

BIZ & IT —

# Anatomy of a hack: How crackers ransack passwords like “qeadzcvrsfxv1331”

For Ars, three crackers have at 16,000+ hashed passcodes—with 90 percent success.

DAN GOODIN - 5/28/2013, 6:30 AM



Aurich Lawson

Thanks to the XKCD comic, every password cracking word list in the world probably has correcthorsebatterystaple in it already.

In March, readers followed along as [Nate Anderson](#), Ars deputy editor and a self-admitted newbie to password cracking, downloaded a list of more than 16,000 cryptographically hashed passcodes. Within a few hours, [he deciphered almost half of them](#). The moral of the story: if a reporter with zero training in the ancient art of password cracking can achieve such results, imagine what more seasoned attackers can do.

Imagine no more. We asked three cracking experts to attack the same list Anderson targeted and recount the results in all their color and technical detail [Iron Chef](#) style. The results, to say the least, were eye opening because they show how quickly even long passwords with letters, numbers, and symbols can be discovered.

The list contained 16,449 passwords converted into hashes using the [MD5 cryptographic hash function](#). Security-conscious websites never store passwords in plaintext. Instead, they work only with these so-called one-way hashes, which are incapable of being mathematically converted back into the

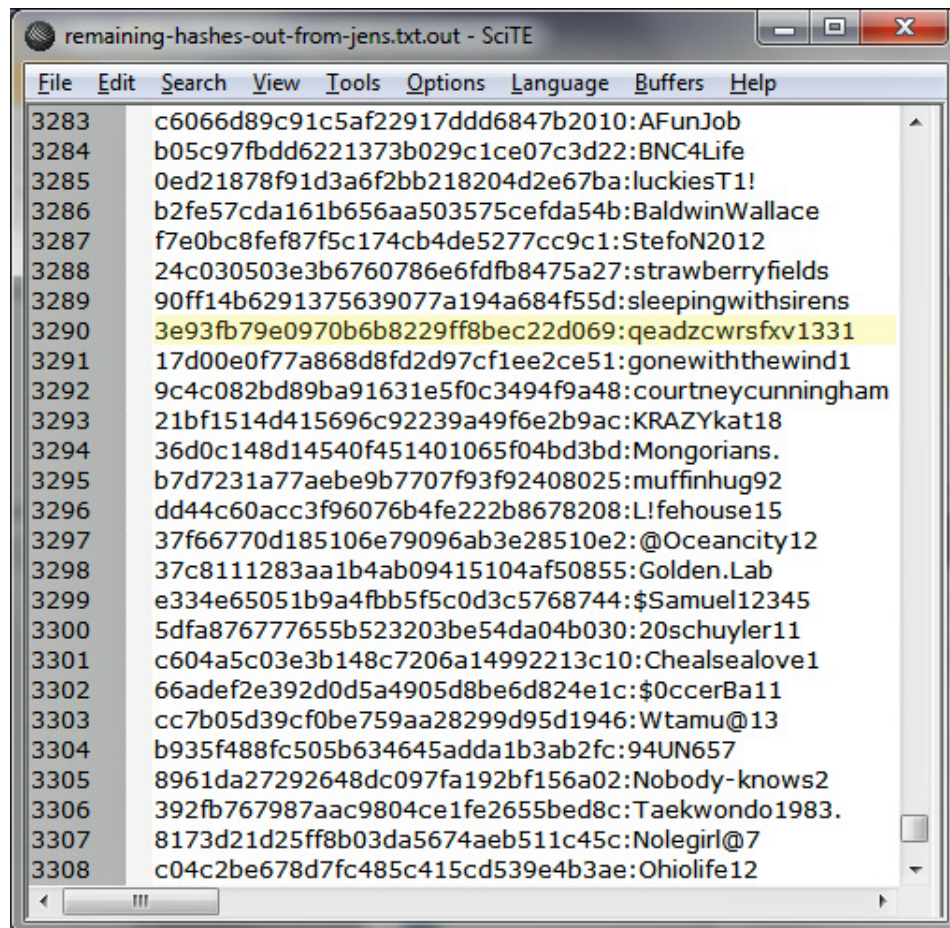
letters, numbers, and symbols originally chosen by the user. In the event of a security breach that exposes the password data, an attacker still must painstakingly guess the plaintext for each hash—for instance, they must guess that "5f4dcc3b5aa765d61d8327deb882cf99" and "7c6a180b36896a0a8c02787eeafb0e4c" are the MD5 hashes for "password" and "password1" respectively. (For more details on password hashing, see the earlier Ars feature "[Why passwords have never been weaker—and crackers have never been stronger.](#)")

While Anderson's 47-percent success rate is impressive, it's miniscule when compared to what real crackers can do, as Anderson himself made clear. To prove the point, we gave them the same list and watched over their shoulders as they tore it to shreds. To put it mildly, they didn't disappoint. Even the least successful cracker of our trio—who used the least amount of hardware, devoted only one hour, used a tiny word list, *and* conducted an interview throughout the process—was able to decipher 62 percent of the passwords. Our top cracker snagged 90 percent of them.

The Ars password team included a developer of cracking software, a security consultant, and an anonymous cracker. The most thorough of the three cracks was carried out by [Jeremi Gosney](#), a password expert with [Stricture Consulting Group](#). Using a commodity computer with a single [AMD Radeon 7970 graphics card](#), it took him 20 hours to crack 14,734 of the hashes, a 90-percent success rate. Jens Steube, the lead developer behind [oclHashcat-plus](#), achieved impressive results as well. (oclHashcat-plus is the freely available password-cracking software both Anderson and all crackers in this article used.) Steube unscrambled 13,486 hashes (82 percent) in a little more than one hour, using a slightly more powerful machine that contained two [AMD Radeon 6990 graphics cards](#). A third cracker who goes by the moniker radix deciphered 62 percent of the hashes using a computer with a single 7970 card—also in about one hour. And he probably would have cracked more had he not been peppered with questions throughout the exercise.

The list of "plains," as many crackers refer to deciphered hashes, contains the usual list of commonly used passcodes that are found in virtually every breach involving consumer websites. "123456," "1234567," and "password" are there, as is "letmein," "Destiny21," and "pizzapizza." Passwords of this ilk are hopelessly weak. Despite the additional tweaking, "p@\$s\$word," "123456789j," "letmein1!," and "LETMEin3" are equally awful. But sprinkled among the overused and easily cracked passcodes in the leaked list are some that many readers might assume are relatively secure. ":LOL1313le" is in there, as are "Coneyisland9/," "momof3g8kids," "1368555av," "n3xtb1gth1ng," "qeadzcxrsfxv1331," "m27bufford," "J21.redskin," "Garrett1993\*," and "Oscar+emmy2."





A screenshot showing a small sampling of cracked passwords.

As big as the word lists that all three crackers in this article wielded—close to 1 billion strong in the case of Gosney and Steube—none of them contained "Coneyisland9/," "momof3g8kids," or the more than 10,000 other plains that were revealed with just a few hours of effort. So how did they do it? The short answer boils down to two variables: the website's unfortunate and irresponsible use of MD5 and the use of non-randomized passwords by the account holders.

