Content Management Systems: Data Security

| **A. Greaney and M. Kinneberg**  Computer Science and Information Systems, Minnesota State University Moorhead, Moorhead, MN, USA |
| --- |

This project examines the use of content management systems (CMS) by small organizations or individual users, with a focus on protection of personal data and ease of use. The purpose of this project is to develop a schema for a content management system that fulfills the needs of small organizations or individual users, protects their data, is easy to use, and is fully functional. The project examines various methods of data storage and encryption, current content management systems on the market, and user needs within the specified niche. The project is based upon agile development methodology and utilizes open-source licensing.

**Keywords:** Content Management Systems, Data Protection, Usability

# Introduction

The term Content Management System, CMS for short, does not carry a solid definition, yet it is still a commonly used term on the web. For the purposes of this research, a CMS is a system that holds, manages, and produces data for use by a single organization on an online portal [1]. This means that any built system that is used to power a website and manage its data without requiring the user to manually write code is a Content Management System.

There are many popular CMS on the market and the most popular ones are often built and used for a broad spectrum of websites. 25% of websites currently available for viewing on the internet are powered by WordPress [2], a CMS originally designed for blogs. Two other popular CMS are Joomla and Drupal, who together power 5% of websites [2], these were both designed as general website support systems. In this paper, our statistics and information will be based on the usage and distribution of WordPress.

Content Management Systems allow any person who wants to create a website the opportunity to do so. A WordPress website takes on average 5 minutes to install, and within 15 minutes the user can have content live on the internet. A CMS takes away all need to write code, design pages, and understand user-experience. [3] It isn’t fool proof, as even a website driven by a CMS can be extremely difficult to use, but it is a lot harder to go wrong. Once the basic website is installed, using a CMS is like using a text editor such as Microsoft Word. The system takes care of all the formatting, which can be overridden by the writer to a certain extent, but it is then matched to the front-end style of the website. This is what makes CMS so popular, as they take away the hard part of running a website and allow the website owner to focus on the content of the site. [4]

The three most popular CMS are all free, open-source software. This means that they can be freely used, distributed, and modified by anyone. While this allows for a populous and feature-rich software, it also opens many gaps for potential security flaws. The issue lies in two different areas, the free distribution and modification of the software and all related software that hooks in, and the lack of knowledge or awareness from the user base. [5] While there are many ways of securing a system, the loss of data is usually a result of missed updates, poor username and password choices, lack of firewalls and security measures put in place during install and use of themes and plugins from unsecure sources. [2]

The biggest problem with open-source software is the lack of checks and verification before distribution. Many times, the software is not published with malicious intent, the security gaps occur due to code not being properly checked, poor software design, and lack of knowledge from the programmer. [6] Many plugins[[1]](#footnote-1) are often given access to more information than they need or access the data they need in an incorrect manner. This leads to data being exposed, often without the end user being aware as this data was never meant to be accessed. It then doesn’t take much for this availability to be exploited and abused.

On February 10, 2017, 1.5 million websites using WordPress for their content management were hacked due to a single flaw in the main WordPress software distribution involving REST-API management. [7] The main WordPress distribution has a team of developers behind it, who check code, possible vulnerabilities, and manage new version distribution. This is a managed process that passes through many hands, and still vulnerabilities and exploits can arise. Plugins and themes are often built and distributed by a single person, meaning very few checks and verifications occur. Therefore, many possible security threats exist.

In 2009, a vulnerability occurred in a version of WordPress 2.7 that allowed unauthorized users to access and view pages that they did not have access to, this included pages that displayed user data. The vulnerability also gave unauthorized users access to pages where they could modify plugins on the website [7]. A vulnerability such as this would have had a huge impact on websites that stored large amounts of private information, allowing malicious usage of phone numbers, emails and other personal information.

In 2011, a vulnerability was discovered in a popular plugin used for image resizing. This plugin was bundled[[2]](#footnote-2) with nearly 1,500 themes [7] and led to the development of plugins specifically designed to detect and scan for vulnerabilities related to the exploited plugin. These scanners have been downloaded over 60,000 times [8]. This exploited plugin had direct access to the database behind the website as it manipulated stored images. The vulnerabilities enabled people to perform malicious injections of code that allowed direct manipulation of the website. Through this, all data stored on the website was accessible to anyone.

WordPress was originally developed as a blogging[[3]](#footnote-3) platform but is now used to power a variety of websites. 74.6 million sites depend on WordPress for their content management, and there are over 29,000 plugins available [8]. Over 1 million businesses use WordPress for their website, yet only a fraction of these have a blog or regularly updated content. WordPress became popular due to its flexibility, ease of use, and availability of themes[[4]](#footnote-4) and plugins. However, there are also many other content management systems that were built for specific purposes. There are CMS available for businesses, restaurants, tax management, non-profit organizations, churches, and many more. Each of these systems were designed to provide functionality specifically for that type of organization.

While these CMS are smaller and more specific, they can be more expensive to manage and maintain as they are often not open-source, or if they are, there are less developers interested in maintaining them. There occurs a trade-off between cost, functionality, security, and availability. [9] In many cases, people choose to avoid the cost and revert to using a larger open-source software, such as WordPress, that doesn’t suit their needs. [10] Often this need is secure management of data, and unknowingly the organization puts their own, and their customers, data at risk.

There is a common belief that a strong password is enough to protect a website, which unfortunately is not the case. There are a variety of ways that passwords can be managed and stored. The data they are protecting can also be stored in many ways, varying in complexity and in turn a level of security.

