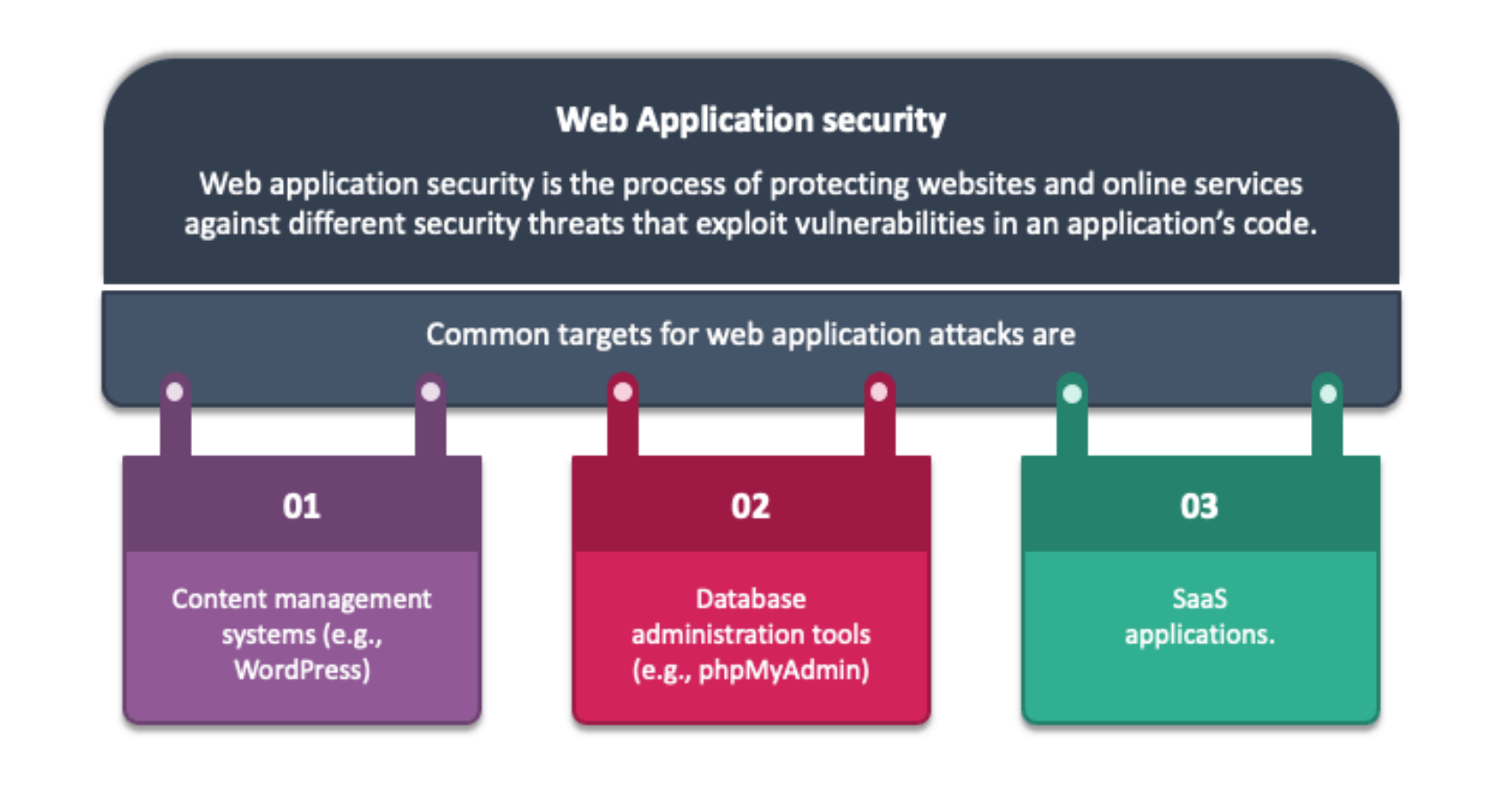
Securing ASP.NET Core Web Applications

Amal Dev

What is Web Application Security



OWASP

- Open Web Application Security Project - is a non-profit global community that strives to promote application security across the web
- Is considered highly credible, and developers have come to count on it for essential web application security, and API security guidance
- OWASP Top 10 - Every few years, OWASP revises and publishes its list of the top 10 web application vulnerabilities
- List includes not only the OWASP Top 10 threats but also the potential impact of each vulnerability and how to avoid them

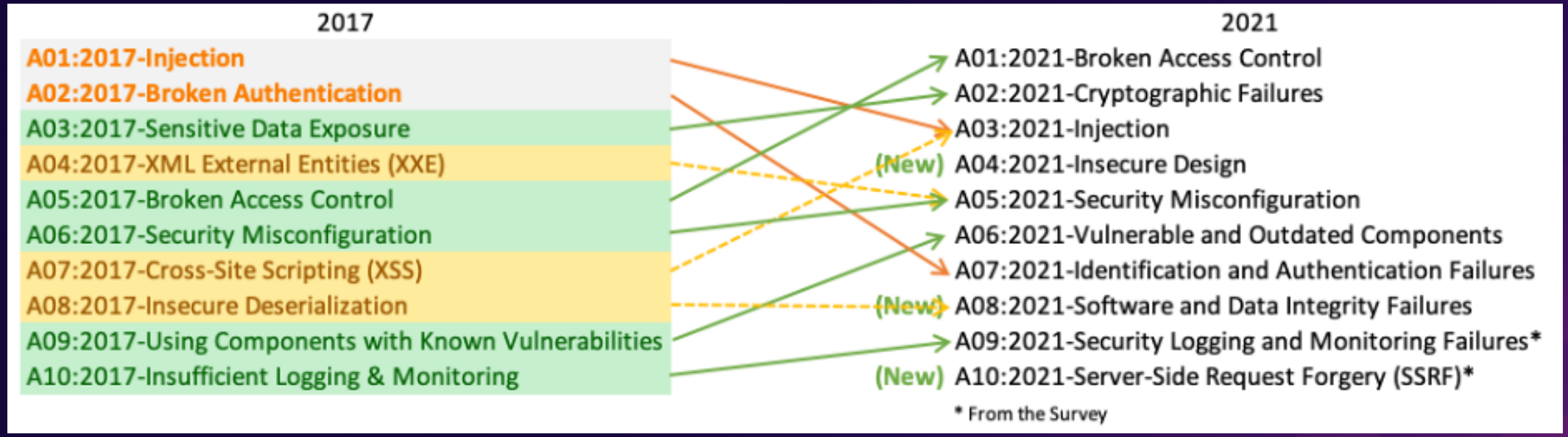
https://owasp.org/

OWASP Top 10 - 2021

- A01 - Broken Access Control
- A02 - Cryptographic Failures(earlier known as Sensitive Data Exposure)
- A03 - Injection
- A04 – In-secure Design
- A05 - Security misconfiguration(includes XML External Entities (XXE[A4, 2017])
- A06 - Vulnerable and Outdated Components
- A07 - Identification and Authentication Failures
- A08 - Software and Data Integrity Failures(includes Insecure deserialization[A8, 2017])
- A09 - Security Logging and Monitoring Failures(includes Insufficient Logging & Monitoring[A10,2017])
- A10 - Server-Side Request Forgery

https://owasp.org/www-project-top-ten/

OWASP 2017 vs 2021



Reference : https://owasp.org/www-project-top-ten/

A01 – Broken Access Control

Reasons

- Elevation of privileges
- Metadata manipulation
- CORS misconfiguration
- Viewing on editing someone else's account/data

Preventive Measures

- Deny by default
- Principle of Least Privilege
- Implement access control mechanisms
- Minimize CORS Usage
- Rate limit API and Controller access

Broken Access Control – Insecure Direct Object References

- Problem

```
public IActionResult GetOrder(int id)
{
    Order order = _orderRepository.GetById(id);
    return Ok(order);
}
```

- Resolution

```
[HttpGet("order/{id}")]
[Authorize]
public IActionResult GetOrder(int id)
{
    Order order = _orderRepository.GetById(id);
    return Ok(order);
}
```

Broken Access Control – Insecure Direct Object References

- Problem

```
[HttpGet("order/{id}")]
[Authorize]
public IActionResult GetOrder(int id)
{
    Order order = _orderRepository.GetById(id);
    return Ok(order);
}
```

- Resolution

```
[HttpGet("order/{id}")]
[Authorize]
public IActionResult GetOrder(int id)
{
    var loggedInUser = HttpContext.User;
    var customerId = loggedInUser.Claims.FirstOrDefault(x => x.Type
== ClaimTypes.NameIdentifier).Value;

    Order order = _orderRepository.GetById(id);
    if(order.CustomerId != customerId)
    {
        return Unauthorized();
    }

    return Ok(order);
}
```

Broken Access Control – CORS

- Problem

```
public void Configure(IApplicationBuilder app, IHostingEnvironment
env)
{
      app.UseCors(options =>
options.AllowAnyOrigin().AllowAnyMethod());
      app.UseMvc();
}
```

- Resolution

```
public void Configure(IApplicationBuilder app, IHostingEnvironment
env)
{
      app.UseCors(options =>
options.WithOrigins("http://www.mydomain.com").AllowAnyMethod());
      app.UseMvc();
}
```

A02 – Cryptographic Failures

Reasons

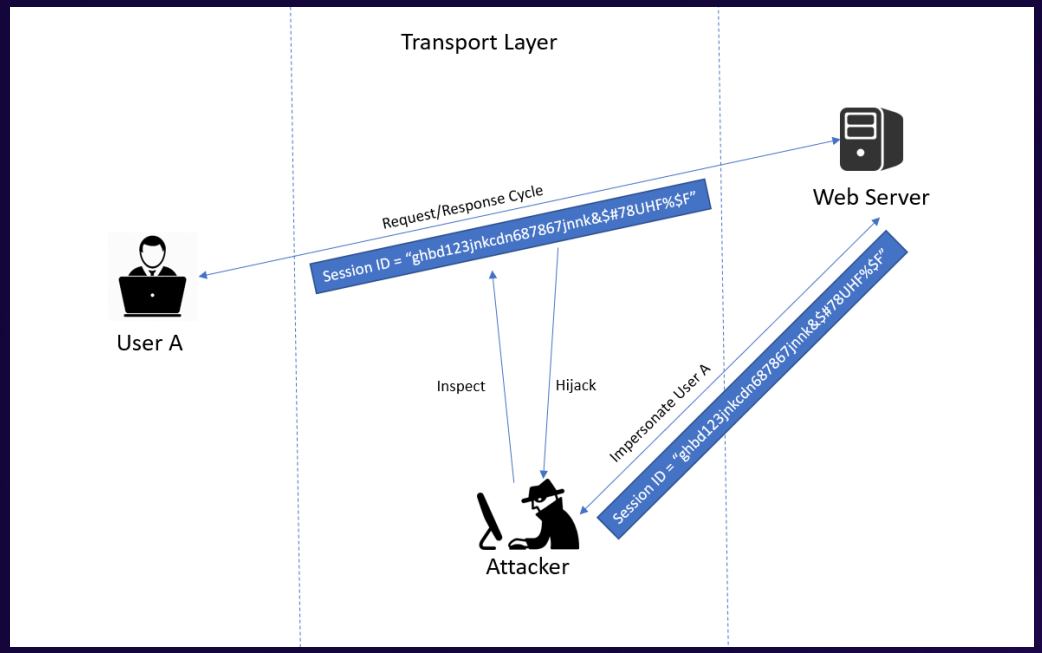
- Data transmitted in clear text
- Old or weak cryptographic algorithms or protocols used either by default or in older code

Preventive Measures

- Prevent storage of unwanted data
- Encrypt all sensitive data stored
- Encrypt all data in transit
- Disable caching for response that contain sensitive data

Cryptographic Failures – Not enforcing TLS or Weak Encryption

Problem



Resolution

```
public void Configure(IApplicationBuilder app, IWebHostEnv
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        // The default HSTS value is 30 days. You may want
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapRazorPages();
    });
```

