MONASH University

FIT5032 - Internet Applications Development

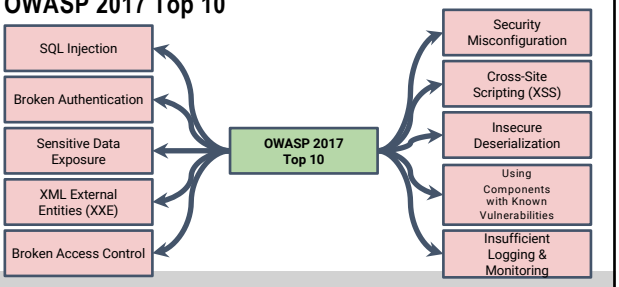
Security and Microsoft Identity

Prepared - Jian Liew
Updated by ABM Russel

OWASP

- The Open Web Application Security Project (OWASP) is a non-profit organization dedicated to providing **unbiased, practical information about application security**.
- The OWASP Top 10 Web Application Security Risks was updated in 2017 to provide guidance to developers and security professionals on the most critical vulnerabilities that are commonly found in web applications, which are also easy to exploit.
- These 10 application risks are dangerous because they may allow attackers to plant malware, steal data, or completely take over your computers or web servers.
- As many as 25 percent of web apps today are vulnerable to eight of the entries on the OWASP Top 10, according to Contrast Security research, and 80 percent had at least one vulnerability.
- As a developer, there is a need to understand the potential vulnerabilities our web application has.

OWASP 2017 Top 10



OWASP Top 10

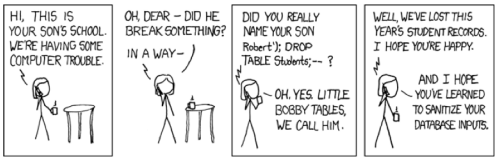
No	Description
A1:2017-**Injection**	Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2:2017-**Broken Authentication**	Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.
A3:2017-**Sensitive Data Exposure**	Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption **at rest or in transit**, and requires special precautions when exchanged with the browser.

OWASP Top 10

No	Description
A4:2017-**XML External Entities (XXE)**	Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.
A5:2017-**Broken Access Control**	**Restrictions on what authenticated users are allowed to do are often not properly enforced**. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.
A6:2017-**Security Misconfiguration**	**Security misconfiguration** is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched and upgraded in a timely fashion.

No	Description
A7:2017-**Cross-Site Scripting (XSS)**	XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to **execute scripts in the victim's browser** which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A8:2017-**Insecure Deserialization**	**Insecure deserialization** often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.
A9:2017-**Using Components with Known Vulnerabilities**	Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with **known vulnerabilities** may undermine application defenses and enable various attacks and impacts.
A10:2017-**Insufficient Logging & Monitoring**	**Insufficient logging and monitoring**, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data.

Relevant XKCD



Lack of careful parsing is a **common SQL vulnerability;** this type of exploit is referred to as SQL injection. Mrs. Roberts (stick figure above) thus reminds the school to make sure they have added **data filtering code to prevent code injection exploits in the future.**

.NET Security Guidance

Data Access

- Use Parameterized SQL commands for all data access, without exception. (This prevents SQL Injection)
- Do not use SqlCommand with a string parameter made up of a concatenated SQL String. (If you are using an ORM, you would avoid this, in other words **you must use the ORM correctly.)**
- Use of the Entity Framework is a very effective SQL injection prevention mechanism. Remember that building your own ad hoc queries in EF is just as susceptible to SQLi as a plain SQL query.
- Whitelist allowable values coming from the user. Use enums, TryParse or lookup values to assure that the data coming from the user is as expected.

Encryption

- **Never, ever write your own encryption**.
- Use a strong hash algorithm. In .NET (both Framework and Core) the strongest hashing algorithm for general hashing requirements is System.Security.Cryptography.SHA512.
- In the .NET framework the strongest algorithm for password hashing is PBKDF2, implemented as System.Security.Cryptography.Rfc2898DeriveBytes.
- In .NET Core the strongest algorithm for password hashing is PBKDF2, implemented as Microsoft.AspNetCore.Cryptography.KeyDerivation.Pbkdf2 which has several significant advantages over Rfc2898DeriveBytes.
- When using a hashing function to hash non-unique inputs such as passwords, use a salt value added to the original value before hashing.

Security Question

- There is no industry standard either for providing guidance to users or developers when using or implementing a Forgot Password feature.
- In the 2000s, security questions came into widespread use on the Internet.
- Theoretically, a security question is a **shared secret** between the user and the website.
- As a form of self-service password reset, security questions have reduced information technology help desk costs.
- It is important to remember that a security question is just another password.