Passwords are usually stored using a hashing algorithm. Hashing is the transformation of a string of characters into a usually shorter fixed length value or key, that represents the original string. [7] The process of hashing data is used to store access codes and passwords but relies on specific algorithms for translation. This means that anyone who has the key to the hash, can decode the simple string that the hash represents. Encryption is designed so that a human cannot simply decipher the representation of a string of characters, however a computer can process this data a lot faster. This means that simple number and letter replacements, often taught in high mathematics classes, are not complex enough to manage any type of data stored online. Encryption is the process used in the transfer of data from one means to another, such as from the database to the front-end output that a website visitor sees. The interception of this data as it is being transferred is where a security vulnerability can occur, and improper access to the data can be obtained. [11]

There are also other vulnerabilities that occur when handling data, specifically data that is input by the user. Any time a user is given the option to enter data to a website, the server is going to be exposed directly to the data. Therefore, this data must be handled and controlled appropriately to avoid harming the server. This is often done by controlling the types of data submitted, for example using a method called escaping strings, that converts all the entered data into formatted HTML, which will stop SQL statements from executing if they have been passed in (This malicious method of attempting to obtain data is known as a SQL injection). [11] Vulnerabilities can also occur when users choose to save data in caches or cookies as this data must be saved correctly or it can be visible as plain English to the user, for example in URLs). While the visible data may not directly be a problem, it can show a malicious user how the data is being handled and/or stored, giving them a method upon which to base their access attempt.

Over the last 12 months, there have been several huge data breaches around the web. E-Sports Entertainment Association (ESEA) suffered a breach where 1.5 million records were added to its database, and records had been leaked. The records included personal information as well as gaming account details. Similarly, Xbox and PSP were also hacked, with 1.2 million accounts each being accessed. This breach included access to IP addresses, usernames and passwords. InterContinental Hotels Group (IHG) had 4 months’ worth of credit card transaction data stolen from 12 different properties, including internal verification codes that gave access to specific system processes. In all these cases, these websites were powered by a content management system. Systems that manage private user data such as IHG have much greater security than WordPress, yet they were still affected by vulnerabilities. IHG later reported that the original number of affected properties was 12, numbers later showed that 1,200 were affected.[12]

Data security is a huge issue in society today, and a hot topic in the media. Despite this, data on thousands of people is being obtained daily, and there are no signs of this slowing down. Even though people are concerned about their information being obtained, they are still many times oblivious to how their data is being handled, or what is happening with information they enter on a website. As software engineers, there are multiple approaches we can choose to take. Education is an option, informing users in detail of how websites work, how their data and private information is managed and obtainable, and detailing how they can protect themselves online. Education is important, as unfortunately not everyone is ethical and honest in their online intentions. However, as software engineers we also have a responsibility to create safe software that people can use without fear of their information being obtained for malicious uses.

Through this research, we have chosen to examine the different types of data encryption, password handling, and data passing that occurs in a content management system. We will be looking at the strength of different algorithms, and how a system can be strengthened without compromising usability and performance. Our hope is to develop algorithms and methods that can be used to develop secure content management systems that only allow external plugins access to the least amount of data they need to perform their tasks, encrypt data correctly and securely, manage user passwords and account information securely, all while not compromising user experience and website performance.

# Application

## Password Algorithms

### Common Passwords

One of the most common causes of security breaches on websites is weak passwords. A user will often select a password that is quick and convenient for them to use, which often means that convenience is prioritized over security. Among the top 25 passwords used in 2017 were “Password”, “Qwerty”, or part of the user’s e-mail or username that they used for that same website. [13]

The problem with common passwords is that they allow malicious users access through brute force within a minimal time span. Therefore, developing a method for avoiding this will help stop the effectiveness of brute force attacks. The beginning of our algorithm will therefore not allow the user to use commonly used passwords, while also limiting the use of words and phrases included in, or similar to, the user’s login name or e-mail. See Table 1 for a full list of common passwords, phrases, and words that users will not be allowed to use for their passwords, based on data collected on the most common passwords used in 2016 and 2017. [14]

Table 1: Top 100 Most Common Passwords of 2017

|  |  |  |  |
| --- | --- | --- | --- |
| Rank | Password | Rank | Password |
| 1 | 123456 | 2 | password |
| 3 | 12345678 | 4 | qwerty |
| 5 | 12345 | 6 | 123456789 |
| 7 | letmein | 8 | 1234567 |
| 9 | football | 10 | iloveyou |
| 11 | admin | 12 | welcome |
| 13 | monkey | 14 | login |
| 15 | abc123 | 16 | starwars |
| 17 | 123123 | 18 | dragon |
| 19 | passw0rd | 20 | master |
| 21 | hello | 22 | freedom |
| 23 | whatever | 24 | qazwsx |
| 25 | 654321 | 26 | 654321 |
| 27 | jordan23 | 28 | harley |
| 29 | password1 | 30 | 1234 |
| 31 | robert | 32 | matthew |
| 33 | jordan | 34 | 111111 |
| 35 | daniel | 36 | andrew |
| 37 | lakers | 38 | andrea |
| 39 | buster | 40 | joshua |
| 41 | 1qaz2wsx | 42 | 12341234 |
| 43 | Ferrari | 44 | cheese |
| 45 | computer | 46 | corvette |
| 47 | blahblah | 48 | george |
| 49 | mercedes | 50 | 121212 |
| 51 | maverick | 52 | 1234567890 |
| 53 | nicole | 54 | hunter |
| 55 | sunshine | 56 | tigger |
| 57 | 1989 | 58 | merlin |
| 59 | ranger | 60 | solo |
| 61 | banana | 62 | chelsea |
| 63 | summer | 64 | 1990 |
| 65 | 1991 | 66 | phoenix |

|  |  |  |  |
| --- | --- | --- | --- |
| 67 | amanda | 68 | cookie |
| 69 | ashley | 70 | bandit |
| 71 | killer | 72 | aaaaaa |
| 73 | pepper | 74 | jessica |
| 75 | zaq1zaq1 | 76 | jennifer |
| 77 | test | 78 | hockey |
| 79 | dallas | 80 | passwor |
| 81 | michelle | 82 | admin123 |
| 83 | qwertyuiop | 84 | pass |
| 85 | asdf | 86 | william |
| 87 | soccer | 88 | london |
| 89 | 1q2w3e | 90 | 1992 |
| 91 | biteme | 92 | maggie |
| 93 | querty | 94 | rangers |
| 95 | charlie | 96 | martin |
| 97 | ginger | 98 | golfer |
| 99 | yankees | 100 | thunder |

