Common password vulnerabilities and how to avoid them

Tomasz Andrzej Nidecki | March 21, 2022

Weak passwords and password reuse are still some of the most serious concerns for cybersecurity. There are several ways to increase password security but they are often not adopted by users and administrators. Here's how you can make sure that sensitive data in your web application is not compromised by malicious hackers due to insecure user passwords.

Length or complexity?

The most common password policy enforced by administrators, both in the case of web applications as well as other systems, is a length and complexity policy. For example, a complex password may be required to contain at least 8 characters, uppercase and lowercase letters, numbers, and special characters. However, this policy is actually quite weak and should not be recommended. It's best to set the minimum length to at least 16 characters, allow for spaces, and introduce no length limit.

Several websites exist, where you can check how long it would take to break your password using a brute-force attack. If you enter an 8-character password with numbers, uppercase/lowercase, and special characters in How Secure Is My Password, it says that a computer could break your password in 9 hours. On the other hand, if you enter a 16-character password that uses only lowercase letters, the result is 224 million years.

The reason for this is mathematical. The formula for password entropy is *log2(RL)*, where *R* is the number of unique characters and *L* is the length of the password. If you use only lowercase characters, *L*=26. If you add uppercase characters and digits, it becomes 62. With special characters, it depends on which characters are allowed, but let us say it becomes 70. For a one-letter password, this means that adding the extra characters increases the complexity threefold. But a two-letter password based on just lowercase characters has 262 = 676 entropy. Therefore, adding just one lowercase character to a lowercase password increases its complexity not three but 26 times. You can clearly see, that it's the length of the password that really matters, not the complexity.

Is length enough?

If password cracking were only based on the brute-force method (trying every single possible combination), password length would be the best way to practically make attacks impossible. However, there is also a cyberattack technique called dictionary attacks, which basically means password guessing based on commonly used words. For example, a password *the quick brown fox jumps over the lazy dog* would be cracked by a dictionary attack almost instantly. On the other hand, a passphrase with exactly the same letters: *vromjon tobki huhet qecar dzowyf xup selg*, would be nearly impossible to crack.

Luckily, dictionary attacks are also very easy to avoid if you use fake words that are easy to memorize because of the way they sound. For example, you can use a passphrase such as *bargle zaws gubble meh brudda dulgly*. Those who know obscure languages have it even easier because most dictionary attacks are based on English vocabulary and several other popular world languages. For example, *kapaċi niekol ħġieġ għax ma tweġġanix* could be considered a very strong password.

Another interesting technique for creating secure passwords is using just the first couple of letters from each word to form a long password that is based on a real sentence. For example, you can make a safer password from *the quick brown fox…* sentence: *thequibrofoxjumovethelazdog* (although it still contains several dictionary words, which is not optimal). You can also, for example, take the first letter of every word from a longer phrase that you know well: *skwklbfhkskppp* (the *Soft kitty* song lyrics). Of course, in all these cases changing some letters into uppercase and adding numbers between the actual words even increases the security (for example, *The7Qui0Bro1Fox2Jum3Ove4The5Laz6Dog*).

Note that password-breaking tools also replicate user tricks such as replacing some characters with numbers that look similar (e.g. *1* = i, *2* = z), reversing words (e.g. *drowssap*), adding a number at the end, etc. Therefore, passwords such as *dr0w554P123!* are easily cracked.

The false sense of security

Another very common mechanism used by web applications and other systems to increase password security is forcing the user to regularly change their password. Such mechanisms usually store the hashes for old passwords and therefore do not let the user reuse any of their previous passwords.

Unfortunately, such password requirements introduce a very false sense of security because users find easy ways to go around them. Let's be honest, what do we do when the system asks us for a password reset? We usually add the next consecutive number at the end and just keep replacing it every three months when asked (*password1*, *password2*, *password3*…).

This technique does not increase password entropy and does not in any way prevent dictionary attacks. Therefore, more and more big players including Microsoft are moving away from recommending regular password changes. Even large institutions such as FTC are now recommending against this, so don't implement this mechanism in your web application.

The danger of password reuse

Even if you come up with the most secure password, it becomes insecure if you use it on every website and in every application. With the number of global data breaches, there is a big chance that your password for some sites has been compromised. If you think this is unlikely, you may be surprised: just enter your email address on Have I Been Pwned.

Luckily, most sites do not actually store unencrypted passwords (although even the biggest players on the market have been guilty of it). This means, that in the case of a data breach, it's only the password hash that is compromised. However, in many cases, the hashes are not secure. Many web applications use old and easy to compromise hash algorithms such as MD5. In such a case, the attacker needs much less time to find the password on the basis of a hash. Of course, the simpler your password, the faster it will be compromised.

Unfortunately, there is no way to check for password reuse in your web application. Therefore, you must simply educate your users and trust that they follow your suggestions.

Security of password managers

To combat password reuse, we resort to password managers. They became more popular in recent years and there are quite a few to choose from for every platform. Many of them are very easy to use and cross-platform. This means that you can, for example, store your password database (secured with a single long and very complex password) in the cloud and access it from your PC, from your mobile, and from anywhere else via a web interface. Password managers combat password reuse and they can also generate complex unique passwords for you, although such passwords will not be easy to remember.

Some password managers make it even easier for web application users and introduce their own web browser plugins. Such plugins take the password out of the password database and automatically use it on the website. However, such plugins are the Achilles heel of password managers and should be avoided, as proven for example by a LastPass vulnerability from 2019. You will be much more secure if you manually open the password manager, copy the password, and paste it into the web application.

Multi-factor authentication

The strongest defense against password-based attacks nowadays is multi-factor authentication (MFA), often simplified to two-factor authentication (2FA). The basic assumption is that the user must supply something that they know (the password) and use something that they have (for example, a smartphone or a hardware token). MFA is even part of the most recent compliance requirements for PCI DSS and more. However, not all MFA mechanisms are equally safe.

The most common MFA technique is for the web application to send an SMS with a one-time code to the mobile number supplied by the user. This is also the worst idea for MFA. Attackers are using SIM-swap attacks to compromise such mechanisms. They scam the mobile operator into issuing a duplicate of the user's SIM card, for example by assuming the user's identity, and use it to receive the one-time code. One of the most popular cases of such an attack was the compromise of the account that belongs to the Twitter CEO.

One way to avoid this is to use a mobile app. You can develop your own application or use standard one-time-password (OTP) solutions such as Google Authenticator or FreeOTP. Mobile apps either use one-time codes generated by the application or they send a push notification to your mobile (which you must confirm).

However, using a mobile app may present an extra problem for users. If they buy a new phone or need to reset the current one, they usually need to visit a branch office to get back their access to online services. On the other hand, if they can restore access by calling on the phone, this makes them very susceptible to social engineering scams. Also, mobile apps based on standard one-time codes do not protect the users from phishing attacks: a phishing page may act as a proxy to the real page and intercept the code and the password.

Hardware tokens to the rescue

The ultimate security option for MFA seems to be hardware tokens. Such tokens have been around for a while, in different forms. For example, since the late 90s, many banks issued tokens to their customers to let them access online services. Such tokens required you to get the access code based on the seed code displayed by the website. Devices like this were not only used by banks but, for example, even by game manufacturers (for example, by Blizzard since 2008, still supported till recently). There are banks that still use such physical tokens instead of or in addition to mobile apps.

Another form of hardware tokens is now becoming more and more popular – hardware keys. However, the idea is not new – they were used for years by certain software manufacturers to combat piracy. Solutions such as YubiKey, Google Titan, Thetis, and more allow access to many applications and are based on common standards (such as OTP, Smart Card, OpenPGP, FIDO U2F, FIDO2). There are keys that you can use for computers and mobile phones: they may use a physical interface (USB-A, USB-C) or a wireless one (Bluetooth, NFC). You can bind your account to multiple keys at the same time (if you are afraid of losing the token or if you need different tokens for different devices due to technology requirements).

Nowadays, hardware security mechanisms are also part of your regular computing equipment. For several years, desktop and laptop computers have been equipped with hardware modules (TPM/SE) that provide encryption technologies. The Windows 10 operating system is compatible with the FIDO2 standard, which means that any Windows 10 device with the hardware module may function as a hardware key. Mobile phones often come with biometric mechanisms such as fingerprint scanners or facial recognition. All these, especially in combination with passwords, provide extra security. It's true that someone can steal your phone/laptop and fingerprint/face scanners can be hacked, but the probability that the thief will also have your password is very small.

All passwords matter

As a web application developer, you may think that if an unprivileged end-user password is breached, and that user has no access to sensitive information, it does not endanger the web application at all. Unfortunately, this is not always the case. For example, if a cybercriminal uses an SQL injection to get the list of password hashes and then cracks even one user's password, they may use this user account to access the system and use privilege escalation to gain access to a privileged account. Unfortunately, an attacker might also get the user password via social engineering, phishing, or malware and this is something that you, as a web application developer, cannot avoid.

Therefore, you should ensure that your users are always using secure passwords. This comes down to two activities: using password security mechanisms in your web application and testing for weak passwords. In the first case, you can either develop your own mechanisms or use open-source solutions (such as nowsecure or zxcvbn) and in the second case: use a web vulnerability scanner that can test for default passwords and weak passwords.

THE AUTHOR

Tomasz Andrzej Nidecki
Principal Cybersecurity Writer
LinkedIn

Tomasz Andrzej Nidecki (also known as tonid) is a Primary Cybersecurity Writer at Invicti, focusing on Acunetix. A journalist, translator, and technical writer with 25 years of IT experience, Tomasz has been the Managing Editor of the hakin9 IT Security magazine in its early years and used to run a major technical blog dedicated to email security.

Related Posts:

Secure Password Recommendations and Research
Read more →

Testing for weak passwords: a common oversight without a great solution
Read more →

Web Passwords are Often the Weakest Link
Read more →

← Older | Newer →

Learn More

IIS Security
Apache Troubleshooting
Security Scanner
DAST vs SAST
Threats, Vulnerabilities, & Risks
Vulnerability Assessment vs Pen Testing
Server Security
Google Hacking

Blog Categories

Articles
Web Security Zone
News
Events
Product Releases
Product Articles

PRODUCT INFORMATION	USE CASES	WEBSITE SECURITY	LEARN MORE	COMPANY	DOCUMENTATION
AcuSensor Technology	Penetration Testing Software	Cross-site Scripting	White Papers	About Us	Case Studies
AcuMonitor Technology	Website Security Scanner	SQL Injection	TLS Security	Customers	Support
Acunetix Integrations	External Vulnerability Scanner	Reflected XSS	WordPress Security	Become a Partner	Videos
Vulnerability Scanner	Web Application Security	CSRF Attacks	Web Service Security	Careers	Vulnerability Index
Support Plans	Vulnerability Management Software	Directory Traversal	Prevent SQL Injection	Contact	Webinars

Login | Invicti Subscription Services Agreement | Privacy Policy | Terms of Use | Sitemap