A03 – Injection

Vulnerability

- User input data is not validated, filtered, or sanitized by the application
- Dynamic queries or non-parameterized inline SQL statements
- Types of Injections – SQL, OS Command, LDAP, ORM

Preventive Measures

- Escape all user input
- User input validation based on user input
- Use parameterized queries
- Avoid calling OS commands directly
- Enforcing Least Privilege

Injection – SQL

Vulnerability

```
using (SqlConnection sqlConnection = new SqlConnection("Data Source=.;Initial Catalog=Mv
{
    string commandText = "SELECT [UserName] FROM dbo.[Login] WHERE [Username] = '"
        + Login.Username
        + "' AND [Password]='"
        + Login.Password
        + "' ";
    try
    {
        using (SqlCommand sqlCommand = new SqlCommand(commandText, sqlConnection))
        {
            sqlConnection.Open();
            if (sqlCommand.ExecuteScalar() == null)
            {
                // Invalid Login
                Login.Message = "Invalid Login.";
```

Username: `admin`

Password: `' OR 1=1 --`

```
SELECT [Username]
FROM   dbo.[Login]
WHERE  [Username] = 'admin'
       AND [Password] = ''
        OR 1 = 1 --'
```

Resolution

```
using (SqlConnection sqlConnection = new SqlConnection("Data Source=.;Initial Catalog=Mv
{
    string commandText = "SELECT [UserName] FROM dbo.[Login] WHERE [Username] = @usernam
    try
    {
        using (SqlCommand sqlCommand = new SqlCommand(commandText, sqlConnection))
        {
            sqlCommand.Parameters.Add(new SqlParameter("username", Login.Username));
            sqlCommand.Parameters.Add(new SqlParameter("password", Login.Password));

            sqlConnection.Open();
            if (sqlCommand.ExecuteScalar() == null)
            {
```

```
using (SqlConnection sqlConnection = new SqlConnection("Data Source=.;Initial Catalog=M
{
    string commandText = "[dbo].[CheckLogin]";

    try
    {
        using (SqlCommand sqlCommand = new SqlCommand(commandText, sqlConnection))
        {
            sqlCommand.CommandType = CommandType.StoredProcedure;

            sqlCommand.Parameters.Add(new SqlParameter("@username", Login.Username));
            sqlCommand.Parameters.Add(new SqlParameter("@password", Login.Password));
```

A04 – Insecure Design

Vulnerability

- Focuses on design and architectural flaws
- Centers around missing or ineffective control design
- Unprotected storage of credentials
- Trust Boundary Violations

Preventive Measures

- Perform effective threat modelling in the design phase
- Document the secure design recommendations and requirements for the proposed system
- Setup continuous unit and integration tests
- Implement System and Network Layer Tier Segregation

A05 – Security Misconfiguration

Vulnerability

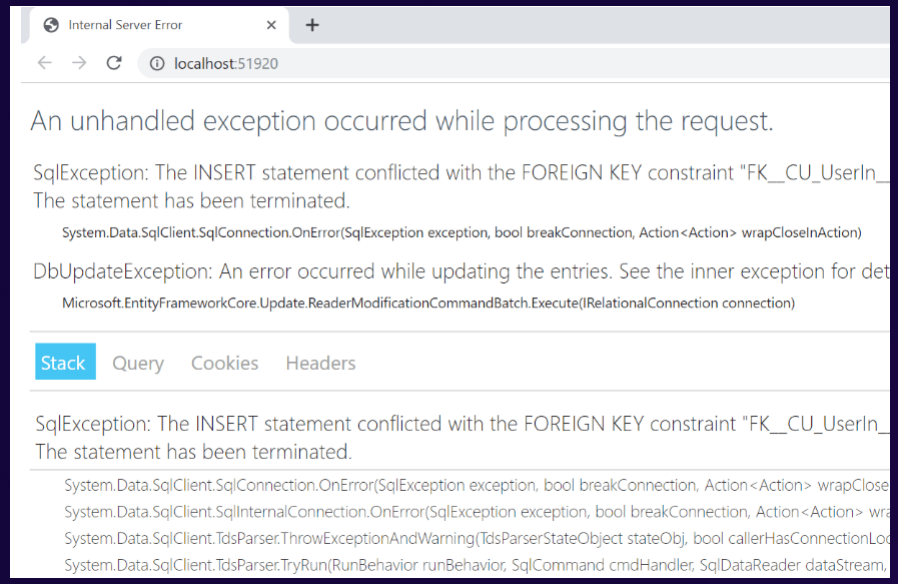
- Out of date security patches
- Application framework feature not turned on or improperly configured
- Default accounts, passwords etc left enabled and unchanged
- Unused default ports or services enabled on the server