### Password Salting

A salt is a pre-defined string of characters that is added to the password during hashing. This salt means that the password input itself changes, and therefore doesn’t directly represent the password the user selected. It doesn’t require the user to know the salt or do anything different than enter their password as usual. The salt process is all performed behind the scenes, after the user submits the login request.

Using a salt means that brute force attacking becomes very nearly impossible as the attacker will have to search through passwords for each value of the salt. The attacker must then observe and monitor a high number of data submissions to attempt to crack the algorithm for the salt before brute forcing can be attempted. Even then, the amount of salts that can occur, depending on the algorithm created for the salt, can make generation so time consuming that the process becomes near impossible. See Table 2 for examples of password salting.

Table 2: How Salts Change Password Hashes

|  |  |  |
| --- | --- | --- |
| Password | Salt | Result |
| hello | - | 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e7304336293b9824 |
| hello | QxLUF1bgIAdeQX | 9e209040c863f84a31e719795b257752394739fe5ed3b58a75cff2127075ed1 |
| hello | bv5PehSMfV11Cd | d1d3ec2e6f20fd420d50e26429928418338a314b8ea157c9e18477aaef226ab |
| hello | YYLmfY6 IehjZMQ | a49670c3c18b9e079b9cfaf51634f563dc8ae3070db2c4a8544305df1b60f007 |

The next step of developing our algorithm was to develop a salt algorithm for use in password management. [15]

#### Static Salt

One of the simplest algorithms for using salts is creating a “secret salt hash” that is hard-coded into the password management process and called by the system every time passwords are managed. This is known as a static hash. While this does add a certain level of security and protects from brute force attacks, it requires that the hash is secured appropriately and not revealed to the user. If an attacker does discover the hash, they would just need to append it to all the passwords they are attempting to force with, therefore rendering the hash pointless. The process of using static hashing is also referred to as salt reuse, which is considered bad practice by many security researchers. [15]

#### Dynamic Salt

Dynamic hashing means that a random salt is generated for each encryption and should be performed using a Cryptographically Secure Pseudo-Random Number Generator (CSPRNG). In this case, the salt generated is unique to each user and every time a user changes their password they are given a new salt.

Salts should also be long enough that the likelihood of the same salt occurring twice is very slim. A general rule of thumb is that the salt is at least as long as the hash function’s output. The salt is stored in a separate entry in the user account table, alongside the hashed password. Hashing should always be performed server side as hashing in the browser can inadvertently give an attacker direct access to the server using the hash, without knowing the user’s password. [16]

#### Salting Algorithm

We developed our password algorithm to use dynamic hashing due to it having a higher level of security than static hashing. Despite dynamic being more complex to develop, which may be a reason it is often overlooked, it offers the end user a higher level of security in their system.

### Lengths and Complexity

The security world continues to debate how security is affected by the lengths and complexity of password. Research by Microsoft TechNet developed a formula for the selection of a password strength where Length was an emphasized factor. In formula 1, C is the size of the character set and L is the length of the password. This formula shows that the length is important, while the character set size also plays a role. However, it shows that complexity is relative in the size of the character set, and not necessarily in how the characters are stringed together.

(1)

Table 3 shows comparisons of complexity between different character set sizes when the password is 10 characters in length.

Table 3: Impact of Character Set Size on Complexity

|  |  |
| --- | --- |
| Character Set | Complexity |
| Numbers (0-9) | 33.22 |
| Alphabet characters (A-Z, a-z) | 57 |
| Alpha-numeric characters (A-Z, a-z, 0-9) | 59.54 |
| Alpha-numeric characters and common symbols (A-Z, a-z, 0-9, !, ?, etc.) | 65.55 |

This algorithm shows that a 16-character long password made with only numbers is just as strong as an 8 character long password made using all 94 possible ASCII characters. Based on this research, we elected our algorithm to require passwords to be at least 12 characters long, and contain at least one number, uppercase letter, lowercase letter, and two different symbols.

### User Recall

Another common cause of security breaches is that the user chooses a password that is too complex for them to remember. This means that they choose to write the password down on paper, use the auto-save feature on their browser, or save the password in a word document on their computer. Every time a password is saved somewhere as a physical copy, that password becomes slightly less secure. Therefore, it becomes important to allow the user to have a password they can still remember, while it is still secure enough to keep the system safe.

Research in psychology has shown that on average, a person can remember 7 chunks of unrelated data. A chunk of data is one or more character that are somehow related to each other in a way that is relatable to the person. An example of this is: “4 A 3 O P E L”, another is “Four A Three O P E L”.

With our algorithm requiring passwords to be at least 12 characters long, this means that passwords will need to contain words or phrases that are memorable in some way to the user. If they string together 12 unrelated characters they will be forced to write down or save the password somewhere. This makes it even more important for us to not allow commonly used words in the password.

### User Prompting

Sometimes the simplest solution is to inform the user. When a user understands why they need to do something a certain way, they are more likely to consider doing it. An important part of our algorithm is to inform the user of the password requirements in a way that they can understand what is needed, and how to themselves develop a password that is secure and memorable. We will be including an information box, similar to Figure 1, for password selection during account creation and password reset.

