DEMYSTIFYING OWASP TOP 10 MOBILE RISKS

ConFoo 2024

Lester Botello

$WHOAMI

@lesterbotello

- Team Lead at nventive
- 20+ years of experience developing software
- Cross platform (Uno Platform, .NET MAUI, Flutter, Kotlin)
- NOT a security expert, just concerned about mobile security

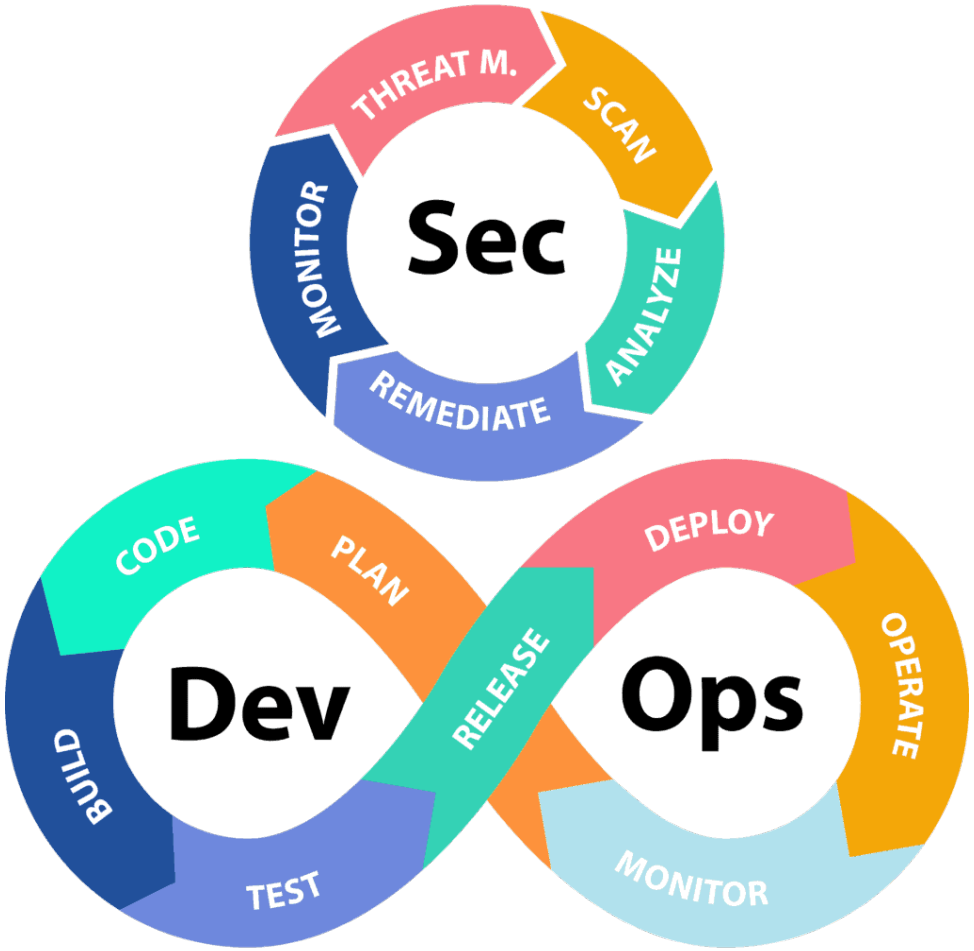
“

OUR MOBILE APP NEEDS TO BE SECURE, BUT WE CAN’T ALTER THE RELEASE SCHEDULE

”

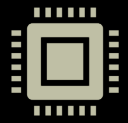
- YOUR APP’S STAKEHOLDERS, PROBABLY

DEVSECOPS



COLLABORATION FRAMEWORK THAT ADDS SECURITY PRACTICES TO THE SOFTWARE DEVELOPMENT AND DELIVERY PROCESS

DEVSECOPS



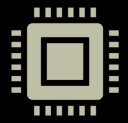
Development, Security, Operations

Integrates security practices within the DevOps process

Security is integrated into the software development life cycle (SDLC) from the start.

Goal is identifying threats early into the SLDC without slowing it down

SECDEVOPS



Security, Development, Operations

Security is the prime directive, even if it means missing a deadline.

Security is not just integrated into the SLDC, it's the central component.

Goal is to make all stakeholders equally responsible for security, not just the security team

MOBILE SECURITY SCANNING TOOLS


OWASP
®

ROLES AND PURPOSE

Aims to improve software security through community-led projects, local chapters, documentation and training

Key initiatives: OWASP Top Ten, OWASP projects, local chapters

Funded through memberships, corporate sponsorships and conferences

Community-driven collaboration

OWASP MOBILE TOP 10

HOW IT WORKS

1. Initial Planning/Data Call

Core team gets together and plans a rough schedule, a data call is released.

2. Industry Survey

We determine content in the survey and release for industry participation.

3. Data Analysis

After the data is collected, it is normalized and analyzed.

4. Draft Top Ten

Once we determine the eight risks from the data and the two from the survey, we draft a new list. The Draft is publicly released for review. All issues raised and decisions made are recorded in GitHub issues.

5. Release

Once we have reached a consensus and the core team agrees, we release the new OWASP Top Ten.

Source:

OWASP MOBILE TOP 10 (2024)

- M1: Improper Credential Usage
- M2: Inadequate Supply Chain Security
- M3: Insecure Authentication/Authorization
- M4: Insufficient Input/Output Validation
- M5: Insecure Communication
- M6: Inadequate Privacy Controls
- M7: Insufficient Binary Protections
- M8: Security Misconfiguration
- M9: Insecure Data Storage
- M10: Insufficient Cryptography

OWASP TOP 10 2016 VS. 2024

Comparison Between 2016-2024		
OWASP-2016	**OWASP-2024-Release**	**Comparison Between 2016-2024**
M1: Improper Platform Usage	M1: Improper Credential Usage	**New**
M2: Insecure Data Storage	M2: Inadequate Supply Chain Security	**New**
M3: Insecure Communication	M3: Insecure Authentication / Authorization	**Merged M4&M6 to M3**
M4: Insecure Authentication	M4: Insufficient Input/Output Validation	**New**
M5: Insufficient Cryptography	M5: Insecure Communication	**Moved from M3 to M5**
M6: Insecure Authorization	M6: Inadequate Privacy Controls	**New**
M7: Client Code Quality	M7: Insufficient Binary Protections	**Merged M8&M9 to M7**
M8: Code Tampering	M8: Security Misconfiguration	**Rewording [M10]**
M9: Reverse Engineering	M9: Insecure Data Storage	**Moved from M2 to M9**
M10: Extraneous Functionality	M10: Insufficient Cryptography	**Moved from M5 to M10**

M1: Improper Credential Usage

Can occur when mobile apps use hardcoded credentials or when credentials are misused.

- Exploitability: Easy
- Prevalence: Common
- Detectability: Easy
- Technical Impact: Severe
- Business Impact: Severe

M1: Improper Credential Usage

Threat agents

- Hardcoded Credentials
- Insecure Credential Transmission
- Insecure Credential Storage
- Weak User Authentication

M1: IMPROPER CREDENTIAL USAGE

Program.cs

```
public class InsecureCredentialHandling
{
    private string username = "admin";
    private string password = "password123"; // Hardcoded credentials

    public void Login()
    {
        // Using hardcoded credentials for authentication
        if (this.username == "admin" && this.password == "password123")
        {
            // Do something that only applies to the superuser
        }
    }
}
```

M1: IMPROPER CREDENTIAL USAGE

Program.cs

```
using Microsoft.AspNetCore.Identity;

public class SecureAuth
{
    private readonly UserManager<IdentityUser> userManager;

    public SecureAuth(UserManager<IdentityUser> userManager)
    {
        this.userManager = userManager;
    }

    public async Task<bool> Authenticate(string username, string password)
    {
        var user = await userManager.FindByNameAsync(username);
        if (user != null)
        {
            // Secure: Verifying password hash
            return await userManager.CheckPasswordAsync(user, password);
        }

        return false;
    }

    public async Task AccessControl(string username, string password)
    {
        if (await Authenticate(username, password))
        {
            // Implement fine-grained authorization checks
            if (await userManager.IsInRoleAsync(username, "Admin"))
            {
                PerformSensitiveOperations();
            }
        }
    }

    private void PerformSensitiveOperations()
    {
        // Operations that require secure access
    }
}
```

M1: Improper Credential Usage

Prevention

- Avoid hard-coded credentials
- If you need to store user credentials, handle and transmit them properly
- Favor the use of authentication tokens instead of handling credentials directly

M2: Inadequate Supply Chain Security

Most associated with untrusted or vulnerable third-party libraries.