## Life in the fast lane

"These are terrible passwords," radix, who declined to give his real name, told Ars just a few minutes into run one of his hour-long cracking session. "There's probably not a complexity requirement for them. The hashing alone being MD5 tells me that they really don't care about their passwords too much, so it's probably some pre-generated site."

Like SHA1, SHA3, and most other algorithms, MD5 was designed to convert plaintext into hashes, also known as "message digests," quickly and with a minimal amount of computation. That works in the favor of crackers. Armed with a single graphics processor, they can cycle through more than eight billion password combinations each second when attacking "fast" hashes. By contrast, algorithms specifically designed to protect passwords require significantly more time and computation. For instance, the SHA512crypt function included by default in Mac OS X and most Unix-based operating systems passes text through 5,000 hashing iterations. This hurdle would limit the same one-GPU cracking system to slightly less than 2,000 guesses per second. Examples of other similarly "slow" hashing algorithms include [bcrypt](#), [scrypt](#), and [PBKDF2](#).

The other variable was the account holders' decision to use memorable words. The characteristics that made "momof3g8kids" and "Oscar+emmy2" easy to remember are precisely the things that

allowed them to be cracked. Their basic components—"mom," "kids," "oscar," "emmy," and numbers—are a core part of even basic password-cracking lists. The increasing power of hardware and specialized software makes it trivial for crackers to combine these ingredients in literally billions of slightly different permutations. Unless the user takes great care, passwords that are easy to remember are sitting ducks in the hands of crackers.

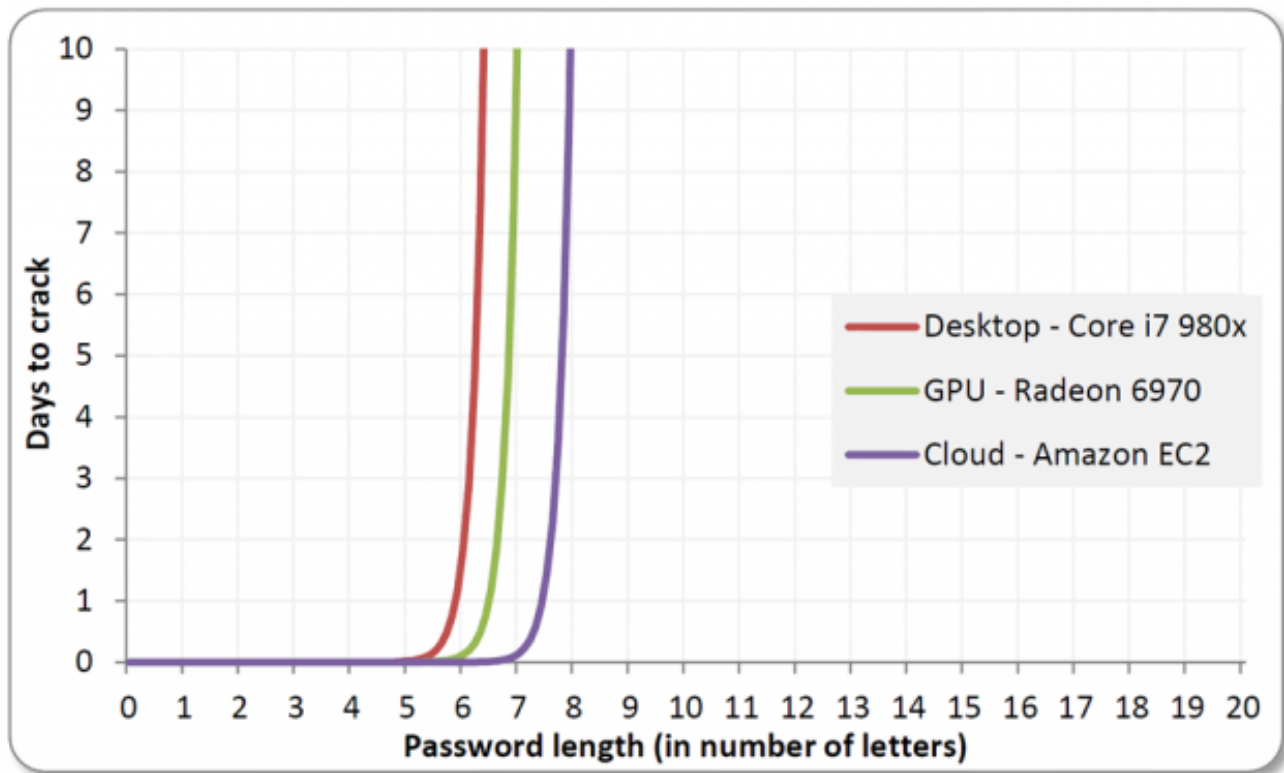
What's more, like the other two crackers profiled in this article, radix didn't know where the password list was taken from, eliminating one of the key techniques crackers use when deciphering leaked hashes. "If I knew the site, I would go there and find out what the requirements are," he said. The information would have allowed radix to craft custom rule sets targeted at the specific hashes he was trying to crack.

## Anatomy of a crack

The longer answer to how these relatively stronger passwords were revealed requires comparing and contrasting the approaches of the three crackers. Because their equipment and the amount of time they devoted to the exercise differed, readers shouldn't assume one cracker's technique was superior to those of the others. That said, all three cracks resembled video games where each successive level is considerably harder than the last. The first stage of each attack typically cracked in excess of 50 percent of the hashes, with each stage that came later cracking smaller and smaller percentages. By the time they got to the latest rounds, they considered themselves lucky to get more than a few hundred plains.

True to that pattern, Gosney's first stage cracked 10,233 hashes, or 62 percent of the leaked list, in just 16 minutes. It started with a brute-force crack for all passwords containing one to six characters, meaning his computer tried every possible combination starting with "a" and ending with "/////." Because guesses have a maximum length of six and are comprised of 95 characters—that's 26 lower-case letters, 26 upper-case letters, 10 digits, and 33 symbols—there are a manageable number of total guesses. This is calculated by adding the sum of  $95^6 + 95^5 + 95^4 + 95^3 + 95^2 + 95$ . It took him just two minutes and 32 seconds to complete the round, and it yielded the first 1,316 plains of the exercise.

Beyond a length of six, however, Gosney was highly selective about the types of brute-force attacks he tried. That's because of the exponentially increasing number of guesses each additional character creates. While it took only hours to brute-force all passwords from one to six characters, it would have taken Gosney days, weeks, or even years to brute-force longer passwords. Robert Graham, the CEO of Errata Security who has calculated the requirements, refers to this limitation as the "exponential wall of brute-force cracking."



**Enlarge** / Brute-force cracks work well against shorter passwords. The technique can take days or months for longer passcodes, even when using Amazon's cloud-based EC2 service.

Recognizing these limits, Gosney next brute-force cracked all passwords of length seven or eight that contained only lower letters. That significantly reduced the time required and still cracked 1,618 hashes. He tried all passwords of length seven or eight that contained only upper letters to reveal another 708 plains. Because their "keyspace" was the sum of  $26^8 + 26^7$ , each of these steps was completed in 41 seconds. Next, he brute-forced all passwords made up solely of numbers from one to 12 digits long. It cracked 312 passcodes and took him three minutes and 21 seconds.