A screenshot of a cell phone

Description generated with very high confidence

Figure 1: Information Box for Prompting User

We avoid suggesting any specific numbers, letters, or symbols, as this can cause the user to naturally gravitate towards these selections, therefore making it easier for attackers to create brute force algorithms. We also try to educate the user on the use of saved and common passwords, and why these are not secure, even though they may seem to be.

#### Real-time Information

In this, we will also use a real-time password manager that measures the security of the password using a mathematical algorithm we have developed.

## Password Management

The creation of strong passwords is only the first step to strong user security. It then comes down to the CMS to efficiently manage the password once the user has created it. This involves password hashing (see section 2.3), secure storage in the database, and managing the user’s interaction with their account data.

### Password Resets

One of the hardest parts of password management is allowing the user to reset their password when they forget it, while still ensuring the integrity of the system. Most websites will now require the user to somehow verify through their e-mail that they are the owner of the account, often with a password reset e-mail, or an e-mail being sent to the registered account that informs the user that the password has been reset, and if they weren’t the one that did it to contact the administrator. The problem with this is that this can easily be intercepted.

Our system will use a token confirmation system that sends a unique link containing a token to the e-mail the user account is registered to. This token will be required for the user to reset their password, which will then go through the same level of screening as a new account creation. The token will be time-limited and expire and become unusable after 20 minutes. At this point the user would have to request a new reset token. The token will also expire immediately after use. The URLs themselves will not directly reference a data entry or token, but will instead be a hash representing the token, user account the token is linked to, and expiration time. Not having any exact data in the URL stops an attacker from being able to intercept or guess a token.

Users will also be required to answer security questions to reset their password. For administrative level users, password resets will require answers to security questions as well as their first level password.

### Two-level Administrative Security

Another important factor to consider is users that have a higher level of access than just a regular user. In this paper, all these users will be referred to as administrative users. An administrative user has direct ability to access user data, make system changes, and even export information. This means that it is very important that their security is higher than that of a regular user.

There are multiple ways of making an administrative user account more secure. One of these is a two-level authentication system where the user has two different passwords that grant them access to different areas on the system. This means that even if an account is compromised, a second password would also have to be cracked to grant access to any operations that could damage the system.

The password requirements on this secondary password will be different from the regular password. The password will need to be at least 12 characters long, contain uppercase letters, lowercase letters, numbers, and at least two different symbols. The password would have to differ completely from the regular user password and will also have a verification algorithm that looks at more aspects than the regular password.

*(Secondary level password algorithm still in development)*

This method differs from two-factor authentication that is often used by high security systems or e-mail providers. This system requires two different passwords instead of requiring an outside source verification method such as an e-mail, phone app, or software installed on the user’s computer. This method was chosen as it is quicker and more efficient as it does not require the user to leave the login screen.

### Security Questions

All users will also be required to enter three security questions. These security questions will be randomly selected out of a bank of 15 questions. Table 4 lists the bank of questions that the user will be prompted to answer from.

Table 4: Bank of Security Questions

|  |  |
| --- | --- |
| # | Question |
| 1 | What was your favorite color in high school? |
| 2 | What was your favorite class in high school? |
| 3 | How many pets did you have growing up? |
| 4 | Where was your first vacation? |
| 5 | What was the name of your first pet? |
| 6 | What is the name of your most memorable stuffed animal? |
| 7 | Where were you when you first heard about 9/11? |
| 8 | What is the name of a college you applied to but didn’t attend? |
| 9 | What is the name of your first crush? |
| 10 | What was your favorite place to visit as a child? |
| 11 | What was your dream job as a child? |
| 12 | How and where did your parents meet? |
| 13 | What is your least favorite nickname? |
| 14 | Who was your childhood hero? |
| 15 | What was your favorite movie during your childhood? |

These questions were selected based upon information that an attacker could not easily obtain about a user. A lot of personal information that is commonly asked for in security questions is publicly available, especially with a simple search engine query.

## Password Hashing

Passwords are stored in the database using a cryptographic hashing algorithm so that passwords are never directly visible as plain strings. There are many different hashing algorithms in existence, each that have their own pros and cons. We examined the most popular hashing methods that are used in C# .NET as this is the language our software is being developed in.

A hash is a one-way function that turns any length of data into a fixed-length unique string that cannot be reversed. Hashes are created in such a way that the string of characters (numbers and lowercase letters) are completely different, even when just one character in the original input is changed. Hashes are however not nearly enough on their own to secure access to a user’s account. CrackStation.com supplies a testing environment for brute force attacks on passwords, with a database over 19GB in size of possible passwords devised by scraping Wikipedia and other popular websites. In just a couple of seconds this database can find the un-hashed version of nearly all common passwords, if a salt is not used.

hash("hello") = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824

hash("hbllo") = 58756879c05c68dfac9866712fad6a93f8146f337a69afe7dd238f3364946366

The process of hashing is simple. When the user registers for the first time they type in their chosen password. If being used, a salt is then applied to the password. This new password with its salt is then passed through the hashing algorithm and the fixed length hash is created. This hash is then passed to the database and saved with the user’s account information. When the user attempts a sign-in, the hash of the password they entered is compared to the hash stored in the database. The actual passwords are never compared. If the hashes match access is granted, if not, access is denied.