- Exploitability: Average
- Prevalence: Common
- Detectability: Difficult
- Technical Impact: Severe
- Business Impact: Severe

M2: Inadequate Supply Chain Security

Threat agents

- Using outdated libraries
- Weak code security

M2: INADEQUATE SUPPLY CHAIN SECURITY

PREVENTION

- USE TRUSTED 3RD PARTY LIBRARIES AND COMPONENTS
- ESTABLISH THOROUGH CODE REVIEW
- KEEP LIBRARIES UP-TO-DATE
- IF OPEN-SOURCED, ANALYZE THE SOURCE CODE TO ENSURE INTEGRITY

M3: Insecure Authentication/Authorization

The app doesn't properly verify the identity of its users or if they're authorized to execute actions that require elevation.

- Exploitability: Easy
- Prevalence: Common
- Detectability: Average
- Technical Impact: Severe
- Business Impact: Severe

M3: INSECURE AUTHENTICATION/AUTHORIZATION

Program.cs

```
public class InsecureAuth
{
    // Basic user authentication without proper security measures
    public bool Authenticate(string username, string password)
    {
        // Insecure: Using plain text comparison and static user data
        return username == "admin" && password == "password123";
    }

    public void AccessControl()
    {
        if (Authenticate("admin", "password123"))
        {
            // Granting access without proper authorization checks
            // Insecure: Any user with correct credentials has full access
            PerformSensitiveOperations();
        }
    }

    private void PerformSensitiveOperations()
    {
        // Operations that require secure access
    }
}
```

M3: INSECURE AUTHENTICATION/ AUTHORIZATION

Program.cs

```
using Microsoft.AspNetCore.Identity;

public class SecureAuth
{
    private readonly UserManager<IdentityUser> userManager;

    public SecureAuth(UserManager<IdentityUser> userManager)
    {
        this.userManager = userManager;
    }

    public async Task<bool> Authenticate(string username, string password)
    {
        var user = await userManager.FindByNameAsync(username);
        if (user != null)
        {
            // Secure: Verifying password hash
            return await userManager.CheckPasswordAsync(user, password);
        }

        return false;
    }

    public async Task AccessControl(string username, string password)
    {
        if (await Authenticate(username, password))
        {
            // Implement fine-grained authorization checks
            if (await userManager.IsInRoleAsync(username, "Admin"))
            {
                PerformSensitiveOperations();
            }
        }
    }

    private void PerformSensitiveOperations()
    {
        // Operations that require secure access
    }
}
```

M3: Insecure Authentication/Authorization

Prevention

- Implement secure patterns for authentication (server-side auth, use revocable tokens, etc)
- Unless it's a business requirement, avoid offline authentication
- Use the device biometrics as an additional layer of security.

M4: Insufficient Input/Output Validation

The app doesn't properly validate data that comes from external sources (for ex., user input or data coming from an API).

- Exploitability: Difficult
- Prevalence: Common
- Detectability: Easy
- Technical Impact: Severe
- Business Impact: Severe

M4: INSUFFICIENT INPUT/OUTPUT VALIDATION

Program.cs

```
using System.Data.SqlClient;

public class InsecureDataHandling
{
    public void AddUserData(string username, string password)
    {
        // Insecure: Directly using user input in SQL query
        string query = $"INSERT INTO Users (Username, Password) VALUES ('{username}', '{password}')";

        using (SqlConnection connection = new SqlConnection("YourConnectionString"))
        {
            SqlCommand command = new SqlCommand(query, connection);
            connection.Open();
            command.ExecuteNonQuery();
        }
    }
}
```

M4: INSUFFICIENT INPUT/OUTPUT VALIDATION

Program.cs

```
using System.Data.SqlClient;

public class SecureDataHandling
{
    public void AddUserData(string username, string password)
    {
        if (IsValidInput(username) && IsValidInput(password))
        {
            // Secure: Using parameterized query
            string query = "INSERT INTO Users (Username, Password) VALUES (@username, @password)";

            using (SqlConnection connection = new SqlConnection("YourConnectionString"))
            {
                SqlCommand command = new SqlCommand(query, connection);
                command.Parameters.AddWithValue("@username", username);
                command.Parameters.AddWithValue("@password", password);
                connection.Open();
                command.ExecuteNonQuery();
            }
        }
    }

    private bool IsValidInput(string input)
    {
        // Basic validation for demonstration
        // In practice, use more comprehensive validation and sanitization
        return !string.IsNullOrWhiteSpace(input) && Regex.IsMatch(input, @"^[a-zA-Z0-9]*$");
    }
}
```

M4: INSUFFICIENT INPUT/OUTPUT VALIDATION

PREVENTION

- SANITIZE USER INPUT AND OUTPUT DATA
- IMPLEMENT DATA INTEGRITY CHECKS
- USE PARAMETRIZED QUERIES WHEN WORKING WITH DATABASES.

M5: INSECURE COMMUNICATION

The app connects to other devices or resources through insecure means.

- Exploitability: Easy
- Prevalence: Common
- Detectability: Average
- Technical Impact: Severe
- Business Impact: Moderate

M5: INSECURE COMMUNICATION

Program.cs

```
using System.Net.Http;
using System.Threading.Tasks;

public class InsecureCommunicationExample
{
    public async Task<string> GetDataFromServer()
    {
        // Using a custom handler to override built-in unsecure communication prevention
        var handler = new HttpClientHandler();
        handler.ClientCertificateOptions = ClientCertificateOption.Manual;
        handler.ServerCertificateCustomValidationCallback =
        (httpRequestMessage, cert, cetChain, policyErrors) ⇒
        {
            return true;
        };

        var client = new HttpClient(handler)
        var response = await client.GetAsync("http://example.com/data");
        return await response.Content.ReadAsStringAsync();
    }
}
```

M5: INSECURE COMMUNICATION

Program.cs

```cs
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;

public class SecureCommunicationExample
{
    public async Task<string> GetDataFromServer()
    {
        // Secure: Using HTTPS for encrypted communication
        var client = new HttpClient();

        // Optional: Set up client certificates, headers, etc.
        client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

        var response = await client.GetAsync("https://secureexample.com/data");
        return await response.Content.ReadAsStringAsync();
    }
}
```

M5: INSECURE COMMUNICATION

Prevention

- Don't override the default HTTP client handler to allow unsecure connections.
- Always require SSL chain verification
- Consider certificate pinning

M6: Inadequate Privacy Controls

Measures to obscure the identity of the user/device are light or non-existent.

- Exploitability: Average
- Prevalence: Common
- Detectability: Easy
- Technical Impact: Low
- Business Impact: Severe

M6: Inadequate Privacy Controls

In Canada, the Personal Information Protection and Electronic Documents Act (PIPEDA) and Privacy Act defines "personal information" as data that on its own, or combined with other pieces of data, can identify an individual.

M6: Inadequate Privacy Controls

Personal Identifiable information (PII).

- Identity: Name, date of birth, signature, gender, race, familial situation
- Contact information: Address, phone number, email address
- Professional information: Job, company, position, date of hire, HR evaluation, salary
- Administrative documents: ID, passport number, driver's license, vehicle identification number (VIN), Social Security number / NAS
- Healthcare: Biometric data, medical records
- IT related: Internet Protocol (IP) address, password(s), cookies, logs

M6: INADEQUATE PRIVACY CONTROLS

Program.cs

```
public class User
{
    public string Name { get; set; }
    public string Email { get; set; }
    // Sensitive data
    public string SocialInsuranceNumber { get; set; }
}

public class PrivacyViolationExample
{
    public void ProcessUserData(User user)
    {
        // Inadequate privacy control: Logging sensitive information
        Console.WriteLine($"Processing user data: Name={user.Name}, Email={user.Email}, SIN={user.SocialInsuranceNumber}");

        // Further processing...
    }
}
```

M6: Inadequate Privacy Controls

Prevention

- Don't manipulate PII unless absolutely necessary
- Replace PII by less critical information if possible
- Anonymize PII if possible
- Always provide the user for a way to delete PII.

M7: Insufficient Binary Protection

The app doesn't provide sufficient validation of its own binary integrity or that of its environment.

