Rachel Engel – rachel[at]isecpartners[dot]com

iSEC Partners, Inc
123 Mission Street, Suite 1020
San Francisco, Ca 94105
https://www.isecpartners.com

July 3, 2013

Abstract

Login and password reset services exist in just about every web application. They're easy pieces of functionality to think about, but include a few common bugs used by attackers to compromise account credentials. This paper discusses security vulnerabilities related to web login services, highlighting possible implementation pitfalls along the way.

1 OVERVIEW OF LOGIN SERVICE SECURITY

Login forms are deceptively simple to develop. They take an HTTP request containing a username and password as input and result in a decision on whether the user is permitted to use the service. Intuitively, if the user submits the right password for the username submitted, they are the correct user. However, the user submitting the username and password might actually be an attacker trying to guess the user's password. How best to decide which is which?

You cannot stop an attacker from trying to guess the right password for a user. An attacker may well guess the correct answer on the first attempt. The best you can do is make successful password guessing less likely. If the occasional correctly guessed password was the only threat to login form security, compromised user accounts would be far less common. In reality, login services often enable rapid guessing on the part of the attacker, making breaches far more likely. With such flaws at the disposal of attackers, account compromises are all but certain. The trick in constructing login forms correctly is to remove these common technical flaws, and in the process ensure that success for the attacker will require a huge and unlikely stroke of luck.

2. THE ATTACKER IS A COMPUTER

The attacker's goal is to successfully submit the correct password for a user on a website. The attacker doesn't need to sit around trying to imagine what the user's password is, and the attacker definitely doesn't need to type it. Attackers will generally write a program or script to automatically submit guess after guess to the login service until they hit upon the correct answer. There are two ways of making this guessing process harder:

- Make the password more complicated. It is not sufficient for passwords to be a secret that only you know. The attacker is not guessing the password, a very fast computer is. The password must be a very complicated sequence of characters, ideally random. The attacker is relying on exhausting all possible password values, meaning a very simple password will be discovered early in the process. Using an

utterly contrived example, if the secret is a letter between *a* and *z*, the attacker can submit every letter between *a* and *z* and be logged in once they get the correct one. If you could convince all users on your website to have passwords of adequate length/complexity, it would take too long for the attacker to guess it to be worth it. Many people have berated users over the years for having easy to guess passwords, but this overlooks the UI complexity of remembering complex passwords. The user wants to accomplish a task, not spend time memorizing 12-16 character strings. Ideally we should have users be able to log in *and* not be unduly burdened by password requirements.

- Make guessing take a lot longer. A person sitting at a terminal would take a very long time to guess the correct password if they had to submit every possible password of 6-10 characters. A clever brute forcing program will go through common password choices at a rate of thousands of passwords per second. The advantage the attacker has here is speed. The appropriate defense is to slow them down. If the attacker can only submit answers as quickly as a person would, they can no longer exhaust the possible passwords as quickly.

3. PASSWORD QUALITY IN PRACTICE

Much is made of how terrible user passwords are in popular news articles about computer security. People are given a cacophony of rules to follow that make passwords more complicated. Users are told to think up something secret and put a number in it; perhaps substitute an exclamation point or an @ sign for a letter. To the average user, their grandmother's maiden name probably does seem secret, and it is easy to remember. They are really given no guidance as to what specifically the security model of the website is expecting of them. With the rise in computational power and advancement in the state of the art in brute force guessing, passwords must not only be secrets another person cannot guess, but they must also be a secret that computers cannot guess.

Ideally, the password will remain unguessed for as long as the user needs the data to remain secret. A person may give up guessing after a few minutes of trying common names and dates, whereas a computer can spin through every imaginable character sequence without ceasing for days, weeks, months, or years. It takes a very complex password to prevent such a determined and speedy attacker – ideally that password would be a long random sequence of bits. This would force the brute forcing algorithm to try every combination of password in order to guess correctly. This is not often communicated to the user. Users are not security engineers.