Desired Characteristics of Security Questions

Memorable	If users can't remember their answers to their security questions, you have achieved nothing.
Consistent	The user's answers should not change over time. For instance, asking "What is the name of your significant other?" may have a different answer 5 years from now.
Safe	The answers to security questions should not be something that is easily guessed, or research (e.g., something that is matter of public record).
Nearly Universal	The security questions should apply to a wide an audience of possible.

Memorable

- A security question should be easy to remember but still not available to others. Ideally, the user should immediately know the answer without looking up a reference or having to write down the answer.
- **Bad examples:**
  - What is your driver's license number?
  - What is your car registration number?
  - What was your first car, favorite elementary school teacher, first kiss, etc?
- Childhood questions can be difficult for older people. Try to use questions that are more prominent or memorable.

Consistent

- The answer to a good security question doesn't change over time.
- **Bad examples:**
  - Where did you vacation last year?
  - Where do you want to retire?
  - What is your favorite… anything
- One of the most common and worst type of question is "what is your favorite…."
- Current favorites can change. But historical favorites may be acceptable, because they can't change.
- For example, "what was your favorite sport in high school?" As long as the person is no longer in high school, that shouldn't change.

Safe

- The most important characteristic of a good security question is security – it does not compromise the very thing it is trying to protect.
- A good security question:
  - **cannot be easily guessed whether or not the attacker knows the user** (family member, close friend, relative, ex-spouse, or significant other)
  - **cannot be easily researched** – Facebook, G+, other social media, blogs, or research sites

Nearly Universal

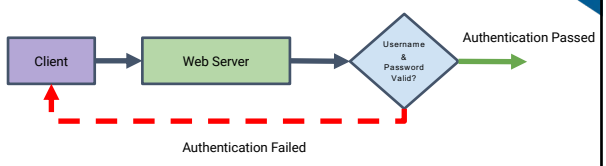
- It is recommended to use a question that is nearly universal and applies to everyone, however sometimes it is not possible.
- No one question works for all people.
- People need enough questions to select those that will work for them. Therefore, it is best to offer 2-3 sets of questions (more if site data is more sensitive) with a variety of questions.
- Permitting the user to create a question may increase user frustration and potential for security breach. Thus, it is recommended not to do so.

Authorization vs Authentication

Authentication is the process of verifying who you are. When you log on to a PC with a username and password you are authenticating. **Authentication is the process of ascertaining that somebody is really is who he claims to be.**

Authorization is the process of verifying that you have access to something. Gaining access to a resource (e.g. directory on a hard disk) because the permissions configured on it allow you access is authorization. In short, **authorization refers to rules that determine who is allowed to do what.**

Traditional Authentication



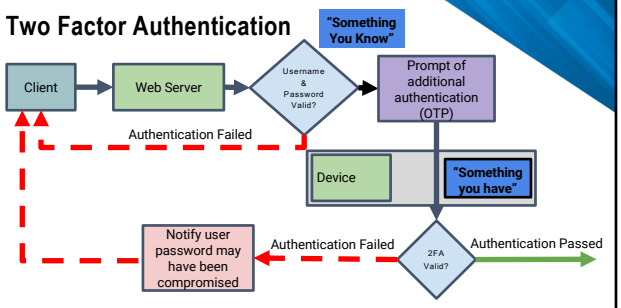
Multi-factor authentication

- Multi-factor authentication is one of the most effective controls an organisation can implement to prevent an adversary from gaining access to a device or network and accessing sensitive information (Our website in this case).
- Multi-factor authentication **differs** from multi-step authentication. **(So, two-factor authentication differs from two step authentication)**
- The Australian Cyber Security Centre (ACSC) recommends that **multi-factor authentication** is implemented for users using remote access solutions, users performing privileged actions and users accessing important (sensitive or high-availability) data repositories.
- Using multi-factor authentication provides a secure authentication mechanism that is not as susceptible to brute force attacks as traditional single-factor authentication methods using passwords or passphrase

What is multi factor authentication?