- Exploitability: Easy
- Prevalence: Common
- Detectability: Easy
- Technical Impact: Moderate
- Business Impact: Moderate

M7: INSUFFICIENT BINARY PROTECTION

Program.cs

```
private static bool IsTeamIdValid()
{
    var expectedTeamID = "A1B2C3D4E5";

    SecRecord result;

    var query = new SecRecord(SecKind.GenericPassword)
    {
        Service = NSBundle.MainBundle.BundleIdentifier
    };

    SecStatusCode status;
    result = SecKeyChain.QueryAsRecord(query, out status);

    if (status == SecStatusCode.ItemNotFound)
    {
        status = SecKeyChain.Add(query);
        result = SecKeyChain.QueryAsRecord(query, out status);
    }

    if (status == SecStatusCode.Success)
    {
        var dic = result.ToDictionary();
        var accGrp = (dic[K_SEC_ATTR_ACCESS_GROUP] as NSString)?.ToString();
        if (accGrp != null)
        {
            var retrievedTeamID = accGrp.Split('.').FirstOrDefault();
            return retrievedTeamID == expectedTeamID;
        }
    }

    return false;
}
```

M7: INSUFFICIENT BINARY PROTECTION

Program.cs

```
public static bool AmIJailbroken()
{
    // This is an incomplete list, IRL you'd want to check
    // for a lot more other apps
    var paths = new[]
    {
        "/Applications/Cydia.app",
        "/Applications/FakeCarrier.app",
        "/Applications/SBSettings.app",
        "/Applications/blackra1n.app",
    };

    return (paths.Any(path ⇒ File.Exists(path)));
}
```

M7: INSUFFICIENT BINARY PROTECTION

Program.cs

```
private static bool AmIRunningOnSimulator()
{
    return ObjCRuntime.Runtime.Arch ≠ ObjCRuntime.Arch.DEVICE;
}

private static bool IsDebuggerAttached()
{
    return System.Diagnostics.Debugger.IsAttached;
}
```

M7: Insufficient Binary Protection

Prevention

- Consider obfuscation.
- Add integrity checks on app's start-up and kill the app immediately if any checks fail.
- Include automated app signing for different environments on CI/CD

M8: Security Misconfiguration

The app's configuration settings are unsafe or insecure.

- Exploitability: Difficult
- Prevalence: Common
- Detectability: Easy
- Technical Impact: Severe
- Business Impact: Severe

M8: SECURITY MISCONFIGURATION

AwesomeApp.csproj

```xml
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <!-- ... other settings ... -->

    <!-- Security Misconfiguration: Debugging enabled in Release configuration -->
    <PropertyGroup Condition="'$(Configuration)|$(Platform)'=='Release|AnyCPU'">
      <DebugType>full</DebugType>
      <Optimize>false</Optimize>
    </PropertyGroup>

  </PropertyGroup>

</Project>
```

M8: SECURITY MISCONFIGURATION

AwesomeApp.csproj

```xml
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <!-- ... other settings ... -->

    <!-- Correct Configuration: Debugging disabled and optimizations enabled for Release configuration -->
    <PropertyGroup Condition="'$(Configuration)|$(Platform)'=='Release|AnyCPU'">
      <DebugType>none</DebugType>
      <Optimize>true</Optimize>
    </PropertyGroup>

  </PropertyGroup>

</Project>
```

M8: SECURITY MISCONFIGURATION

AndroidManifest.xml

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">

    <!-- ... other manifest entries ... -->

    <application>
        <!-- Insecure: Exposing root path in exported FileProvider -->
        <provider
            android:name="androidx.core.content.FileProvider"
            android:authorities="com.example.myapp.fileprovider"
            android:exported="true"
            android:grantUriPermissions="true">
            <meta-data
                android:name="android.support.FILE_PROVIDER_PATHS"
                android:resource="@xml/file_paths" />
        </provider>
    </application>
</manifest>
```

file_paths.xml

```xml
<paths>
    <!-- Insecure: Exposing the entire app's file directory -->
    <external-path name="external_files" path="." />
</paths>
```

M8: SECURITY MISCONFIGURATION

AndroidManifest.xml

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">

    <!-- ... other manifest entries ... -->

    <application>
        <!-- Secure: Limiting access to specific directories and not exporting FileProvider -->
        <provider
            android:name="androidx.core.content.FileProvider"
            android:authorities="com.example.myapp.fileprovider"
            android:exported="false"
            android:grantUriPermissions="true">
            <meta-data
                android:name="android.support.FILE_PROVIDER_PATHS"
                android:resource="@xml/file_paths" />
        </provider>
    </application>
</manifest>
```

file_paths.xml

```xml
<paths>
    <!-- Secure: Exposing only specific, non-sensitive directories -->
    <external-path name="external_files" path="some/safe/directory/" />
</paths>
```

M8: Security Misconfiguration

Other threat agents.