It was only then that Gosney turned to his word lists, which he has spent years fine tuning. Augmenting the lists with the "best64" rule set built into Hashcat, he was able to crack 6,228 hashes in just nine minutes and four seconds. To complete stage one, he ran all the plains he had just captured in the previous rounds through a different rule set known as "d3ad0ne" (named after its **creator** who is a recognized password expert). It took one second to complete and revealed 51 more plains.

"Normally I start by brute-forcing all characters from length one to length six because even on a single GPU, this attack completes nearly instantly with fast hashes," Gosney explained in an e-mail. He continued:

And because I can brute-force this really quickly, I have all of my wordlists filtered to only include words that are at least six chars long. This helps to save disk space and also speeds up wordlist-based attacks. Same thing with digits. I can just brute-force numerical passwords very quickly, so there are no digits in any of my wordlists. Then I go straight to my wordlists + best64.rule since those are the most probable patterns, and larger rule sets take much

longer to run. Our goal is to find the most plains in the least amount of time, so we want to find as much low-hanging fruit as possible first.

Cracking the weakest passwords first is especially helpful when hashes contain cryptographic salt. Originally devised to thwart rainbow tables and other types of precomputed techniques, salting appends random characters to each password before it is hashed. Besides defeating rainbow tables, salting slows down brute-force and dictionary attacks because hashes must be cracked one at a time rather than all of them at once.

But the thing about salting is this: it slows down cracking only by a multiple of the number of unique salts in a given list. That means the benefit of salting diminishes with each cracked hash. By cracking the weakest passwords as quickly as possible first (an optimization offered by Hashcat) crackers can greatly diminish the minimal amount of protection salting might provide against cracking. Of course, none of this applies in this exercise since the leaked MD5 wasn't salted.

With 10,233 hashes cracked in stage one, it was time for stage two, which consisted of a series of **hybrid attacks**. True to the video game analogy mentioned earlier, this second stage of attacks took considerably longer than the first one and recovered considerably fewer plains—to be exact, five hours and 12 minutes produced 2,702 passwords.

As the name implies, a hybrid attack marries a dictionary attack with a brute-force attack, a combination that greatly expands the reach of a well-honed word list while keeping the keyspace to a manageable length. The first round of this stage appended all possible two-characters strings containing digits or symbols to the end of each word in his dictionary. It recovered 585 plains and took 11 minutes and 25 seconds to run. Round two appended all possible three-character strings containing digits or symbols. It cracked 527 hashes and required 58 minutes to complete. The third round, which appended all four-digit number strings, took 25 minutes and recovered 435 plains. Round four appended all possible strings containing three lower-case letters and digits and acquired 451 more passwords.

As fruitful as these attacks were, Gosney said they were handicapped by his use of a single graphics card for this exercise.

"For example, you'll notice that when I was doing hybrid attacks, I appended 2-3 digits/special but then only did digits with length 4," he explained. "This is because doing digits/special for length 4 would have taken a really long time with just one GPU, so I skipped it. Same with when I started appending lower alpha/digits, I only did length 3 because length 4 would have taken too long with just one GPU."

No doubt, Gosney could have attacked much larger keyspaces had he used the **monster 25-GPU cluster he unveiled in December**. Because the graphics cards in the five-server system scale almost linearly, it's able to harness almost all of their combined power. As a result, it can achieve 350 billion guesses per second when cracking password hashes generated by Microsoft's NTLM algorithm. And it could generate similar results when going up against MD5 and other fast hash functions.

The remaining hybrid attacks in stage two continued in the same vein. By the time it was completed, he had cracked a total of 12,935 hashes, or 78.6 percent of the list, and had spent a total of just 5 hours and 28 minutes doing it.

One of the things Gosney and other crackers have found is that passwords for a particular site are remarkably similar, despite being generated by users who have never met each other. After cracking such a large percentage of hashes from this unknown site, the next step was to analyze the plains and mimic the patterns when attempting to guess the remaining passwords. The result is a series of statistically generated brute-force attacks based on a mathematical system known as **Markov chains**. Hashcat makes it simple to implement this method. By looking at the list of passwords that already have been cracked, it performs probabilistically ordered, per-position brute-force attacks. Gosney thinks of it as an "intelligent brute-force" that uses statistics to drastically limit the keyspace.

Where a classic brute-force tries "aaa," "aab," "aac," and so on, a Markov attack makes highly educated guesses. It analyzes plains to determine where certain types of characters are likely to appear in a password. A Markov attack with a length of seven and a threshold of 65 tries all possible seven-character passwords with the 65 most likely characters for each position. It drops the keyspace of a classic brute-force from  $95^7$  to  $65^7$ , a benefit that saves an attacker about four hours. And since passwords show surprising uniformity when it comes to the types of characters used in each position—in general, capital letters come at the beginning, lower-case letters come in the middle, and symbols and numbers come at the end—Markov attacks are able crack almost as many passwords as a straight brute-force.

"This is where your attack plan deviates from the standard and becomes unique, because now you're doing site-specific attacks," Gosney said. "From there, if you start hitting upon any interesting patterns, you just start chasing those patterns down the rabbit hole. Once you've fully exploited one pattern you move on to the next."

In all, it took Gosney 14 hours and 59 minutes to complete this third stage, which besides Markov attacks included several other custom wordlists combined with rules. Providing further evidence of the law of diminishing returns that dictates password cracking, it yielded 1,699 more passwords. It's interesting to note that the increasing difficulty is experienced even within this last step itself. It took about three hours to cover the first 962 plains in this stage and 12 hours to get the remaining 737.

The other two password experts who cracked this list used many of the same techniques and methods, although not in the same sequence and with vastly different tools. The only wordlist used by radix, for example, came directly from the **2009 breach of online games service RockYou**. Because the SQL-injection hack exposed more than 14 million unique passwords in plaintext, the list represents the largest corpus of real-world passwords ever to be made public. radix has a much bigger custom-compiled dictionary, but like a magician who doesn't want to reveal the secret behind a trick, he kept it under wraps during this exercise.

## Killing hashes

Like Nate Anderson's foray into password cracking, radix was able to crack 4,900 of the passwords, nearly 30 percent of the haul, solely by using the RockYou list. He then took the same list, cut the last four characters off each of the words, and appended every possible four-digit number to the end. Hashcat told him it would take two hours to complete, which was longer than he wanted to spend. Even after terminating the run two after 20 minutes, he had cracked 2,136 more passcodes. radix then tried brute-forcing all numbers, starting with a single digit, then two digits, then three digits, and so on (259 additional plains recovered).



He seemed to choose techniques for his additional runs almost at random. But in reality, it was a combination of experience, intuition, and possibly a little luck.

"It's all about analysis, gut feelings, and maybe a little magic," he said. "Identify a pattern, run a mask, put recovered passes in a new dict, run again with rules, identify a new pattern, etc. If you know the source of the hashes, you scrape the company website to make a list of words that pertain to that specific field of business and then manipulate it until you are happy with your results."

He then ran the 7,295 plains he recovered so far through PACK, short for the **Password Analysis and Cracking Toolkit** (developed by password expert Peter Kacherginsky), and noticed some distinct patterns. A third of them contained eight characters, 19 percent contained nine characters, and 16 percent contained six characters. PACK also reported that 69 percent of the plains were "stringdigit" meaning a string of letters or symbols that ended with numbers. He also noticed that 62 percent of the recovered passwords were classified as "loweralphanum," meaning they consisted solely of lower-case letters and numbers.

