markov model Second, we evaluated the **Markov-model** password guesser presented by Ma et al. [40], which implemented a number of variants differing by order and approaches to smoothing. We use the order-5 Markov-chain model, which they found most effective for English-language test sets. We tried using both our combined training data (dictionaries and paswords) using the same weighting as with PCFG, as well as only the passwords from our training data. The combined training data and passwords-only training data performed nearly identically. We report only on the combined training data, which was slightly more effective for Basic passwords and is most consistent with the other approaches.

We used Ma et al.'s code [40], which they shared with us, to enumerate a list of guesses in descending probability. We used a separate program to remove guesses that did not conform to the given password-composition policy. Because this approach is extremely resource-intensive, both conceptually (traversing a very large tree) and in its current implementation, we were not able to analyze as many guesses as for other approaches. As with PCFG, we found prepending the training data improved performance, albeit only marginally for Markov. Therefore, we used this tweak. We simulated over 10^{10} guesses for Basic passwords, similar to Ma et al. [40].

John the Ripper We also tested variants of a mangled wordlist attack implemented in two popular software tools. The first tool, John the Ripper (termed **JTR**), has been used in a number of prior studies as a security metric, as described in Section 2. In most cases, these prior studies used JTR with its stock mangling rules. However, pairing the stock rules with our 19.4-million-word wordlist produced only 10^8 guesses for Basic passwords. To generate more guesses, we augment the stock rules with 5,146 rules released for DEF CON's "Crack Me If You Can" (CMIYC) password-cracking contest in 2010 [35]. Specifically, we use Trustwave SpiderLabs' reordering of these rules for guessing efficiency [64]. Our JTR tests therefore use the stock mangling rules followed by the Spiderlabs rules. For completeness, Appendix A.2 presents these rules separately.

Instrumenting JTR to calculate precise guess numbers was an involved process. We used `john-1.7.9-jumbo` with the `--stdout` flag to output guesses to standard out. We piped these guesses into a program we wrote to perform a regular expression check filtering out guesses that do not conform to the given password policy. This program then does a fast hash table lookup with GNU `gperf` [27] to quickly evaluate whether a guess matches a password in our dataset. Using this method, we achieved a throughput speed of 3 million guesses per second and made more than 10^{13} guesses for Basic passwords.

Hashcat While Hashcat is conceptually similar to JTR, we chose to also include it in our tests for two reasons. First, we discovered in our testing that JTR and Hashcat iterate through guesses in a very different order, leading to significant differences in the efficacy of guessing specific passwords. JTR iterates through the entire wordlist using one mangling rule before proceeding to the subsequent mangling rule. Hashcat, in contrast, iterates over all mangling rules for the first wordlist entry before continuing to the subsequent wordlist entry.

Second, the GPU-based oclHashcat, which is often used in practice [23, 24, 36, 51], does not permit users to filter guesses that do not meet password-composition requirements except for computationally expensive hash functions. We accept this limitation both because it represents the actual behavior of a popular closed-source tool and because, for fast hashes like MD5 or NTLM, guessing without filtering cracks passwords faster in practice than applying filtering.

Unlike JTR, Hashcat does not have a default set of mangling rules, so we evaluated several. We generally report on only the most effective set, but detail our tests of four different rule sets in Appendix A.3. This most effective rule set, which we term **Hashcat** throughout the paper, resulted from our collaboration with a Hashcat user and password researcher from MWR InfoSecurity [25, 44], who shared his mangling rules for the purpose of this analysis. We believe such a configuration represents a typical expert configuration of Hashcat.

We used `oclHashcat-1.21`. While, like JTR, Hashcat provides a debugging feature that streams guesses to standard output, we found it extremely slow in practice relative to Hashcat's very efficient GPU implementation. In support of this study, Hashcat's developers generously added a feature to oclHashcat to count how many guesses it took to arrive at each password it cracked. This feature is activated using the flag `--outfile-format=11` in `oclHashcat-1.20` and above. We therefore hashed the passwords in our datasets using the NTLM hash function, which was the fastest for Hashcat to guess in our benchmarks. We then used Hashcat to actually crack these passwords while counting guesses, with throughput of roughly 10 billion guesses per second on our system. We made more than 10^{13} guesses for Basic passwords, along with nearly 10^{15} guesses for some alternate configurations reported in Appendix A.3.

Professional cracker An open question in measuring password guessability using off-the-shelf, automated tools is how these attacks compare to an experienced, real-world attacker. Such attackers manually customize and dynamically update their attacks based on a target set's characteristics and initial successful cracks.

To this end, we contracted an industry leader in professional password recovery services, KoreLogic (termed **Pros**), to attack the password sets we study. We believe KoreLogic is representative of expert password crackers because they have organized the DEF CON "Crack Me If You Can" password-cracking contest since 2010 [36] and perform password-recovery services for many Fortune-500 companies [38]. For this study, they instrumented their distributed cracking infrastructure to count guesses.

Like most experienced crackers, the KoreLogic analysts used tools including JTR and Hashcat with proprietary wordlists, mangling rules, mask lists, and Markov models optimized over 10 years of password auditing. Similarly, they dynamically update their mangling rules (termed *freestyle rules*) as additional passwords are cracked. To unpack which aspects of a professional attack (e.g., proprietary wordlists and mangling rules, freestyle rules, etc.) give experienced crackers an advantage, we first had KoreLogic attack a set of 4,239 Complex passwords (distinct from those reported in our other tests) in artificially limited configurations.

We then had the professionals attack the Complex, LongBasic, and LongComplex passwords with no artificial limitations. An experienced password analyst wrote freestyle rules for each set before cracking began, and again after 10^{13} guesses based on the passwords guessed to that point. They made more than 10^{14} guesses per set.

LongBasic and LongComplex approaches are rare in corporate environments and thus relatively unfamiliar to real-world attackers. To mitigate this unfamiliarity, we randomly split each set in two and designated half for training and half for testing. We provided analysts with the training half (in plaintext) to familiarize them with common patterns in these sets. Because we found that automated approaches can already crack most Basic passwords, rendering them insecure, we chose not to have the professionals attack Basic passwords.

3.4 Computational Limitations

As expected, the computational cost of generating guesses in each approach proved a crucial limiting factor in our tests. In three days, oclHashcat, the fastest of our approaches, produced 10^{15} guesses using a single AMD R9 290X GPU (roughly a \$500 value). In contrast, the Markov approach (our slowest) required three days on a roughly \$10,000 server (64 AMD Opteron 6274 CPU cores and 256 GB of RAM) to generate 10^{10} guesses without computing a single hash. In three days on the same machine as Markov, PCFG simulated 10^{13} guesses.

The inefficiency of Markov stems partially from our use of a research implementation. Even the most efficient implementation, however, would still face substantial conceptual barriers. Whereas Hashcat and JTR incur the same performance cost generating the quadrillionth guess as the first guess, Markov must maintain a tree of substring probabilities. As more guesses are desired, the tree must grow, increasing the cost of both storing and traversing it. While Markov produced a high rate of successful guesses per guess made (see Section 4.2), the cost of generating guesses makes it a poor choice for computing guessability beyond billions of guesses.

Further, our automated approaches differ significantly in how well they handle complex password-composition policies. For PCFG, non-terminal structures can be pruned before guessing starts, so only compliant passwords are ever generated. As a result, it takes about equal time for PCFG to generate Basic passwords as LongComplex passwords. In contrast, Markov must first generate all passwords in a probability range and then filter out non-compliant passwords, adding additional overhead per guess. JTR has a similar generate-then-filter mechanism, while Hashcat (as discussed above) does not allow this post-hoc filtering at all for fast hashes. This means that Markov and JTR take much longer to generate valid LongComplex guesses than Basic guesses, and Hashcat wastes guesses against the LongComplex set.

As a result of these factors, the largest guess is necessarily unequal among approaches we test, and even among test sets within each approach. To account for this, we only compare approaches directly at equivalent guess numbers. In addition, we argue that these computational limitations are important in practice, so our findings can help researchers understand these approaches and choose among them appropriately.

4 Results

We first show, in Section 4.1, that for each automated guessing approach we evaluated, different seemingly reasonable configurations produce very different cracking results, and that out-of-the-box configurations commonly used by researchers substantially underestimate password vulnerability.

Next, in Section 4.2, we examine the relative performance of the four automated approaches. We find they are similarly effective against Basic passwords. They have far less success against the other password sets, and their relative effectiveness also diverges.

