Ver 2.00.02

Windows Phone 7 Internals and Exploitability

プラットフォームのセキュリティに関する初期報告

Fourteenforty Research Institute, Inc.

株式会社フォティーンフォティ技術研究所

目次

著作権

免責事項

更新履歴

2012-01-26	Ver. 0.1	大居　司

文書情報

発行元: 株式会社フォティーンフォティ技術研究所

連絡先: 株式会社フォティーンフォティ技術研究所

sales@fourteenforty.jp

〒150-0013

東京都渋谷区恵比寿 1-18-18　恵比寿東急ビル 4 階

1. 概要

Windows Phone 7 は、Apple の iOS、Google の Android と競合するスマートフォン向けのオペレーティングシステムである。現状、競合と比べて後発であるなどの問題から普及はあまり進んでいないものの、アプリケーション配信プラットフォームとサンドボックスを備えた先進的なスマートフォン用オペレーティングシステムであり、第 3 の選択肢として普及が期待される。しかし現時点では国内外を問わず、Windows Phone 7 のセキュリティに関する詳細な分析報告は少ない。当文書では、我々が Windows Phone 7 プラットフォームに対して行ったセキュリティ評価に関する初期報告を行う。

2. 解析環境

解析に使用した機種は HTC 7 Mozart、オペレーティングシステムのバージョンは 7.0.7004 である。なおこの文書で示した解析結果は、将来のバージョンの Windows Phone において異なる可能性がある。

3. Windows Phone 7 概要

3.1. アプリケーションプラットフォーム

Windows Phone 7 は、Apple の iOS と同様に専用のアプリケーション流通プラットフォームと連携し、審査を通過したアプリケーションのインストールのみを許可している。一方で Android のパーミッションに似た仕組みであるケーパビリティ (Capability) を持ち、ユーザーが使用する権限を確認できるようになっている。

原則として、Windows Phone 7 端末では Microsoft による電子署名が施された Marketplace のアプリケーションしか実行できない。例外として、Windows Phone 7 には開発中のアプリケーションを実機でテストするための開発者アンロックという仕組みがある。具体的には、特定の端末が認証されると特殊なレジストリ値が置き換えられ、Microsoft の署名がないアプリケーションを動作させること (Side loading と呼ばれる) が可能になる。

3.2. アプリケーションの仕組み

すべての Windows Phone 7 アプリケーションは、実行コードである DLL ファイルやアプリケーション情報を格納する WMAppManifest.xml などを ZIP 形式で圧縮した、XAP という拡張子を持つパッケージとして頒布される。WMAppManifest.xml (マニフェストファイル) は Android における AndroidManifest.xml と同様に、アプリケーションのメタデータや必要な権限、実行するクラス名を格納している。ただし Windows Phone 7 が Android と大きく異なるのは、Android が場合によって複数のエントリポイントを持つことができるのに対し、Windows Phone 7 アプリケーションはマニフェストで宣言された単一のエントリポイントだけを持つことができるということである。

アプリケーションの本体は .NET アセンブリを格納した DLL ファイルである。Windows Phone 7 の実行環境は中間言語である MSIL を含んだアセンブリを厳密に検証し、ネイティブコードの実行やポインタの使用など、実行環境を壊す可能性のあるあらゆる操作を禁止する。またこの実行環境においては、クラスのオブジェクトを動的に生成したり動的に呼び出したりするリフレクションの仕組みも大幅に制限されている。この方法の大きな利点は、コードの正当性をほぼ全自動で解析することができるということである。加えて.NET の機能のひとつであるコードアクセスセキュリティ (CAS) をアプリケーション隔離のために用いることができる。

3.3. セキュリティ

では、Windows Phone 7 プラットフォームのセキュリティについて見てみよう。

前述したように、Windows Phone 7 アプリケーションはケーパビリティの仕組みを持っている。ここでは Android と共通するケーパビリティやパーミッションを対比させてみよう。

可能になる操作	Android	Windows Phone 7
ネットワークへの接続	INTERNET	ID_CAP_NETWORKING
位置情報の取得	ACCESS_*_LOCATION	ID_CAP_LOCATION
電話の発信	CALL_PHONE	ID_CAP_PHONEDIALER

ここでは Android と同等の権限を持つものだけを対比させたが、全体的に Windows Phone 7

の権限は Android と比べて制限されている。例えば他のアプリケーションの情報を取得したり、システムの設定を変更したりするためのケーパビリティは、Windows Phone 7 にはそもそも存在しない。しかしこれは同時に、不用意なリソースへのアクセスを避けることができるということも意味する。Android の場合、アプリケーションの権限によっては秘密情報やシステムの重要な情報を抜き出したりすることが可能である。しかしこのような可能性は、Windows Phone 7 では大きく制限されている。