This information gave him fodder for his next series of attacks. In run 4, he ran a **mask attack**. This is similar to the hybrid attack mentioned earlier, and it brings much of the benefit of a brute-force attack while drastically reducing the time it takes to run it. The first one tried all possible combinations of lower-case letters and numbers, from one to six characters long (341 more plains recovered). The next step would have been to try all combinations of lower-case letters and numbers with a length of eight. But that would have required more time than radix was willing to spend. He then considered trying all passwords with a length of eight that contained only lower-case letters. Because the attack excludes upper case letters, the search space was manageable,  $26^8$  instead of  $52^8$ . With radix's machine, that was the difference between spending a little more than one minute and six hours respectively. The lower threshold was still more time than he wanted to spend, so he skipped that step too.

So radix then shifted his strategy and used some of the rule sets built into Hashcat. One of them allows Hashcat to try a random combination of 5,120 rules, which can be anything from swapping each "e" with a "3," pulling the first character off each word, or adding a digit between each character. In just 38 seconds the technique recovered 1,940 more passwords.

"That's the thrill of it," he said. "It's kind of like hunting, but you're not killing animals. You're killing hashes. It's like the ultimate hide and seek." Then acknowledging the dark side of password cracking, he added: "If you're on the slightly less moral side of it, it has huge implications."

Steube also cracked the list of leaked hashes with aplomb. While the total number of words in his custom dictionaries is much larger, he prefers to work with a "dict" of just 111 million words and pull out the additional ammunition only when a specific job calls for it. The words are ordered from most to least commonly used. That way, a particular run will crack the majority of the hashes early on and then slowly taper off. "I wanted it to behave like that so I can stop when things get slower," he explained.

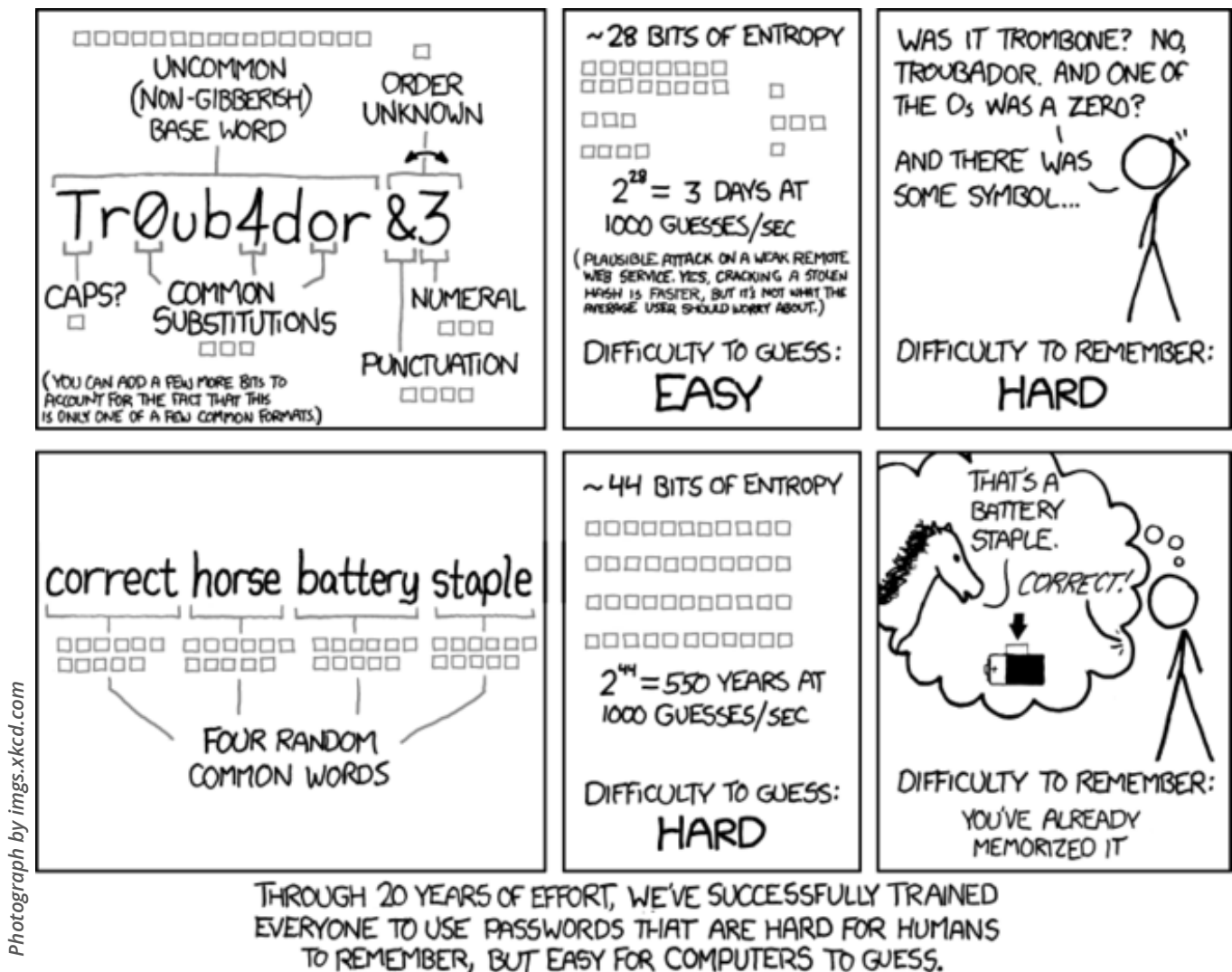
Early in the process, Steube couldn't help remarking when he noticed one of the plains he had recovered was "momof3g8kids."

"This was some logic that the user had," Steube observed. "But we didn't know about the logic. By doing hybrid attacks, I'm getting new ideas about how people build new [password] patterns. This is why I'm always watching outputs."

