Windows Phone App Security
for builders and breakers

Luca De Fulgentis ~ luca@securenetwork.it

May 22nd, 2015 | Amsterdam

OWASP AppSecEU 15

Amsterdam, The Netherlands

About /me

- Luca De Fulgentis ~ @_daath
- Chief Technology Officer at Secure Network
- OWASP Mobile Project Contributor
- Nibble Security co-founder - blog.nibblesec.org
- Consuming brain-power with InfoSec since 2001

Agenda

- Introduction
  - Research overview and motivations
- Mobile Top Ten for Windows Phone
  - Examples of real-world vulnerable code
  - Discussion on potential insecure APIs usage
  - Secure coding tips for builders
- Final considerations

Introduction

- In 2014 we collected examples of insecure code for Windows Phone apps
  - Set of 60+ samples, of which 30% of mobile banking apps
  - Mostly developed with the Silverlight 8.x technology
- Statistics on the initial study has been shared with the OWASP Mobile Project for the MTT 2015 definition
- Later, we extended our research developing an automated script that allowed downloading 160+ AppX from US and IT regions of the WP Store
  - We needed to cover WP 8.1 Windows Runtime (WinRT) security as well

Introduction – motivations

- Too few (public) resources on WP apps security
  - MWR's «Navigation a Sea of Pwn?» (SyScan, 2014) - pretty amazing paper on the topic
  - XDA Forum represents an invaluable source of information
- We want both builders and breakers to be happy!
  - Provide a wide range of common APIs (MSDN) categorized on the basis of MTT 2014 and also define methods and strategies to mitigate these risks
- We defined a public *catalog* of potentially insecure APIs
  - Focus on C#/XAML apps, still the most relevant development technologies
- The talk will detail, for each MTT 2014 risk, these APIs !

OWASP Mobile Top 10 Risks (2014)

M1: Weak Server Side Controls	M2: Insecure Data Storage	M3: Insufficient Transport Layer Protection	M4: Unintended Data Leakage
M5: Poor Authorization and Authentication	M6: Broken Cryptography	M7: Client Side Injection	M8: Security Decisions via Untrusted Inputs
	M9: Improper Session Handling	M10: Lack of Binary Protections	

OWASP Mobile Top 10 Risks (2014)

M1: Weak Server Side Controls	M2: Insecure Data Storage	M3: Insufficient Transport Layer Protection	M4: Unintended Data Leakage
M5: Poor Authorization and Authentication	M6: Broken Cryptography	M7: Client Side Injection	M8: Security Decisions via Untrusted Inputs
	M9: Improper Session Handling	M10: Lack of Binary Protections	

M1 – Weak Server Side Controls

- The risk is referring to server-side security
  - Mobile platform *agnostic*
- Why bother about server-side security ?
  - Back-end can be *directly* attacked and mobile users data stolen
  - Back-end functionalities and *trust relationship* can be abused to hack into victim's mobile apps
- Our research has focused on mobile apps code only

OWASP Mobile Top 10 Risks (2014)

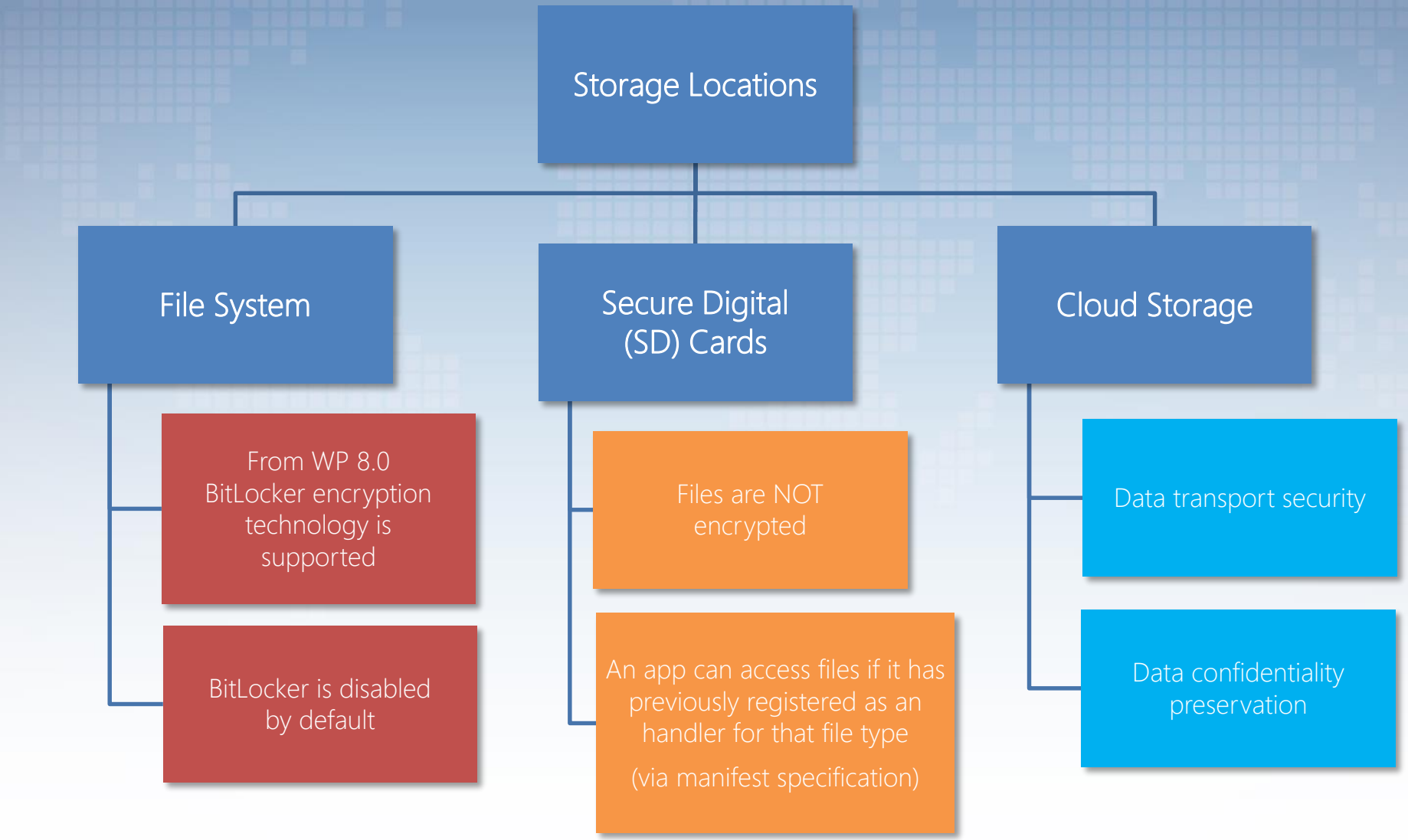
M1: Weak Server Side Controls	M2: Insecure Data Storage	M3: Insufficient Transport Layer Protection	M4: Unintended Data Leakage
M5: Poor Authorization and Authentication	M6: Broken Cryptography	M7: Client Side Injection	M8: Security Decisions via Untrusted Inputs
	M9: Improper Session Handling	M10: Lack of Binary Protections	

M2 – Insecure Data Storage

- Clear-text storage of sensitive/confidential/private data
- Different kind of critical information
  - Account credentials
  - Authentication/authorization tokens
  - Application-specific data containing user's sensitive information
- A privileged access to target device file system is required to properly exploit these issues

M2 – Insecure Data Storage

- Starting from Windows Phone 8, Microsoft's mobile platform supports BitLocker disk encryption technology (AES 128)
- BitLocker is disabled by default
- Built-in encryption can be activated with Exchange ActiveSync policy "RequiredDeviceEncryption" or MDM policies only
- Data encryption represents a crucial security requirement in the WP universe!