For the three non-Basic sets, we also compare the automated approaches to the professional attack. Pros outperform the automated approaches, but only after a large number of guesses. As Pros crack more passwords, their manual adjustments prove quite effective; automated approaches lack this feedback mechanism. We also find that, at least through 10^{14} guesses, automated approaches can conservatively approximate human password-cracking experts, but only if a password is counted as guessed when *any* of the four automated approaches guesses it. A single approach is not enough.

In Section 4.3, we explore the degree to which different cracking approaches overlap in which particular passwords they guess. While multiple approaches successfully guess most Basic passwords, many passwords in the other classes are guessed only by a single approach. We also find that different cracking approaches provide systematically different results based on characteristics like the number of character classes in a password.

In Section 4.4, we revisit how the choice of guessing approach impacts research questions at a high level (e.g., how composition policies impact security) and lower level (e.g., if a particular password is hard to guess). While we find analyses on large, heterogeneous sets of passwords to be fairly robust, security estimates for a given password are very sensitive to the approach used.

4.1 The Importance of Configuration

We found that using any guessing approach naively performed far more poorly, sometimes by more than an order of magnitude, than more expert configurations.

Stock vs advanced configurations We experimented with several configurations each for Hashcat and JTR, including the default configurations they ship with, and observed stark differences in performance. We detail a few here; others are described in Appendices A.2 and A.3.

For example, Hashcat configured with the (default) Best64 mangling rules guessed only about 2% of the Complex passwords before running out of guesses. Using the mangling rules described in Section 3, it made far more guesses, eventually cracking 30% (Figure 1).

Similarly, JTR guessed less than 3% of Complex passwords before exhausting its stock rules. The larger set of rules described in Section 3 enabled it to guess 29% (see Appendix A.2 for details). We found similar configuration effects for LongComplex passwords, and analogous but milder effects for the Basic and LongBasic sets.

We also compared the PCFG implementation we use throughout the paper [32] with our approximation of the originally published algorithm [70], which differs in how probabilities are assigned (see Section 3). As we detail in Appendix A.1, the newer PCFG consistently outperforms the original algorithm; the details of the same conceptual approach greatly impact guessability analyses.

Choices of training data The performance of PCFG and Markov depends heavily on the quality of training data. Our group previously found that training with closely related passwords improves performance [31].

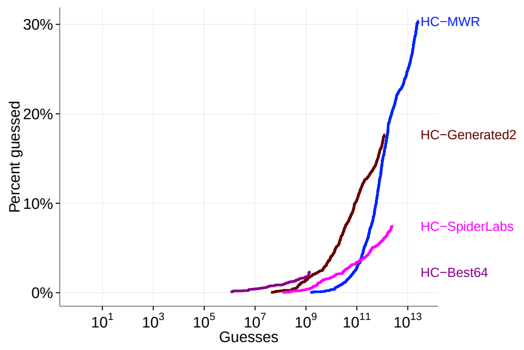


Figure 1: Results of Hashcat configured using the same wordlist, but different sets of mangling rules (described in Appendix A.3), to guess Complex passwords.

For our non-basic password sets, however, closely matched data is not available in publicly leaked sets.

In tests reported in Appendix A.1, we thus incorporated closely matched data via cross-validation, in which we iteratively split the test set into training and testing portions. Using cross-validation improved guessing efficiency for three of the four password sets, most dramatically for LongBasic. This result demonstrates that an algorithm trained with generic training data will miss passwords that are vulnerable to an attacker who has training data that closely matches a target set. To minimize differences across approaches, however, PCFG results in the body of the paper use generic training data only.

Actionable takeaways Together, these results suggest that a researcher must carefully manage guessing configuration before calculating password guessability. In particular, tools like JTR and Hashcat will "out of the box" systematically underestimate password guessability. Unfortunately, many existing research studies rely on unoptimized configurations [11, 14, 15, 20, 21, 28, 34, 71].

While we report on the configurations we found most effective in extensive testing, we argue that the research community should establish configuration best practices, which may depend on the password sets targeted.

4.2 Comparison of Guessing Approaches

We first show that automated approaches differ in effectiveness based on the nature of the password sets being cracked and the number of guesses at which they are compared. We then compare these automated approaches to cracking by an expert attacker making dynamic updates, finding that the expert lags in initial guessing efficiency, yet becomes stronger over time. We find the minimum guess number across automated approaches can serve as a conservative proxy for guessability by an expert attacker.

4.2.1 Guessing by Automated Approaches

On some password sets and for specific numbers of guesses, the performance of all four approaches was similar (e.g., at 10^{12} guesses all but Markov had guessed 60-70% of Basic passwords). In contrast, on other sets, their performance was inconsistent at many points that would be relevant for real-world cracking (e.g., PCFG cracked 20% of Complex passwords by 10^{10} guesses, while Hashcat and JTR had cracked under 3%).

As shown in Figure 2, all four automated approaches were quite successful at guessing Basic passwords, the most widely used of the four classes. Whereas past work has found that, for password sets resembling our Basic passwords, PCFG often guesses more passwords than JTR [16] or that Markov performs significantly better than PCFG [40], good configurations of JTR, Markov, and PCFG performed somewhat similarly in our tests. Hashcat was less efficient at generating successful guesses in the millions and billions of guesses, yet it surpassed JTR by 10^{12} guesses and continued to generate successful guesses beyond 10^{13} guesses.

The four automated approaches had far less success guessing the other password sets. Figure 3 shows the guessability of the Complex passwords under each approach. Within the first ten million guesses, very few passwords were cracked by any approach. From that point until its guess cutoff, PCFG performed best, at points having guessed nearly ten times as many passwords as JTR. Although its initial guesses were often successful, the conceptual and implementation-specific performance issues we detailed in Section 3.4 prevented Markov from making over 100 million valid Complex guesses, orders of magnitude less than the other approaches we examined. A real attack using this algorithm would be similarly constrained.

Both Hashcat and JTR performed poorly compared to PCFG in early Complex guessing. By 10^9 guesses, each had each guessed under 3% of Complex passwords, compared to 20% for PCFG. Both Hashcat and JTR improve rapidly after 10^{10} guesses, however, eventually guessing around 30% of Complex passwords.

JTR required almost 10^{12} guesses and Hashcat required over 10^{13} guesses to crack 30% of Complex passwords. As we discuss in Section 4.3, there was less overlap in which passwords were guessed by multiple automated approaches for Complex passwords than for Basic passwords. As a result, the Min_{auto} curve in Figure 3, representing the smallest guess number per password across the automated approaches, shows that just under 10^{11} guesses are necessary for 30% of Complex passwords to have been guessed by at least one automated approach. Over 40% of Complex passwords were guessed by at least one automated approach in 10^{13} guesses.

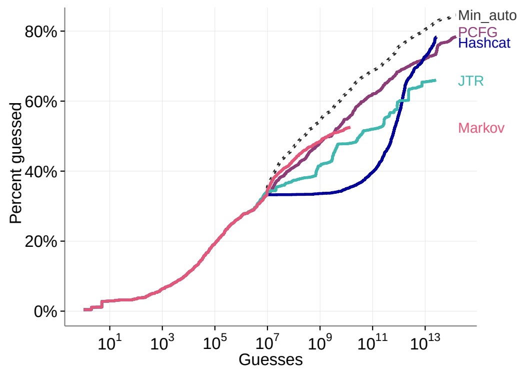


Figure 2: Automated approaches' success guessing Basic passwords. Min_{auto} represents the smallest guess number for a password by any automated approach.

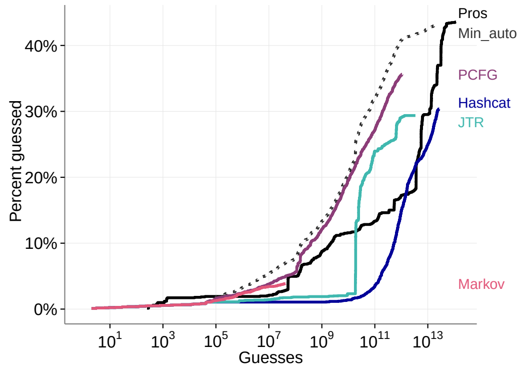


Figure 3: Success guessing Complex passwords. Pros are experts updating their guessing strategy dynamically.