### MD5

Message Digest Algorithm 5 (MD5) is one of the oldest and still most commonly used hashing functions, despite it being one of the easiest hashes to crack. MD5 always creates a 32-digit hexadecimal number and was designed to use collision-proof hashing meaning that two different strings would never have the same hash. However, since its creation in 1991 there have been multiple cases of identical hashes occurring. Since this became public, MD5 has often become the victim of collision hacking attempts where attempts are made to hack sites by finding one of these collision values. In 2008, Homeland Security declared the use of MD5 to protect websites as unsafe and recommended against its use altogether after it was found MD5 hashes could be used to falsely display SSL browsing (VU#836068).

### SHA256

Secure Hash Algorithm 256 (SHA256) was developed by the NSA and also uses a 32-bit algorithm. 256 stands for the number of rounds the string being hashed is passed through the algorithm. While SHA256 is widely regarded as one of the most secure basic hashing algorithms, it still suffers from collision issues and brute force attacks. At best, it has tested at 46 out of 64 rounds in a collision attack. The hash is still widely used by many common web applications and security certificates, for example SSL and SSH. The algorithms for SHA is still being actively developed, ensuring it has a higher level of security that MD5, but this also requires constant updates from developers, especially as they cannot update existing user passwords in their systems to the new hashes.

### RipeMD

Race Integrity Primitives Evaluation Message Digest (RipeMD) is a hashing algorithm developed by the open academic community and is less frequently used than the aforementioned hashes. The most common version is the 160-bit version, and 256-bit & 320-bit versions also exist. RipeMD uses 40-digit hexadecimal number hashes and has not had any collision attack success reported for the 160-bit version. This hash is popular in C# .NET applications, but not as much is known about it due to it not being as widespread in use.

### BCrypt

BCrypt was designed in 1999 and uses a built-in salting algorithm during encryption. It is based off the Blowfish algorithm which uses block encryption to replace keys in sections. BCrypt is also one of the few hashing algorithms that uses a technique known as key stretching which slows down the encryption process. Key stretching slows down brute force attacks and minimizes their impact as each password they try takes longer to process and return a response. This makes BCrypt popular, but it is also more resource heavy.

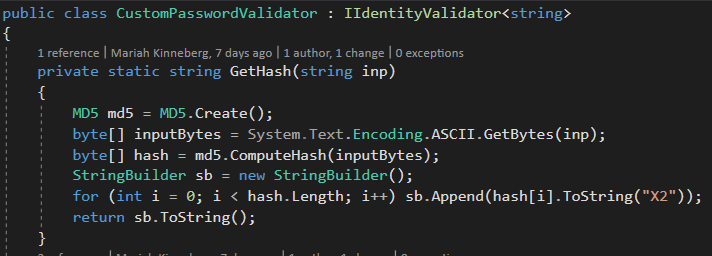
# Conclusions

To conclude our research and test our theories, we implemented a login system for an ASP.NET web framework. We constructed an algorithm to strengthen the security of the user sign-in, and the protection of the user’s data being stored on the site.

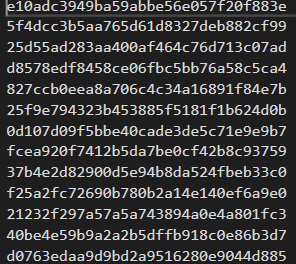
## Algorithm

Our algorithm consists of three parts, the first being data storage. To enhance the security of the data stored by the website, all e-mails are hashed using SHA-256 hashing. This means that if the database were to be compromised, no information that is of any value can be obtained.

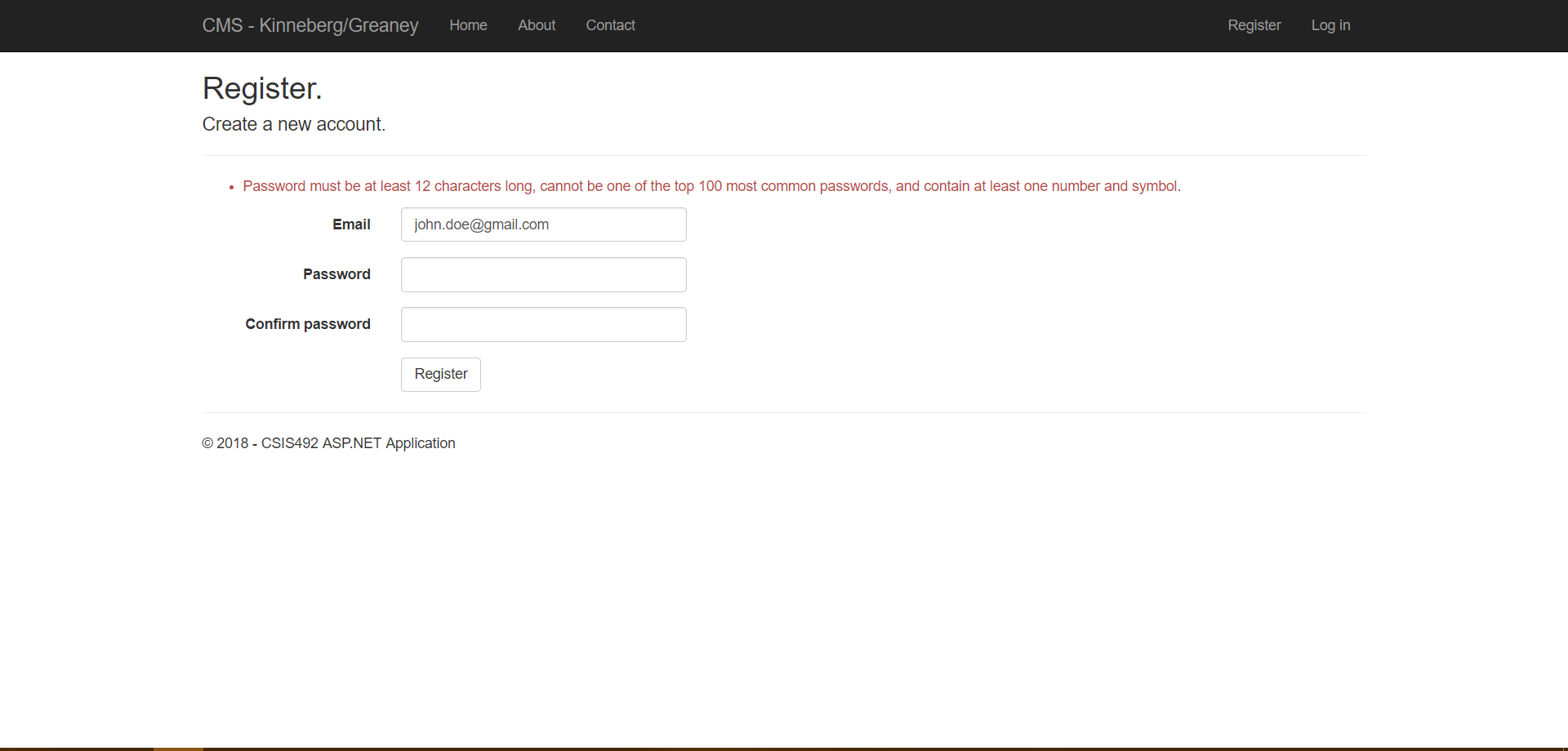
The second part of the algorithm focused on the complexity of the password. The length of the password is set to be at least 12 characters long. The password is also required to be unusual and uncommon, so that it is not likely be present in a brute force dictionary. This verification is performed by comparing the user’s entered password to a text file containing commonly used passwords stored as MD5 hashes. Passwords were hashed to preserve the integrity of the text file should it be compromised.