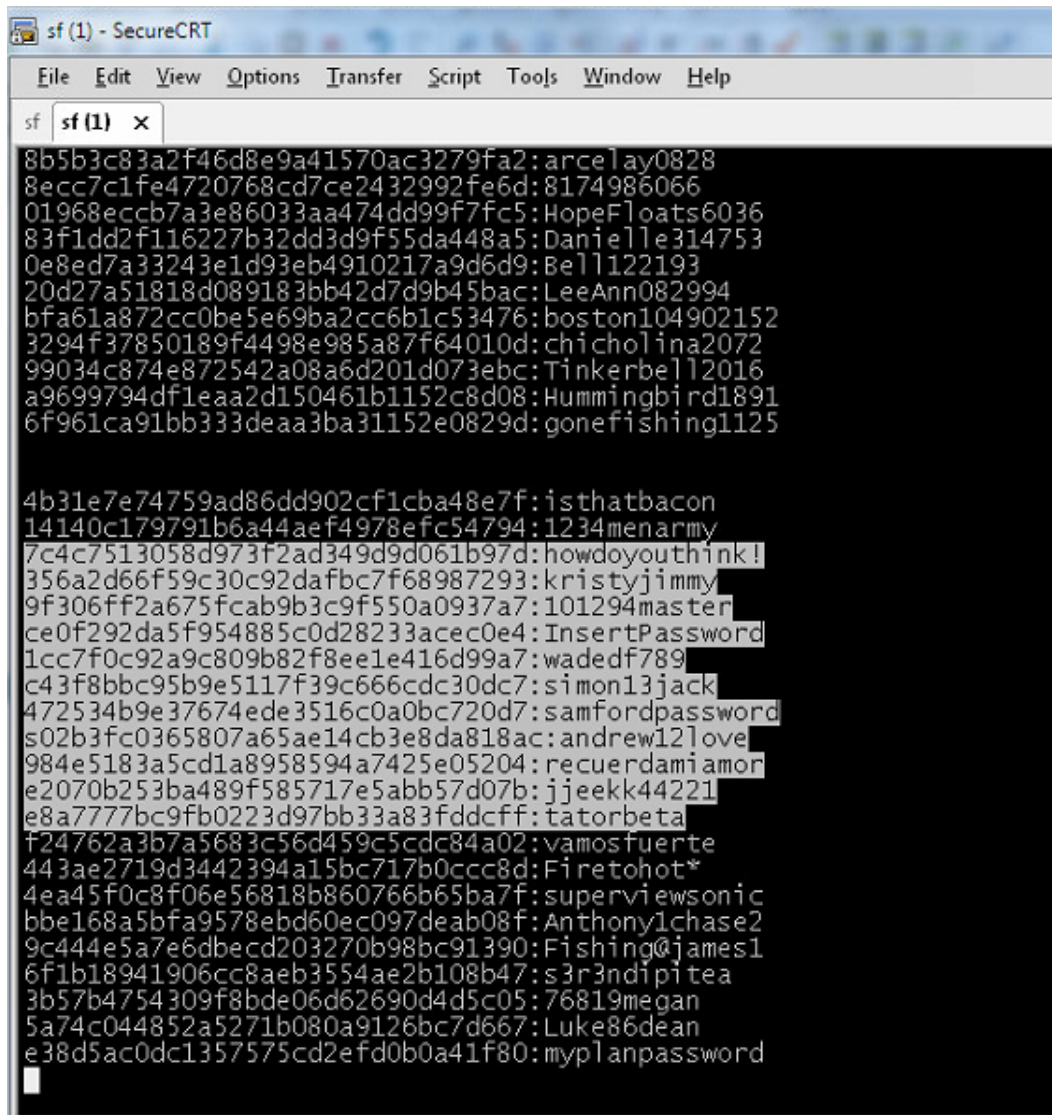


The specific type of hybrid attack that cracked that password is known as a **combinator attack**. It combines each word in a dictionary with every other word in the dictionary. Because these attacks are capable of generating a huge number of guesses—the square of the number of words in the dict—crackers often work with smaller word lists or simply terminate a run in progress once things start slowing down. Other times, they combine words from one big dictionary with words from a smaller one. Steube was able to crack "momof3g8kids" because he had "momof3g" in his 111 million dict and "8kids" in a smaller dict.

"The combinator attack got it! It's cool," he said. Then referring to the **oft-cited xkcd comic**, he added: "This is an answer to the batteryhorsestaple thing."



What was remarkable about all three cracking sessions were the types of plains that got revealed. They included passcodes such as "k1araj0hns0n," "Sh1a-labe0uf," "Apr!l221973," "Qbesancon321," "DG091101%," "@Yourmom69," "ilovetofunot," "windermere2313," "tmdmmj17," and "BandGeek2014." Also included in the list: "all of the lights" (yes, spaces are allowed on many sites), "i hate hackers," "allineedislove," "ilovemySister31," "iloveyousomuch," "Philippians4:13," "Philippians4:6-7," and "qeadzcxwrsfxv1331." "gonefishing1125" was another password Steube saw appear on his computer screen. Seconds after it was cracked, he noted, "You won't ever find it using brute force."



```
sf sf (1) x
8b5b3c83a2f46d8e9a41570ac3279fa2:arcelay0828
0ecc7c1fe4720768cd7ce2432992fe6d:8174986066
01968eccb7a3e86033aa474dd99f7fc5:HopeFloats6036
83f1dd2f116227b32dd3d9f55da448a5:Danielle314753
0e8ed7a33243e1d93eb4910217a9d6d9:Bell122193
20d27a51818d089183bb42d7d9b45bac:LeeAnn082994
bfa61a872cc0be5e69ba2cc6b1c53476:boston104902152
3294f37850189f4498e985a87f64010d:chicholina2072
99034c874e872542a08a6d201d073ebc:Tinkerbell2016
a9699794df1eaa2d150461b1152c8d08:Hummingbird1891
6f961ca91bb333deaa3ba31152e0829d:gonefishing1125

4b31e7e74759ad86dd902cf1cba48e7f:isthatbacon
14140c179791b6a44aef4978efc54794:1234menarmy
7c4c7513058d973f2ad349d9d061b97d:howdoyouthink!
356a2d66f59c30c92dafbc7f68987293:kristyjimmy
9f306ff2a675fcab9b3c9f550a0937a7:101294master
ce0f292da5f954885c0d28233acac0e4:InsertPassword
1cc7f0c92a9c809b82f8ee1e416d99a7:wadedf789
c43f8bbbc95b9e5117f39c666cdc30dc7:simon13jack
472534b9e37674ede3516c0a0bc720d7:samfordpassword
s02b3fc0365807a65ae14cb3e8da818ac:andrew12love
984e5183a5cd1a8958594a7425e05204:recuerdamiamor
e2070b253ba489f585717e5abb57d07b:jjeekk44221
e8a7777bc9fb0223d97bb33a83fddcff:tatorbeta
f24762a3b7a5683c56d459c5cdc84a02:vamosfuerte
443ae2719d3442394a15bc717b0ccc8d:Firetohot*
4ea45f0c8f06e56818b860766b65ba7f:superviewsonic
bbe168a5bfa9578ebd60ec097deab08f:Anthony1chase2
9c444e5a7e6dbecd203270b98bc91390:Fishing@james1
6f1b18941906cc8aeb3554ae2b108b47:s3r3ndipitea
3b57b4754309f8bde06d62690d4d5c05:76819megan
5a74c044852a5271b080a9126bc7d667:Luke86dean
e38d5ac0dc1357575cd2efd0b0a41f80:myplanpassword
```

The ease these three crackers had converting hashes into their underlying plaintext contrasts sharply with the assurances many websites issue when their password databases are breached. Last month, when daily coupons site LivingSocial disclosed a hack that exposed names, addresses, and password hashes for 50 million users, company executives downplayed the risk.

"Although your LivingSocial password would be difficult to decode, we want to take every precaution to ensure that your account is secure, so we are expiring your old password and requesting that you create a new one," CEO Tim O'Shaughnessy told customers.

In fact, there's almost nothing preventing crackers from deciphering the hashes. LivingSocial used the SHA1 algorithm, which as mentioned earlier is woefully inadequate for password hashing. He also mentioned that the hashes had been "salted," meaning a unique set of bits had been added to each users' plaintext password before it was hashed. It turns out that this measure did little to mitigate the potential threat. That's because salt is largely a protection against rainbow tables and other types of precomputed attacks, which almost no one ever uses in real-world cracks. The file sizes involved in rainbow attacks are so unwieldy that they fell out of vogue once GPU-based cracking became viable. (LivingSocial later said it's in the process of transitioning to the much more secure bcrypt function.)