Preventive Measures

- Patch your systems periodically
- Reduce attack surface by removing unwanted features, disabling default accounts and services
- Remove passwords and tokens from your source code

Misconfiguration – Error Pages

Vulnerability

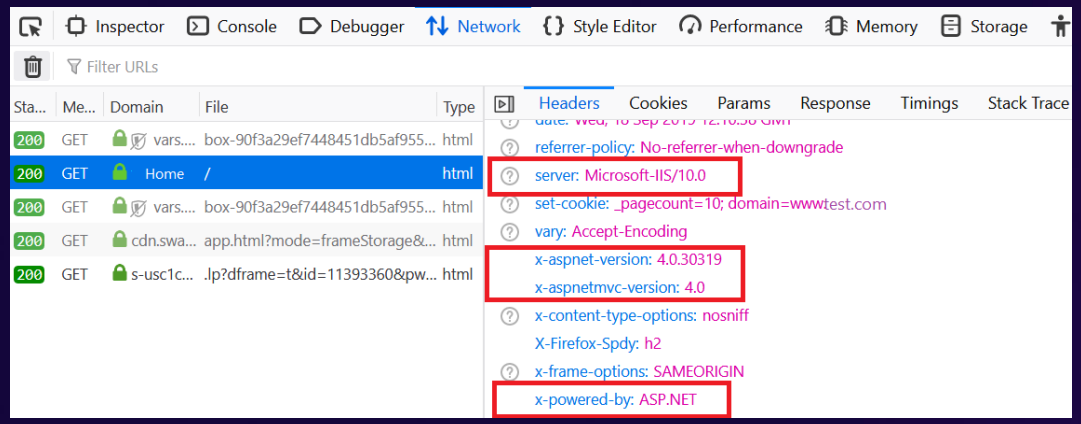


Resolution

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseHttpsRedirection();
    }
    else if (env.IsProduction())
    {
        app.UseExceptionHandler("/error/customerror");
    }
```

Misconfiguration – Server Headers

Vulnerability



Resolution

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args
    WebHost.CreateDefaultBuilder(args)
        .UseKestrel(c => c.AddServerHeader = false)
        .UseStartup<Startup>();
```

```
<system.webServer>
  <httpProtocol>
    <customHeaders>
      <remove name="X-Powered-By" />
    </customHeaders>
  </httpProtocol>
</system.webServer>
```

```
<system.web>
  <httpRuntime enableVersionHeader="false" />
</system.web>
```

A06 – Vulnerable and Outdated Components

Vulnerability

- Unsupported or outdated software
- Outdated third party components or libraries
- Not scanning for vulnerabilities regularly
- Absence of proper patching or remediation process

Preventive Measures

- Update your software and components periodically
- Remove any unused dependencies, unnecessary features, components, files
- Regularly inventory the versions of client-side and server-side components and their dependencies

A07 – Identification and Authentication Failures

Vulnerability

- Not protected against automated attacks such as credential stuffing
- Weak or ineffective credential recovery and forgotten password procedures
- Does not use or has ineffective MFA
- Improper invalidation of user sessions and authentication tokens during logout or when inactive

Preventive Measures

- Implement multi-factor authentication
- Perform weak password checks
- Setup password policies to determine password length, complexity, and rotation policies
- Limit or progressively delay repeated login attempts after failure

A08 – Software and Data Integrity Failures

Vulnerability

- Relates to vulnerabilities in software updates, critical data, and CI/CD pipelines whose integrity is not verified.
- Auto-update functionality of most applications that do not necessarily include a thorough integrity check
- Failures that arise due to data encoded or serialized into a structure visible to an attacker.