- The Australian Cyber Security Centre (ACSC), defines multi-factor authentication as 'a method of authentication that uses two or more authentication factors to authenticate a single claimant to a single authentication verifier.'
- The authentication factors that make up a multi-factor authentication request must come from two or more of the following:
  - something the claimant knows (e.g. a personal identification number (PIN), password or response to a challenge) **(Something you know)**
  - something the claimant has (e.g. a physical token, smart card or software certificate) **(Something you have)**
  - something the claimant is (e.g. a fingerprint or iris scan). **(Something you are)**

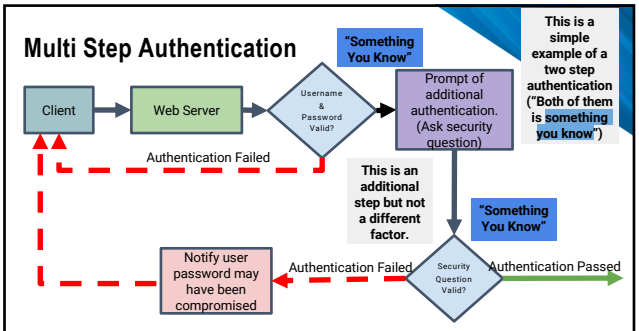
Two Factor Authentication



What is multi step authentication?

- In comparison to multi factor authentication, a multi step authentication is a scheme that has **multiple steps.**
- The main difference between multi-step in comparison to the multi-factor authentication is that in multi-step authentication **the steps can belong to the same factor.**
- So, a multi-step authentication scheme might require **two physical keys**, or **two passwords**, or **two forms of biometric identification** is not two-factor, but the two steps.
- In comparison to the multi factor, is that the attacker must **successfully pull off two different types** of theft to impersonate you in multi factor while in the multi-step, the theft does not.
- The type of multi-step authentication provided by Google or Facebook or Twitter is still strong enough to thwart most attackers, **but from a purist point of view, it technically isn't multi-factor authentication.**

Multi Step Authentication

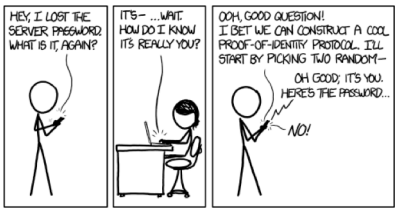


Advantages & Disadvantages of MFA

Multi factor authentication is important for **websites** because

- Passwords are the most common form of login authentication across the spectrum of technology. But they're also incredibly fallible (They are easily obtained by other means, social engineering , hack and etc) (Advantage)
- The primary benefit of multi factor authentication is that it provides **additional security by adding protection in layers. (Advantage)**
- Users must carry a mobile phone, charged, and kept in range of a cellular network, whenever authentication might be necessary. (Disadvantage)
- The user must share their personal mobile number with the provider, reducing personal privacy and potentially allowing spam. (Disadvantage)

Relevant XKCD



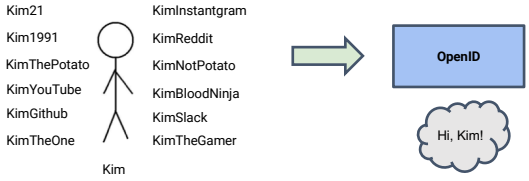
It is a good idea not to implement your own authorization and authentication features.

OpenID

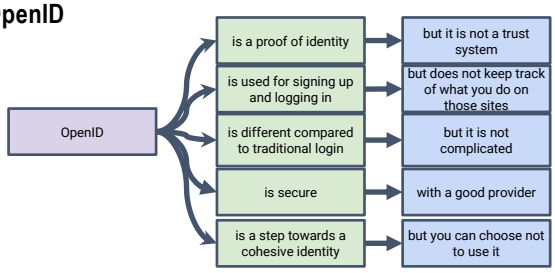
- An OpenID is a way of identifying yourself no matter which website you visit.
- OpenID is an **open standard and decentralized authentication protocol**.
- OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol.
- Current version of OpenID is OpenID Connect 1.0, finalized and published in February 2014. Older versions of OpenID is deprecated.
- It allows Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner.
- As of 2018, OpenID is no longer used by a number of companies in favour of OAuth 2.0 protocol. (Facebook used to use open ID but since moved to Facebook Connect)

OpenID explained…..

With OpenID you only have to **remember one username and one password.**



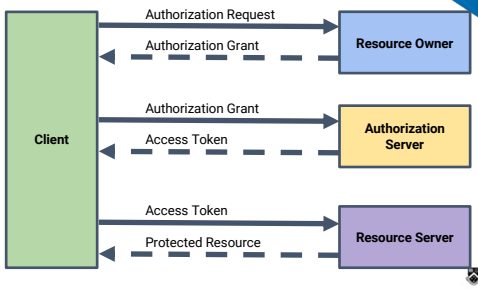
OpenID



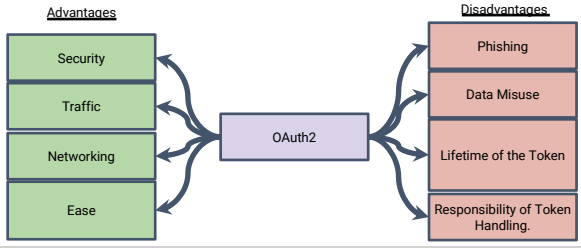
OAuth 2.0

- OAuth 2.0 is the industry-standard protocol for authorization.
- **Most social login are implemented using the OAuth standard.** (Facebook Login, Google Login)
- OAuth 2.0 supersedes the work done on the original OAuth protocol created in 2006.
- OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices.
- Authorization **can be used as a form of pseudo-authentication.**
- OAuth 2.0 protocol describes a **3 legged authentication.**
- **OAuth provides a method for clients to access a protected resource of behalf of a resource owner.**