アプリケーション審査の仕組みが高度に自動化されている (と推測される) ことも特筆すべき点であろう。Windows Phone 7 には、iOS や Android などと同様にインタフェースが明らかにされていない非公開 API が存在する。また、これらの非公開 API を使うための非公開ケーパビリティなども存在する。しかしこれらを使用したアプリケーションを審査のために送信した場合、自動的に技術的問題があることが通知され、Marketplace に並ぶことはおろか人手による審査に進むことすらない。これは、アプリケーションの中間言語やマニフェストが自動解析に向いているという利点を生かした仕組みで、場合によっては承認されていない API を使用するアプリケーションが配信されてしまう iOS の App Store とは対照的といえるだろう。

結論として、Windows Phone 7 のセキュリティシステムは権限の制約とアプリケーション審査の自動化によって高いセキュリティが確保されているといえる。

4. Windows Phone 7 内部分析

Windows Phone 7 のアプリケーションモデルは高いセキュリティを実現するが、内部でそれらが十分に機能しているかどうかとは別の問題である。実際に内部構造を解析しなければ、セキュリティ機構の評価を厳密に行うことはできないと考えられる。

ただしここには問題がある。Windows Phone 7 アプリケーションの実行環境は厳しく制限されており、通常実行できる .NET コードだけではシステム内部を詳細に分析することは不可能である。そのため解析には ARM 向けのネイティブコードを使用した。

もちろん、通常であればアプリケーションでネイティブコードを実行することはできない。しかし、非公開ケーパビリティである ID_CAP_INTEROP と特定の非公開 API を使用することで、少なくとも開発者アンロックされた Windows Phone OS 7.0 端末においては ARM 用の COM DLL が使用可能となる。これを用いて OS 内部を明らかにし、脆弱性攻撃の可能性を調査した。

4.1. 内部環境

Windows Phone OS 7.0 の内部環境は、Windows Embedded CE 6.0 をベースにしているものと推測される。しかしセキュリティ機構に関しては大幅に強化されており、Windows Phone 7 アプリケーションが動作する環境である TaskHost.exe (Android における app_process に相当) はポリシーによって動作が厳格に制限されている。そのため、単にネイティブコードが実行可能なだけではシステム内で権限昇格を行うことは困難と予想される。

また、ネイティブコードの露出も他のプラットフォームと比べると部分的である。iOS の場合はほぼ全てのコードがネイティブであるのに対して、Windows Phone 7 において攻撃可能な箇所にネイティブコードが露出しているのは実行環境である .NET Framework の一部と Internet Explorer コンポーネント程度である。

ネイティブコードが少ないから脆弱性が少ないとは一概に言い切れるものではないが、保護された実行環境は脆弱性の露出を最小限に抑える役割を果たしている。

4.2. ASLR

次に、脆弱性攻撃からの防御を行うメモリ保護機能を見てみよう。

ASLR (Address Space Layout Randomization) は、脆弱性攻撃に利用される実行可能なコード領域の配置をメモリ空間上でランダム化することにより、脆弱性攻撃を阻止するメモリ保護機能である。最近のモバイルプラットフォームについて見てみると、iOS においてはバージョン 4.3 で実装され、Android においてはバージョン 4.0 で部分的に実装された。対して、Windows Phone 7 は最初から ASLR を備えている。

プロセス空間内にロードされる EXE ファイルや DLL ファイルのロードアドレスは、アプリケーションを起動する毎に変化する。繰り返しアプリケーションを起動してメモリマップを取得する実験を繰り返した結果、エントロピーは 8 ビットであると推測した。言い換えると、ほとんどの EXE ファイルや DLL は 256 通りのアドレスからランダムに配置されるということである。

ただし再配置にかかる負荷を軽減するために、頻繁に使用される DLL は 256 通りの中から選ばれる 1 つの基底アドレスから、相対的に配置されているものと推測される。言い換えると、頻繁に使用される 2 つのシステム DLL があった場合、これらのアドレスの差 (距離) は常に一定にな

る。この簡易的な ASLR は iOS 4.3 と酷似しており、セキュリティ上の強度とパフォーマンス上の問題とのバランスを取ったものと推測される。

スタックに関してもランダム化されているため、古典的なスタックオーバーフローを利用する攻撃の多くを成立させることは難しい。

ヒープに関しては懸念が残る。アプリケーションが確保したヒープメモリは常にメモリ空間の低位アドレスから確保され、高いランダム性を持っていない。そのため、ヒープの構造やアドレスを脆弱性攻撃に使われる可能性は十分にある。これは Android と同様の結果で、ヒープのランダム化を行う iOS と比べてセキュリティ上問題がある。

4.3. DEP

DEP (Data Execution Prevention) とは、CPU が持つメモリ制御機能を用いてデータ領域でコードを実行することを禁止する機能である。ベンダーによって呼び名に差があるが、ハードウェア DEP の名で知られる Windows 向けのセキュリティ機構が初めて実装されたのは Windows XP Service Pack 2 であり、現在では主要なすべてのオペレーティングシステムが DEP または類似の保護機能をサポートしている。最近のモバイルプラットフォームについていえば、iOS は初期から適切な DEP を搭載しており、Android も 2.3 以降では適切な DEP が適用される仕組みになっている。では、Windows Phone 7 の場合はどうだろうか。