Preventive Measures

- Use mechanisms such as digital signatures to verify that software or data
- Use only trusted repositories for libraries and dependencies
- Implement a review process for code and configuration changes to reduce the risk
- Integrity check or digital signature to detect tampering or replay of the data

A10 – Server-Side Request Forgery (SSRF)

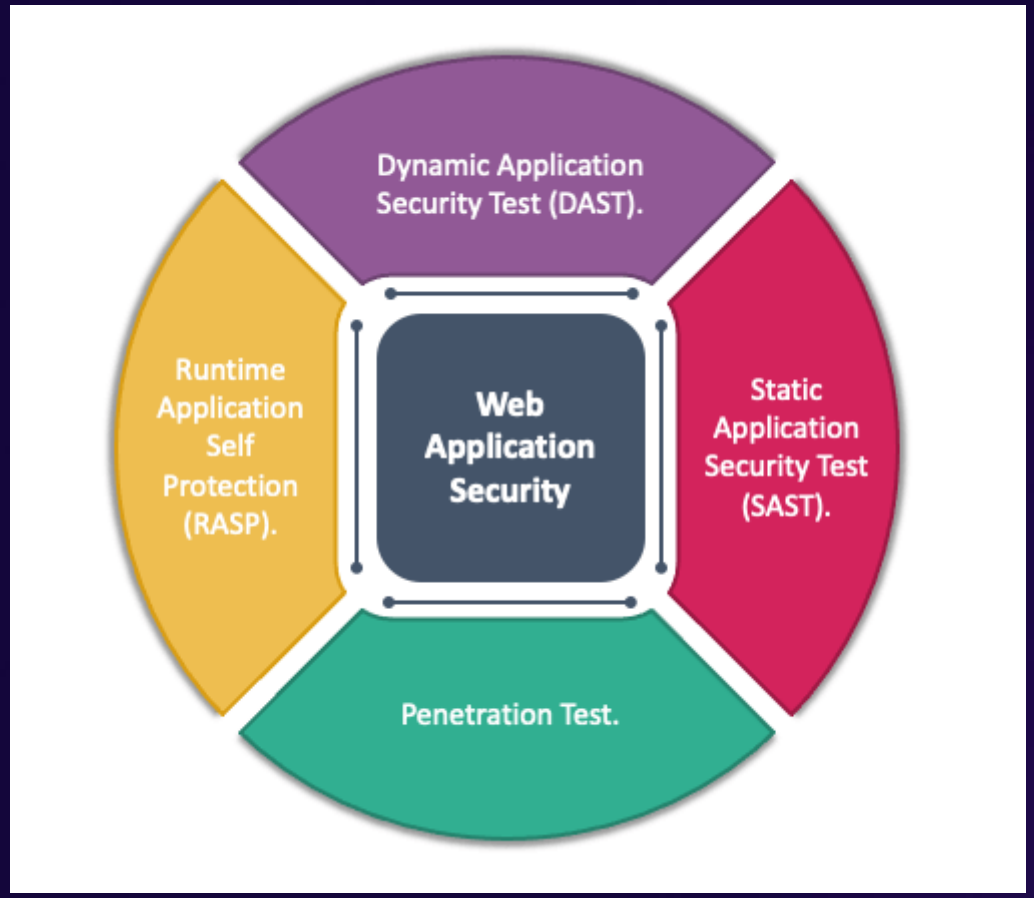
Vulnerability

- Non validation of the user-supplied URL when fetching a remote resource

Preventive Measures

- Block all but essential traffic by instituting “deny by default” network policies or network access control rules
- All client-supplied input data must be sanitized and validated
- Use a positive allow list to enforce the URL schema, port, and destination

Types of Security Testing



- SAST Tools
  - https://owasp.org/www-community/Source_Code_Analysis_Tools
- DAST Tools
  - https://owasp.org/www-community/Vulnerability_Scanning_Tools
- Open Source
  - https://owasp.org/www-community/Free_for_Open_Source_Application_Security_Tools

Q&A

Thank you

@amaldevv