Storage Locations
File System
Secure Digital (SD) Cards
Cloud Storage
From WP 8.0 BitLocker encryption technology is supported
BitLocker is disabled by default
Files are NOT encrypted
An app can access files if it has previously registered as an handler for that file type
(via manifest specification)
Data transport security
Data confidentiality preservation

Storage locations and physical paths

Locations	Windows Runtime Apps
Local data store	ApplicationData.Current.LocalFolder - URI - **ms-appdata:///local/** C:\Data\Users\DefApps\APPDATA\Local\Packages\%packageName%**LocalState**
Roaming data store	ApplicationData.Current.RoamingFolder - URI - **ms-appdata:///roaming/** C:\Data\Users\DefApps\APPDATA\Local\Packages\%packageName%\RoamingState
Temporary data store	ApplicationData.Current.TemporaryFolder - URI - **ms-appdata:///temporary/** C:\Data\Users\DefApps\APPDATA\Local\Packages\%packageName%**TempState**
Package installation	Windows.ApplicationModel.Package.Current.InstalledLocation URI: **ms-appx://** or **ms-appx-web://** C:\Data\SharedData\PhoneTools\AppxLayouts\{GUID}\
Cache data store	ApplicationData.Current.LocalCacheFolder C:\Data\Users\DefApps\APPDATA\Local\Packages\%packageName%**LocalCache**

Storage locations and physical paths

Locations	Windows Runtime Apps
Media Library	KnownFolders.MusicLibrary, KnownFolders.CameraRoll, KnownFolders.PicturesLibrary, KnownFolders.VideosLibrary
SD Card	KnownFolders.RemovableDevices
Local Settings	Windows.Storage.ApplicationData.Current.LocalSettings
Roaming Settings	Windows.Storage.ApplicationData.Current.RoamingSettings

Local and Roaming Setting save data in C:\Data\Users\DefApps\APPDATA\Local\Packages\%packageName%\Settings\settings.dat, which is a Windows NT registry file (REGF) - and NOT encrypted

Storage locations and physical paths

Locations	Silverlight Apps
Application local folder	C:\Data\Users\DefApps\APPDATA\{GUID}\Local
Application Settings	IsolatedStorageSettings.ApplicationSettings C:\Data\Users\DefApps\APPDATA\{GUID}\Local__ApplicationSetting
Package installation	Windows.ApplicationModel.Package.Current.InstalledLocation C:\Data\Programs\{GUID}\Install
Cached data	C:\Data\Users\DefApps\APPDATA\{GUID}\INetCache
Cookies	C:\Data\Users\DefApps\APPDATA\{GUID}\INetCookies
SD Card	(read only)

And media library as well.. but it is enough for M2 ☺

Hunting for insecure data storage

Locations	Classes, Methods and Properties	
Local folders	StorageFile	OpenReadAsync() - OpenAsync() GetFileFromApplicationUriAsync() - GetFileFromPathAsync()
	StorageFolder	GetFilesAsync() - GetFileAsync() - CreateFileAsync()
	IsolatedStorageFile.CreateFile() IsolatedStorageFile.OpenFile()	
Application or Roaming Settings	IsolatedStorageSettings.ApplicationSettings – property ApplicationData.LocalSettings – property ApplicationData.RoamingSettings - property	
SD Card (WP 8.1 only)	KnownFolders.RemovableDevices returns a StorageFolder object that can be sequentially used to read/write data from the SD card	
Local database	Identify objects that inherit from System.Data.Linq.DataContext. Verify the existence of reserved data stored in the local .sdf file	

The Pandora's box

- Local databases – reside in app's local folder
  - LINQ to SQL object model is used to interact with the local db
    - The DataContext object is used as a *proxy* for the local database
  - Data is stored in clear-text in «dbname».sdf files
  - SQLite is also a widely adopted solution for local data storage
- Application often relies on settings files which are located into app's local folder - e.g. __ApplicationSettings for Silverlight apps
- Obviously, a custom file format can be adopted as well

Password stored in clear-text

```
private async void DoLogin()
{
    bool? isChecked = this.checkBoxRicordami.IsChecked;
    if ((!isChecked.GetValueOrDefault() ? 0 : (isChecked.HasValue ? 1 : 0)) != 0)
        this.saveCredentials();

    // [...]

    private void saveCredentials()
    {
      if (!(this.textBlockUsername.Text != "") || !(this.textBlockPassword.Password != ""))
        return;

      this.storageSettingsRememberMe.Remove("Username");
      this.storageSettingsRememberMe.Remove("Password");
      this.storageSettingsRememberMe.Remove("isChecked");

      this.storageSettingsRememberMe.Add("Username", this.textBlockUsername.Text);
      this.storageSettingsRememberMe.Add("Password", this.textBlockPassword.Password);
      this.storageSettingsRememberMe.Add("isChecked", true);
      this.storageSettingsRememberMe.Save();
 }
```

credentials saved in application setting file

Session cookies stored in application setting

```
public void SaveState(IDictionary<string, object> stateDictionary)
{
     if (App.ViewModel.LoginMgr.IsLoggedIn)
     {
       // [...]
       List<Cookie> list = new List<Cookie>();
       foreach (Cookie cookie in App.ViewModel.LoginMgr.Cookies)
         list.Add(cookie);

       this.AddToDictionary("Cookies", (object) list);
     }
     // [...]
 }

 private void AddToDictionary(string key, object value)
 {
     IsolatedStorageSettings applicationSettings = IsolatedStorageSettings.ApplicationSettings;

     if (applicationSettings.Contains(key))
       applicationSettings.Remove(key);

     applicationSettings.Add(key, value);
 }
```

session cookies saved in application settings

Secure coding tips

- Private data should *always* be encrypted before storing on device
  - Including data in the local database!
- Windows Phone provides the Data Protection API (DPAPI) as a built-in mechanism to preserve data confidentiality – see M6 for more details
- Credentials should be stored using PasswordVault

Secure coding tips

- Encrypt local databases
  - Define a database access password in ConnectionString
    - The database will be encrypted with AES-128
  - The password still persists in the app's code as a hardcoded *secret*
  - Solution: store the password with PasswordVault, then recover the secret when necessary
  - Encrypt data to be inserted into the db with the DPAPI
- Consider adopting SQLCipher instead of SQLite

OWASP Mobile Top 10 Risks (2014)

M1: Weak Server Side Controls	M2: Insecure Data Storage	M3: Insufficient Transport Layer Protection	M4: Unintended Data Leakage
M5: Poor Authorization and Authentication	M6: Broken Cryptography	M7: Client Side Injection	M8: Security Decisions via Untrusted Inputs
	M9: Improper Session Handling	M10: Lack of Binary Protections	

M3 – Insufficient Transport Layer Security

- Confidentiality and integrity with the app-to-(endpoint) data transmission
  - http-based communication ⊂ WP-supported mechanisms
- Common issues
  - Communication over an unencrypted channel – e.g., http instead of https → MiTM attacks
  - Communication over a poorly encrypted channel – e.g., use of weak encryption mechanisms
  - Issues related to digital certificates – e.g., failures in certificates validation or absence of certificate pinning
    - Overlap with M10 – Lack of binary protections - but with a different "meaning"

HTTP login page loaded via http

```
<phone:PhoneApplicationPage
  x:Class="App.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="PortraitOrLandscape" Orientation="Portrait"
    shell:SystemTray.IsVisible="True">

  <phone:WebBrowser Height="Auto" IsScriptEnabled="true" Source="http://m.WONT-SAY.com/login1.html?continua=true"
    HorizontalAlignment="Stretch" Name="WONT-SAY" VerticalAlignment="Stretch" Width="Auto"
     Margin="-12,0,0,0" Grid.ColumnSpan="2" />

</phone:PhoneApplicationPage>
```

Italian mobile banking app of a leading Danish bank: an attacker can replace the login page with a malicious one

Remote URL loaded via http

```
namespace VulnApp
{
  public class MainPage : PhoneApplicationPage
  {

    private Uri home;
    internal WebBrowser Browser;
    [...]

    public MainPage()
    {
      this.InitializeComponent();
      this.home = new Uri("http://vulnerable.com");
      [...]
    }

    [...]
    private void Home_Click(object sender, EventArgs e)
    {
      this.Browser.Navigate(this.home);
    }
```

an attacker can manipulate the entire app layout

Remote JS file loaded via http

```
public CordovaView()
 {
      this.InitializeComponent();
      if (DesignerProperties.IsInDesignTool)
        return;

      // [...]
      if (this.configHandler.ContentSrc != null)
        this.StartPageUri = !Uri.IsWellFormedUriString(this.configHandler.ContentSrc, UriKind.Absolute) ?
            new Uri(CordovaView.AppRoot + "www/" + this.configHandler.ContentSrc, UriKind.Relative) :
                new Uri(this.configHandler.ContentSrc, UriKind.Absolute);
```

```
<!-- ... -->

<link rel="stylesheet" href="style.css" />
<link rel="stylesheet" href="style-icons.css" />

<script type="text/javascript"
src="http://maps.googleapis.com/maps/api/js?v=3.4&key=ABCDE[...]&libraries=places"></script>

<!-- ... -->
```

www/index.html loads
a remote JS via http == XSS

Dealing with digital certificates

- Windows Phone 8.0 *automagically* discards invalid certificates
  - There are no public APIs to programmatically disable this behavior
- Windows Phone 8.1 allows developers to specify errors to ignore with HttpBaseProtocolFilter.IgnorableServerCertificateErrors.Add()
  - (fortunately) not all exceptions can be ignored

Reference	Ignorable	Not Ignorable
ChainValidationResult (Enumeration)	Expired	Success
	IncompleteChain	Revoked
	WrongUsage	InvalidSignature
	InvalidName	InvalidCertificateAuthorityPolicy
	RevocationInformationMissing	BasicConstraintsError
	RevocationFailure	UnknownCriticalExtension
	Untrusted	OtherErrors

Certificate Pinning on Windows Phone

- In a standard configuration, an attacker may still violate mobile app transmission confidentiality by
  - Inducing the victim to install a malicious certificate (e.g., sent as an email attachment) or
  - Hacking a Certificate Authority (CA) and forging valid certificates
- We need to "pin" the digital certificate to properly mitigate these category of attacks
  - WP 8.0 apps require third parties libraries (e.g., EldoS SecureBlackbox)
  - WP 8.1 provides StreamSocket.Information that can be use to access StreamSocketInformation.ServerCertificate, which allows getting the remote server digital certificate – and its details [2]

Hunting for transport issues – part I

Category	Namespaces	Classes, Methods and Properties	
HTTP	System.Net.Http.HttpClient	DeleteAsync() GetAsync() PostAsync() PutAsync()	GetByteArrayAsync() GetStreamAsync() GetStringAsync() SendAsync()
	Windows.Web.Http.HttpClient	DeleteAsync() GetAsync() PostAsync() PutAsync()	GetStringAsync() SendRequestAsync() GetBufferAsyn() GetInputStreamAsync()
TCP and UDP Sockets	Windows.Networking.Sockets	StreamSocket.ConnectAsync() SocketProtectionLevel.PlainSocket - property StreamSocket.UpgradeToSslAsync() StreamSocketListener - does not support SSL/TLS DatagramSocket.ConnectAsync()	

Hunting for transport issues – part II

Category	Namespaces	Classes, Methods and Properties
Web	Microsoft.Phone.Controls	WebBrowser.Navigate() WebBrowser.Source property
	Windows.UI.Xaml.Controls	WebView.Navigate() WebView.Source property
	Microsoft.Phone.Tasks	WebBrowserTask.Uri property
	Windows.System	Launcher.LaunchUriAsync(uri)
WebSocket	Windows.Networking.Sockets	MessageWebSocket.ConnectAsync() – with ws:// uri scheme StreamWebSocket.ConnectAsync() – with ws:// uri scheme

Hunting for transport issues – part II

Category	Namespaces	Classes, Methods and Properties
XAML Object Element Usage	-	«Source» property for **WebBrowser** and **WebView** «uri» property for **WebBrowserTask** "NavigateUri" for **HyperlinkButton**
Push Notifications	Microsoft.Phone.Notification	**HttpNotificationChannel(string)**
Brutal approach	-	Grep for **Uri()** and look at http:// instead of https://
Digital Certificates	Windows.Web.Http.Filters	**HttpBaseProtocolFilter.IgnorableServerCertificateErrors.Add()**
Windows.Web.AtomPub, Windows.Networking.BackgroundTransfer, Windows.Web.Syndication classes/methods should be reviewed as well		

Secure coding tips

- *Uri() → https://*
- Adopt standard encryption solutions instead of custom ones
- Implement Certificate Pinning

OWASP Mobile Top 10 Risks (2014)

M1: Weak Server Side Controls	M2: Insecure Data Storage	M3: Insufficient Transport Layer Protection	M4: Unintended Data Leakage
M5: Poor Authorization and Authentication	M6: Broken Cryptography	M7: Client Side Injection	M8: Security Decisions via Untrusted Inputs
	M9: Improper Session Handling	M10: Lack of Binary Protections	

M4 – Unintended Data Leakage

- Involuntary data exposure caused by OS or frameworks *side-effects*
- Potential sources of information leakage
  - System caching
  - Application backgrounding
  - System logging
  - Telemetry frameworks, which expose sensitive data
    - e.g. (plain-text) transmission of exception messages containing private data
- A privileged access to target device file system - or connected network - is required to properly exploit these issues

Hunting for potential data leakage

Conditions	Classes, Methods and Properties
Application Backgrounding and Closing	Handler for the Application.Suspending event, typically the **OnSuspending()** method in App.xaml.cs
	Handler for the Application.Deactivated event, typically the **Application_Deactivated()** method in App.xaml.cs
	Handler for the Application.Closing event, typically the **Application_Closing()** method in App.xaml.cs
	Handler for the Application.UnhandledException event, typically the **Application_UnhandledException()** method in App.xaml.cs
Use of Telemetry Frameworks	HockeyApp, BugSense, etc.
Dump of app memory	check for encryption keys or passwords stored as «string» (immutable) object (.NET's **System.Security.SecureString** object is not supported by WP Silverlight | WinRT)

Leakage via cached data and cookies


```
private void Application_Deactivated(object sender, DeactivatedEventArgs e)
{
}

private void Application_Closing(object sender, ClosingEventArgs e)
{
}
```

On closing or deactivation the app does not "clean" data, which are saved by the OS

Cached data + saved cookies by WebBrowser or WebView are NOT cleaned from:

C:\Data\Users\DefApps\APPDATA\{GUID}\INetCookies
C:\Data\Users\DefApps\APPDATA\{GUID}\INetCache

Secure coding tips

Actions	Classes, Methods or Properties	
Remove cached data on app closing, suspension or deactivation	server-side	Cache-Control: no-store
	client-side	WebBrowserExtensions.ClearInternetCacheAsync() WebBrowser.ClearInternetCacheAsync() WebView - no programmatic way
Remove stored cookies	WebBrowser.ClearCookiesAsync() WebBrowserExtensions.ClearCookie() WebView – use HttpCookieManager.GetCookies() + HttpCookieManager.DeleteCookie()	
Avoid sensitive data disclosure (dump of app's memory)	Use of byte[] array instead of System.String objects, and re-assign bytes when the "secret" is no longer necessary	

OWASP Mobile Top 10 Risks (2014)

M1: Weak Server Side Controls	M2: Insecure Data Storage	M3: Insufficient Transport Layer Protection	M4: Unintended Data Leakage
M5: Poor Authorization and Authentication	M6: Broken Cryptography	M7: Client Side Injection	M8: Security Decisions via Untrusted Inputs
	M9: Improper Session Handling	M10: Lack of Binary Protections	

M5 – Poor Authorization and Authentication

- Security *decisions* without server-side engagement
- Common client-side issues
  - Offline authentication
  - Issues related to password complexity (e.g., 4 digits PIN)
  - Absence of anti-guessing or brute forcing mechanisms
  - Authorization issues on apps critical functions/data access
  - Predictable authentication/authorization tokens
- Similar issues also affect the *server-side*