Officials with Reputation.com, a service that helps people and companies manage negative search results, borrowed liberally from the same script when [disclosing their own password breach](#) a few days later. "Although it was highly unlikely that these passwords could ever be decrypted, we

immediately changed the password of every user to prevent any possible unauthorized account access," a company e-mail told customers.

Both companies should have said that, with the hashes exposed, users should presume their passwords are already known to the attackers. After all, cracks against consumer websites typically recover 60 percent to 90 percent of passcodes. Company officials also should have warned customers who used the same password on other sites to change them immediately.

To be fair, since both sites salted their hashes, the cracking process would have taken longer to complete against large numbers of hashes. But salting does nothing to slow down the cracking of a single hash and does little to slow down attacks on small numbers of hashes. This means that certain targeted individuals who used the hacked sites—for example, bank executives, celebrities, or other people of particular interest to the attackers—weren't protected at all by salting.

The prowess of these three crackers also underscores the need for end users to come up with better password hygiene. Many Fortune 500 companies tightly control the types of passwords employees are allowed to use to access e-mail and company networks, and they go a long way to dampen crackers' success.

"On the corporate side, its so different," radix said. "When I'm doing a password audit for a firm to make sure password policies are properly enforced, it's madness. You could go three days finding absolutely nothing."

Websites could go a long way to protect their customers if they enforced similar policies. In the coming days, Ars will publish a detailed primer on passwords managers. It will show how to use them to generate long, random passcodes that are unique to each site. Because these types of passwords can only be cracked by brute force, they are the hardest to recover. In the meantime, readers should take pains to make sure their passwords are a minimum of 11 characters, contain upper- and lower-case letters, and numbers, and aren't part of a pattern.

The ease these crackers had in recovering as many as 90 percent of the hashes they targeted from a real-world breach also exposes the inability many services experience when trying to measure the relative strength or weakness of various passwords. A recently launched site from chipmaker Intel asks users "How strong is your password?" and it estimated it would **take six years to crack the passcode "BandGeek2014"**. That estimate is laughable given that it was one of the first ones to fall at the hands of all three real-world crackers.

As **Ars explained recently**, the problem with password strength meters found on many websites is they use the total number of combinations required in a brute-force crack to gauge a password's strength. What the meters fail to account for is that the patterns people employ to make their passwords memorable frequently lead to passcodes that are highly susceptible to much more efficient types of attacks.

"You can see here that we have cracked 82 percent [of the passwords] in one hour," Steube said. "That means we have 13,000 humans who did not choose a good password." When academics and some websites gauge susceptibility to cracking, "they always assume the best possible passwords, when it's exactly the opposite. They choose the worst."

## Promoted Comments

**ringlord**

/ Smack-Fu Master, in training

**JUMP TO POST****ror** wrote:

"That means we have 13,000 humans who did not choose a good password."

how is Qbesancon321 not a "good" password? It could be strengthened by using a symbol, but sooner or later, Qbe\$@ncon321 won't be a good password either. (For all I know, that's already the case).

seems like not reusing the same password is more important than strength. even if I have a password that I believe to be "strong", it's still in my best interests to change my password once its hash has been released.

Capital at the beginning, digits at the end, all lowercase in the middle. It's a poor password because it's using a common pattern. That was the point of the article.

Everyone needs to use truly random passwords and, thus, a password manager.

Edit: and, no, doing that common symbol substitution makes it no stronger. Again, a common pattern.

38 posts | registered Oct 8, 2006

**Jon Brodtkin**

/ Senior IT reporter

**JUMP TO POST****RoninX** wrote:

Articles like this make me seriously consider using a password manager. However, I wonder if there are any risks to using a password manager. Maybe it's just a remnant of my initial visceral reaction when Microsoft first suggested this approach ("Just let Microsoft have all of your passwords!").

My questions are:

- 1) Do the password manager sites ever go down? And if they do, does this mean you are locked out of every single account you're managing?
- 2) What if the password manger site itself is hacked? Has this ever happened? What safeguards exist if the password manager database is stolen?

We're covering this in another article quite soon, but password managers generally keep working even when the password manager site is offline. In the case of 1Password, everything is local (while you sync to other computers via Dropbox). With LastPass, your passwords are stored in the cloud but there is a local cache that keeps you up and running in case LastPass were to ever go down.

In both cases, these companies don't actually store your master password, and all encryption and decryption takes place locally, so a hack of the password manager site alone wouldn't be enough to get your passwords.

If you use Dropbox to sync a password manager file, enabling two-factor authentication in Drobbox is obviously a good idea.

752 posts | registered Aug 29, 2011



epixoip

/ Password Expert

[JUMP TO POST](#)**technon** wrote:

How does a cracker know he correctly recovered the plaintext password from the hash? Does he run it back through the hash function and see if he gets the same result? Also how does he know which hash function he's dealing with anyway if all he has is a list of hashes?

given the target hash T and an input of P, using algorithm H, we simply check to see if  $H(P) == T$ .

most of the time, experience allows us to know what hash function we are dealing with at a quick glance. when that fails, use the source. if no source is available, reverse engineer it.

109 posts | registered Aug 21, 2012

epixoip

/ Password Expert

[JUMP TO POST](#)**robrob** wrote:

It took me forever to figure out the qeadzcxrsfxv1331 one, then I looked at my keyboard. I guess there's a limited number of patterns people will regularly use on one.

it's a simple keyboard walk. common enough to be directly in our dicts, but if it wasn't, a keyboard walk generator would have snagged it up quickly.

109 posts | registered Aug 21, 2012

ws3

/ Ars Tribunus Militum

[JUMP TO POST](#)**ChrisC** wrote:**topham** wrote:

The xkcd example is poor, the states level of entropy is "ideal" not "real". If I have a 100,000 word dictionary then each word represents about 10 bits of entropy, however people do not pick random words, they pick words they know and the average person uses about 1/5th of a dictionary.

If you had a dictionary with only 20000 words in it, would all the words you chose be in the dictionary? If the answer is yes then your entropy calculation is against the smaller dictionary.

Phrases and random words are NOT the answer to good passwords.

The oft cited xkcd comic has always set off the "is this really as good as it seems" itch in my brain.

I don't think the name space is as big as people think where pass phrases are concerned.

People are creatures of habit. And lazy ones at that. They are going to make up pass phrases from words in their working vocabulary - probably well less than 20,000 words for the average person.

I'd also not be surprised to learn that the vast majority are also likely to gravitate to using phrases that are 3-5 words with a word length of 3-5 characters. This probably trims the number of words the average person will build pass phrases from to well less than 10,000.

And they will use common substitutions to meet complexity rules. Start with a capital, end with a symbol, toss in a 0 for o, etc... Stuff like that.

I would submit they are also likely to favor sentence style pass phrases - Maryk3ptsheep. Further reducing the complexity of the attack.

Figuring out the math for this is on my to do list...when I have a few days spare time to research and do it. But suspect the average pass phrase is not that much more secure than the average password is today.

