Windows Phone 7 Application Security Survey: a look at popular apps and their data storage practices

Andy Grant — agrant[at]isecpartners[dot]com

iSEC Partners, Inc
123 Mission Street, Suite 1020
San Francisco, CA 94105
https://www.isecpartners.com

August 21, 2013

Abstract

As more people use mobile devices for sensitive tasks, such as online banking and password storage, the data stored on the device increases in value. With each new mobile platform there are more opportunities for a mobile application developer to store data in an insecure manner. This paper looks at how popular Windows Phone 7 apps address data storage with a focus on the platform's initial lack of data protection APIs and how that influenced the type of and manner in which data was kept on a user's device.

1 Introduction

Windows Phone 7 (WP7) was initially released without data protection APIs (DPAPIs), a system-level support for secure storage of cryptographic keys. The current versions of iOS and Android both have a keystore for this purpose. It is a basic tenet of cryptography not to store an encryption key on the same disk it is encrypting, and many operating systems address this through DPAPIs that protect secrets through keystores. However, without DPAPIs, developers had no system-level option to securely store data on WP7 devices. Instead, developers created other ways for deriving keys or retrieving keys from off the device. These solutions introduced problems of their own. For example, using a user-entered PIN/password or similar adds friction to the user flow, requiring the user to enter a password, PIN, or similar each time the application is launched or returns from tombstoning. Storing the key off the device, such as on a server, requires the device to have a network connection before the application is able to access the protected data, and still requires the user to authenticate themselves in some manner to the server.

This lack of appropriate support for secure data storage by Windows Phone 7 was likely to result in apps that did not properly handle or store sensitive information. Some developers would not put in the effort to research the best solution, and the security conscious developers would have to balance security with user friction.

The idea for this project began in the summer of 2010, before any operating system updates for Windows Phone 7 were released. As is often the case with the tech industry and particularly the mobile sector, the initial plan was quickly invalidated by advances and updates. Now, as the work is wrapped up, Windows Phone 8 is already on the market. Microsoft has announced that Windows Phone 7 devices will be unable to upgrade to Windows Phone 8, thus the results below will still be relevant for some time going forward.

1.1 Windows Phone 7 DPAPIs

Windows Phone 7's Data Protection APIs solves the problem of cryptographic key storage for WP7 developers. The DPAPIs generate and store a cryptographic key securely so a developer does not need to be concerned about key management. The WP7 DPAPIs use the user and device credentials to encrypt and decrypt data. Each app on a WP7 device gets its own cryptographic key. This key is created the first time the app is run and persists across app updates. A developer can use the DPAPIs by calling into the `ProtectedData`[1] class. This class provides the `Protect` and `Unprotect` methods that make use of the DPAPIs. The `Protect` method encrypts the data and the `Unprotect` method decrypts the data; both of these methods implicitly use the app's cryptographic key.

2 The Idea

This look into data storage by popular WP7 apps was inspired by some client work I performed. The client's app handled sensitive personal information, and the client wanted this data to be cached on the device for performance reasons. This was before the release of Windows Phone OS 7.1 (also referred to as Windows Phone 7.5 or Mango) and the data protection APIs it provided. In order to securely store the data on the device, the client had limited and non-optimal choices. These choices were: 1) require the user to enter a passcode that was then massaged into a key used in the encryption and decryption of the data; and 2) store a cryptographic key on a server that the app requests upon launch. The first option increased friction for users, and most passcodes entered on a mobile device lack the complexity for proper security. The second solution did not allow for offline access of the cached data.

The above situation is a perfect example of when DPAPIs should be used, but there was no support for it. If this one client was running into this problem, how many other apps in the WP7 Marketplace faced the same issue and how did they handle it? The goal of this survey was to answer that question. But a new issue arose — there were not many "interesting" and popular apps in the Marketplace. "Interesting" apps included mainstream, popular apps such as Facebook, Twitter, LinkedIn, OpenTable and Netflix, financial apps such as Bank of America and Chase, and password stores such as PasswordCrypt and Keeper. Most of these did not store or cache any data beyond the expected saving of user settings. The lack of data kept on the device by these apps prevents many of them from working without a data connection and may have been a result of lack of support for secure data storage. Even the apps supporting Windows Phone 7.5 did not take advantage of the DPAPIs, and this is may be due to the fact that those APIs were not present during initial development of the app.