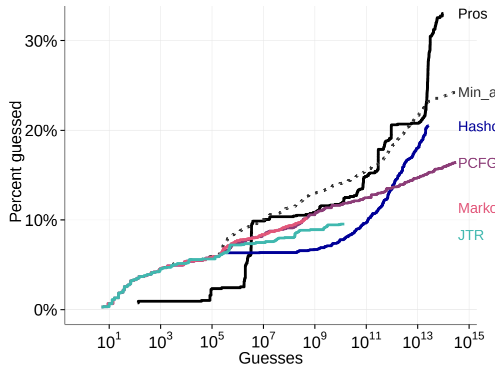


Figure 4: Success guessing LongBasic passwords.

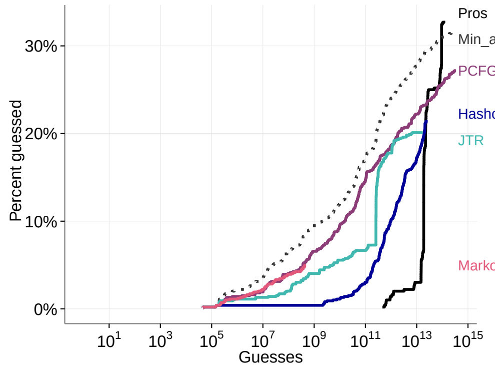


Figure 5: Success guessing LongComplex passwords.

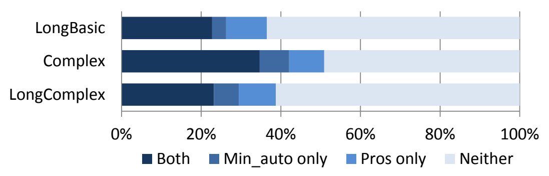


Figure 6: The proportion of passwords guessed by Min_{auto}, Pros, both, or neither within 10^{14} guesses.

LongBasic passwords were also challenging for all approaches to guess, though relative differences across approaches are not as stark as for Complex passwords. Markov was marginally more successful than other approaches at its cutoff just before 10^9 guesses. JTR and PCFG both continued to generate successful guesses through when JTR exhausted its guesses after guessing 10% of the passwords. Hashcat lagged slightly behind JTR at 10^9 guesses (7% cracked vs ~9%), but was able to make more guesses than either, eventually guessing over 20% of the passwords, compared to 16% for PCFG and 10% for JTR at those approaches' guess cutoffs.

As with LongBasic passwords, all approaches had difficulty guessing LongComplex passwords. As shown in Figure 5, nearly 70% of LongComplex passwords were not guessed by any of the approaches we examined even after trillions of guesses. The relative performance of the four automated guessing approaches for LongComplex passwords again differed noticeably. Markov and PCFG again outperformed other approaches early. Markov guessed 5% of the passwords after 10^8 guesses, yet reached its guess cutoff soon thereafter. At 10^9 guesses PCFG and JTR had both also guessed at least 5% of the passwords, compared to almost no passwords guessed by Hashcat. PCFG's and JTR's performance diverged and then converged at higher guess numbers. Hashcat caught up at around 10^{13} guesses, cracking 20% of LongComplex passwords.

4.2.2 Guessing by Pros

As we expected, Pros guessed more passwords overall than any of the automated approaches. As we discussed in Section 3, we chose not to have Pros attack Basic passwords because those passwords could be guessed with automated approaches alone. As shown in Figures 3–5, within 10^{14} guesses Pros cracked 44% of Complex passwords, 33% of LongBasic passwords, and 33% of LongComplex passwords, improving on the guessing of the best automated approach.

Three aspects of guessing by Pros were particularly notable. First, even though Pros manually examined half of each password set and adjusted their mangling rules and wordlists before making the first guess against each set, automated approaches were often more successful at early guessing. For example, Markov surpassed Pros at guessing Complex passwords in the first 10^2 guesses and again from around 10^6 till Markov's guess cutoff at 5×10^7. Similarly, all four automated approaches guessed LongComplex passwords more successfully than Pros from the start of guessing until past 10^{13} guesses. All approaches guessed LongBasic passwords better than Pros for the first 10^6 guesses.

Second, while Pros lagged in early guessing, the freestyle rules an experienced analyst wrote at 10^{13} guesses proved rather effective and caused a large spike in successful guesses for all three password sets. Hashcat, the only automated approach that surpassed 10^{13} guesses for all sets, remained effective past 10^{13} guesses, yet did not experience nearly the same spike.

Third, while Pros were more successful across password sets once a sufficiently high number of guesses had been reached, the automated approaches we tested had guessing success that was, to a very rough approximation, surprisingly similar to Pros. As we discussed in Section 4.1 and discuss further in the appendix, this success required substantial configuration beyond each approach's performance out of the box.

We found that our Min_{auto} metric (the minimum guess number for each password across Hashcat, JTR, Markov, and PCFG) served as a conservative approximation of the success of Pros, at least through our automated guess cutoffs around 10^{13} guesses. As seen in Figures 3–6, Pros never substantially exceeded Min_{auto}, yet often performed worse than Min_{auto}.

Professional cracking with limitations To unpack why professional crackers have an advantage over novice attackers, we also had KoreLogic attack a different set of Complex passwords in artificially limited configurations. These limitations covered the wordlists they used, the mangling rules they used, and whether they were permitted to write freestyle rules. To avoid biasing subsequent tests, we provided them a comparable set of 4,239 Complex passwords [31] distinct from those examined in the rest of the paper. We call this alternate set $\textbf{Complex}_{pilot}$.

As shown in Table 2, we limited Pros in Trial 1 to use the same wordlist we used elsewhere in this paper and did not allow freestyle rules. In Trial 2, we did not limit the wordlist, but did limit mangling rules to those used in the 2010 Crack Me If You Can contest [35]. In Trial 3 and Trial 4, we did not limit the starting wordlist or mangling rules. In Trial 4, however, KoreLogic analysts dynamically adjusted their attacks through freestyle rules and wordlist tweaks after 10^{14} guesses.

We found that KoreLogic's set of proprietary mangling rules had a far greater impact on guessing efficiency than their proprietary wordlist (Figure 7). Furthermore, as evidenced by the difference between Trial 3

Table 2: The four trials of Pros guessing Complex$_{pilot}$. We artificially limited the first three trials to uncover why Pros have an advantage over more novice attackers.

Trial	Wordlist	Rules	Freestyle Rules
1	CMU wordlist	Anything	None
2	Anything	2010 CMIYC rules	None
3	Anything	Anything	None
4	Anything	Anything	Unlimited

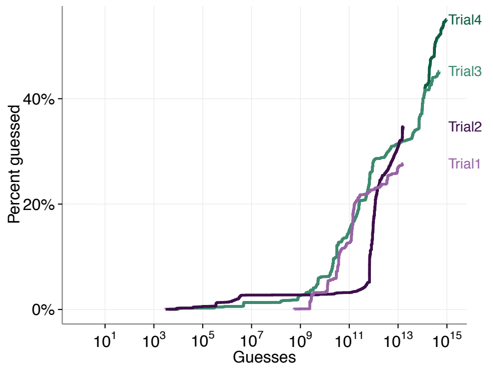


Figure 7: Complex$_{pilot}$ guessability by trial.

and Trial 4, freestyle rules also had a major impact at the point the analyst wrote them.

Actionable takeaways One conceptual advantage of parameterized metrics is that they model an attack using existing cracking approaches. However, it has long been unclear whether automated cracking approaches used by researchers effectively model the dynamically updated techniques used by expert real-world attackers. Our results demonstrate that only by considering multiple automated approaches in concert can researchers approximate professional password cracking.

One of our primary observations, both from comparing Pros to the automated approaches and from our trials artificially limiting Pros (Section 4.2.2), is that dynamically updated freestyle rules can be highly effective. This result raises the question of to what extent automated approaches can model dynamic updates. Although the adversarial cracking community has discussed techniques for automatically generating mangling rules from previous cracks [41], researchers have yet to leverage such techniques, highlighting an area ripe for future work.

Contrary to prior research (e.g., [16, 40]), we found that Hashcat, JTR, Markov, and PCFG all performed relatively effectively when configured and trained according to currently accepted best practices in the cracking and research communities. That said, our tests also highlighted a limitation of the guessability metric in not considering the performance cost of generating a guess. Despite its real-world popularity, Hashcat performed comparatively poorly until making trillions of guesses, yet generated guesses very quickly.