Choosing pass phrases from the 1000 (1e3) most common words:

2 words =  $1e6$  combinations /  $8e9$  per second = 0.000125 seconds to crack

3 words =  $1e9$  combinations /  $8e9$  per second = 0.125 second to crack

4 words =  $1e12$  combinations /  $8e9$  per second = 125 seconds to crack

5 words =  $1e15$  combinations /  $8e9$  per second = 125000 seconds = 34.72 hours to crack, etc.

Choosing from the 10000 (1e4) most common words yields:

2 words =>  $1e8$  => 0.0125 seconds

3 words =>  $1e12$  => 125 seconds

4 words =>  $1e16$  => 1.25e6 seconds = 12.46 days

5 words =>  $1e20$  => 1.25e10 seconds = 396.1 years

1505 posts | registered Nov 2, 2010

epixoip

/ Password Expert

JUMP TO POST

joppek wrote:

if i'm given a table of hashes to crack, wouldn't my first order of business be to identify the algorithm?

the vast majority of the time, this can be determined at a quick glance.

joppek wrote:

or, when someone gains access to a hash database, is it common for them to also gain information on what algorithm(s) are used to generate those hashes?

if the algorithm used is not obvious, then yes, most of the time whatever vulnerability was used to dump the password database can also be leveraged to see the exact algorithm used to store the passwords in the database.

joppek wrote:

while md5 and sha1 are both weak, would it be significantly more secure to use a number of iterations of both in some sequence?

yes, this is what pbkdf2, crypt(3), etc do.

109 posts | registered Aug 21, 2012

epixoip

/ Password Expert

[JUMP TO POST](#)

Abresh wrote:

trs8 wrote:

Brute forcing MD5 stopped being impressive years ago.

Agreed. No website worth it's salt that is truly worried about password cracking (mainly sites that deal in financial data) uses MD5 anymore. They have moved on to other, much stronger, cryptographic means.

Except they haven't. The reality is most financial sites are still using DES-based crypt schemes for legacy reasons. Most all small- to medium- size sites are still using raw MD5. And most all large sites are still using raw or salted MD5 or SHA1, and have no incentive to change until after they've been compromised. And even then, it's usually just a heartfelt apology followed by the reassurance that your passwords are safe because they were "cryptographically scrambled."

109 posts | registered Aug 21, 2012

epixoip

/ Password Expert

[JUMP TO POST](#)

jameskatt2 wrote:

What if the user's password was not in English though uses the Roman alphabet? For example, what if it was in Japanese. If the cryptographer did not know the language and their word lists are in English, would they not be at a disadvantage?

no. nobody has wordlists that are only in english. we have wordlists for most any language you can think of. it just so happens that this particular dump that Dan had us crack was all english. we crack passwords in foreign languages all the time, not only in unicode, but in native codepages as well.

109 posts | registered Aug 21, 2012

epixoip

/ Password Expert

[JUMP TO POST](#)

Anne Ominous wrote:

The assertion that "the thing about salting is this: it slows down cracking only by a multiple of the number of unique salts in a given list." is ludicrous. This presumes that salts are only used in the weakest possible way.

If salts are fairly random (as they should be), they add characters to the length of the password, thereby multiplying the computation time by a factor of approximately 82 for EACH character in the salt (and that's if it only contained standard keyboard characters, which would be silly). Salts can also be used in a more effective way, to not only add to a password but to also randomize it before encryption, which does not just reduce the utility of rainbow tables, it renders them COMPLETELY useless. The only method that is effective against this AT ALL is brute force.

This is of course given that the cracker does not have both the password and the salt, AND knowledge of how the two are used together. If they don't, and the passwords are salted well, the "encryption algorithm" could be ROT13 and the crackers still would not get it.

What on earth are you talking about?

First of all, please do not speak with such authority on topics you clearly know nothing about. It confuses people who are here to learn, makes yourself appear completely ignorant to those who know better, and frustrates those who are here trying to teach.

Second, the salts are always a known value. They are no more secret than the password hashes themselves. With most password hash functions, the salt is part of the hash string itself (see bcrypt, crypt(3), pbkdf2, etc.) With custom schemes, the salt is usually stored alongside the password hash in the same table.

Attempting to make the salts more secret than the hashes is nothing more than security through obscurity. Even if the salts were stored in a separate table, or even a separate database, the application we've exploited to dump the hashes has to have sufficient privileges to read the salt values, therefore we as attackers have sufficient privileges to read the salt values as well. Side note, this is also precisely why we do not store passwords using reversible encryption.

Last, you seem to be under the assumption that we do not know what algorithm we are dealing with when we dump a database, and this is absolutely false. If it is not obvious at first glance (and believe me, rot13 would certainly be obvious) and we cannot quickly guess the correct algorithm based on experience, then we can always use the source. We can either leverage the same vulnerability we used to dump the hashes to examine the webapp source and find the precise function used, or if the software is COTS we can simply obtain and analyze the software to determine the algorithm. And in the rarest of cases, if all of the above fails, we will simply reverse engineer the algorithm, which we are very good at doing.

109 posts | registered Aug 21, 2012

ChrisC

/ Ars Tribunus Militum

[JUMP TO POST](#)

**cromage** wrote:

Combinator attacks aren't an "answer" to xkcd, at least not yet.

In the example given, the number of guesses was "only" the square of 100 million, or  $10^{16}$ .

Four english words would be the the quad(?) of 100,000 (a lowball estimate to the number of words in the English language), or  $10^{20}$ ; it would take 10,000x the time.

If the combinator attack took an hour the xkcd combinator attack would take a year.

Now, if you really were paranoid and didn't care so much about the problem of passwords being hard to remember (let's say it's one you use every day) you could easily make things several times more complicated through substitutions, making a similar technique take 1,000 years instead...

The number of people who read this article and completely missed the key points of it is astounding.



The three subjects in the article only used pure brute force attacks where it made sense - i.e. where they could be done quickly. Beyond that they used attacks based on human tendencies and pattern analysis that essentially targeted the attacks where they were likely to produce the best results.

They subjects in the article weren't trying to get my, truly random, 16 character LastPass vault password. They were going for the low hanging fruit, so to speak.

The same thing that makes the majority of passwords weak - human behavior - will also make most pass phrases weak.

The pass phrases typical users create:

- will contain usage patterns that can be detected and exploited.
- won't use the entire name space, The pool of words used will likely be less than 10,000 instead of greater than 100,000. Let's face it, most people know the word 'onomatopoeia' but no one is putting it in a pass phrase. They are going to use basic, working vocabulary words like house, cove, Audi, sunset, etc...
- will be used for login at multiple sites, making dictionary attack possible.

2561 posts | registered Feb 19, 1999

---

**Dan Goodin**

/ Security Editor

[JUMP TO POST](#)

**Grieviant** wrote:

While I agree this is another interesting article by Goodin, it's sad to see the mass downvoting of posts which point out the fact that the weakness of the (obsolete) MD5 was key to the high success rate of cracking in a such a short amount of time. Can we seriously not allow any legitimate questions to be raised about the methodology without treating them as trolling? Get a grip.

Why the focus on MD5 when SHA1, SHA3 and the vast majority of other hash functions are just as unsuitable for password storage?

