You’re hashing it wrong

The “Collection 1” data breach, containing around 773 million unique emails and passwords, dropped at the beginning of this year and more has been promised to come. In light of this, I want to talk about the weaknesses in current password-handling best practices on both the user and business end. More specifically I want to complain about the common and extremely out-of-date delusion that salted hashes are somehow safe that pervades the computing community.

The aforementioned databreach, like most, contained passwords stored in a mixture of plaintext, hashes and salted hashes.
Its not even worth explaining why storing passwords in plaintext is a catastrophe, but the other two storage types have their own, lesser-known, problems too. The three main weaknesses of password storage today I think are:

1. The wrong hashing algorithms are being used
2. Developers are lulled into complacency by salting
3. Common password advice given to users is useless

Now, one obvious statement is that old hashing algorithms should never be used for cryptography. SHA-1 and MD5 are both hopelessly outdated, and MD5 has been fundamentally broken after a paper published in 2004. And yet, unbelievably, in 2013 Adobe had an enormous databreach and were found to be still using unsalted MD5 hashing for their password storage, proving yet again that you literally cannot set the bar low enough for the public. Either the public needs to start pushing back stronger against this kind of negligence, or there needs to be regulatory punishments introduced by governments to fine companies for being so irresponsible with customer data.

But even modern algorithms like SHA-512, commonly used in Linux distros, are no longer up to task for large datasets simply because they are just too fast, especially with fast-improving GPU technology spurred on by last years’ bitcoin boom. The issue of hashes being too fast is compounded because people use the same algorithms for different things! When you are checking the integrity of datafiles, you want a fast hash which is antithetical to security. Developers desperately need to start actually using purpose-built modern day cryptographic hashes, instead of just slapping a salt into SHA-512 and calling it a day. Modular hashes which can vary hash speeds based on the specific use case exist and would be absolutely ideal, for example PBKDF2 or bcrypt2.

The passwords in this databreach would almost invariably be cracked with a password dictionary, which basically takes in the hundreds of millions of previously leaked passwords and ranks them by popularity. More sophisticated attackers will then also run “modification” options, i.e instead of searching only for direct matches they will also try simple permutations such as replacing “O” with “0” or “i” with 1. In this way, an attacker can extremely easily crack the top, say, 80% weakest passwords in a breach, more than enough for their purposes, and never have to worry about actually cracking every last one. In this way, choosing a password is a lot like running away from a bear - you don’t have to outrun the bear, you just have to outrun the person next to you. And yet current password advice practically encourages identical passwords from users! Things like requiring a number or a capital letter are pointless, as the majority of users will just capitalise the first letter or put a “1” on the end of their password, defeating a prehistoric dictionary attack but folding instantly to a password dictionary.

So as a user, what can you do? There are three takeaways from this article:

1. Prioritize password length above all else - the amount of variance grows exponentially, thus the security of a 20 digit password is unfathomably greater than one of half its size. 15 characters should be the absolute minimum if you are using english words inside your password.
2. Don't even bother with "normal" substitutions, like l33tspeak. Instead insert your l33tspeak into the middle of words or substitute the wrong letters to throw off permutation seekers - for example, instead of "Z3RO" you could have "Z3ERO" or "Z5RO" as stronger alternatives. Adding a "1" or "123" to the end of your password is similarly useless, try inserting it into the middle of your password instead if you must.
3. Change your passwords! If you use weak passwords for small and/or incompetent companies, they *will* be broken eventually and you may not even notice. Make sure your passwords for important services (email, banking, etc) are completely different from those you use for other accounts.

Other than that, there is very little you can do except hope.