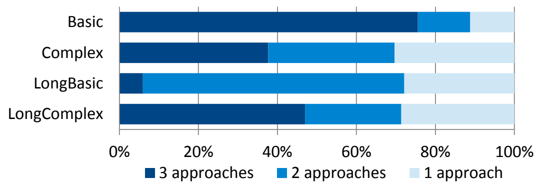


Figure 8: Number of automated approaches, excluding Markov, that cracked a particular password. We ignore passwords not guessed by any approach and use the same guess cutoff for all guessing approaches within a set.

If hashing a guess is the dominant time factor, as is the case for intentionally slow hash functions like bcrypt, PBKDF2, and scrypt, probabilistic approaches like Markov and PCFG are advantageous for an attacker. For fast hash functions like MD5 or NTLM, Hashcat's speed at generating and hashing guesses results in more passwords being guessed in the same wall-clock time. As discussed in Section 3.4, Markov proved comparatively very resource-intensive to run to a large guess number, especially for password sets with complex requirements. These practical considerations must play a role in how researchers select the best approaches for their needs.

4.3 Differences Across Approaches

Next, we focus on differences between approaches. We first examine if multiple approaches guess the same passwords. We then examine the guessability of passwords with particular characteristics, such as those containing multiple character classes or character substitutions. To examine differences across how approaches model passwords, for analyses in this section we do not prepend the training data to the guesses generated by the approach.

4.3.1 Overlap in Successful Guesses

While one would expect any two cracking approaches to guess slightly different subsets of passwords, we found larger-than-expected differences for three of the four password sets. Figure 8 shows the proportion of passwords in each class guessed by all four approaches, or only some subset of them. We exclude passwords guessed by none of the automated approaches. Within a password set, we examine all approaches only up to the minimum guess cutoff among Hashcat, JTR, and PCFG; we exclude Markov due to its low guess cutoffs.

The three approaches guessed many of the same Basic passwords: Three-fourths of Basic passwords guessed by any approach were guessed by all of them. Only 11% of Basic passwords were guessed only by a single ap-

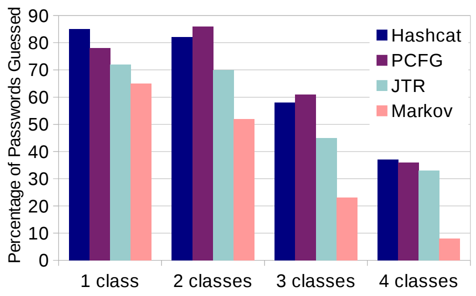


Figure 9: Percentage of Basic passwords each approach guessed, by character-class count.

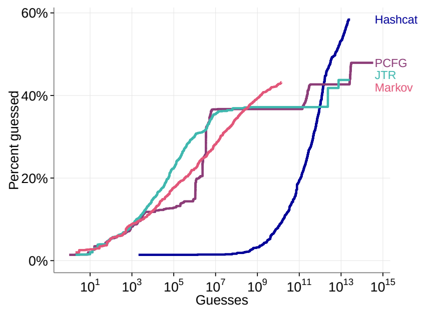


Figure 10: Approaches' effectiveness guessing passwords composed entirely of lowercase letters across sets.

proach. In contrast, only 6% of LongBasic passwords were guessed by all approaches, while 28% of Complex, LongBasic, and LongComplex passwords were guessed only by a single approach.

4.3.2 Guessing Success by Password Characteristics

While it is unsurprising that different approaches do better at guessing distinct types of passwords, we found differences that were large and difficult to predict.

Character classes and length We first considered how efficiently automated approaches guessed passwords relative to their length and character-class count. These two characteristics are of particular interest because they are frequently used in password-composition policies.

As shown in Figure 9, the impact of adding character classes is not as straightforward as one might expect. While the general trend is for passwords with more character classes to be stronger, the details vary. Markov experiences a large drop in effectiveness with each increase in character classes (63% to 52% to 23% to 8%). JTR, by contrast, finds only a minor difference between one and two classes (72% to 70%). PCFG actually increases in effectiveness between one and two classes (78% to 86%). Since changes in security and usability as a result of different policies are often incremental (e.g., [8]), the magnitude of these disagreements can easily affect research conclusions about the relative strength of passwords.

In contrast, we did not find surprising idiosyncrasies based on the length of the password. For all approaches, cracking efficiency decreased as length increased.

Character-level password characteristics As the research community seeks to understand the characteristics of good passwords, a researcher might investigate how easy it is to guess all-digit passwords, which are common [6], or examine the effect of character substitutions (e.g., *$Hplovecraft!$* $\rightarrow$ *$Hpl0v3cr@ft!$*) on guessability. Despite their sometimes similar effectiveness overall, approaches often diverged when guessing passwords that had these characteristics. As a result, researchers using different approaches could draw different conclusions about the guessability of these properties.

The guessability of the 1,490 passwords (across sets) composed entirely of lowercase letters varied starkly by guessing approach. This variation is particularly notable because such passwords made up 29% of Basic and LongBasic passwords, and were impermissible under the other two composition policies. As shown in Figure 10, Hashcat guessed few such passwords until well into the billions of guesses, whereas Markov successfully guessed passwords composed entirely of lowercase letters throughout its attack. In contrast, PCFG had a large spike in successful guesses between 1 million and 10 million guesses, but then plateaued. JTR had early success, but similarly plateaued from 10 million guesses until into the trillions of guesses.

Similarly, approaches differed in their efficiency guessing passwords containing character substitutions, which we identified using crowdsourcing on Amazon's Mechanical Turk. Passwords identified by crowdworkers as containing character substitutions included *4Everblessed*, *B1cycle_Race*, and *Ca$hmoneybr0*. PCFG performed poorly relative to JTR and Markov at guessing passwords with character substitutions. A researcher using only PCFG could mistakenly believe these passwords are much stronger than they actually are. We found similar differences with many other common characteristics, potentially skewing research conclusions.

Actionable takeaways Given the many passwords guessed by only a single cracking approach and the systematic differences in when passwords with certain characteristics are guessed, we argue that researchers must consider major cracking approaches in parallel.

Our results also show how comparative analyses uncover relative weaknesses of each approach. Upon close examination, many of these behaviors make sense. For example, PCFG abstracts passwords into structures of non-terminal characters based on character class, ignoring contextual information across these boundaries. As a result, *P@ssw0rd* would be split into "P," "@," "ssw," "0," and "rd," explaining PCFG's poor performance guessing passwords with character substitutions.

4.4 Robustness of Analyses to Approach

In this section, we examine whether differences among automated cracking approaches are likely to affect conclusions to two main types of research questions.

We first consider analyses of password sets, such as passwords created under particular password-composition policies. We find such analyses to be somewhat, but not completely, robust to the approach used.

In contrast, per-password analyses are very sensitive to the guessing approach. Currently, such analyses are mainly used in security audits [61] to detect weak passwords. In the future, however, per-password strength metrics may be used to provide detailed feedback to users during password creation, mirroring the recent trend of data-driven password meters [10, 33]. The ability to calculate a guess number per-password is a major advantage of parameterized metrics over statistical metrics, yet this advantage is lost if guess numbers change dramatically when a different approach is used. Unfortunately, we sometimes found huge differences across approaches.

4.4.1 Per Password Set

As an example of an analysis of large password sets, we consider the relative guessability of passwords created under different composition policies, as has been studied by Shay et al. [56] and Kelley et al. [31].

Figure 11 shows the relative guessability of the three password sets examined by the Pros. LongBasic passwords were most vulnerable, and LongComplex passwords least vulnerable, to early guessing (under 10^9 guesses). Between roughly 10^9 and 10^{12} guesses, LongBasic and Complex passwords followed similar curves, though Complex passwords were cracked with higher success past 10^{12} guesses. Very few LongComplex passwords were guessed before 10^{13} guesses, yet Pros quickly guessed about one-third of the LongComplex set between 10^{13} and 10^{14} guesses.

Performing the same analysis using Min_{auto} guess numbers instead (Figure 12) would lead to similar conclusions. LongBasic passwords were again more vulnerable than Complex or LongComplex under 10^8 guesses. After 10^{12} guesses, Complex passwords were easier to guess than LongBasic or LongComplex passwords. Basic passwords were easy to guess at all points. The main difference between Min_{auto} and Pros was that LongComplex passwords appear more vulnerable to the first 10^{12} guesses under Min_{auto} than Pros.