我々はアプリケーションのプロセス空間のアクセス権をメモリダンプとともにネイティブコードから取得した。その結果、Windows Phone 7 (少なくとも Windows Phone OS 7.0) には脆弱性攻撃を容易とする致命的な欠点があることを発見した。具体的にいえば、アプリケーションが確保するヒープメモリはすべて読み書きおよび実行可能権限でマップされているという点である。これは二重の意味で、脆弱性攻撃を容易にする可能性がある。

この弱点を悪用する方法のひとつが、この部分に存在する JIT コンパイルされたコードを書き換えることである。既に存在するコード領域を書き換えることによって、攻撃者が任意のコードを実行できる可能性がある。もうひとつの方法は、Heap Spray である。Heap Spray とは、ヒープメモリ上にシェルコードやそのアドレスを多数格納することにより、脆弱性攻撃が成功する確率を上げるテクニックである。ヒープ上にシェルコードを多数書き込んでから既存の脆弱性を突くことにより、高い確率で攻撃者の与えたコードを実行させることができると考えられる。特に、Windows Phone 7

においてはアプリケーションが読み込む文字列にシェルコードを埋め込む手段が脅威になる可能性がある。文字列を多数書き込ませることは、脆弱性の存在しないアプリケーションでも比較的容易である。また悪用できるデータ構造はアプリケーションや読み取るデータの種類によって異なるが、文字列は比較的多くのアプリケーションが扱うと考えられる。

4.4. バックドアインタフェース

解析に用いた HTC 製端末だけでなく、LG、Samsung などは自身の Windows Phone 7 端末にベンダー用のバックドアインタフェースを搭載させている。これらはアプリケーションのポリシーが定める範囲を超えてファイルを読み書きする、あるいはレジストリを書き換えることを可能にする。

解析用の HTC 製端末で読み取れたデータを解析した結果、アプリケーションの一覧や設定、“メール”アプリケーションに保存されたメールの一部など、プライバシーに関わるものが多数存在することを確認できた。

現状では、バックドアインタフェースを効率的に利用する、全自動の攻撃は存在しない。しかしながら、将来的にはこれを悪用する攻撃が出現する可能性は否定できない。

4.5. 結論

Windows Phone 7 はセキュアな設計が行われていると思われるものの、主にメモリ保護が十分でないこととベンダー固有のバックドアが存在することに起因する懸念要素が幾つか存在する。

5. 既存の Windows Phone 7 exploitation

本来の意味で、外部からの攻撃により全自動で exploit が成されたことは (iOS や Android と異なり) 一度もない。しかし、Windows Phone 7 をターゲットにした exploit 手法には様々なものがあり、その中には Windows Phone 7 のセキュリティ強度に疑念を抱かせるものも存在する。ここでは WindowBreak という、一部ベンダーの Windows Phone 7 端末においてレジストリの書き換えを非公式に実現する exploit について簡単に解説する。

5.1. WindowBreak の仕組みと懸念

WindowBreak の本体は XML ファイルが格納された ZIP ファイルである。このファイルをブラウザで開き、中にある XML ファイルを開くと、ZIP ファイルビューアー (ZipView.exe) が持つディレクトリトラバーサル脆弱性が悪用され XML ファイルがシステムの使用するディレクトリに展開される。この XML ファイルをベンダー固有のバックドアツールで開くと、本来許可されていないレジストリを書き換えることが可能、というのが exploit の仕組みである。

ここで懸念が生じるのは、Windows Phone 7 にプリインストールされている ZipView.exe というアプリケーションに脆弱性があり、加えてシステムやバックドアツールが使用するディレクトリに書き込みが成功しているという点である。サードパーティーのベンダーがセキュリティを弱めている可能性も否定できないものの、Windows Phone 7 が持つサンドボックスの仕組みが十分でないということも考えられる。これに関しては追加の調査および検証が必要である。

6. 結論

Windows Phone 7 は、整理されたセキュリティ設計とサンドボックス化、高度に自動化されたアプリケーション検証の仕組みを組み合わせることによって、Android、iOS を含めた最近のモバイルプラットフォーム 3 種の中では最も高いセキュリティを実現しているといえる。また実行環境の攻撃可能性は極めて小さく、また環境が Microsoft 自身によって管理されているため、脆弱性攻撃の可能性は極めて小さいと考えられる。

しかしながら、前述したように Windows Phone 7 は強力なメモリ保護の仕組みを部分的にしか実装していない。これにより、もし実行環境やライブラリに脆弱性が見つかった場合には 3 種中最も容易に攻略されてしまうと推測される。また Windows Phone 7 の安全性は実行環境の安全性に強く依存している。そのため、特にベンダーが端末のセキュリティを弱めることによって、Windows Phone 7 端末は深刻な脅威にさらされる可能性がある。

現時点で行った解析は極めて限定的である。そのため、セキュリティ評価を厳密かつ正確にするにはさらなる継続調査が求められる。