- Unnecessary permissions (ex., requesting camera permission for a flashlight app)
- Overly permissive storage permissions
- Any unused functionality or packages

M8: Security Misconfiguration

Prevention

- Least privilege principle: Default the app to the least elevated functionality
- Audit the permissions of your app regularly.
- Refrain from using default credentials

M9: Insecure Data Storage

The app stores data insecurely.

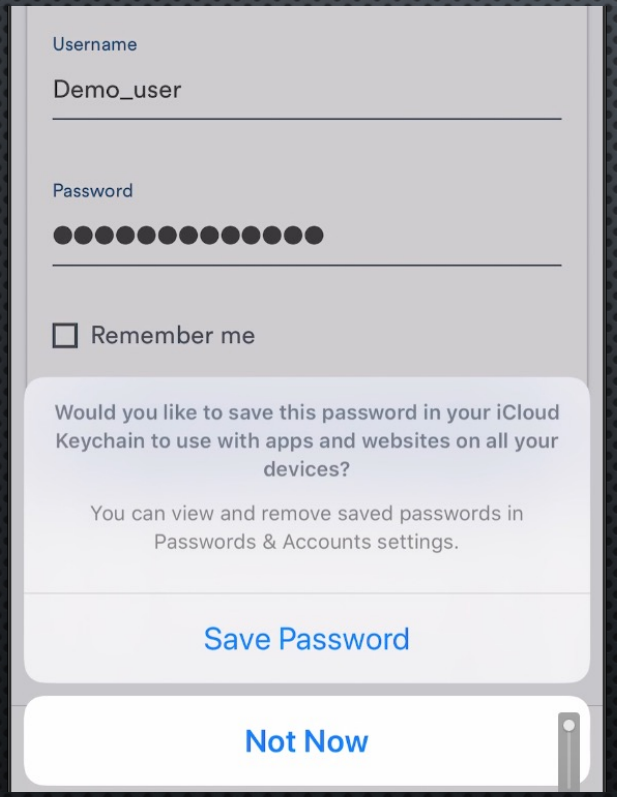
- Exploitability: Easy
- Prevalence: Common
- Detectability: Average
- Technical Impact: Severe
- Business Impact: Severe

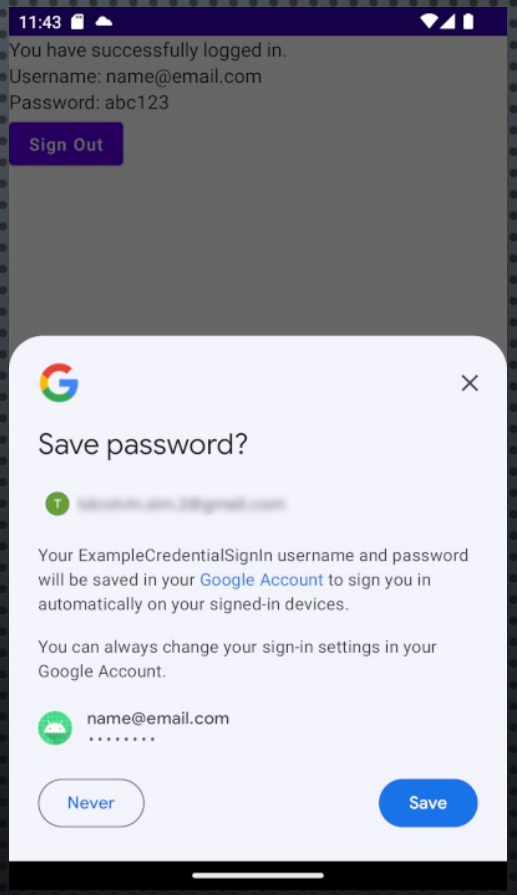
M9: INSECURE DATA STORAGE

Program.cs

```
using System.IO;

public class InsecureStorageExample
{
    public void SaveUserData(string username, string password)
    {
        // Insecure: Storing sensitive data in plain text in a local file
        string filePath = "user_credentials.txt";
        File.WriteAllText(filePath, $"Username: {username}, Password: {password}");
    }
}
```

M9: INSECURE DATA STORAGE





M9: INSECURE DATA STORAGE

Program.cs

```
public void SaveCredentials(string username, string password)
{
    // Storing the username
    var credsRecord = new SecRecord(SecKind.GenericPassword)
    {
        Service = "com.yourappname.credentials",
        Account = username,
        ValueData = NSData.FromString(password),
        Accessible = SecAccessible.WhenUnlockedThisDeviceOnly
    };
    SecKeyChain.Add(credsRecord);
}
```