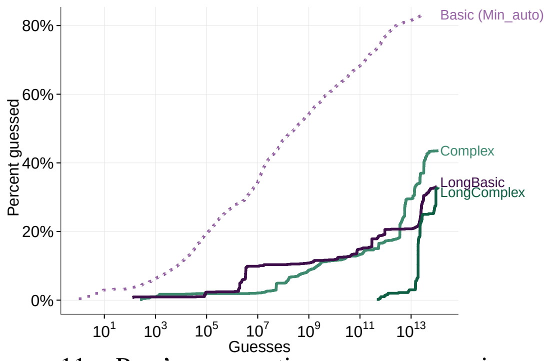


Figure 11: Pros' comparative success guessing each password. For reference, the dotted line represents the Min_{auto} guess across automated approaches for Basic passwords, which the Pros did not try to guess.

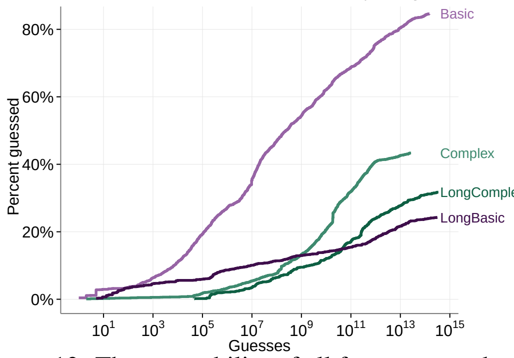


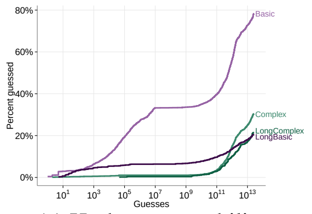
Figure 12: The guessability of all four password sets under Min_{auto}, representing the smallest guess number for each password across all four automated approaches.

Based on this data, a researcher comparing composition policies would likely reach similar conclusions using either professionals or a combination of automated approaches. As shown in Figure 13, we repeated this analysis using each of the four automated approaches in isolation. Against every approach, Basic passwords are easily guessable, and LongBasic passwords are comparatively vulnerable during early guessing. After trillions of guesses, Hashcat, PCFG, and JTR find Long Complex passwords more secure than Complex passwords. In each case, a researcher would come to similar conclusions about the relative strength of these password sets.

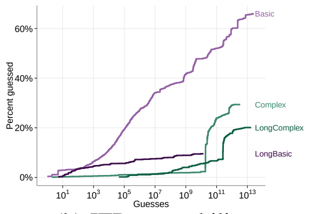
4.4.2 Per Individual Password

Analyses of the strength of individual passwords, in contrast, proved very sensitive to the guessing approach. Although one would expect different approaches to guess passwords at somewhat different times, many passwords' guess numbers varied by orders of magnitude across approaches. This state of affairs could cause a very weak password to be misclassified as very strong.

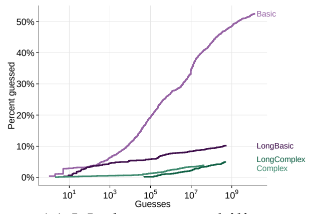
We examined per-password differences pairwise among JTR, Markov, and PCFG, using the same guess



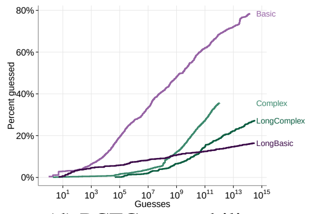
(a) Hashcat guessability.



(b) JTR guessability.



(c) Markov guessability.



(d) PCFG guessability.

Figure 13: The relative guessability of the four different password sets under each of the four automated cracking approaches considered in isolation. The research conclusions would be fairly similar in each case.

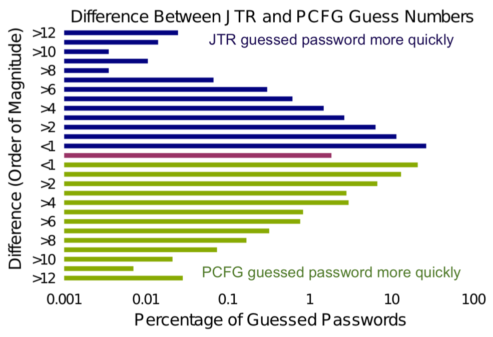


Figure 14: The % (log scale) of passwords guessed by JTR or PCFG whose guess numbers differed by a given order of magnitude. e.g., the blue > 6 bar represents passwords guessed by JTR more than 6, but no more than 7, orders of magnitude more quickly than by PCFG.

cutoff for each approach in a pair. Because Hashcat's early guesses were often unsuccessful, we exclude it from this analysis. Passwords not guessed by the guess cutoff were assigned a guess number one past the cutoff, lower-bounding differences between passwords guessed by one approach but not the other. For each password, we calculated the log_{10} of the ratio between guess numbers in the two approaches. For example, *iceman1232* was guess 595,300,840 for JTR and 61,554,045 for Markov, a 0.985 order of magnitude difference.

Among passwords guessed by JTR, PCFG, or both, 51% of passwords had guess numbers differing by more than an order of magnitude between approaches, indicating large variations in the resulting security conclusions. Alarmingly, some passwords had guess numbers differing by over 12 orders of magnitude (Figure 14). For example, *P@ssw0rd!* took JTR only 801 Complex guesses, yet PCFG never guessed it in our tests. Similarly, *1q2w3e4r5t6y7u8i* was the 29th LongBasic JTR guess, yet it was not among the 10^{14} such guesses PCFG made. In contrast, PCFG guessed *Abc@1993* after 48,670 guesses and *12345678password* after 130,555 guesses. JTR never guessed either password.

We found similar results in the two other pairwise comparisons. Among passwords guessed by Markov, PCFG, or both, 41% of guess numbers differed by at least one order of magnitude. In an extreme example, the passwords *1qaz!QAZ* and *1q2w3e4r5t6y7u8i* were among the first few hundred Markov guesses, yet not guessed by PCFG's guess cutoff. Conversely, *unitedstatesofamerica* was among PCFG's first few dozen LongBasic guesses, yet never guessed by Markov. For 37% of passwords, JTR and Markov guess numbers differed by at least one order of magnitude. Markov was particularly strong at guessing long passwords with predictable patterns. For instance, *password123456789*, *1234567890123456*, and *qwertyuiopasdfgh* were among Markov's first thirty guesses, yet JTR did not guess any of them by its cutoff.

Actionable takeaways As researchers and system administrators ask questions about password strength, they must consider whether their choice of cracking approach biases the results. When evaluating the strength of a large, heterogeneous password set, any of Hashcat, JTR, Markov, or PCFG—if configured effectively—provide fairly similar answers to research questions. Nonetheless, we recommend the more conservative strategy of calculating guessability using Min_{auto}.

In contrast, guessability results per-password can differ by many orders of magnitude between approaches even using the same training data. To mitigate these differences, we again recommend Min_{auto} for the increasingly important tasks of providing precise feedback on password strength to users and system administrators.

5 Conclusion

We report on the first broad, scientific investigation of the vulnerability of different types of passwords to guessing by an expert attacker and numerous configurations of off-the-shelf, automated approaches frequently used by researchers. We instrument these approaches, including both adversarial tools and academic research prototypes, to enable precise, guess-by-guess comparisons among automated approaches and between them and the expert.

We find that running a single guessing algorithm, particularly in its out-of-the-box configuration, often yields a very poor estimate of password strength. However, using several such algorithms, well-configured and in parallel, can be a good proxy for passwords' vulnerability to an expert attacker. We also find that while coarse-grained research results targeting heterogeneous sets of passwords are somewhat robust to the choice of (well-configured) guessing algorithm, many other analyses are not. For example, investigations of the effect on password strength of password characteristics, such as the number of character classes and the use of character substitutions, can reach different conclusions depending on the algorithm underlying the strength metric.

Finally, we hope our investigation of the effectiveness of many configurations of popular guessing approaches will help facilitate more accurate and easily reproducible research in the passwords research community. To that end, we have created a Password Guessability Service [9] that enables researchers to submit plaintext passwords and receive guessability analyses like those presented in this paper. We particularly encourage researchers investigating password-cracking algorithms to contribute to this service to improve the comparability of experiments.