*Figure 2: Generation of MD5 hash of user's entered password to compare against text file of common passwords*

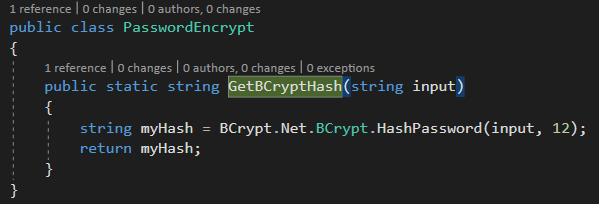


*Figure 3: Example of the text file storing common passwords*

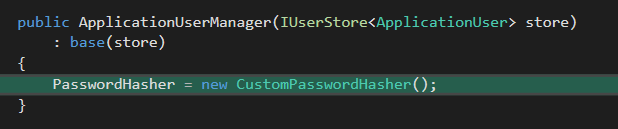


*Figure 4: Example of the error message when a user tries to register with the password "password"*

The default hashing method for the password by ASP .NET was changed to use a more secure password hashing method. BCrypt was selected due to its level of encryption, built in salt handling and ability to control the speed of hashing. A work factor of 12 was selected to slow down the hashing, so as to render brute force attacks useless, but not compromise user experience by slowing down the process too much. The current process operates at approximately .25 of a second. The BCrypt hashing was perfomed using the BCrypt-Official NuGet package. Two methods were created for hashing the password, HashPassword() returns a BCrypt hashed password that is created using GetBCryptHash(). The password is then verified against the database using VerifyHashedPassword(). It was then required to modify the Application Manager to allow the use of a custom password hash.



*Figure 5: Generation of a BCrypt Hash*



*Figure 6: Uses the CustomPasswordHasher to hash all passwords for the system*

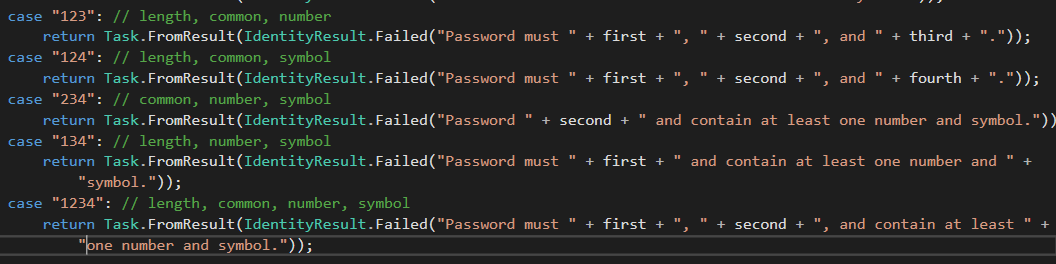
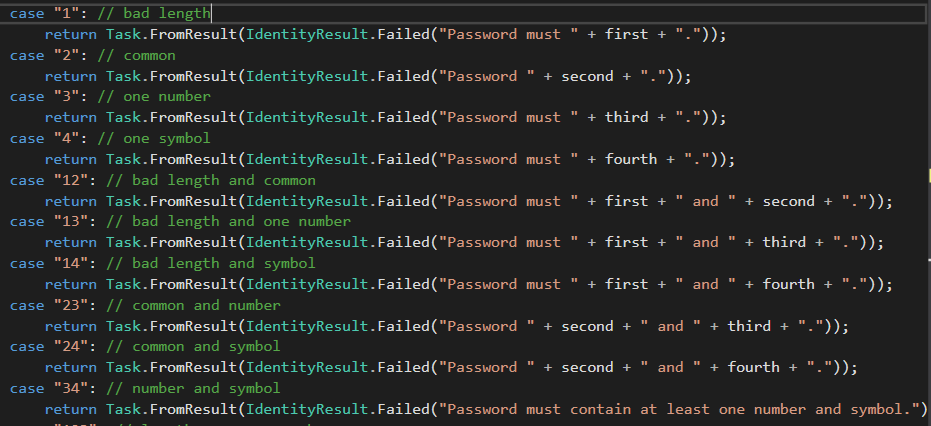
The third part of the algorithm requires the user’s password to be of a certain complexity. Passwords are required to consist of at least one number, one symbol, and both upper and lower-case letters. Examples below are of the number and symbol verifications, and the tests the password must pass to be considered a valid password.