A study by Microsoft research in 2007 (http://research.microsoft.com/pubs/74164/www2007.pdf) gives some insight into how many guesses an attacker needs to be able to guess a password correctly. The study provides data on the password complexity of a population of actual users. Many of the users surveyed have very low password complexity, with a maximum of a few million possible passwords, assuming that passwords are evenly spread out across the set of possible passwords. In practice, users base their passwords on common words, phrases, names, dates, and other relevant personal information, limiting even further the number of guesses needed by an attacker. The attacker doesn't need to try every possible password; a carefully chosen dictionary of common passwords will reduce the number of needed guesses dramatically. With the number of recent password leaks, attackers have ample data on common passwords to source from.

4. BRUTE FORCE MITIGATION ANTI-PATTERNS

Attackers have efficient scripts for making guesses. Instead of submitting each password guess manually, an attacker can build a script with hundreds of threads, each one submitting password guess upon password guess to maximize the throughput of the attacker's connection. If the attacker can make an endless number of guesses, and user passwords are of very low complexity, the attacker will likely guess one or more passwords for valid user accounts before long. We can mitigate this somewhat by taking aim at the attacker's

ability to submit password guesses. A number of ineffective approaches are often taken, so we'll step through each approach before discussing more effective strategies.

Approach 1: Lock the user account after a number of unsuccessful password attempts

If an attacker is sending endless password attempts, why not lock a user's account after a few tries? If the account is locked, the attacker cannot access it, and the correct user can call customer service to have the account unlocked. This stops brute force attacks against user accounts, but the attacker can take a different approach and deny service to users on the system by locking their account out on purpose. Once an attacker begins such an attack, a patch and redeployment of the login service is required to mitigate the attack.

Approach 2: Lock the user account upon repeated incorrect passwords, but automatically re-enable the accounts after a short period of time

Although this approach attempts to stop the denial of service threat by re-enabling the account, the denial of service attack is still possible. The attacker submits the incorrect passwords as before, waits until the lockout period ends, and resubmits incorrect passwords again. The effect of this defense is that the attacker is still able to deny service for user accounts on the website. Any scheme denying service to the user ultimately runs into this problem. What we need instead is a way to stop the attacker that doesn't prevent the legitimate user from accessing the login service.

Approach 3: Slow the attacker down

The goal should be to find an approach that prevents attackers from submitting requests at computer speed (many thousands of guesses per second), while permitting users to enter their password with little noticeable delay. A user typing their password as fast as possible will still take a few tenths of a second to enter their password. Instead of just accepting password after password as rapidly as our service can process them, we can require a human-length delay between password submissions. Take the following algorithm as a starting point:

- A failed login attempt is made.
- If a second password attempt is received a few milliseconds later, the request is blocked for short period.
- After every password attempt, we increase the delay period up to a maximum of several seconds (something long enough to thwart the script but short enough to not irritate the user).

A user is unlikely to really notice a tenth of a second delay between password attempts, as it takes them longer than that to type their password in. After a suitable length of time without a login attempt, the delay can be reduced back to the default. This gets at a couple of the goals we initially outlined for a successful login form.

- If a user attempts to log in during the brute forcing attack, they will still be able to type in their password (after a small delay).
- Automated attack scripts are forced to guess passwords at the same rate as normal users.

Non-technical users often have a mental model of an attacker sitting at a keyboard typing random guesses hoping to log in as the user. By making a computer take as long to guess passwords as a human would, we make the model of the security of the login service match our mental model a little more closely, and make the penalty incurred by a user whose account is under attack a little less severe.

A few things should be kept in mind when implementing this delay. An attacker can open up multiple connections to the login service when guessing, making implementing the delay on a per-connection basis insufficient. The delay must exist across all login attempts for the user.

Similarly, the attacker can attempt password guessing across all users on the system. The attacker could measure the delay between attempts to get a profile of the throttling algorithm used by the website, and when the throttling gets too onerous, move on to brute forcing a different account. This permits attackers to keep the rate at which they guess passwords fairly constant.

APPROACH 4: CAPTCHAS

A full treatment of Captchas is beyond the scope of this paper, but the basic approach is easy to summarize here. Include an image that displays a message, and have the user type the message. The message should be difficult or impossible for a computer to extract from the image. The idea is to detect specifically if a computer is submitting requests.