Acknowledgments

We thank the authors of Hashcat and John the Ripper, Matt Marx, Jerry Ma, Weining Yang, Ninghui Li, KoreLogic, Michael Stroucken, Jonathan Bees, Chuck Cranor, and Dustin Heywood. This research was supported in part by NSF grants DGE-0903659 and CNS-1116776, and by a gift from Microsoft Research. It was also conducted with government support awarded by DoD, Air Force OSR, via the NDSEG Fellowship, 32 CFR 168a.

References

[1] The "web2" file of english words. `http://www.bee-man.us/computer/grep/grep.htm#web2`, 2004.

[2] Bonneau, J. The Gawker hack: How a million passwords were lost. *Light Blue Touchpaper* Blog, December 2010. `http://www.lightbluetouchpaper.org/2010/12/15/the-gawker-hack-how-a-million-passwords-were-lost/`.

[3] Bonneau, J. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *Proc. IEEE Symp. Security & Privacy* (2012).

[4] Bonneau, J. Statistical metrics for individual password strength. In *Proc. WPS* (2012).

[5] Bonneau, J., Herley, C., van Oorschot, P. C., and Stajano, F. The quest to replace passwords: A framework for comparative evaluation of Web authentication schemes. In *Proc. IEEE Symp. Security & Privacy* (2012).

[6] Bonneau, J., and Xu, R. Of contraseñas, sysmawt, and mìmǎ: Character encoding issues for web passwords. In *Proc. W2SP* (2012).

[7] Brodkin, J. 10 (or so) of the worst passwords exposed by the LinkedIn hack. *Ars Technica*, June 2012.

[8] Burr, W. E., Dodson, D. F., and Polk, W. T. Electronic authentication guideline. Tech. rep., NIST, 2006.

[9] Carnegie Mellon University. Password guessability service. `https://pgs.ece.cmu.edu`, 2015.

[10] Castelluccia, C., Dürmuth, M., and Perito, D. Adaptive password-strength meters from Markov models. In *Proc. NDSS* (2012).

[11] Chou, H.-C., Lee, H.-C., Yu, H.-J., Lai, F.-P., Huang, K.-H., and Hsueh, C.-W. Password cracking based on learned patterns from disclosed passwords. *IJICIC* (2013).

[12] Chrysanthou, Y. Modern password cracking: A hands-on approach to creating an optimised and versatile attack. Master's thesis, Royal Holloway, University of London, 2013.

[13] curlyboi. Hashtopus. `http://hashtopus.nech.me/manual.html`, 2009-.

[14] Das, A., Bonneau, J., Caesar, M., Borisov, N., and Wang, X. The tangled web of password reuse. In *Proc. NDSS* (2014).

[15] de Carné de Carnavalet, X., and Mannan, M. From very weak to very strong: Analyzing password-strength meters. In *Proc. NDSS* (2014).

[16] Dell'Amico, M., Michiardi, P., and Roudier, Y. Password strength: An empirical analysis. In *Proc. INFOCOM* (2010).

[17] Dev, J. A. Usage of botnets for high speed md5 hash cracking. In *INTECH* (2013).

[18] Dürmuth, M., Angelstorf, F., Castelluccia, C., Perito, D., and Chaabane, A. OMEN: Faster password guessing using an ordered markov enumerator. In *Proc. ESSoS* (2015).

[19] Dürmuth, M., Chaabane, A., Perito, D., and Castelluccia, C. When privacy meets security: Leveraging personal information for password cracking. *CoRR* (2013).

[20] Fahl, S., Harbach, M., Acar, Y., and Smith, M. On The Ecological Validity of a Password Study. In *Proc. SOUPS* (2013).

[21] Forget, A., Chiasson, S., van Oorschot, P., and Biddle, R. Improving text passwords through persuasion. In *Proc. SOUPS* (2008).

[22] Goodin, D. Hackers expose 453,000 credentials allegedly taken from Yahoo service. *Ars Technica*, July 2012.

[23] Goodin, D. Anatomy of a hack: How crackers ransack passwords like "qeadzcwrsfxv1331". *Ars Technica*, May 2013.

[24] Goodin, D. "thereisnofatebutwhatwemake"-turbo-charged cracking comes to long passwords. *Ars Technica*, August 2013.

[25] Goodin, D. Meet wordhound, the tool that puts a personal touch on password cracking. *Ars Technica*, August 2014.

[26] Google. Web 1T 5-gram version 1, 2006. `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2006T13`.

[27] haible. gperf. `https://www.gnu.org/software/gperf/`, 2010-.

[28] Haque, S. T., Wright, M., and Scielzo, S. A study of user password strategy for multiple accounts. In *Proc. CODASPY* (2013).

[29] Imperva. Consumer password worst practices, 2010. `http://www.imperva.com/docs/WP_Consumer_Password_Worst_Practices.pdf`.

[30] InsidePro. PasswordsPro. `http://www.insidepro.com/eng/passwordspro.shtml`, 2003-.

[31] Kelley, P. G., Komanduri, S., Mazurek, M. L., Shay, R., Vidas, T., Bauer, L., Christin, N., Cranor, L. F., and Lopez, J. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *Proc. IEEE Symp. Security & Privacy* (2012).

[32] Komanduri, S. *Modeling the adversary to evaluate password strengh with limited samples*. PhD thesis, Carnegie Mellon University, 2015.

[33] Komanduri, S., Shay, R., Cranor, L. F., Herley, C., and Schechter, S. Telepathwords: Preventing weak passwords by reading users' minds. In *Proc. USENIX Security* (2014).

[34] Komanduri, S., Shay, R., Kelley, P. G., Mazurek, M. L., Bauer, L., Christin, N., Cranor, L. F., and Egelman, S. Of passwords and people: Measuring the effect of password-composition policies. In *Proc. CHI* (2011).

[35] KoreLogic. "Crack Me If You Can" - DEFCON 2010. `http://contest-2010.korelogic.com/rules.html`, 2010-.

[36] KoreLogic. "Crack Me If You Can" - DEFCON 2013. `http://contest-2013.korelogic.com`, 2010-.

[37] KoreLogic. Pathwell topologies. *KoreLogic Blog*, 2014. `https://blog.korelogic.com/blog/2014/04/04/pathwell_topologies`.

[38] KoreLogic. "Analytical Solutions: Password Recovery Service. `http://contest-2010.korelogic.com/prs.html`, 2015.

[39] Love, D. Apple on iCloud breach: It's not our fault hackers guessed celebrity passwords. *International Business Times*, September 2014.

[40] Ma, J., Yang, W., Luo, M., and Li, N. A study of probabilistic password models. In *Proc. IEEE Symp. Security & Privacy* (2014).

[41] Marechal, S. Automatic wordlist mangling rule generation. *Openwall Blog*, 2012. `http://www.openwall.com/presentations/Passwords12-Mangling-Rules-Generation/`.

[42] Mazurek, M. L., Komanduri, S., Vidas, T., Bauer, L., Christin, N., Cranor, L. F., Kelley, P. G., Shay, R., and Ur, B. Measuring password guessability for an entire university. In *Proc. CCS* (2013).

[43] Morris, R., and Thompson, K. Password security: A case history. *CACM 22*, 11 (1979).

[44] MWR InfoSecurity. MWR InfoSecurity, 2014. `https://www.mwrinfosecurity.com/`.

[45] Narayanan, A., and Shmatikov, V. Fast dictionary attacks on passwords using time-space tradeoff. In *Proc. CCS* (2005).

[46] Openwall. Wordlists. `http://download.openwall.net/pub/wordlists/`, 2015.

[47] Percival, C. Stronger key derivation via sequential memory-hard function. In *Proc. BSD Conference* (2009).

[48] Perlroth, N. Adobe hacking attack was bigger than previously thought. *The New York Times Bits Blog*, October 2013.

[49] Peslyak, A. John the Ripper. `http://www.openwall.com/john/`, 1996-.

[50] Peslyak, A. John the Ripper. `http://www.openwall.com/john/doc/MODES.shtml`, 1996-.

[51] PHDays. "Hash Runner"– Positive Hack Days. `http://2013.phdays.com/program/contests/`, 2013.

[52] Provos, N., and Mazieres, D. A future-adaptable password scheme. In *Proc. USENIX* (1999).

[53] Rao, A., Jha, B., and Kini, G. Effect of grammar on security of long passwords. In *Proc. CODASPY* (2013).

[54] Schneier, B. MySpace passwords aren't so dumb. *Wired*, December 2012.

[55] SCOWL. Spell checker oriented word lists. `http://wordlist.sourceforge.net`, 2015.