How OAuth2 Works



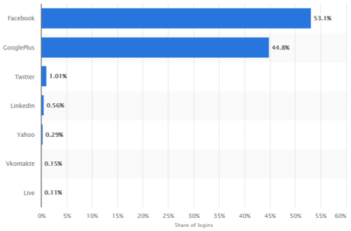
Pros and Cons of OAuth2



List of OAuth Providers

- Amazon
- BattleNet
- BitBucket
- Dropbox
- Github
- Google
- Instagram
- LinkedIn
- Twitter
- Yahoo

Social Login



Preferred social login as of 2016.

These days, it is almost required for you to provide a way to use a social login in any websites that you create.

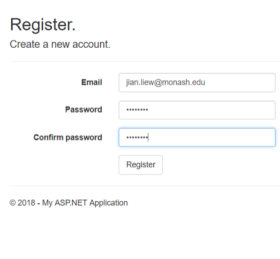
ASP.NET Identity Goals

- One ASP.NET Identity system
- Ease of plugging in profile data about the user
- Persistence control
- Unit testability
- Role provider
- Claims based
- Social Login Providers
- Azure Active Directory
- OWIN Integration
- NuGet package

ASP.NET Identity

- ASP.NET Identity is a **membership system** which allows you to add login functionality to your application.
- Users can create an account and login with a username and password.
- They can use an external login provider such as Facebook, Google, Microsoft Account, Twitter or others. These can be enabled at a later time.
- The implementation of Identity is done via CodeFirst and it can be extended to suit the needs of the developer.

Automatic Generated Registration Page

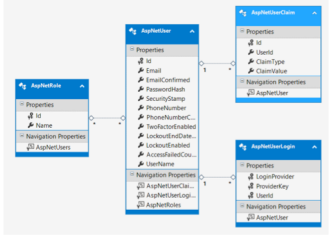


The default automatically generated Registration page only requires an email and password.

Most modern website only require this two information for a user to be registered to their website.

Other information can be obtained at a later time.

Model of the AspNet Users



By default, the identity features is generated via **Code First**. This model is generated via reverse engineering.

It is possible to look at the database schema of it once a user has registered to the web site.

It can be seen that there is a **many to many relationship between AspNetRole and AspNetUser**

This will be shown in the tutorials.

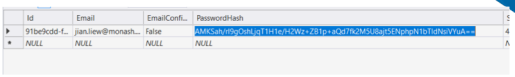
Hashing

- Hashing is an ideal way to store passwords, as hashes are inherently one-way in their nature.
- By storing passwords in hash format, it's very difficult for someone with access to the raw data to reverse it (assuming a strong hashing algorithm and appropriate salt has been used to generate it).
- When storing a password, **hash it with a salt**, and then with any future login attempts, hash the password the user enters and compare it with the stored hash.
- If the two match up, then it's virtually certain that the user entering the password entered the right one.
- **MS Identity uses the PBKDF2 to hash passwords.**

Encryption

- Encryption turns data into a series of unreadable characters, that aren't of a fixed length. The key difference between encryption and hashing is that encrypted strings can be reversed back into their original decrypted form if you have the right key.
- There are two primary types of encryption, symmetric key encryption and public key encryption. In symmetric key encryption, the key to both encrypt and decrypt is exactly the same. This is what most people think of when they think of encryption.
- Encryption should only ever be used over hashing when it is a necessity to decrypt the resulting message.
- This is the main reason why there is **no** "Retrieve Password" in any modern web applications. Passwords are always **hashed and not encrypted.** However, the general public tend to misuse these words. You **should never remind** the user what their password was.