3 The Methodology

Isolated Storage is the WP7 private application storage mechanism. It provides application developers a way to store information locally on the device. Auditing what apps store in their isolated storage identifies if there is need for DPAPI. Since Windows Phone 7's application sandbox does not permit one app to access the Isolated Storage of another, the test device needed to be unlocked. There are four different levels of unlocking WP7 devices:

1. **Developer Unlock:** Enable sideloading of homebrew (unsigned) applications. Number of homebrew apps installed limited to 10.
2. **Interop Unlock:** Enable access to restricted resources, such as the registry and file system.
3. **Rooting:** Enable installation of higher privileged apps.
4. **Full Unlock:** Enabled installation of native EXEs.

[1] http://msdn.microsoft.com/en-us/library/windowsphone/develop/system.security.cryptography.protecteddata(v=vs.105).aspx

The device used for testing was a HD7 by HTC. In order to view the files the device needed at least the second level of unlocking, Interop Unlock. To achieve this the Deepshining[2] custom ROM was installed on the device. The Deepshining ROM enables all the unlock levels, allowing the access necessary to inspect arbitrary apps' Isolated Storages. Ability to view and download the files off the device came through TouchXperience, a WP7 app, and Windows Phone Device Manager, a Windows desktop application.[3] Device Manager allowed for most files to be extracted from the device and inspected on a computer using common file inspection tools, including hex/text editors, strings, and grep.

4 The Results

There was surprisingly little on-device storage used by applications. The majority of the apps audited chose not to use Isolated Storage for storage of sensitive information, thus the developers had no need for DPAPIs. This, combined with the lack of interesting apps, brought about less-than-thrilling results.

When this project began in late 2010, there was only one financial app of interest in the Marketplace, Bank of America. Recently, Chase released their Chase Mobile app for WP7. Neither of these major financial apps were storing or caching inappropriate data to disk. Some interesting data in the Chase Mobile application settings file was the presence of a DeviceId value and a binary blob labeled "UsernameSetting", regardless of the "Remember Me" setting being enabled. Attempting to decode the "UsernameSetting" value did not result in any noteworthy revelations and the data did not appear to be sent back to the server.

Similarly, the official apps for Twitter, Facebook, and Netflix stored very little of interest. Facebook, Twitter, and LinkedIn do store an authenticating token to establish authenticated connections without requiring the user to enter their username and password each time the app is launched. For the Facebook app, this token is essentially a long-lived cookie that authenticate the app. For the Twitter and LinkedIn apps, these are the Oauth token and token secret used for making authenticated, signed Oauth 1.0 requests. Ideally these would be stored in a keystore, but they are handled similar to other sensitive cookie values. These two apps also cache personal data; however, it was deemed non-sensitive data that does not demand being encrypted, such as "tweets", posts, Timelines and Friends. While those with private accounts may consider this data sensitive, it is not necessarily sensitive to the point that it needs to be encrypted when stored on a personal device to prevent exposure in the case of a lost or stolen device.

While there was not a significant amount of data stored by the apps inspected, there were at least a few that did store data on the device. Of the apps that stored data on the device, OpenTable and the password utility applications stored the most sensitive information without use of DPAPIs. The OpenTable app stored the user's saved password in cleartext within the application settings file. The password apps are designed to allow a user to enter one password and view and store all the other passwords they use. This way the user only has to remember one (hopefully strong) password, allowing them to avoid password reuse. Being apps related to security and entrusted with protecting highly sensitive data (the keys to all your other data) one would expect the data to be protected appropriately.

PasswordCrypt did not use any form of encryption to protect the credentials stored: the usernames and passwords were found in cleartext form in an XML file stored within the app's Isolated Storage. In addition to the user's usernames and passwords, the "Master" password used to access the app and view the data was contained within the same XML file.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<Vault>
  <Site>
    <Id>1</Id>
    <UserId>Master</UserId>
    <Password>MyPassword</Password>
    <Note />
```

[2] http://forum.xda-developers.com/showthread.php?t=1271118
[3] http://www.touchxperience.com/