M9: INSECURE DATA STORAGE

Program.cs

```
public void SaveCredentials(string username, string password)
{
    // Generate a key to encrypt/decrypt data
    var keyGenerator = KeyGenerator.GetInstance(KeyProperties.KeyAlgorithmAes, "AndroidKeyStore");
    keyGenerator.Init(new KeyGenParameterSpec.Builder("myKeyAlias",
            KeyProperties.PurposeEncrypt | KeyProperties.PurposeDecrypt)
            .SetBlockModes(KeyProperties.BlockModeGcm)
            .SetEncryptionPaddings(KeyProperties.EncryptionPaddingNone)
            .Build());
    var secretKey = keyGenerator.GenerateKey();

    // Store the key in the Keystore
    var keyStore = KeyStore.GetInstance("AndroidKeyStore");
    keyStore.Load(null);
    keyStore.SetEntry("myKeyAlias", new KeyStore.SecretKeyEntry(secretKey), null);

    // Encrypt the username and password
    var cipher = Cipher.GetInstance("AES/GCM/NoPadding");
    cipher.Init(CipherMode.EncryptMode, secretKey);
    var iv = cipher.GetIV();
    var encryptedUsername = cipher.DoFinal(Encoding.UTF8.GetBytes(username));
    var encryptedPassword = cipher.DoFinal(Encoding.UTF8.GetBytes(password));

    // Save the encrypted username, password, and IV in SharedPreferences
    var preferences = GetSharedPreferences("myApp", FileCreationMode.Private);
    var editor = preferences.Edit();
    editor.PutString("username", Convert.ToBase64String(encryptedUsername));
    editor.PutString("password", Convert.ToBase64String(encryptedPassword));
    editor.PutString("iv", Convert.ToBase64String(iv));
    editor.Apply();
}
```

M9: INSECURE DATA STORAGE

PREVENTION

- ALWAYS USE THE INTEGRATED STORAGE MECHANISM OF THE MOBILE OS
- ADD SHORT CACHE EXPIRATION WHENEVER POSSIBLE.
- KEEP LIBRARIES UP TO DATE -- ***DON'T ACCUMULATE TECH DEBT!***

M10: Insufficient Cryptography

The app doesn't implement strong cryptographic protections all across.

- Exploitability: Average
- Prevalence: Common
- Detectability: Average
- Technical Impact: Severe
- Business Impact: Severe

M10: INSUFFICIENT CRYPTOGRAPHY

Program.cs

```
public string EncryptData(string data)
{
    // Insecure: Using DES, which is considered weak due to its small key size
    using (DESCryptoServiceProvider des = new DESCryptoServiceProvider())
    {
        des.Key = Encoding.UTF8.GetBytes("WeakKey!"); // Weak key
        des.IV = Encoding.UTF8.GetBytes("WeakIV!"); // Weak initialization vector

        ICryptoTransform encryptor = des.CreateEncryptor();

        using (MemoryStream ms = new MemoryStream())
        {
            using (CryptoStream cs = new CryptoStream(ms, encryptor, CryptoStreamMode.Write))
            {
                using (StreamWriter sw = new StreamWriter(cs))
                {
                    sw.Write(data);
                }

                return Convert.ToBase64String(ms.ToArray());
            }
        }
    }
}
```