[56] Shay, R., Komanduri, S., Durity, A. L., Huh, P. S., Mazurek, M. L., Segreti, S. M., Ur, B., Bauer, L., Cranor, L. F., and Christin, N. Can long passwords be secure and usable? In *Proc. CHI* (2014).

[57] Steube, J. Hashcat. `https://hashcat.net/oclhashcat/`, 2009-.

[58] Steube, J. Mask Attack. `https://hashcat.net/wiki/doku.php?id=mask_attack`, 2009-.

[59] Steube, J. Rule-based Attack. `https://hashcat.net/wiki/doku.php?id=rule_based_attack`, 2009-.

[60] Stobert, E., and Biddle, R. The password life cycle: User behaviour in managing passwords. In *Proc. SOUPS* (2014).

[61] Stricture Group. Password auditing services. `http://stricture-group.com/services/password-audits.htm`.

[62] Trustwave. eHarmony password dump analysis, June 2012. `http://blog.spiderlabs.com/2012/06/eharmony-password-dump-analysis.html`.

[63] Trustwave. 2014 business password analysis. *Password Research*, 2014.

[64] Trustwave Spiderlabs. SpiderLabs/KoreLogic-Rules. `https://github.com/SpiderLabs/KoreLogic-Rules`, 2012.

[65] Ur, B., Kelly, P. G., Komanduri, S., Lee, J., Maass, M., Mazurek, M., Passaro, T., Shay, R., Vidas, T., Bauer, L., Christin, N., and Cranor, L. F. How does your password measure up? The effect of strength meters on password creation. In *Proc. USENIX Security* (August 2012).

[66] Ur, B., Noma, F., Bees, J., Segreti, S. M., Shay, R., Bauer, L., Christin, N., and Cranor, L. F. "i added '!' at the end to make it secure": Observing password creation in the lab. In *Proc. SOUPS* (2015).

[67] Vance, A. If your password is 123456, just make it hackme. New York Times, 2010.

[68] Veras, R., Collins, C., and Thorpe, J. On the semantic patterns of passwords and their security impact. In *Proc. NDSS* (2014).

[69] Weir, M., Aggarwal, S., Collins, M., and Stern, H. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *Proc. CCS* (2010).

[70] Weir, M., Aggarwal, S., Medeiros, B. d., and Glodek, B. Password cracking using probabilistic context-free grammars. In *Proc. IEEE Symp. Security & Privacy* (2009).

[71] Yazdi, S. H. Analyzing password strength and efficient password cracking. Master's thesis, The Florida State University, 2011.

[72] Zhang, Y., Monrose, F., and Reiter, M. K. The security of modern password expiration: An algorithmic framework and empirical analysis. In *Proc. CCS* (2010).

A Appendix

We provide additional measurements of how guessing approaches perform in different configurations. To support the ecological validity of our study, we also repeat analyses from the body of the paper on password sets leaked from RockYou and Yahoo. We provide these details in hopes of encouraging greater accuracy and reproducibility across measurements of password guessability.

A.1 Alternate PCFG Configurations

We tested four different PCFG configurations. As in the body of the paper, **PCFG** represents Komanduri's implementation of PCFG [32], which assigns letter strings probabilities based on their frequency in the training data and assigns unseen strings a non-zero probability. For consistency across approaches, we prepend all policy-compliant elements of the training data in lieu of enabling Komanduri's similar hybrid structures [32].

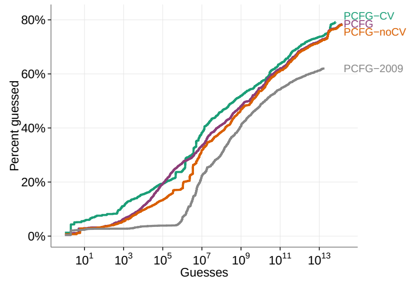
PCFG−noCV is the same as PCFG, but without the training data prepended. **PCFG−CV** is equivalent to PCFG−noCV except for using two-fold cross-validation. In each fold, we used half of the test passwords as additional training data, with a total weighting equal to the generic training data, as recommended by Kelley et al. [31]. **PCFG−2009** is our approximation of the original 2009 Weir et al. algorithm [70] in which alphabetic strings are assigned uniform probability and unseen terminals are a probability of zero.

As shown in Figure 15, prepending the training data and performing cross-validation both usually result in more efficient guessing, particularly for Long and LongBasic passwords. All three other configurations outperform the original PCFG−2009 implementation.

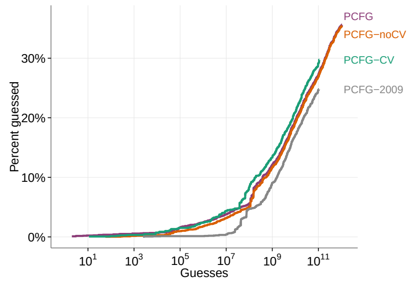
Figure 16 shows the guessability of the 350 passwords comprised only of digits across the Basic and LongBasic sets. Similar to the results for passwords of other common characteristics (Section 4.3.2), approaches differed. Of particular note is PCFG−2009, which plateaued at around 50% of such passwords guessed in fewer than 10 million guesses. Idiosyncratically, through 10^{14} guesses, it would never guess another password of this type because of the way it assigns probabilities.

A.2 Alternate JTR Configurations

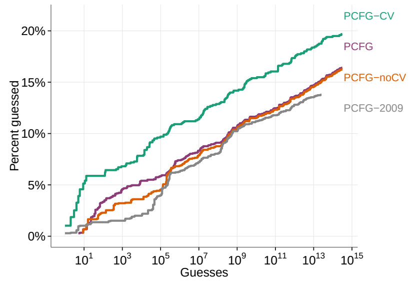
We next separately analyze the sets of JTR mangling rules we combined in the body of the paper. **JTR_stock** represents the 23 default rules that come with JTR. **JTR_SpiderLabs** represents 5,146 rules published by KoreLogic during the 2010 DEF CON "Crack Me If You Can" password-cracking contest [35], later reordered for guessing efficiency by Trustwave Spiderlabs [64].



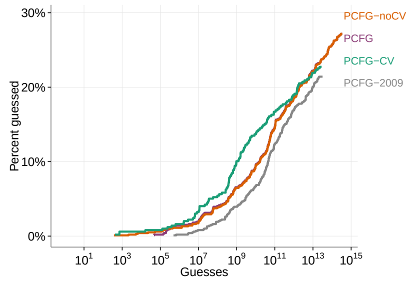
(a) Basic



(b) Complex



(c) LongBasic



(d) LongComplex

Figure 15: The guessing efficiency of the different PCFG configurations we tested.

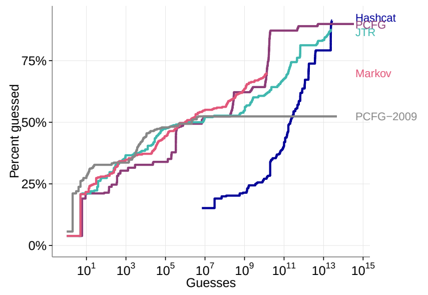


Figure 16: Guessing efficiency for the 350 Basic and LongBasic passwords composed entirely of digits.

As described in Section 3.3, our standard JTR configuration used JTR_stock followed by JTR_SpiderLabs. In isolation (Figure 17), JTR_stock rules were far more efficient on a guess-by-guess basis than JTR_SpiderLabs. Unfortunately, however, they quickly ran out of guesses. We exhausted JTR_stock in making fewer than 10^9 guesses for Basic passwords. More crucially, we made fewer than 10^5 guesses that were valid Complex passwords before exhausting these rules. Thus, any analysis of passwords that uses only the stock rules will vastly underestimate the guessability of passwords that contain (or are required to have) many different character classes.

The sharp jumps in the proportion of Complex and LongComplex passwords guessed by JTR_SpiderLabs result from one group of 13 rules. These rules capitalize the first letter, append digits, append special characters, and append both digits and special characters.

A.3 Alternate Hashcat Configurations

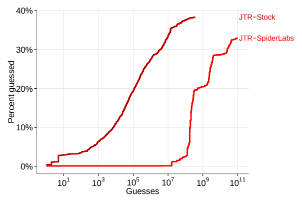
We tested eight Hashcat configurations and chose the one that best combined efficient early guessing with successfully continuing to guess passwords into the trillions of guesses. These configurations consist of four different sets of mangling rules, each with two different wordlists. The smaller wordlist was the same one we used in all other tests (Section 3.2). The larger wordlist augmented the same wordlist with all InsidePro wordlists[1] in descending frequency order and with duplicates removed.