SUCCESSFUL PATTERN: CAPTCHA + EXPONENTIAL BACKOFF

An adequate deterrent to the problem of computerized brute forcing can be found by exponentially increasing the delays between attempts after every failed login and requiring the user to solve a well-engineered Captcha. For every failed login, double the length of the delay between permitted logins for the account up to a maximum of five to ten minutes, then reset. After the first few failed logins, require the user to solve a Captcha. Even if the attacker successfully breaks the Captcha, the system will produce severe enough delays in processing password attempts that brute forcing would take too long to be worth it to the attacker, while the user is likely to get the password typed in correctly in the first few logins before the Captcha is applied.

MULTI-ACCOUNT ATTACKS

It is often assumed that attackers will go through one account at a time, attempting every password for that one account before moving to the next account. Often, attackers will brute force many accounts on a server at the same time, attempting a common dictionary against every known account. While a simple dictionary may not work against a single account (depending on the password complexity), it is likely that a simple dictionary applied across a great many accounts will result in multiple successful logins. Consider applying a monitoring strategy that looks for the same password being applied across many accounts in a limited period. If the same password is applied within a short time range across many accounts, this is a red flag that a brute forcing attack is likely in progress.

5. PASSWORD RESET SECURITY

Handling the need of users to reset forgotten passwords is another basic web application service that's easy to get wrong. Someone is attempting to reset a user's password, but there is no reason to believe that the person resetting the password is the actual user. Worse yet, a user is identified by the web application as the authenticated user by having the correct password. In this scenario, the user no longer has the correct password. This often makes password reset functionality behave as a less secure second login form for people who don't know their password. While every user has at one time or another forgotten their password, attackers also don't know the user's password, and have good reason to go after the soft target of the password reset form.

It is often said that the three ways to authenticate a person are by something the user knows, something the user has, or something the user is (biometric authentication). In the case of a login form, authentication relies on something the user knows (their password). If the user has forgotten their password, we need to rely

on something else. Email accounts fit the bill of being something the user has, making them the most commonly used authentication factor to fall back on.

Approach 1: Send the password to the user by email

Most users have access to an email account and most web applications have the user's email address, making it a convenient way to help the user. It's also fairly common for email to be sent unencrypted over the network, making password reset emails easy to intercept. It's not just possible to read and edit traffic on local networks for an attacker, it's downright easy. An entire genre of security tools and techniques that are well understood by security engineers and security enthusiasts exist to let people read and modify traffic on computer networks. Ensuring that there are no attackers reading traffic on a local network is next to impossible. The protocols used to enable transport of data on computer networks just aren't designed with security in mind. Going back to the email scenario, if you send the password to the user's email account, the password will travel in the clear over the user's local network. An attacker with a few hours and access to the internet can download the tools needed to view the password.

Approach 2: Send the user a single-use link by email

The website looks up the random number in a table of outstanding password resets, gets the user record associated with the random number, and presents a password reset form. The website at this point either displays the password associated with the user, or offers a field for the user to enter their new password. The logic of this again rests on the idea that only the user will have access to the URL in their email account, and the number used is too long to guess by brute force.

Once again, the problem comes down to the insecurity of email. The link in the example above is HTTPS, which is good, but the https link is sent to the user's email client, possibly in the clear. Like the previous method, an attacker on the local network can fill out the password reset form on the website, watch the link sail by on the network, load the page, and reset the password. None of the methods that rely solely on the user being the only one with access to their email account work very well. The link to the password reset form must be hosted over SSL, and the form itself must rely on something the user knows.

Approach 3: What's your favorite sports team?

Now we are coming to methods that are a little stronger. The next step is usually to ask the user for something that only the user would know before resetting their password. Sadly, the password is the most complicated piece of information that only the user would know, and the user is resetting their password because they don't have it. The next algorithm usually looks something like:

1. User goes to the website, requesting that their password be reset.
2. An HTTPS URL is sent to the user, directing the user to a form that requests the user answer a few questions.
3. The user answers the questions, and if the answers are correct, the password is reset.

The user can be identified by a long, unique nonce included in the URL set to the user. When the user clicks on the URL sent to their email, it sends the unique nonce to the password reset form. We now know that at the very least the user has access to the email of the person who registered the account. To authenticate this person, we ask them a question the user is certain to know. The recipient doesn't know the user's password, and may either be the user or an attacker. We can't really have the user set up a second password, meaning any information we can ask them for will be less complex than the ideal. The ability to select the correct user comes down to the quality of the information selected.