*Figure 7: Checking if the password contains at least one number*

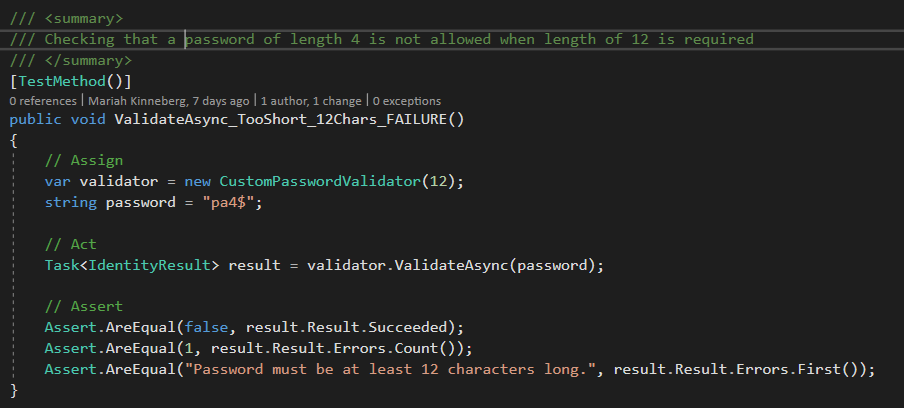


*Figure 8: Checking if the password contains at least one symbol*



*Figure 9: Generating the error message*

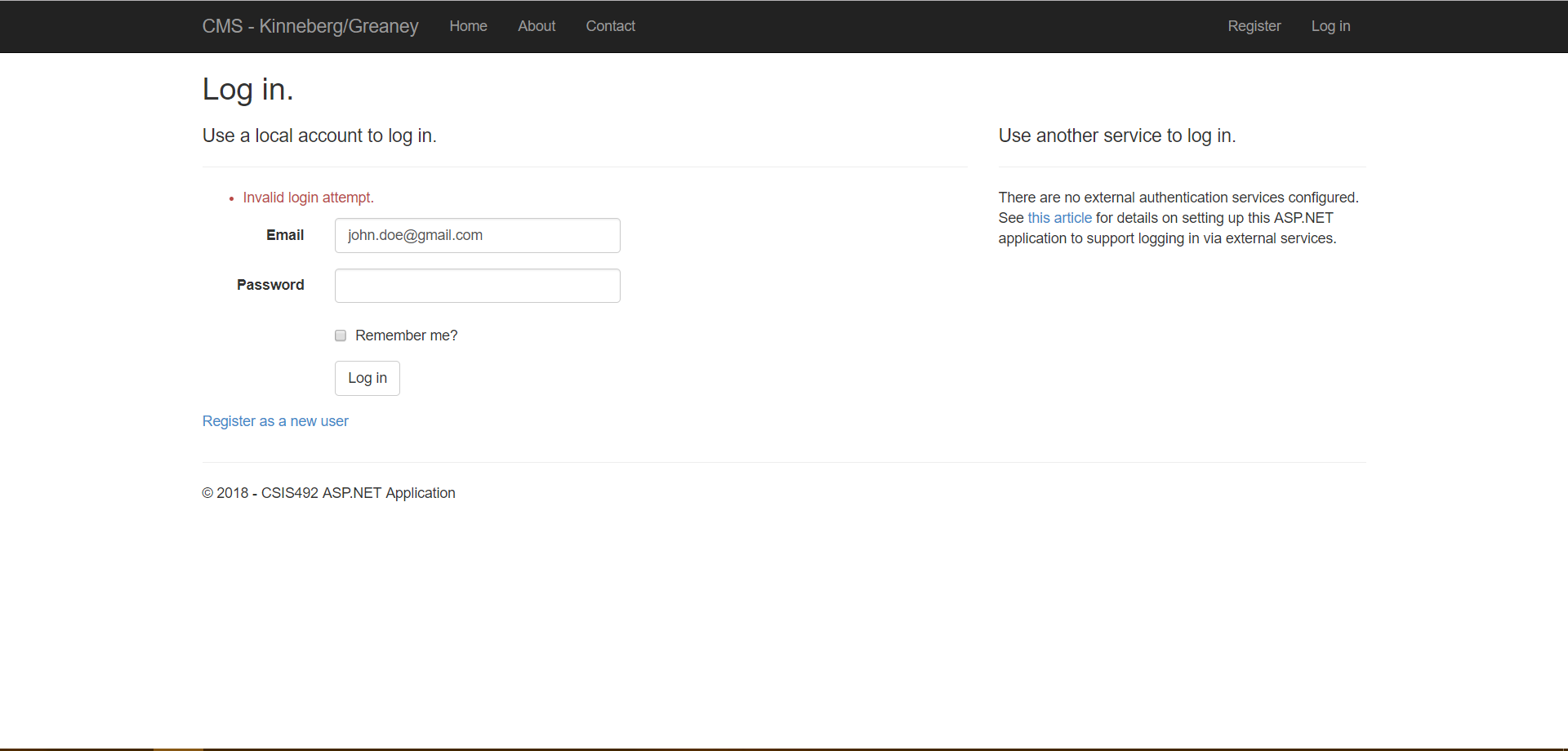
These rules also show how the error messages to the user attempting to create the password were generated. To verify that all the criteria were being filled and no cases were being missed, 18 unit tests were created to verify the password complexity. One example of these is shown below.