Our four sets of mangling rules are the following:

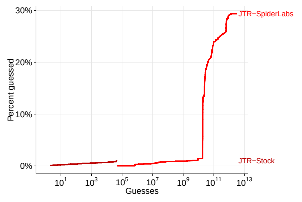
Hashcat_best64: Although Hashcat does not have a default set of mangling rules, the Best64 mangling rules are often used analogously to JTR's stock rules.

Hashcat_generated2: Hashcat comes with a second set of mangling rules, "generated2." This set comprises 65,536 rules. Dustin Heywood of ATB Financial created them by randomly generating and then testing hundreds of millions of mangling rules over 6 months (2013-2014) on a 42-GPU cluster. The rules were optimized by Hashcat developers by removing semantic equivalents.

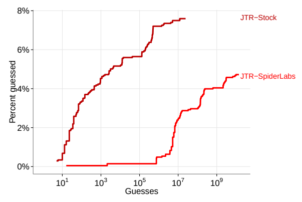
[1]`http://www.insidepro.com/dictionaries.php`



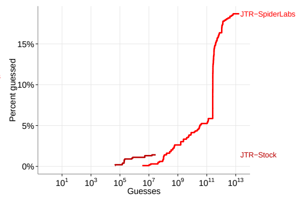
(a) Basic



(b) Complex

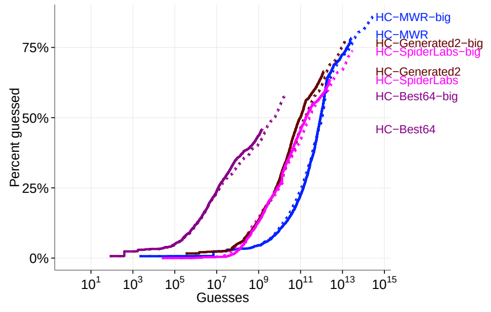


(c) LongBasic

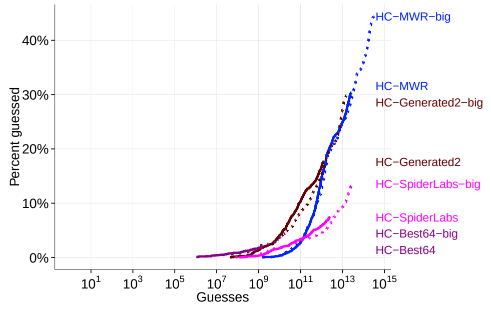


(d) LongComplex

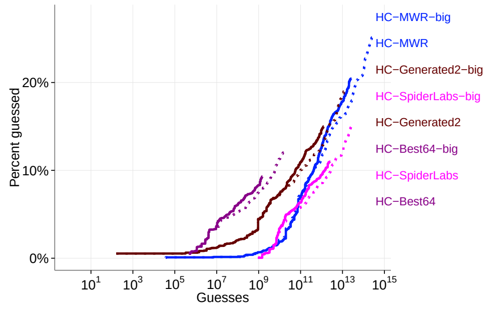
Figure 17: The guessing efficiency of JTR rules.



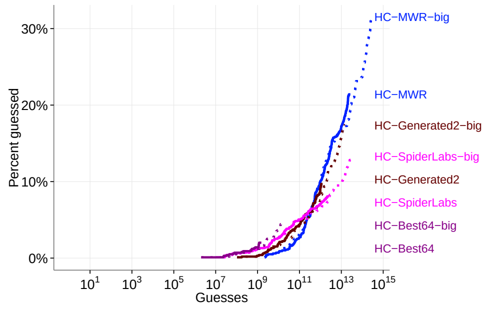
(a) Basic



(b) Complex



(c) LongBasic



(d) LongComplex

Figure 18: The guessing efficiency of Hashcat using four different sets of mangling rules. We tested each set with the wordlist used elsewhere in this paper, as well as a larger (**-big**) wordlist adding the InsidePro dictionaries.

Hashcat_SpiderLabs: We performed a manual translation to Hashcat of the SpiderLabs JTR rules (Section 3), which entailed removing clauses mandating minimum criteria; such rules are not permitted in oclHashcat.
Hashcat_MWR: We collaborated with with Matt Marx of MWR InfoSecurity to obtain the set of 1.4 million mangling rules he uses for password auditing [25, 44]. Following his suggestion, we augmented these rules with the aforementioned SpiderLabs rules.

Using the smaller wordlist, we exhausted all four sets of mangling rules. With the larger wordlist, we did not exhaust any set of rules. The curves in Figure 18 that use this larger dictionary have **-big** appended to the name and are graphed with dotted, rather than solid, lines.

We present the results of these eight configurations in Figure 18. True to their name, the Hashcat_best64 rules were the most efficient at guessing passwords. Unfortunately, they ran out of guesses using the smaller wordlist after only 10^9 guesses. For Complex and LongComplex passwords, Hashcat_best64 therefore guesses only a fraction of the number possible using the other sets of mangling rules, albeit in far fewer guesses. While not the most efficient guess-by-guess, the Hashcat_MWR rules eventually guessed the largest proportion of the different sets, most notably the Complex and LongComplex sets.

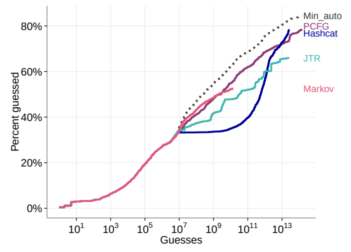
A.4 Ecological validity

To better understand how well our password sets, which we collected for research studies, compare to real plaintext passwords revealed in major password leaks, we compared the efficiency of the four automated cracking approaches in guessing Basic passwords, as well as the following two comparable sets of leaked passwords:
Basic$_{rockyou}$: 15,000 passwords randomly sampled from those containing 8+ characters in the RockYou gaming website leak of more than 32 million passwords [67]
Basic$_{yahoo}$: 15,000 passwords randomly sampled from those containing 8+ characters in the Yahoo! Voices leak of more than 450,000 passwords [22]

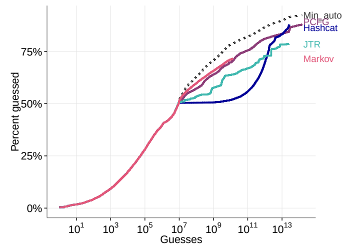
We found a high degree of similarity in the guessability of the Basic passwords collected for research and the leaked passwords. As shown in Figure 19, the four automated cracking approaches followed similar curves across the research passwords and the leaked passwords.

This similar guessability is notable because our analyses depend on using passwords collected by researchers for two reasons. First, no major password leak has contained passwords contained under strict composition requirements. Furthermore, in contracting experienced humans to attack the passwords, it was important to have them attack passwords they had not previously examined or tried to guess. Presumably, these experienced analysts would already have examined all major password leaks.

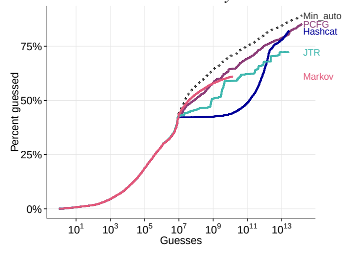
In the body of the paper, we reported how different approaches were impacted differently by the number of character classes contained in Basic passwords. When we repeated this analysis for Basic$_{rockyou}$ and Basic$_{yahoo}$ passwords, we found similar behavior (Figure 20). PCFG was more successful at guessing passwords containing two character classes, as opposed to only a single character class. PCFG only guesses strings that were found verbatim in its training data, which we hypothesize might be the cause of comparatively poor behavior for passwords of a single character class.



(a) Basic



(b) Basic$_{rockyou}$



(c) Basic$_{yahoo}$

Figure 19: The four automated cracking approaches targeting the Basic password set, 15,000 passwords sampled from the RockYou leak, and 15,000 passwords sampled from the Yahoo leak.

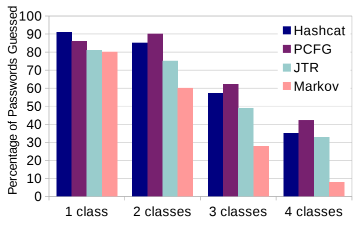


Figure 20: Combined percentage of Basic$_{rockyou}$ and Basic$_{yahoo}$ passwords each approach guessed by the number of character classes in the password.