M10: INSUFFICIENT CRYPTOGRAPHY

Program.cs

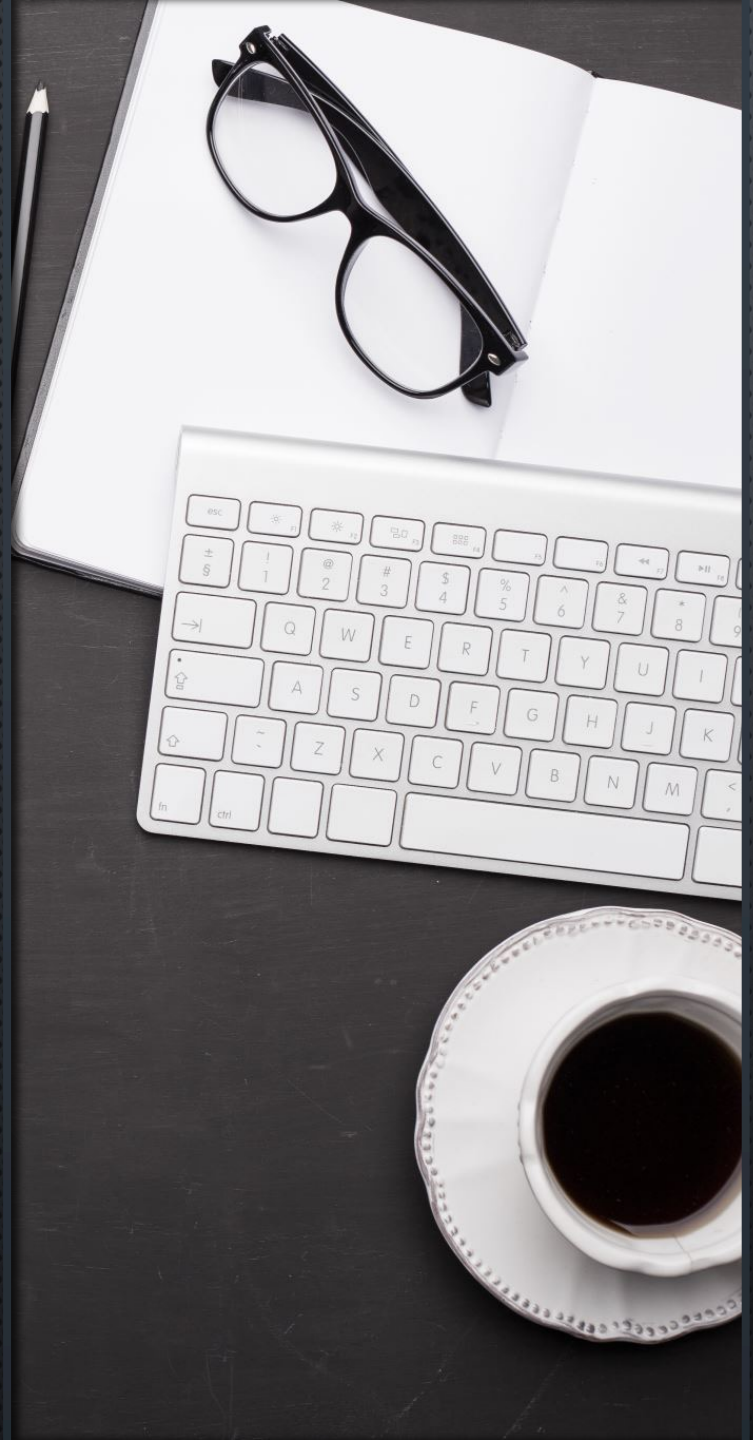
```cs
using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;

public class StrongCryptographyExample
{
    pr_ivate byte[] _key;
    pr_ivate byte[] _iv;

    public StrongCryptographyExample()
    {
        using (var rng = new RNGCryptoServiceProvider())
        {
            // Generate strong key and iv
            _key = new byte[32]; // 256 bits
            rng.GetBytes(_key);
            _iv = new byte[16]; // 128 bits
            rng.GetBytes(_iv);
        }
    }

    public string EncryptData(string data)
    {
        // Secure: Using AES with strong key and iv
        using (Aes aesAlg = Aes.Create())
        {
            aesAlg.Key = _key;
            aesAlg.IV = _iv;

            ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);

            using (MemoryStream ms = new MemoryStream())
            {
                using (CryptoStream cs = new CryptoStream(ms, encryptor, CryptoStreamMode.Write))
                {
                    using (StreamWriter sw = new StreamWriter(cs))
                    {
                        sw.Write(data);
                    }
                    return Convert.ToBase64String(ms.ToArray());
                }
            }
        }
    }
}
```

M10: Insufficient Cryptography

Prevention

- Always use strong, standard encryption algorithms
- If you need to store encryption keys, do so using the mobile OS secure storage
- Use Key Derivation Functions (KDFs) if you need to do encryption

FURTHER READING



FURTHER READING

- OWASP Mobile TOP 10 Official Site
  - https://owasp.org/www-project-mobile-top-10/
- Mobile Application Penetration Testing by Vijay Kumar Velu (2016)
- Secure By Design by Daniel DeogunDan Bergh Johnsson & Daniel Sawano (2019)

THANK YOU!