No authentication on backup access

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);
    using (IsolatedStorageFile storeForApplication = IsolatedStorageFile.GetUserStoreForApplication())
    {
        this.fileExists = storeForApplication.FileExists("wp contacts backup.zip");
        if (!this.fileExists)
        {
            this.infoTextBlock.Text = "No backup file exists! Please create one before trying to download it.";
        }
        else
        {
            try
            {
                this.server = new HttpServer(2, 65536);
                this.server.Start(new IPEndPoint(IPAddress.Parse("0.0.0.0"), 5656));
                this.server.TextReceived += new EventHandler<HttpDataReceivedEventArgs>(this.server_TextReceived);
                this.infoTextBlock.Text = "http://" + this.server.LocalEndpoint.ToString();
            }
            catch
            {
                this.infoTextBlock.Text = "Unable to start WEB Server. Please check your connectivity settings.";
            }
        }
    }
}
```

contacts backup file is stored in app's sandbox

No authentication on backup access

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);
    using (IsolatedStorageFile storeForApplication = IsolatedStorageFile.GetUserStoreForApplication())
    {
        this.fileExists = storeForApplication.FileExists("wp contacts backup.zip");
        if (!this.fileExists)
        {
            this.infoTextBlock.Text = "No backup file exists! Please create one before trying to download it.";
        }
        else
        {
            try
            {
                this.server = new HttpServer(2, 65536);
                this.server.Start(new IPEndPoint(IPAddress.Parse("0.0.0.0"), 5656));
                this.server.TextReceived += new EventHandler<HttpDataReceivedEventArgs>(this.server_TextReceived);
                this.infoTextBlock.Text = "http://" + this.server.LocalEndpoint.ToString();
            }
            catch
            {
                this.infoTextBlock.Text = "Unable to start WEB Server. Please check your connectivity settings.";
            }
        }
    }
}
```

local web server lacks any authentication mechanism

Client-side generation of authorization tokens

```
public byte[] GetPostData()
{
  string date = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss", (IFormatProvider) CultureInfo.InvariantCulture);

  return Encoding.UTF8.GetBytes(string.Format("token={0}&&msisdn={1}&InteractiveType=Web&dt={2}",
    this.GetAuthorizationToken(), HttpUtility.UrlEncode(this.par), HttpUtility.UrlEncode(date)));
}

private string GetAuthorizationToken()
{
  string str = MD5Core.GetHashString("RYKn92938339944005kf8fk9ekwdkwud83jud3" + this.par).ToLower();
  Utils.Log("MD5 Token: " + str);
  return str;
}
```

token generation logic can be easily reproduced

Hunting for weak tokens forgery

Identification Data	Namespaces	Classes, Methods and Properties
Device Name	Microsoft.Phone.Info	DeviceStatus.DeviceName
Hardware Identification	Microsoft.Phone.Info	DeviceExtendedProperties.GetValue("DeviceName")
		DeviceExtendedProperties.GetValue("DeviceUniqueId")
Hashing Functions	Windows.Security.Cryptography	SHA1Managed, SHA256Managed, SHA384Managed and SHA512Managed classes (or any other 3° party libraries implementing these functions)
	Windows.Security.Cryptography.Core	HashAlgorithmProvider.OpenAlgorithm()
Geo Location Coordinates	Windows.Devices.Geolocation	Geolocator / Geoposition / Geocoordinate
	System.Device.Location	GeoCoordinateWatcher / GeoPosition / GeoCoordinate

Secure coding tips

- We don't want you to (client-side) generate authN/authZ tokens
  - However, if you really need a (real) UUID use the HostInformation.PublisherHostId property
  - The generated string is unique per device and per publisher, while the DeviceExtendedProperties.GetValue("DeviceUniqueId") is unique only *per device* – so extremely unsafe
- Implement proper client-side authorization mechanisms on OnNavigatedTo() methods referring to XAML pages that expose critical functionalities

OWASP Mobile Top 10 Risks (2014)

M1: Weak Server Side Controls	M2: Insecure Data Storage	M3: Insufficient Transport Layer Protection	M4: Unintended Data Leakage
M5: Poor Authorization and Authentication	M6: Broken Cryptography	M7: Client Side Injection	M8: Security Decisions via Untrusted Inputs
	M9: Improper Session Handling	M10: Lack of Binary Protections	

M6 – Broken Cryptography

- Risk associated with both *local* and *in-transit* data encryption
  - Clear overlap with M3 – Insufficient Transport Layer Security
- Use of weak cryptographic algorithms
  - Weak "standard" or custom algorithms
  - Exotic "encryption" strategies
- Weak encryption processes
  - Hardcoded encryption keys
  - Encryption keys stored with the encrypted data or in *unsafe areas*

Hardcoded encryption key

```
private static readonly string SaltKey = "*********salt_here_******";

public static string EncryptPlainText(string dataToEncrypt)
{
  AesManaged aesManaged = new AesManaged();
  byte[] bytes1 = new UTF8Encoding().GetBytes(SecurityHelper.SaltKey);

  Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(SecurityHelper.SaltKey, bytes1);
  aesManaged.Key = rfc2898DeriveBytes.GetBytes(16);
  aesManaged.IV = rfc2898DeriveBytes.GetBytes(16);
  aesManaged.BlockSize = 128;

  using (MemoryStream memoryStream = new MemoryStream())
  {
    using (CryptoStream cryptoStream = new CryptoStream((Stream) memoryStream,
      aesManaged.CreateEncryptor(), CryptoStreamMode.Write))
    {
      byte[] bytes2 = Encoding.UTF8.GetBytes(dataToEncrypt);
      cryptoStream.Write(bytes2, 0, bytes2.Length);
      cryptoStream.FlushFinalBlock();
      cryptoStream.Close();
      return Convert.ToBase64String(memoryStream.ToArray());
    }
  }
}
```

hardcoded (and same)
symmetric encryption key and salt

Encoding instead of encryption

```
private void GetUserCompleted(object sender, EventArgs e)
{
  if (e == null)
  {
    // ...
  }
  else
  {
    NetUserCompletedEventArgs completedEventArgs = (NetUserCompletedEventArgs) e;
    byte[] numArray1 = Crypto.encryptString(completedEventArgs.user.username);
    byte[] numArray2 = Crypto.encryptString(completedEventArgs.user.password);
    this.isolatedStorageSettings.StoreValueForKey("Username", (object) numArray1);
    this.isolatedStorageSettings.StoreValueForKey("Password", (object) numArray2);
    CurrentAppConfig.Instance.User = completedEventArgs.user;
    this.storeCurrentUserStoresPreferences(completedEventArgs.user);
  }
}


public class Crypto
{
  public static byte[] encryptString(string input)
  {
    return Encoding.UTF8.GetBytes(input);
  }
```

"encrypted" credentials are stored into the sandbox

Hunting for encryption failures

Functions	Namespaces	Classes, Methods and Properties
Hashing	Windows.Security.Cryptography	SHA1Managed, SHA256Managed, SHA384Managed and SHA512Managed classes (or any other 3° party libraries implementing these functions)
	Windows.Security.Cryptography.Core	HashAlgorithmProvider.OpenAlgorithm()
Symmetric Encryption	System.Security.Cryptography.Core	CryptographicEngine.Encrypt() | Decrypt()
	System.Security.Cryptography	AesManaged.CreateEncryptor() | CreateDencryptor()

Hunting for encryption failures

Functions	Namespaces	Classes, Methods and Properties
Data Encoding	Windows.Security.Cryptography	CryptographicBuffer.[Encode | Decode]ToBase64String() CryptographicBuffer. [Encode | Decode]ToHexString()
	System.Text	Encoding.UTF8
	System	Convert.ToBase64String() | Convert.FromBase64String()
Plenty of third party encryption libraries (e.g., Bouncy Castle for .NET) implement similar algorithms		

Secure coding tips

- Your new mantra: *do not store (even encrypted) critical data on device*
  - I know, user experience could be damaged
- Another mantra: *serialization is just a data representation not encryption at all*
- Store sensitive data on device adopting the Data Protection API (DPAPI)

Platform	Namespaces	Methods
WP 8.0	System.Security.Cryptography	ProtectedData.Protect() ProtectedData.Unprotect()
WP 8.1	Windows.Security.Cryptography	DataProtectionProvider.ProtectAsync() DataProtectionProvider.UnprotectAsync() DataProtectionProvider.ProtectStreamAsync() DataProtectionProvider.UnprotectStreamAsync()

OWASP Mobile Top 10 Risks (2014)

M1: Weak Server Side Controls	M2: Insecure Data Storage	M3: Insufficient Transport Layer Protection	M4: Unintended Data Leakage
M5: Poor Authorization and Authentication	M6: Broken Cryptography	M7: Client Side Injection	M8: Security Decisions via Untrusted Inputs
	M9: Improper Session Handling	M10: Lack of Binary Protections	

M7 – Client Side Injection

- *Feeding* an interpreter with untrusted data
  - Similar to the server-side ones but involve the *app-side*
- Common interpreters that could be attacked
  - Local database querying systems
  - XML parsers
  - HTML rendering engines
  - File handling routines
- Attacks impact depends on data stored on device

Hunting for untrusted data sources

- We need mapping the sources of untrusted data and reviewing the "parsing" routines
- Examples of sources of untrusted data
  - Input from network – e.g., web responses or any other network communications
  - Bluetooth or NFC
  - Inter Processor Communication (IPC) mechanism - e.g., via extensions/protocols registration or toast notifications
  - Files accessed from SD card – which is a shared storage area
  - User typed input – via UI, speech to text, camera (e.g., QR code), USB data, etc.