*Figure 10: Example of one of the unit tests that was written*

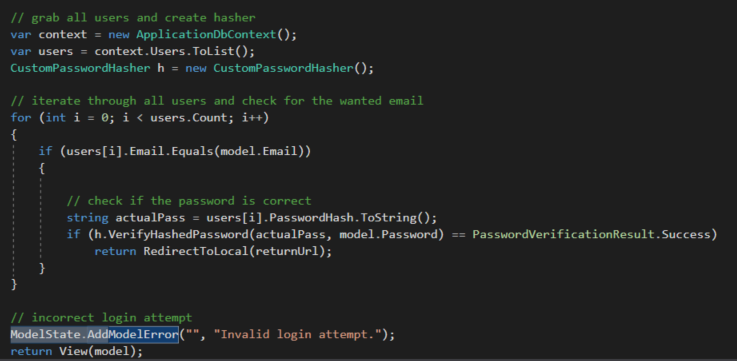
## User Information

Once the user created an account with a valid password, the user is able to sign in and out of that account as needed. One important part of this is informing the user when they have entered incorrect login information. The original information provided by ASP.NET was changed to use a generic “Invalid login attempt” message. This means that less information is provided to a potential malicious user if they are attempting to gain access to someone’s account.



*Figure 11: Example of login attempt with invalid credentials*

This login change was implemented by editing the Login method that checks whether or not the account exists, which is shown in Figure 12. The list of emails is grabbed from the ASP.NET database that stores logins. If the email used to login doesn’t exist in this table or the hashed password doesn’t match the password that is associated with this email in the table, the “Invalid login attempt” message is shown.



*Figure 12: Verifying whether or not there exists an account with the given login credentials*

## Challenges

One of the main challenges that was encountered during this implementation was accessing the user information and data structure. User data is stored in a separate database from the website’s operational database and accessing this requires the use of specific controllers that must be hooked into. This required the system to be developed slightly differently, and an access table to be created in the operational database to implement some of the security functionality. ASP.NET also uses many tables with relational information instead of storing all information in one table.

# References

[1] Christianson, Curt, and Jeff Cochran. *ASP.NET 3.5 content management system development build, manage, and extend your own content management system.* Packt Publishing Ltd., 2009.

[2] Clobridge, Abby. "WordPress: The Most Popular (Open) Web Content Management System." Online Searcher. Information Today, Inc. 2016. *HighBeam Research*. pp. 1-3.

[3] Fraser, Stephen R. G. *Real World ASP.NET: Building a Content Management System*. 1st ed., vol. 1 1, Apress, 2002.

[4] Sara Connell, Ruth. (2013). Content Management Systems: Trends in Academic Libraries. *Information Technology and Libraries,* vol 32, no. 2, 10 June 2013, pp. 42-54.

[5] Chae, Hyun Kyung, et al. “Challenges to Music Documentation: Design and Implementation of a Web-Based Content Management System for East Asian Music Education Documents.” *Fontes Artis Musicae*, vol. 61, no. 3, 2014, pp. 249–259.

[6] Clarke, Russell, et al. “Is Open Source Software More Secure?” *Homeland Security / Cyber Security*, 2005, pp. 1–34.

[7] “WordPress Privileges Unchecked in admin.Php and Multiple Information Disclosures.” *Core Security*, Core Security SDI Corporation, 6 Mar. 2017, www.coresecurity.com/content/WordPress-Privileges-Unchecked.

[8] “WordPress.org.” *Blog Tool, Publishing Platform, and CMS - WordPress*, wordpress.org/.

[9] Souer, Jurriaan, et al. “A Framework for Web Content Management System Operations and Maintenance.” *Journal of Digital Information Management*, vol. 6, no. 4, Aug. 2008.

[10] Reddy, Sandeep, et al. “Identifying an Appropriate Content Management System to Develop Clinical Practice Guidelines: A Perspective.” *Health Informatics Journal*, vol. 23, no. 1, 1 Mar. 2017, pp. 14–34., doi: https://doi.org/10.1177/146045815616264.

[11] Petković, Milan, and Willem Jonker. *Security, Privacy, and Trust in Modern Data Management.* 1st ed., Springer, 2010.

[12] Daitch, Heidi. “2017 Data Breaches - The Worst Breaches, So Far.” *IdentityForce*, IdentityForce, Inc., 19 Dec. 2017, www.identityforce.com/blog/2017-data-breaches.

[13] Korosec, Kirsten . “The 25 Most Common Passwords of 2017 Include 'Star Wars'.” *Fortune*, Time Inc, 19 Dec. 2017, fortune.com/2017/12/19/the-25-most-used-hackable-passwords-2017-star-wars-freedom.

[14] “Worst Passwords of 2017 - Top 100.” *TeamsID*, SplashData, 19 Dec. 2017, 13639-presscdn-0-80-pagely.netdna-ssl.com/wp-content/uploads/2017/12/Top-100-Worst-Passwords-of-2017a.pdf.

[15] Hornby, Taylor. “Salted Password Hashing - Doing it Right.” *CrackStation*, DefuseSecurity, 1 Aug. 2017, crackstation.net/hashing-security.htm#salt.

[16] Steven, John, et al. “Password Storage Cheat Sheet.” *Password Storage Cheat Sheet - OWASP*, The Open Web Application Security Project (OWASP), 19 Feb. 2018, www.owasp.org/index.php/Password\_Storage\_Cheat\_Sheet.

1. Extensions and add-ons available for a software to extend functionality [↑](#footnote-ref-1)
2. Plugins are automatically included and installed with the theme [↑](#footnote-ref-2)
3. A website that has fresh content being updated on a daily or weekly basis, often concerning daily events [↑](#footnote-ref-3)
4. Extension that can be installed to alter the visual appearance of the website [↑](#footnote-ref-4)