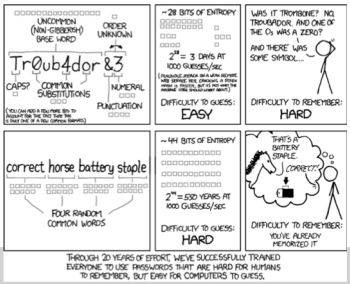
Password Hash



	Id	Email	EmailConfi...	PasswordHash
▶	91be9cdd-f...	jian.liew@monash...	False	AMKSah/rl9gOohLjqT1H1e/H2Wz+ZB1p+aQd7fk2M5U8ajt5ENphpN1bTtdNsiVYuA==
*	NULL	NULL	NULL	NULL

- By default passwords are stored as a password hash using PBKDF2.
- PBKDF2 applies a pseudorandom function, such as hash-based message authentication code (HMAC), to the input password or passphrase along with a salt value and repeats the process many times to produce a derived key, which can then be used as a cryptographic key in subsequent operation.
- The source codes can be found here.

Relevant XKCD

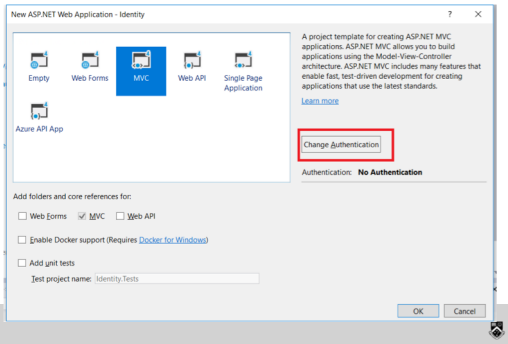


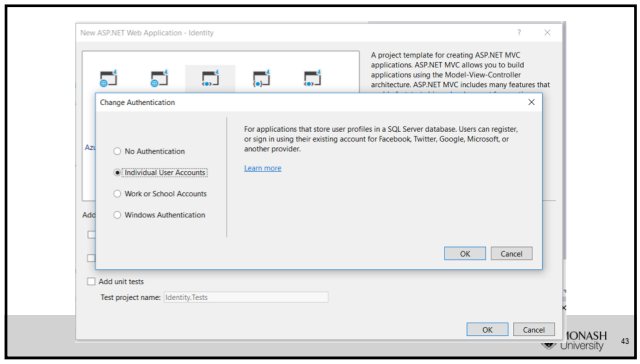
It is absolutely true that people make passwords hard to remember because they think they are "safer", and it is certainly true that length, all other things being equal, tends to make for very strong passwords.

In addition to being easier to remember, long strings of lowercase characters are also easier to type.

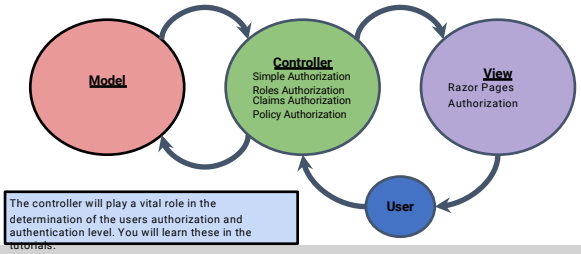
Authorization in ASP.NET MVC

- ASP.NET authorization provides a simple, declarative role and a rich policy-based model.
- Authorization is expressed in requirements, and handlers evaluate a user's claims against requirements. Imperative checks can be based on simple policies or policies which evaluate both the user identity and properties of the resource that the user is attempting to access.
- Authorization can be done in several different layers and ways
  - Razor Pages authorization
  - Simple authorization
  - Roles authorization
  - Claims authorization
  - Policy based authorization
- This can be achieved by having attribute annotations at the methods in the controller.





Overview of Authorization in ASP.NET MVC



The controller will play a vital role in the determination of the users authorization and authentication level. You will learn these in the tutorials.

Importance of HTTPS

HTTPS protects the privacy and security of your users

- HTTPS prevents intruders from being able to passively listen to communications between your websites and your users.
- One common misconception about HTTPS is that the only websites that need HTTPS are those that handle sensitive communications.
- Every unprotected HTTP request can potentially reveal information about the behaviors and identities of your users.
- Although a single visit to one of your unprotected websites may seem benign, some intruders look at the aggregate browsing activities of your users to make inferences about their behaviors and intentions, and to de-anonymize their identities.
- For example, employees might inadvertently disclose sensitive health conditions to their employers just by reading unprotected medical articles.

HTTPS protects the integrity of your website

- HTTPS helps prevent intruders from tampering with the communications between your websites and your users' browsers.
- Intruders include intentionally malicious attackers, and legitimate but intrusive companies, such as ISPs or hotels that inject ads into pages.
- Intruders exploit unprotected communications to trick your users into giving up sensitive information or installing malware, or to insert their own advertisements into your resources.
- For example, some third parties inject advertisements into websites that potentially break user experiences and create security vulnerabilities.
- Intruders exploit every unprotected resource that travels between your websites and your users. Images, cookies, scripts, HTML … they're all exploitable.