It's a fact that a large number of sites continue to use these hashes, despite the very clear benefits of using something like bcrypt. Witness breaches of HB Gary, LinkedIn, eHarmony, and LivingSocial, to name a very small few.

I'm not sure why these comments are getting downvoted. I suspect it's because people recognize complaints about attacking a list of MD5 hashes is a side show and largely beside the point. Ars will stop picking lists with weak hashes when the vast majority of sites stop using the underlying functions. In the meantime, please direct your complaints to sites that continue to put their users at risk because they don't use slow hash functions.

302 posts | registered Jan 30, 2012

---

**gregvp**

/ Smack-Fu Master, in training

[JUMP TO POST](#)

It amazes me, reading the first 150 or so comments, how many people say "so, the takeaway from this is that I need a different rule for generating my passwords."

NO.

The takeaways are these:-

1. You can't use ANY rule to create passwords. Your passwords MUST BE RANDOM.

No rules, no "clever" tweaks, nothing. Random. Anything one human can think of, another can. We're pretty dumb that way. Passwords must be random.

2. You must be ready and able to change any or all passwords at any time. Therefore, coming up with new passwords (random, remember) must be something you can do while you're feeling stressed and exhausted.

SUBSCRIBE

SIGN IN

>:

First, let go. Realise that professional cryptographers know more about this stuff than you do, so if you disagree with their advice, you're wrong. Then, stop trying to do something that computers are better at than you are, and realise you need to work to your strengths as a human. Then, realise that you can use a computer to do this for you.

(I'm fairly reclusive by modern standards, and I have upwards of 50 passwords. I only remember two of them, though. Most of them I've never even seen.)

Lots of commenters have given you a hint: "use a password manager". Bruce Schneier's Password Safe, KeePass2, KeePassX, 1Password, LastPass, others... there are several to choose from. You can wait for Ars's next article on passwords, or you can go ahead now. I chose KeePassX and compatible Android and iOS apps, all using device-local copies of the same password register, helpfully synchronised by DropBox. I'm unlikely to lose all four of my computers at the same time. Even if I do, I can download the list onto replacements.

Get a password manager, and set aside a couple of hours to change your passwords. There's one tiny task to go through first.

Having chosen your password manager, you need to protect access to it. Do what cryptographers do: use a passphrase. That's working to your strengths. Phrases are made of words, and humans are evolved to remember words. Peter Bright pointed out in a comment on the piece about Nathan's password cracking adventures that Randall Munroe's four-word phrase is not strong enough. But Peter didn't allow for a trivial adjustment. With five words instead of four, Peter's argument is blown out of the water. Five words are, for humans, a LOT easier to remember than 12 random keyboard characters.

But why stop at five? Five is only just good enough, and words are what people are good at: they're your strength. Go large: use seven words.

Passphrases with seven RANDOMLY chosen words (from a large-enough list) should be infeasible to decrypt for the foreseeable future, allowing for double the current rate of growth in hardware and software capabilities. Not my opinion, that of the professionals.

Seven words are easy to remember. I can remember two sets of seven words, my wife at least one, possibly three or more. If you want some help: having come up with your set of words, recite it to yourself a few times, and write it down a few times. (Shred or burn the paper afterwards.) You won't forget it -- in fact, the odds are good you'll remember a seven-word phrase the next day, even if you just read it once after generating it.

How do you generate a RANDOM sequence of five words. Here's the cryptic hint dropped by other commenters: diceware.

Diceware?? They mean, "go to diceware.com, and follow the instructions there for generating a passphrase." Diceware's method is as valid now as it was in 1996. (If you're a coder, you can cobble up something that uses a large word list and the computer's cryptographically safe source of randomness -- in python, `random.SystemRandom()` -- but

why not take the chance to get away from the keyboard for a while?)

Those takeaways again: for passwords, clever is stupid. The only thing that works is random. Humans can't do random, but computers can. Use a computer, a secure password manager, to manage your passwords. Use a cryptographer-approved method of generating the one passphrase that you need to remember -- a method that works to your strengths as a human.

8 posts | registered Nov 19, 2010

**Dan Goodin**

/ Security Editor

[JUMP TO POST](#)

**kurkosdr** wrote:

Color me confused, but if the file containing the hashed passwords leaks, isn't it a matter of time before a hacker determined to find every password in the file eventually finds your password? No matter how complex your password is?

We do need better hash functions sure, but what we need more is better ways to protect the file that contains the hashed passwords from leaking. So that all the hackers get is a black box that allows only a fixed number of login efforts before requiring longer and longer waiting times between retries.

Please see the brute force graph on page 2. It can take years or even decades to brute force long passwords. So, yes, it /is/ only a matter of time before every hash in a leaked list is cracked. But if the account user is smart, that time will be measure in years rather than in seconds.

306 posts | registered Jan 30, 2012

## READER COMMENTS 493

### DAN GOODIN

Dan Goodin is Senior Security Editor at Ars Technica, where he oversees coverage of malware, computer espionage, botnets, hardware hacking, encryption, and passwords. In his spare time, he enjoys gardening, cooking, and following the independent music scene.



**Unsolved  
Mysteries Of  
Quantum Leap  
With Donald P.  
Bellisario**



**Unsolved  
Mysteries Of**



WATCH

## Unsolved Mysteries Of Quantum Leap With...

room do while Sam is in their bodies? What happens to Sam's loyal ally AI? 30 years following the series finale, answers to these mysteries and more await.



Warhammer 40K  
With Author Dan Ahnett



SITREP: F-16  
replacement  
search a signal of  
F-35 fail?



More videos

← PREVIOUS STORY

NEXT STORY →

## Related Stories

Why passwords have never been weaker—and crackers have never been stronger

“thereisnofatebutwhat-wemake”—Turbo-charged cracking comes to long passwords

How I became a password cracker

Why you should take hacked sites' password assurances with a grain of salt

## Today on Ars

Our Solar System possibly survived a supernova because of how the Sun formed

The 2024 Rolls-Royce Spectre proves EVs make the best luxury cars

336,000 servers remain unpatched against critical Fortigate vulnerability

AMAs are the latest casualty in Reddit's API war



The link rot spreads: GIF-hosting site Gfycat shutting down Sept. 1

Pornhub cuts off more US users in ongoing protest over age-verification laws

Unsolved Wendy’s outbreak shows challenges of fighting foodborne illnesses

Musk annoys Twitter users by capping number of tweets they can view each day

STORE  
SUBSCRIBE  
ABOUT US  
RSS FEEDS  
VIEW MOBILE SITE

CONTACT US  
STAFF  
ADVERTISE WITH US  
REPRINTS

NEWSLETTER SIGNUP

Join the Ars Orbital Transmission mailing list to get weekly updates delivered to your inbox. [Sign me up →](#)



CNMN Collection  
WIRED Media Group  
© 2023 Condé Nast. All rights reserved. Use of and/or registration on any portion of this site constitutes acceptance of our User Agreement (updated 1/1/20) and Privacy Policy and Cookie Statement (updated 1/1/20) and Ars Technica Addendum (effective 8/21/2018). Ars may earn compensation on sales from links on this site. Read our affiliate link policy.  
Your California Privacy Rights |

The material on this site may not be reproduced, distributed, transmitted, cached or otherwise used, except with the prior written permission of Condé Nast.

[Ad Choices](#)