Hunting hard for injection flaws

Interpreters	Namespaces	Classes, Methods and Properties	
HTML/JavaScript	Microsoft.Phone.Controls	WebBrowser	NavigateToString() InvokeScript() IsScriptEnabled = true (property)
	Windows.UI.Xaml.Controls	WebView	NavigateToString() InvokeScript() InvokeScriptAsync() NavigateToLocalStreamUri() NavigateWithHttpRequestMessage()
XML	System.Xml.Linq	XDocument.Load()	
	System.Xml	XmlReaderSettings.DtdProcessing = DtdProcessing.Parse	
XAML	System.Windows.Markup	XamlReader.Load()	

Hunting hard for injection flaws

Interpreters	Third Parties Libraries	Classes, Methods and Properties
SQL	SQLitePCL	SQLiteConnection.Prepare()
	SQLite-Net-WP8	Query() / Query<T>() / QueryAsync<T>() Execute() / ExecuteAsync() ExecuteScalar<T>() / ExecuteScalarAsync<>() DeferredQuery() / DeferredQuery<T>() FindWithQuery<T>() CreateCommand()
	CSharp-SQLite	IDbCommand.CommandText (property)
	SQLiteWinRT	Database.ExecuteStatementAsync() Database.PrepareStatementAsync()

Hunting for file and path names manipulation

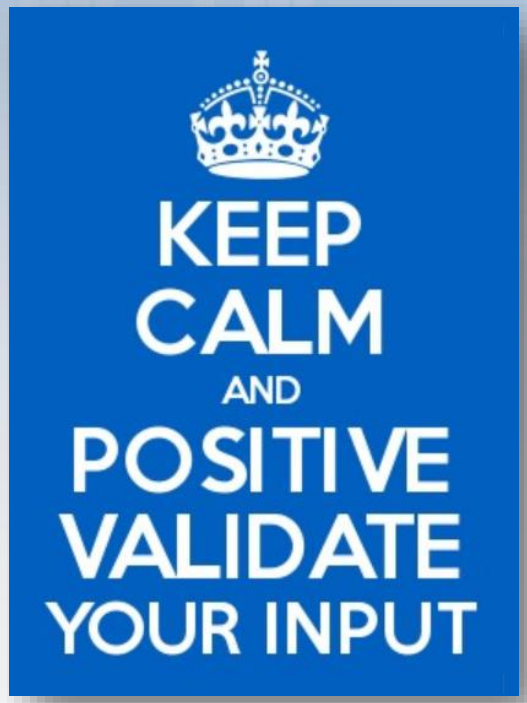
Interpreters	Classes, Methods and Properties	
File handling	StorageFolder	CreateFileAsync() RenameAsync() GetFolderFromPathAsync() GetFolderAsync()
	StorageFile	CopyAsync() GetFileFromApplicationUriAsync() GetFileFromPathAsync() RenameAsync()
	IsolatedStorageFile	OpenFile() CopyFile() CreateDirectory() – CreateFile() DeleteDirectory() - DeleteFile()

Just click to XSS

```
private void ButtonView_Click(object sender, RoutedEventArgs e)
{
    this.ButtonView.IsEnabled = false;
    this.intexttemp = this.TextIP.Text.Trim() + "xxxxxxxx";
    this.WebBrowser1.NavigateToString("<body bgcolor=black>" +
      "<form action='https://www.REMOTE-SITE.com/path/resource.asp' method=post>" +
       "<input name='iname' value='" + this.TextAdmin.Text.Trim() + "' type='hidden >" +
         "<input name='pword' value='" + this.PasswordBox1.Password.Trim() +
           "' type=hidden><input name='ip' value='" + this.TextIP.Text.Trim() +
             "' type=hidden><input name='port' value='" + this.TextPort.Text.Trim() +
             "' type=hidden><input name='vers1' value='or1' type='hidden'></form>" +
             "<script>document.forms[0].submit();</script></body>");
```

app renders user-controlled data without any validation

Secure coding tips



- All input is evil, simply trust no one! [3]
- Adopt *positive validation* strategies
- Adopt parametrized queries to avoid SQL Injection
- Do not allow DTD parsing when working with XML documents

OWASP Mobile Top 10 Risks (2014)

M1: Weak Server Side Controls	M2: Insecure Data Storage	M3: Insufficient Transport Layer Protection	M4: Unintended Data Leakage
M5: Poor Authorization and Authentication	M6: Broken Cryptography	M7: Client Side Injection	M8: Security Decisions via Untrusted Inputs
	M9: Improper Session Handling	M10: Lack of Binary Protections	

M8 – Security Decision via Untrusted Inputs

- Do you apply input validation and properly authorize sensitive actions with Inter Process Communication (IPC)?
- Windows Phone platform provides limited support to IPC
  - WP 7.x does not support IPC
  - WP 8.0 and 8.1 provides files and URI associations

File and protocol handlers – WP 8.0

IPC Mechanism	Supported Platform and Manifest Specifications (WMAppManifest.xml)
File association	<Extensions> <FileTypeAssociation Name="name" TaskID="_default" NavUriFragment="fileToken=%s"> […] <SupportedFileType> <FileType ContentType="application/sdk">.test1</FileType> <FileType ContentType="application/sdk">.test2</FileType> </SupportedFileTypes> </FileTypeAssociation> </Extensions>
	email attachment, SD cards, website via IE, WebBrowser/WebView, NFC-enabled devices or another app from the Store (Launcher.LaunchFileAsync)
URI association	<Extensions> <Protocol Name="luca" NavUriFragment="encodedLaunchUri=%s" TaskID="_default" /> </Extensions>
	click here (IE/WebBrowser/WebView), other apps via Launcher.LaunchUriAsync("luca:..") or NFC-enabled devices

File and protocol handlers – WP 8.1

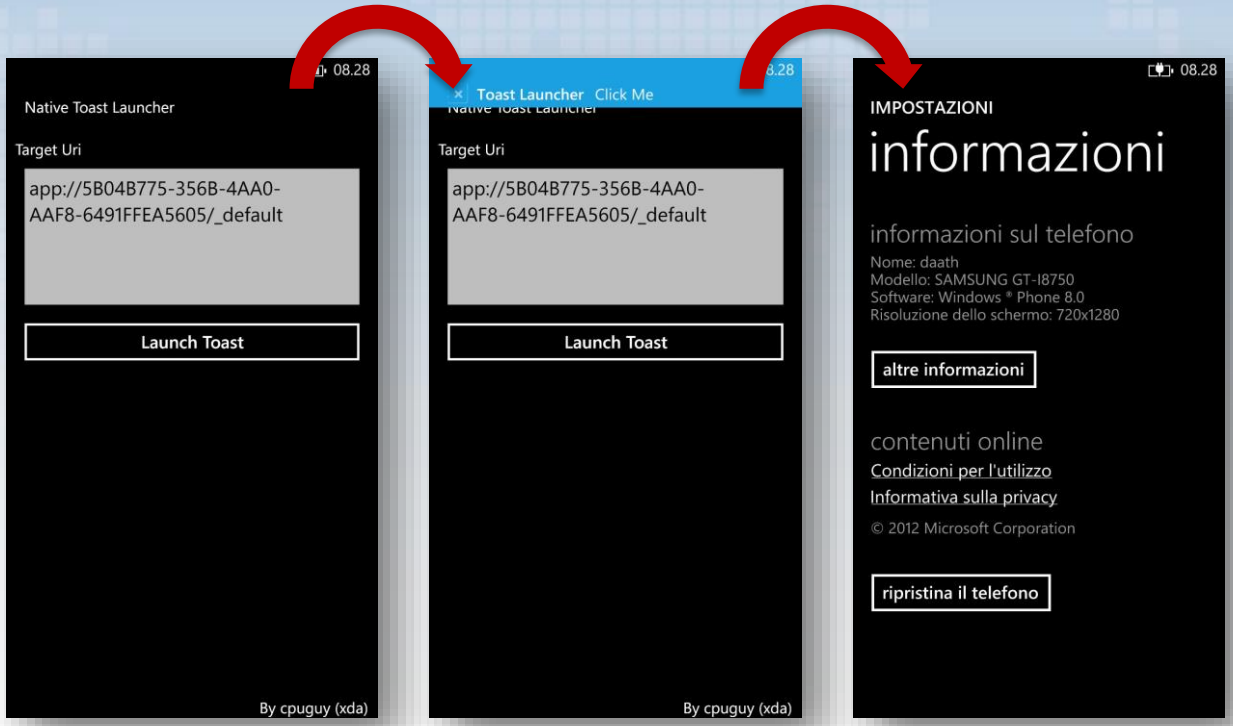
IPC Mechanism	Supported Platform and Manifest Specifications (package.appxmanifest)
File association	<Extensions> <Extension Category="windows.fileTypeAssociation"> <FileTypeAssociation Name="test3"> <DisplayName>My test 3</DisplayName> […] <SupportedFileTypes> <FileType ContentType="image/jpeg">.test1</FileType> </SupportedFileTypes> </FileTypeAssociation> </Extension> </Extensions>
URI association	</Extensions> <Extension Category="windows.protocol"> <Protocol Name="luca" m2:DesiredView="useLess"/> <Logo>images\logo.png</Logo> <DisplayName>My uri has my name</DisplayName> </Extension> </Extensions>

M8 – Security Decision via Untrusted Inputs

- The undocumented Shell_PostMessageToast method (ShellChromeAPI.dll) – discovered by cpuguy from XDA [4] – can be abused to perform *Cross Application Navigation Forgery* attacks
  - The term has been coined by Alex Plaskett and Nick Walke from MWR [5]
- Basically a malicious app can use the Shell_PostMessageToast() method to send a *toast message* that, once tapped, allows to open an arbitrary XAML page of an arbitrary app – XAML page code behind can be fed with malicious input

Cross application navigation forgery attacks

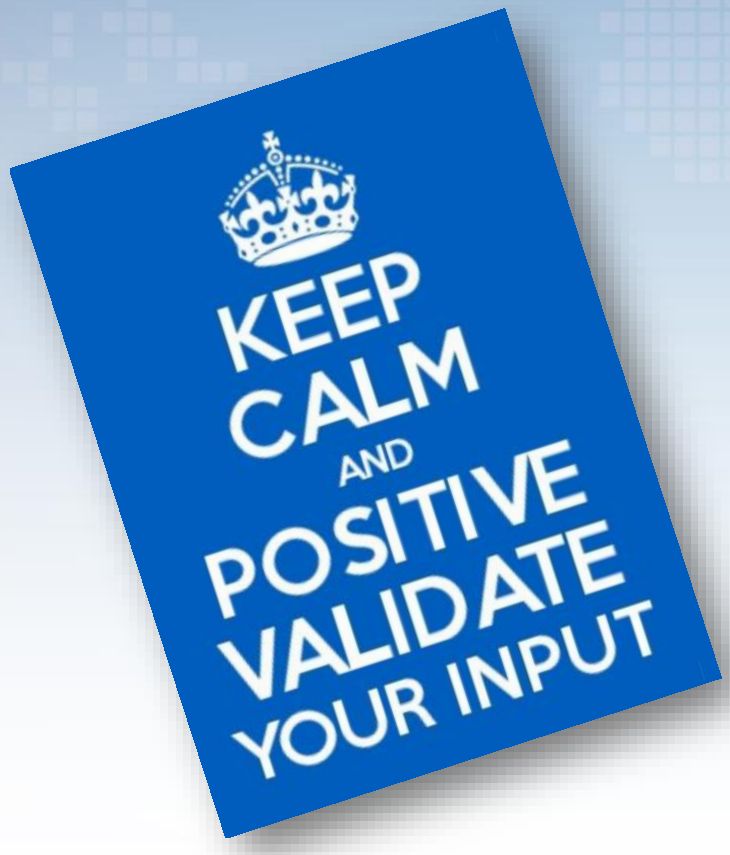
app://{GUID}/_default#/AssemblyName;component/Page.xaml?par=AAAAA

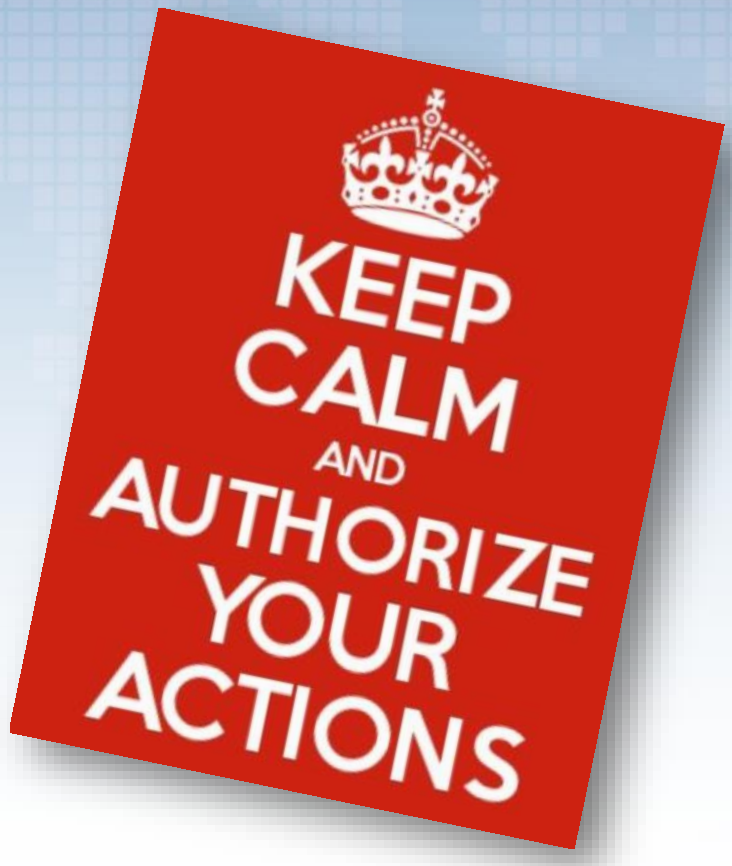


Hunting for IPC mechanisms

IPC Mechanism	Platform	Namespaces	Classes, Methods and Properties
URI associations	WP 8.0	System.Windows.Navigation	overridden **UriMapperBase.MapUri()** method
	WP 8.1	Windows.Ui.Xaml.Application	**OnActivated()** - ActivationKind.Protocol property
File associations	WP 8.0	System.Windows.Navigation	overridden **UriMapperBase.MapUri()** method
	WP 8.1	Windows.Ui.Xaml.Application	**OnFileActivated()** method
(Toast Message)	WP 8.0	System.Windows.Navigation	**OnNavigatedTo()** (NavigationContext.QueryString)

Secure coding tips





OWASP Mobile Top 10 Risks (2014)

M1: Weak Server Side Controls	M2: Insecure Data Storage	M3: Insufficient Transport Layer Protection	M4: Unintended Data Leakage
M5: Poor Authorization and Authentication	M6: Broken Cryptography	M7: Client Side Injection	M8: Security Decisions via Untrusted Inputs
	M9: Improper Session Handling	M10: Lack of Binary Protections	

M9 – Improper Session Handling

- Insecure app sessions life-cycle
  - Issues related to both client and server-side "session handlers"
- Common session-related security issues
  - Failure to invalidate sessions on the backend
  - Lack of adequate timeout protection
  - Failure to properly rotate cookies
  - Insecure tokens creation
    - Clear overlap with M5 – Poor Authorization and Authentication
  - Failure to invalidate sessions on app closing or deactivation

Failure to invalidate sessions

```
public void Logout()
{

 if (this.AppState.StCookies != null && this.AppState.StCookies.Count > 0)
 {
   foreach (KeyValuePair<string, System.Net.Cookie> keyValuePair in this.AppState.StCookies)
   {
     System.Net.Cookie cookie = keyValuePair.Value;
     cookie.Expired = true;
     cookie.Discard = true;
   }
 }

 this.AppState.StCookies = (Dictionary<string, System.Net.Cookie>) null;

 ((Frame) this.rootFrame).Navigate(new Uri(ViewList.PreLogin, UriKind.Relative));
}
```

no server-side session cookies invalidation mechanism is involved in the logout process

Hunting and fixing session issues

- Carefully look at cookie handling routines and usage, and make sure to clean all your cookies when they are not needed

Namespaces	Classes, Methods and Properties
System.Net	Cookie CookieCollection CookieContainer HttpWebRequest.CookieContainer HttpWebResponse.Cookies
Windows.Web.Http (WP 8.1 only)	HttpCookie HttpCookieCollection HttpCookieManager

OWASP Mobile Top 10 Risks (2014)

M1: Weak Server Side Controls	M2: Insecure Data Storage	M3: Insufficient Transport Layer Protection	M4: Unintended Data Leakage
M5: Poor Authorization and Authentication	M6: Broken Cryptography	M7: Client Side Injection	M8: Security Decisions via Untrusted Inputs
	M9: Improper Session Handling	M10: Lack of Binary Protections	

M10 – Lack of Binary Protections

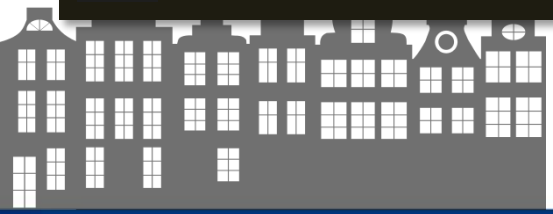
- We want a *self-defending app* from binary attacks
- Preventing app analysis
  - Certificate pinning
  - Anti-debugging and runtime-tampering detection mechanisms
- Preventing reverse engineering
  - Code obfuscation and code encryption
- Preventing app modification
  - Anti-jailbreaking routines
  - App resources integrity verification mechanisms

How (not) to encrypt apps code

```
private void CordovaBrowser_Loaded(object sender, RoutedEventArgs e)
{
  this.resourceStreamInformation =
  Application.GetResourceStream(new Uri(Resource1.WWWPath, UriKind.Relative));

  // [ ... ]
  string encodedPassword = Resource1.EncodedPassword;

  this.strPasswordDecodingSecond =
    CordovaView.Base64Decode(CordovaView.Base64Decode(
      encodedPassword.Substring(0, encodedPassword.Length - 10)));

  this.passwordLength = this.strPasswordDecodingSecond.Length;
  this.stream = this.resourceStreamInformation.Stream;
  this.filebytes = Convert.FromBase64String(CordovaView.StreamToString(this.stream));

  this.Unzip(new MemoryStream(
            this.Decrypt(this.filebytes, this.strPasswordDecodingSecond, this.passwordLength)
          ));

  this.RetrievePage(); // CordovaView.uri setting

  this.CordovaBrowser.Navigate(CordovaView.uri); // Navigate unzipped app index.html page

}
```

pathname of the encrypted ZIP file

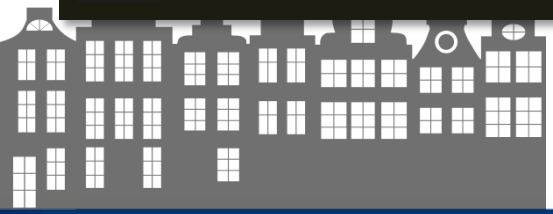
hardcoded password



How (not) to encrypt apps code

```
private void CordovaBrowser_Loaded(object sender, RoutedEventArgs e)
{
  this.resourceStreamInformation =
  Application.GetResourceStream(new Uri(Resource1.WWWPath, UriKind.Relative));

  // [...]
  string encodedPassword = Resource1.EncodedPassword;

  this.strPasswordDecodingSecond =
    CordovaView.Base64Decode(CordovaView.Base64Decode(
      encodedPassword.Substring(0, encodedPassword.Length - 10)));

  this.passwordLength = this.strPasswordDecodingSecond.Length;
  this.stream = this.resourceStreamInformation.Stream;
  this.filebytes = Convert.FromBase64String(CordovaView.StreamToString(this.stream));

  this.Unzip(new MemoryStream(
              this.Decrypt(this.filebytes, this.strPasswordDecodingSecond, this.passwordLength)
          ));

  this.RetrievePage(); // CordovaView.uri setting

  this.CordovaBrowser.Navigate(CordovaView.uri); // Navigate unzipped app index.html page

}
```

Unizip() calls the UnzipAndSaveFiles() method



How (not) to encrypt apps code

```
public void UnzipAndSaveFiles(Stream stream)
{
    // [...]

    using (ZipInputStream zipInputStream = new ZipInputStream(stream))
    {
        storeForApplication.CreateDirectory(Resource1.WWWDirectory);

        ZipEntry nextEntry;
        while ((nextEntry = zipInputStream.GetNextEntry()) != null)
        {
            // [...]
                str1 = Path.Combine(Resource1.WWWDirectory, strArray[index]);
                if (!storeForApplication.DirectoryExists(str1))
                    storeForApplication.CreateDirectory(str1);
```

unzipped file content is saved in the SANDBOX

A note on apps encryption

- Windows Phone Store apps are downloaded as encrypted files
- Apps are then decrypted during the deployment phase
  - A privileged access to the device file system allows "clear-text" apps code extraction
- Code obfuscation and app code encryption are effective strategies to mitigate – but not solve – binaries reversing

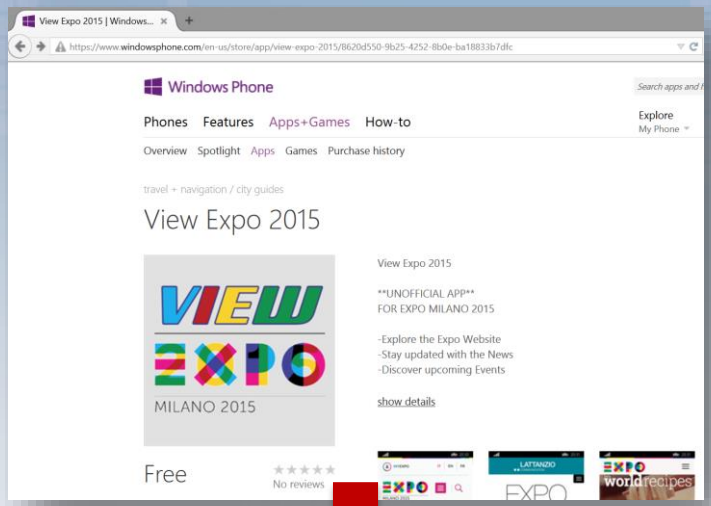
AppX from Store are NOT encrypted

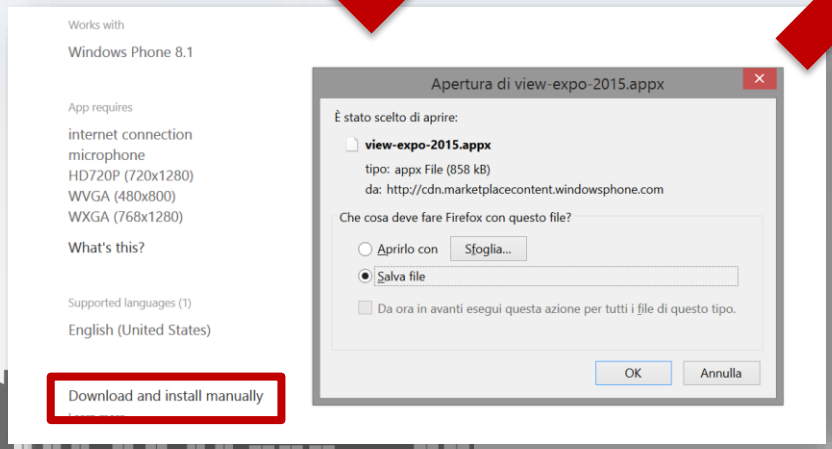
Comparison by feature by package format

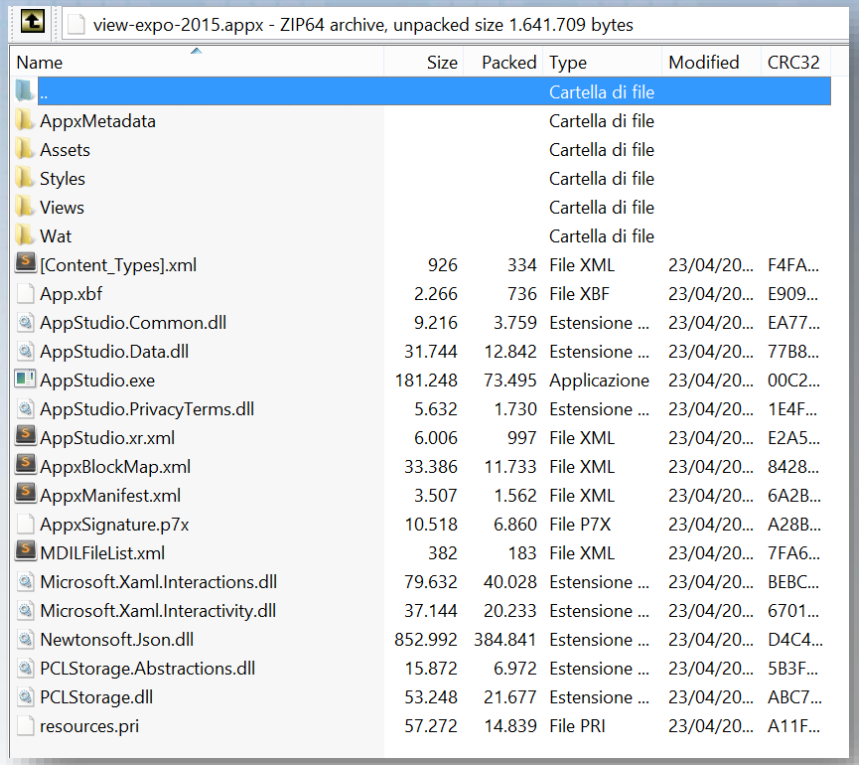
In summary...

Feature	XAP Phone	XAP 8.1 Phone	AppX Phone	AppX Windows
Platform Targeting	7.x and later	8.1 and later	8.1 and later	8.0 and later
Package Encryption	Yes	Yes	No, not yet.	No, not yet.
Package Bundling	No	No	Yes	Yes
Debug Package Signing	No	No	No	Yes
Differential Download/Update	No	No	Yes	Yes
Application File Single Instancing	No	No	Yes	Yes
Formal Versioning Requirements	No	Yes	Yes	Yes
External Volume (SD) App Installation	Yes on 8.1	Yes	Yes	No, not yet.

AppX from Store are NOT encrypted







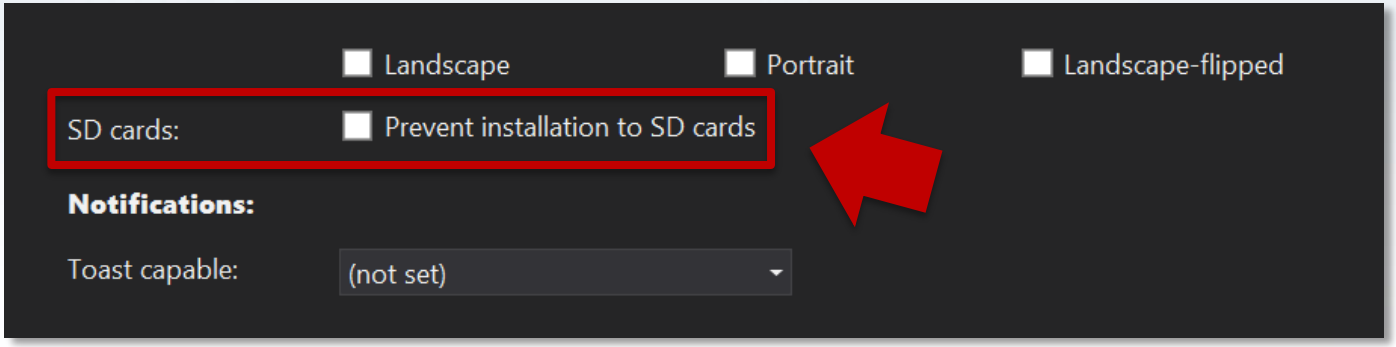
Name	Size	Packed	Type	Modified	CRC32
..			Cartella di file		
AppxMetadata			Cartella di file		
Assets			Cartella di file		
Styles			Cartella di file		
Views			Cartella di file		
Wat			Cartella di file		
[Content_Types].xml	926	334	File XML	23/04/20...	F4FA...
App.xbf	2.266	736	File XBF	23/04/20...	E909...
AppStudio.Common.dll	9.216	3.759	Estensione ...	23/04/20...	EA77...
AppStudio.Data.dll	31.744	12.842	Estensione ...	23/04/20...	77B8...
AppStudio.exe	181.248	73.495	Applicazione	23/04/20...	00C2...
AppStudio.PrivacyTerms.dll	5.632	1.730	Estensione ...	23/04/20...	1E4F...
AppStudio.xr.xml	6.006	997	File XML	23/04/20...	E2A5...
AppxBlockMap.xml	33.386	11.733	File XML	23/04/20...	8428...
AppxManifest.xml	3.507	1.562	File XML	23/04/20...	6A2B...
AppxSignature.p7x	10.518	6.860	File P7X	23/04/20...	A28B...
MDILFileList.xml	382	183	File XML	23/04/20...	7FA6...
Microsoft.Xaml.Interactions.dll	79.632	40.028	Estensione ...	23/04/20...	BEBC...
Microsoft.Xaml.Interactivity.dll	37.144	20.233	Estensione ...	23/04/20...	6701...
Newtonsoft.Json.dll	852.992	384.841	Estensione ...	23/04/20...	D4C4...
PCLStorage.Abstractions.dll	15.872	6.972	Estensione ...	23/04/20...	5B3F...
PCLStorage.dll	53.248	21.677	Estensione ...	23/04/20...	ABC7...
resources.pri	57.272	14.839	File PRI	23/04/20...	A11F...

Binary Protection 101 – app analysis

- Anti-debugging and anti-runtime tampering
  - Code obfuscator also implements this kind of mechanisms (e.g., dotFuscator and ConfuserEx)
- Certificate Pinning
  - Slows down https traffic analysis because the attacker is required to *unpin* the certificate, modifying the victim-app
  - We already discussed the technological solutions in M3
- Anti-jailbreaking* mechanisms
  - In the Windows Phone universe, these mechanisms would require the use of privileged APIs that normally are not granted to Independent Software Vendors (ISV) [6]

Binary Protection 102 – app reversing

- Obfuscate and encrypt your code, always
- Prevent app installation on SD cards
  - Recent Capabilities Hack allows access to code/data on SD card
- OWASP RE and Code Modification Prevention Project [7] provides architectural principles to securely design your apps



Final considerations

- We presented the first public, most accurate and complete catalog of potentially insecure APIs for WP apps
- Our work started back in 2014, when we analyzed several Silverlight apps to contribute with statistics to the MTT 2015
- The research has been extended, assessing a series of Windows Runtime apps and identifying potential insecure usage of related APIs
- Things are changing fast with Microsoft dev technologies - see Build 2015 - but substantial part of our work on APIs security should be valid for the Universal Apps

Thank you!

@_daath ~ luca@securenetwork.it ~ blog.nibblesec.org

References

1. MSDN – API References for Windows Runtime Apps
   - https://msdn.microsoft.com/en-us/library/windows/apps/xaml/br211369.aspx
2. Certificate Pinning in Mobile Applications
   - http://www.slideshare.net/iazza/certificate-pinning-in-mobile-applicationsprosconsv10
3. Input Validation Cheat Sheet
   - https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet
4. Native Toast Notification Launcher
   - http://forum.xda-developers.com/windows-phone-8/help/qa-native-toast-notification-launcher-t2980873
5. MWR's Navigating a Sea of Pwn?
   - https://labs.mwrinfosecurity.com/system/assets/651/original/mwri_wp8_appsec-whitepaper-syscan_2014-03-30.pdf

References

6. OWASP Mobile Jailbreaking Cheat Sheet
   - https://www.owasp.org/index.php/Mobile_Jailbreaking_Cheat_Sheet
7. OWASP Reverse Engineering and Code Modification Prevention Project
   - https://www.owasp.org/index.php/OWASP_Reverse_Engineering_and_Code_Modification_Prevention_Project
8. Windows Phone 8.1 Security Overview
   - https://www.microsoft.com/en-us/download/details.aspx?id=42509
9. Windows Phone 8 Application Security
   - http://erpscan.com/wp-content/uploads/2013/06/Windows-Phone-8-application-security-slides.pdf