If the person using the reset form is asked questions available publicly about the person, the reset form is fairly useless. Information in this category: phone number, home address, etcetera. The information should also not be something commonly available on the web (through web searches or social networks). There are a number of common questions that reset forms ask, all requiring somewhat low quality information that could be easily gained by snooping into the user's background, for example:

- First grade teacher
- Elementary school
- Mother's maiden name
- Grandmother's first name
- Childhood pet
- Favorite sports team

All of these are obviously lower quality secrets than a password. With a password you can attempt to require the user to enter information that would take a truly long time to guess, even for a computer. Most of the above information is probably available to other people in the user's life, as well as random people unaffiliated with the user. Questions like the above are very common, common enough even that it introduces another source of security concern. If a user registers accounts at 10-20 web applications, all of which ask them for their mother's maiden name or favored sports team, that information is no longer terribly secret. Administrators at the web applications the user uses have access to the information, and if one of the applications gets compromised, attackers will also have access to the information. In the password reset scenario, we're trying to verify that the person using the application is the correct user without an adequate secret, so an approach like the above is frequently the best effort a web application can make. If you're going to use the above approach, try to make the questions novel. It's not as good as a secret, and never will be, but it's better than nothing.

APPROACH 4: SEND A PIN CODE TO THE USER'S CELL PHONE

Another promising approach has been used recently, and will hopefully see more use in the future. If the user is asked for their phone number in the initial setup of their account, users will often have access to the ability to use text messages to verify their information. While increasingly under security scrutiny in recent years, cellular networks are nonetheless harder targets to attack than local area computer networks. More and more applications will attempt an algorithm like the following:

1. User enters username into password reset form
2. A random number is sent via text message to the user's phone
3. The user is directed to a form that asks for this number

This has a number of advantages. Although there has been much security research in recent years into cell phone networks, intercepting cellular traffic is still a far harder target, and the likelihood that the attacker of the website has already targeted the user's phone is fairly low. We can assume the number sent to the user's cell phone is at least somewhat more secret.

6. USERNAME HARVESTING

Attacks on the login form or password reset form require that the attacker know valid usernames. Usernames and email addresses are not designed to be secret, but ideally we won't do anything to make the lives of attackers easier by giving them the usernames. Attackers can often harvest usernames from websites by a mild form of vulnerability often found in login forms, password reset forms, and account registration forms. The broken functionality usually goes something like the following algorithms:

Scenario 1:

- User enters a username that doesn't exist
- Website helpfully informs the user that the account doesn't exist on the server

Scenario 2

- User enters a username that does exist, but an incorrect password
- Website helpfully informs the user that the password is incorrect

An attacker can use the structure of the error reporting to validate guessed usernames. If they enter a username that doesn't exist, the website informs them that the user doesn't exist. If they enter a name that does exist, the website tells them that they entered the wrong password. They can use this to guess valid login names on the server by brute force. After a list of valid usernames is harvested, the attacker moves on to the password guessing techniques mentioned earlier. Instead, if we give an identical error message for both incorrect usernames and non-existent accounts, the attacker can no longer use this information to obtain a list of usernames ("invalid login credentials" for example). Similar attacks can be performed against password reset forms, account registration forms, and login forms.

7. CONCLUSION

Login services have many small flaws that can in aggregate make user accounts with weak passwords an easy target, but the difficulty of getting login service implementation correct highlights several larger-grain problems with the general use of login forms on the web. If it takes a background in web application security engineering to get login form basics correct, this leaves most websites out in the cold. Most websites rely on passwords for authenticating users, and are run by people whose time would be far better spent pursuing the interest that led to the development of the website (pet toys, mandolins, sports enthusiasm, etc) than pursuing correct implementation of login services.

From the perspective of the user, life is just as difficult. Most users have no desire to memorize 12-16 character non-dictionary passwords. Worse yet, as users frequently reuse passwords across many websites, the aggregate security of their data on each website reduces to the security of the most poorly secured of these websites. The likelihood that every website with the user's "govikings!" password will have hardened login services is very low. Many articles have been written over the years citing the lack of complexity in passwords chosen by users as a major source of security concern, and while it's true that users choose poor passwords, it's also true that they should be able to securely use websites with credentials that they do not find onerous. Having an engineering solution that takes care of managing adequate security credentials for web accounts in a way that users are not burdened by is an engineering challenge that would protect users and improve website security.