```
    <SiteName>Master</SiteName>
  </Site>
  <Site>
    <Id>2</Id>
    <UserId>test123</UserId>
    <Password>findme</Password>
    <SiteName>ISEC</SiteName>
  </Site>
</Vault>
```

Listing 1: Example Vault.xml file from PasswordCrypt

Keeper, another password storage application, also stored a cleartext version of the user's master password; however, the user's stored passwords where not discovered on the device. Without inspecting the web traffic (beyond the scope of storage examination), it is assumed the credentials are stored on a server and accessed during application launch; this is partially confirmed by the app's requirement of a data connection to operate. With the master password, an attacker would be able to access the app and view all the cleartext passwords, regardless of being stored on or off the device.

The other apps inspected did not reveal any significantly sensitive data. The YouTube app appears to just launch a browser pointed at accessing m.youtube.com. myAT&T stored the username (in this test case, a phone number) and, upon request of the user, a password equivalent for the "remember my password" feature. The Locations app had a database of GPS coordinates. While these pieces of data are not information one would want publicly exposed, aside from the password equivalent, they do not necessarily warrant use of DPAPIs.

Results

App	Data Stored	Comments
Bank of America	• Static content such as EULA and messages displayed to the user • Masked version of saved username • Encrypted username / username equivalent	No personally identifiable information was found. Even the saved username was partially masked when stored to disk.
Facebook	• Name • Facebook ID number • Friends list • Cache of News Feed • Access token used for authentication	The access token is passed with every request made by the app. This token authenticates the user's request and appears to be a long-lived session token. Ideally this token would be further protected, such as with the DPAPIs.
Twitter	• Username • Twitter profile (name, "bio", URL to profile image) • Cache of timeline (recent tweets of users followed) • Oauth token • Value assumed to be the Oauth token secret	It is necessary for the app to store the token and secret so it is not unexpected to see these values. However, data used for authentication should be properly protected, such as by using the DPAPIs.

App	Data Stored	Comments
Amazon Kindle	• File named PSK • File named PSV	PSK and PSV files could be related to DRM.
Netflix	• Cookie store • Server response caches (video metadata and URLs) • Encoded string representing username	No data that should have been further protected through DPAPIs.
YouTube	• No data in the app's data folder	Appears to just launch the browser pointed at m.youtube.com.
Mail	• Unencrypted email messages	This was the default email app, most modern mobile OSes do not encrypt email messages beyond the device's disk encryption.
myAT&T	• Username (phone number in test case) • Upon request, stores authentication token	The authentication token should be protected using the DPAPIs.
Locations	• Database of previous visited GPS coordinates	This app is very similar to Google Earth and it was not at all surprising to see a database of cached GPS locations. While personal, local encryption of such data is not expected.
PasswordCrypt	• Cleartext Vault.xml file – master password – usernames and passwords	The master password should be used to seed a cryptographic key used to protect the usernames and passwords. The DPAPIs should be used to protect the master password.
Keeper	• __ApplicationSettings contained plaintext form of master password	No other passwords or usernames were discovered, nor any file that appeared to contain encrypted forms of the data a user enters. This implies that the data is stored off device (on Keeper's servers).
LinkedIn	• Oauth token • Oauth token secret	It is necessary for the app to store the token and secret so it is not unexpected to see these values. However, data used for authentication should be properly protected, such as by using the DPAPIs.
OpenTable	• Cleartext password • An auth token • Phone number • First and last name • Current location	The password should never be stored in cleartext. A revocable auth token would be preferable. Also a great chance to use the DPAPIs. The application settings has an "IsAdmin" setting that is set to '0', but changing it to '1' did not have any noticeable affect on the app.

App	Data Stored	Comments
Chase Mobile	• "UsernameSetting" - A long base64-encoded string that decoded to unidentifiable data • Cache of unauthenticated HTTPS responses for static content such as "Contact Us" • Cache of map titles for nearby ATM locations	The "UsernameSetting" field within __ApplicationSettings was populated regardless of the "save user ID" setting. The actual username was not identifiable within the decoded blob.

5 Conclusion

The goal of this survey was to investigate the practices of major app developers with regard to data storage. The assumption was that due to the lack of sufficient data protection APIs provided by Windows Phone 7, developers would have to make security trade-offs. Instead, it was found that the majority of the big name apps chose not to store any data at all. The largest collection of data stored was static content by Bank of America's mobile app. Some of these apps may take the same approach on other mobile platforms regardless of DPAPI offerings; however, it seems to be an unnecessary performance trade-off to be dependent on network speed and connectivity when suitable data storages are available on the device.

Would developers have taken a different approach and made different storage decisions if Windows Phone 7 had released with DPAPIs? Would more big name companies have developed for WP7? These are hard questions to answer, but maybe Windows Phone 8 and its enhanced security model will provide insight.

Acknowledgments

Special thanks goes to Alex Garbutt for his assistance in the research. Thank you to David Thiel, Alex Vidergar, Chris Hacking, and Loic Simon for their input and critique of this paper. And a final thank you to iSEC Partners for their support of this project.