# Potential vulnerabilities

SQL Injection: An SQL injection is when malicious and deliberate code is inserted into an entry field, which is then accidentally executed by the receiver. An example is executing SQL code into a login field to gain access, view or destroy contents of a website’s database such as user account details. This type of vulnerability is widely known and is often protected in most websites such as limiting special characters used in code syntax or using prepared statements. Prepared statements are SQL queries that are templated and sent to the database, with values later filled in when needed. This means dynamic SQL is never able to execute, including SQL from injection attacks.

Brute force password access: This is simply automatically and rapidly attempt to use as many passwords as possible, starting from the most common and insecure passwords, hoping to eventually get a correct password out of millions of guesses. This can be easily prevented by limiting the number of logins available at once, with a cooldown being enforced on an account if too many incorrect guesses are used. Optionally, if a cooldown expires and the limit is reached again, the cooldown period can be increased to further prevent the number of login attempts.

Insecure Passwords: A simple vulnerability caused by having insecure passwords. One example is having a common password too easy to guess such as Password12345, personal details such as you or a family member’s name or birthdate, or commonly used words with no variation that is vulnerable to dictionary attacks (guessing passwords using words from the dictionary). Insecure passwords also cover having a password that is physically accessible, such as having it written down near the computer or stored in plaintext somewhere on a computer or online. These issues can be prevented by informing users of common password security, not allowing passwords without a number and a special character allowed by the system, and ensuring no passwords are visible physically in person or in plaintext on the computer system.

Similar Password Hijack: This type of vulnerability is caused by users and insecurities in other services. If a user has the same email and password across multiple websites and services, and if just one of those gets their account details compromised, hackers can use the email and password combination to gain access to other accounts held by the user. For example, if a forum account has their details leaked and the same email and password is used for banking, PayPal and their workplace, the hacker could gain access to all these accounts easily. Users can protect themselves by having unique passwords for every account and keeping up to date on when certain sites get compromised. Services can protect their users by regularly urging them to change their password at specific intervals (I.e. every 3 months), changing passwords when a breach in another major service is known, or requiring a security question or additional information that is stored on no other site.

DDoS Attacks: Short for a Distributed Denial of Service attack, this is flooding a web server or network with innumerable requests until it can no longer handle the load and either drastically slow down or crash. This prevents regular users from accessing the service until the attack stops or the service is restarted. “Distributed” means the requests come from many different sources so it’s impossible to stop the traffic by blocking one source and is often accomplished by utilising bots to have many clients spamming requests. DDoS prevention can be complicated, and websites usually outsource protection, such as cloud service providers that have tools to monitor, detect and scrub botted activity as well as backup servers in case a network does go down, to mitigate the amount of downtime.

Outdated Software: Sometimes vulnerabilities can be found in existing software, protocols, programming languages and even hardware. This requires said components to be updated and fixed so the vulnerability can be fixed. Famous examples of vulnerabilities include Heartbleed, a vulnerability found in OpenSSL from 2012 – 2014, and the recent CVE-2020-0601 vulnerability in Windows this year. These vulnerabilities can be avoided by ensuring that the latest version of software, operating systems, programming languages, etc is downloaded and used to avoid old exploits being a threat.

Social Engineering: The least overtly techy of vulnerabilities but arguably the method most used today to gain unauthorised access, social engineering covers all the potential threats and vulnerabilities caused by human error involving said person being tricked or fooled into giving away sensitive information or unauthorised access. Examples include phishing, where a fake version of a website or service is used to trick the victim into giving away login details. Pretexting, impersonating a colleague, boss or client to fool the victim into giving away details to someone they think is trustworthy. And tailgating, where the attacker gains access to somewhere unauthorised by following someone who does usually by impersonating staff and following right behind them as they go through a secure door. Every person is susceptible to falling for these attacks, and the best way to prevent them is educating staff into how these attacks work and being on the lookout for fake phishing emails, making sure that a caller authorises and confirms who they claim to be before giving information, and making sure no-one directly follows behind you even if they look like a colleague or trustworthy person.

# Implemented Solutions

SQL Injection: We have a Django module, which we are using for the login that automatically implements protection against common SQL injection methods. It incorporates means of protection including data sanitisation for the login. We are planning to implement prepared statements too to cut down on the amount of dynamic SQL statements used which could potentially be used for an SQL attack.

Brute force password access: Once again Django automatically has protection against this. The same login module we used to protect against SQL also innately protects against mass use of passwords, denying attempts to login for a while if 5 password attempts are used. This protects against people just attempting passwords over and over until they finally get lucky with the correct one to log into an account. Shown below is evidence of this in action: