

Windows Phone 7 Internals and Exploitability Black Hat USA 2012

Tsukasa Oi | Research Engineer

Fourteenforty Research Institute, Inc. <u>http://www.fourteenforty.jp/</u>

Agenda

- In aspect of:
 - Looking Internals
 - Security Evaluation
 - Exploitation
- Introduction Windows Phone 7.x
- System Implementation
- Reverse Engineering
- Exploitation, Part I and II
- Analysis & Conclusion



Is it good? INTRODUCTION: WINDOWS PHONE 7.X



3



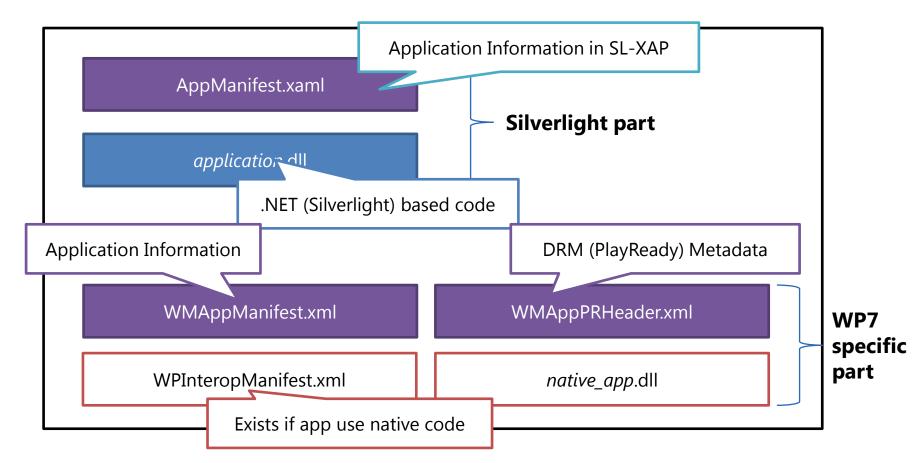
Windows Phone 7.x

- Operating System based on Windows Embedded CE 6.0R3 (core) and Windows Embedded Compact 7 (some features)
- Strong App Sandbox to protect system and user
 - No native code allowed (*in general*)
 - Unlike iOS, executable memory is permitted
 - Capabilities and Chambers



Windows Phone Apps

• Packaged in XAP (actually ZIP) format





Capabilities

- Resources which apps are authorized to use:
 - Devices (ID_CAP_[ISV_CAMERA|MICROPHONE|SENSORS])
 - Identification (ID_CAP_IDENTITY_[DEVICE|USER])
 - Web Browser (ID_CAP_WEBBROWSERCOMPONENT)...



Capabilities

		United States
😹 Windows Phone		search apps and games
Discover Buy Marketplace H	ow-to My Phone	Sign in 🕣
Apps Games		
social		
Facebook		
f	friends all in beautiful Metro Syle de this app but is fully integrated into Mes - New in 2.7: Notifications now show thumbnail display on likes; now respec settings - New in 2.6: Added posting to my made as 'only me'; groups with no pict	w updated time instead of created time; corrected ting except clause on posts with custom privacy wall when posting to groups; custom privacy posts
★★★★ Ratings: 15897 Get free app	resulting and a second se	Configurations Configuration Confi
C Like (2.5k) ★ Tweet (0)	L = 0 = 0 info wall photo: photos info photos info phot	
Required*: Windows Phone 7 or higher	Long the second	
kpp will use location services data services movement and directional sensor push notification service phone identity contacts amera		
Published by: Microsoft Corporation		
keleased: 10/19/2010		
Version: 2.7.0.0		
Download Size: 1 MB		
 Languages (6) 		<u>http://www.wir</u>

App will use:

location services data services movement and directional sensor push notification service phone identity contacts camera

Notifies what resources the app uses (just like Android but no confirmation)

Citation from: ttp://www.windowsphone.com/en-US/apps/82a23635-5bd9-df11-a844-00237de2db9e

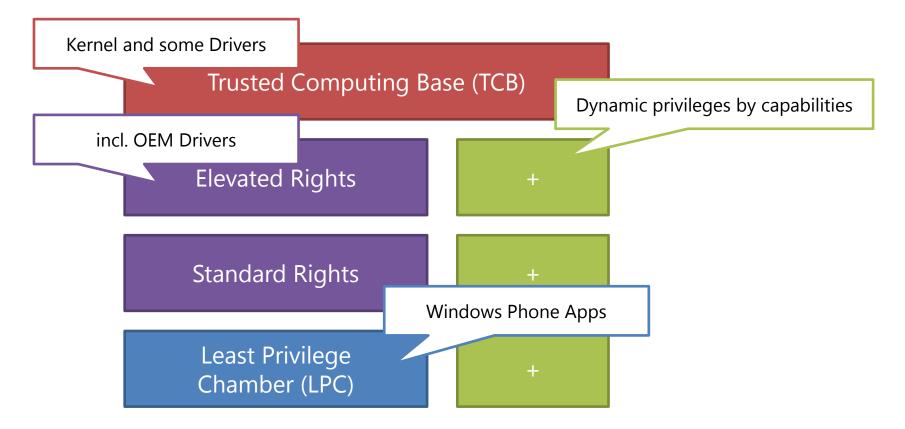


Capabilities

- Resources which apps are authorized to use:
 - Devices (ID_CAP_[/SV_CAMERA|MICROPHONE|SENSORS])
 - Identification (ID_CAP_IDENTITY_[DEVICE|USER])
 - Web Browser (ID_CAP_WEBBROWSERCOMPONENT)...
- Undocumented Capabilities
 - Only selected vendors can use these "privileges"
 - File Type Association (ID_CAP_FILEVIEWER)
 - Native Code and Interop Services (ID_CAP_INTEROPSERVICES)

Chambers

• Isolate app/driver policies by separate chamber





FFRI

Conclusion

- These looks modern, secure and well-designed.
 - We need to know how good they are.



How Windows Phone 7 Operating System works? SYSTEM IMPLEMENTATION



Low Layer Implementation

- OS Mechanisms
 - Process Memory
 - Protected Server Libraries
 - Security System (incl. Sandbox)



How we analyzed the system?

- An unlocked Windows Phone device
 - We mainly used "HTC 7 Mozart"
- Custom Native COM DLL
 - <u>http://forum.xda-developers.com/showthread.php?t=1299134</u>
 - Native COM DLL for Windows Phone 7 can be created using Windows Mobile 6 SDK
- Reverse Engineering (later)



Process Memory Layout



- Similar to Windows Embedded CE 6.0
 - User VM (0x0010000-0x3fffffff)
 - Shared DLLs (0x4000000-0x5fffffff)
 - Contains common DLLs
 - RAM-backed Map Files (0x6000000-0x6ffffff)
 - Contains common files including some .NET assemblies
 - …and specific system areas



Process ASLR (1)



- Stack / Executable / Heap (LocalAlloc) / Virtual Memory (VirtualAlloc)
 - Randomized every launch
 - 64KiB granularity, chosen from 64+MiB low memory range (estimated entropy: about 10-bits)
 - Issue: Only base addresses are randomized (no random "gap" between two allocations – unlike Linux's mmap)
- Some (uncommon) DLLs are loaded in low memory range
 - Also randomized every launch



Process ASLR (2)



- Shared DLLs / Memory Mapped Files
 - Randomized every boot
 - Mapped to RANDOM_BASE_ADDRESS + CONST_OFFSET? (Optimized for Performance; or just no "shared gaps")
 - 64KiB granularity, chosen from 64MiB memory range (estimated entropy: about 10-bits)

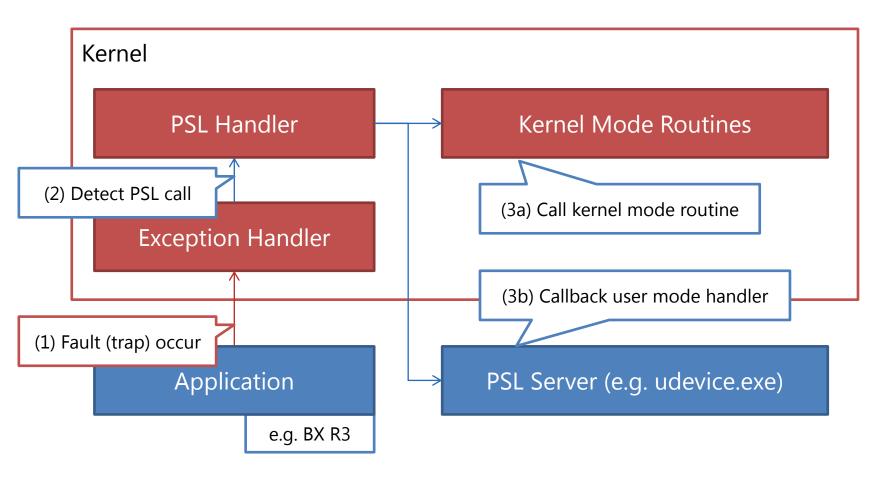


System Calls → Protected Server Library (PSL) calls

- "Branch" specific address (Trap Address) and cause trap
 - Exception handler determines PSL calls (by identifying address which caused the exception)
 - Kernel launches proper PSL routine (kernel mode procedure / user-mode handler)
- Trap Address is Randomized (ASLR for system call)
 - PSL_TRAP_SEED (part of shared memory called UserKData)
 - Randomized every boot
 - 4-bytes granularity, chosen from 1MiB specific memory range (Maximum entropy should be around 18-bits but seems non-uniform)
 - note that location of UserKData is NOT randomized
 - UserKData is located at 0xffffc800



Making PSL: Handling in the system





Making PSL: Disassembly

.text:000DF934

. CONC. 000001 004					
.text:000DF934			hmapViewOfF		
.text:000DF934	xxx_UnmapViewOfFileInPr	ocess	;	÷,	DATA XREF: .text:off_1209C8↓o
.text:000DF934	LDMMIIA	R0!,	{R0,R2,R3,F	R5	,R8,R11,SP-PC}^
.text:000DF938	CODE16				
.text:000DF938	ADDW	R11,	SP, #0×10		
.text:000DF93C	MOVS	R4,	R0		
.text:000DF93E	LDR	R5,	=0×FFFFC854	;	&PSL_TRAP_SEED
.text:000DF940	MOVS	R6,	R1		
.text:000DF942	TST.W	R4,	#1		
.text:000DF946	BNE		DF962		
.text:000DF948	CMP.W		#0×10000		
.text:000DF94C	BLS		DF962		
.text:000DF94E	LDR			÷,	0xF101FFF0 (trap address of UnmapViewOfFileInProcess)
.text:000DF950	MOVS	R2,			
.text:000DF952	LDR		[R5]	÷,	PSL_TRAP_SEED
.text:000DF954	MOVS	R7,	11 · ·		
.text:000DF956	MOVS	R1,			
.text:000DF958	MOVS	R0,			
.text:000DF95A	EORS	R3,	R4	÷,	0xF101FFF0 ^ PSL_TRAP_SEED == trap address
.text:000DF95C	BLX	R3			
.text:000DF95E	MOVS	R4,	R0		



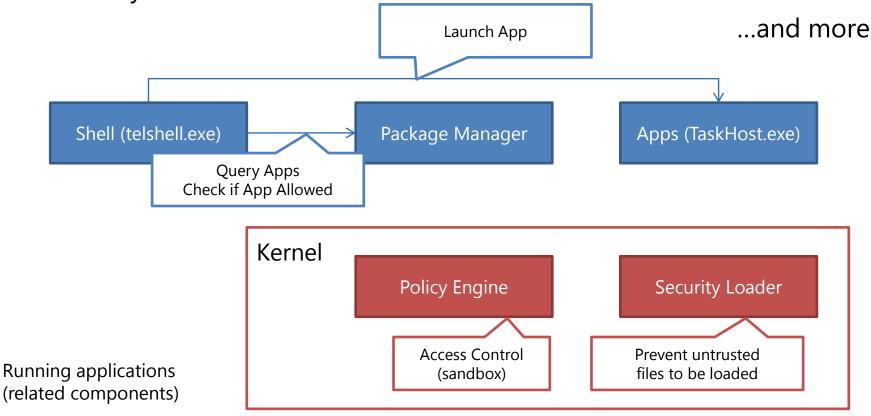
Process ASLR: Evaluation

- Almost all process memory areas are randomized
 - Except shared data regions like UserKData (there's no FS segment register in ARM!)
- However: Only Base addresses are randomized
 - This behavior will make Heap Spray (or similar techniques) easier and more reliable
 - At most about 70MiB?



Security System in WP7

- Policy Engine
- Security Loader





Policy Engine (1) – PolicyEngine.dll

- Actual Access Control
 - Policy Database
 - Policy XML file
- Principle of "Least Privilege"
- ¥Windows¥BasePolicy.xml
 - Policy definition for whole system
 - Generated by merging *(guid)*.policy.xml



Policy Engine (2) – XML format

- <CeSecurityPolicyFile>
 - <Macro> (defines common macro)
 - <Account> (defines account and account group)
 - <MemberOfGroup>
 - <Rule> (policy definition)
 - <Authorize> / <Stop> (permit or reject operations)
 - <Match>
- Achieves Program Isolation



Security Loader – LVMOD

- Following modules can run on default configuration:
 - Modules inside ROM (Stored PE [XIP] files may have invalid signature but that's fine)
 - Modules Authenticode-signed by Microsoft (including Windows Phone Marketplace files)
- Developer Unlock
 - Authorizes limited number of apps (modules) having no Authenticode signature
 - Some checks are done by Package Manager
 - lvmod.dll checks whether "DeveloperUnlockState" registry key is 1 and (conditionally) allows unsigned modules if developer-unlocked



Don't fall into the ditch!

REVERSE ENGINEERING



Reverse Engineering Windows Phone 7 OS

- It's not a black box.
 - Reverse Engineering is possible even if you don't actually own the device
- There are some obstacles but can be cleared
 - We created "WP7 Helper Tools" to make analysis easier.
- It's not difficult.
 - ... if you understand the tools



Retrieving Files

- Types:
 - OS image for recovery
 - Web-based Updater (which Zune downloads)
- These files should be fixed before using IDA Pro



Fixing Files (1) – OS image

- Windows CE ROM can contain PE files in XIP (eXecute-In-Place) format
 - Headers are converted to its original one (E32/O32 header)
 - struct e32_rom / o32_rom
 - Image contents are aligned but the original file seems to be an unaligned PE file⁽¹⁾
- WP7 Helper Tools :: unpack-xip.py
 - Accept dump generated by ImgFsToDump/xidump (<u>http://forum.xda-developers.com/showthread.php?t=572673</u>)

(1) Found while observing IMAGE_DEBUG_DIRECTORY.



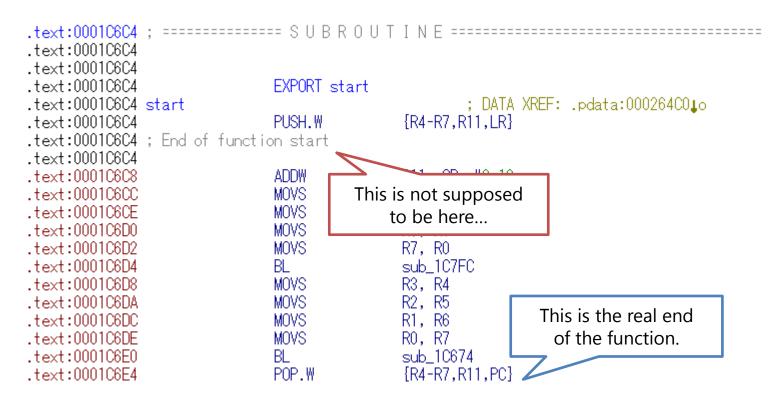
Fixing Files (2) – Web-based Updater

- Zune retrieves updater CAB files from Microsoft website
 - It contains PE files!
- Unaligned (and a bit corrupted) PE file
 - Section is not aligned (!= OptionalHeader.FileAlignment)
 - Some section sizes are corrupted
- WP7 Helper Tools :: unpack-updater.py
 - Accept ordinal (but unaligned) PE file
 - Realign the file to make IDA Pro analyze the file correctly



IDA Pro bug (1) – Auto Analysis Failure

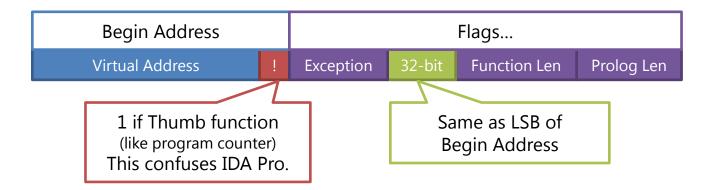
- Found IDA Pro (6.1-6.2) analysis bug for WP7 modules:
 - What?!





IDA Pro bug (2) – Exception Table (PE/ARMI)

- IDA Pro did not correctly handle LSB of Begin Address
 - Conflict with Auto Analysis and result in early function ends
- WP7 Helper Tools has an option to avoid this issue (-p)
 - Reported this issue to Hex-Rays
 - Fixed in IDA Pro version 6.3





IDA Pro bug (2) – Exception Table (PE/ARMI)

- IDA Pro did not correctly handle LSB of Begin Address
 - Conflict with Auto Analysis and result in early function ends
- WP7 Helper Tools has an option to avoid this issue (-p)
 - Reported this issue to Hex-Rays
 - Fixed in IDA Pro version 6.3

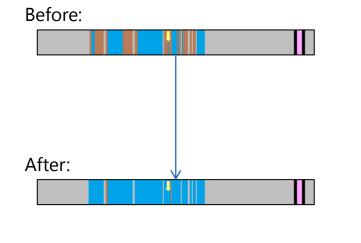
Bugfixes

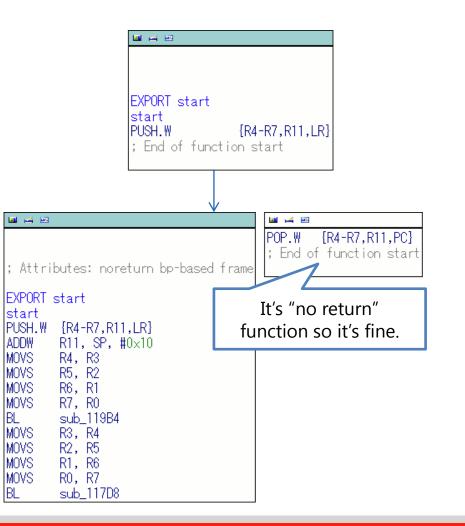
BUGFIX: 'produce exe' command was inviting the user to overwrite the current idb file BUGFIX: .pdata section of PE files for ARMI architecture was not parsed correctly BUGFIX: added a workaround for integer overlow in 'operator new []' if compiled with GCC BUGFIX: AF2_STKARG option was ignored by the analysis engine BUGFIX: an attempt to create a huge segment that can not be created could corrupt the database in some cases BUGFIX: ARM: more correct frame setup in Thumb mode (local variables were lumped together with saved registers) BUGFIX: automatic database snapshots were not working if no snapshots existed before

Citation from http://www.hex-rays.com/products/ida/6.3/index.shtml



IDA Pro bug – Effects by Fixing

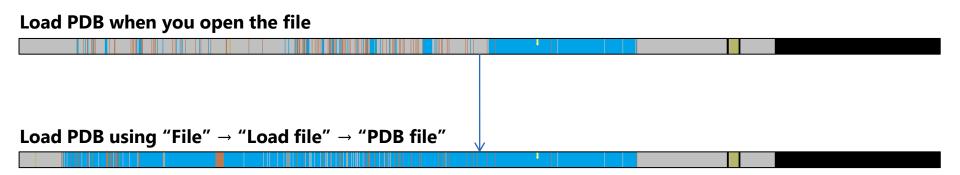






IDA Pro tip – Symbols (1)

- Most of debug symbols can be retrieved!
 - WP7 Helper Tools fix debug directory (with -d option)
 - http://msdl.microsoft.com/download/symbols
- Load PDB file after you load and analyze the program
 - Loading PDB files first will result in auto-analysis failure





IDA Pro tip – Symbols (2)

- Loading PDB files may corrupt analysis
- To solve this:
 - 1. Analyze program
 - 2. Acquire the list of "Thumb" functions using IDAPython
 - 3. Disable Auto Analysis
 - 4. Load PDB file
 - Make "Thumb" functions "Thumb" functions (fix sideeffects by loading PDB file, using IDAPython)
 - 6. Enable Auto Analysis and Reanalyze Program



Demo (WP7 Helper Tools + IDA Pro)



Appendix: Reverse Engineering for WP7 package

- Downloading Applications
 - Marketplace Browser and Downloader for Windows Phone 7 (<u>http://mktwp7.codeplex.com/</u>)
 - We can also use "Unlocked" device to retrieve installed application images (apps are installed on "¥Applications¥Install¥{product-id}")
- Reverse Engineering
 - IDA Pro (.NET or native)
 - .NET Reflector
 (<u>http://www.reflector.net/</u>)



Native code seems vulnerable... but really? EXPLOITATION, PART I



How about Exploitability?

- Memory Protection (kernel / native components)
 - DEP: good
 - ASLR: not bad (except no random "gap")
 - Executable Memory: not good as iOS (executable memory may be unsigned)
- Memory Protection (.NET)
 - DEP: ?
 - Executable Memory: ?
- Native Code / Native Modules
 - How these are used?



Usage of Memory in .NET

- Strings:
 - Dynamic strings are allocated in the VM (access == RWX)
 - Low randomization on heap/VM
 - Strings may be **frequently used** in the program
- Possibility of "String" Spray
 - Spraying executable code as UTF-16LE strings (each string must be unique)
 - Low memory footprint because of low randomization *(estimated: 70MiB at most)*
- Conclusion: Attacking native components
 using .NET might be possible

FFRI

Native Code in Apps

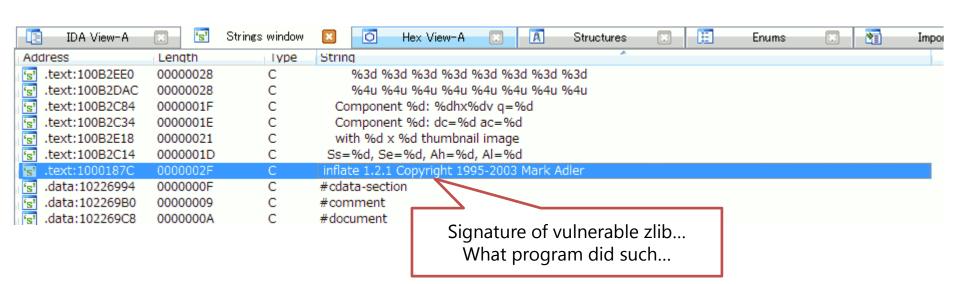
- Very few Apps have Native code
- Most of "native" apps are OEM or career's one but non-OEM apps include:
 - Adobe Reader
 - Tango Video Calls
- Can native code be vulnerable?



Finding Vulnerabilities (in general)

- Static Analysis
 - Using IDA Pro
- Dynamic Analysis
 - Trace and instrumentation with JTAG or something...
 - Fuzzing
- I have planned to do this before I find...

What?!









(not publicly disclosed)





Vulnerability in (not publicly disclosed)

- CVE-2005-2096
 - Buffer overflow vulnerability in zlib before 1.2.3
 - (not disclosed yet) uses zlib 1.2.1
- Heap Overflow
 - Destroys inflate_state struct allocated by zalloc
 - Overwritten address range can be controlled but overwrite pattern cannot (always "invalid" signature)
 - (redacted)'s zalloc function just calls COREDLL's malloc
 - COREDLL's malloc uses LocalAlloc to allocate memory



Privilege escalation with third-party components EXPLOITATION, PART II

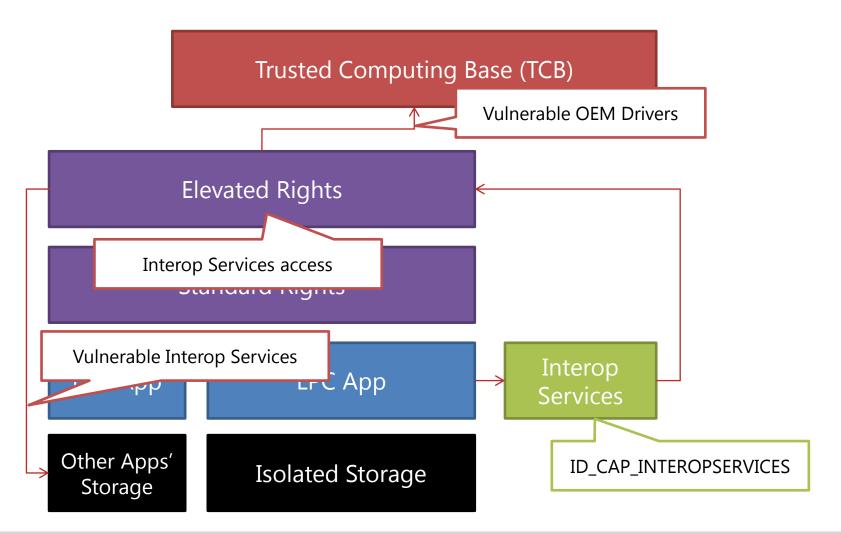


Policy Flaw? – ID_CAP_INTEROPSERVICES

- Undocumented Capability ID_CAP_INTEROPSERVICES allows "third-party service" access
 - Some third-party (and **non-OEM**) apps declare this capability!
 - e.g. Tango Video Calls
 - Why such application *have to* do so?
- ID_CAP_INTEROPSERVICES allows OEM driver access
 - I have no good feelings about OEM drivers...
 - Many "rooting" vulnerabilities in OEM drivers are found in various Android devices...
 - Some OEM drivers allow "backdoor" access



Policy Flaw? – Breaking Chambers





Looking at Demo Device – HTCRegUtility.dll

- Allows registry access
 - Ability to change system settings without permission
- Latest version: fixed
 - By restricting registry access



Looking at Demo Device – HTCFileUtility.dll

- Allows file system access bypassing sandbox/chamber
 - Ability to steal user/system information
- Latest version: Directory traversal vulnerability
 - HTCFileUtility.dll checks whether supplied path has specific (hardcoded) prefixes but there's no other verification ⁽²⁾



Looking at Demo Device – HTCUtility.dll

- Allows RAW RAM read/writes using DeviceIoControl
 - Ability to gain kernel mode privileges
 - Detailed in great work by Alex Plaskett
- Latest version: fixed
 - By removing related features



Demo (What will happen then?)



- Windows Phone is designed secure
 - But some OEM drivers seem to be unconcerned about security
 - This might be a big difference between Android
 - Bad designs, vulnerabilities
- Privilege (Capability) separation is important
 - Microsoft should have been separated OEM capabilities correctly...



Anyway, is Windows Phone 7 secure? ANALYSIS & CONCLUSION



Analysis – Summary

- Sandbox / Application System
 - Designed secure, conforming "Principle of Least Privilege"
- Exploitation (Native Code)
 - Designed well but some concerns here (regarding insufficient ASLR and .NET memory usage)
- Exploitation (OEM components)
 - If the app with ID_CAP_INTEROPSERVICES capability is vulnerable, it may result in sandbox bypass.

Conclusions

- Windows Phone 7.x OS' sandbox is very strong
 - Conforming "Principle of Least Privilege"
- Interop Services and Native Code could be a design failure
 - OEM Code and vulnerable native apps might threaten users (depends on devices)
 - It {have to | will} be fixed in Windows Phone 8





Windows Phone 8

- Shared Windows Core
 - NT kernel is coming.
 - Expecting strong memory protection
- Native Code access and a new framework
 - Minimize ID_CAP_INTEROPSERVICES attack surface
 - May minimize applications which require such privilege



References

- Windows Embedded Compact 7 public headers and libraries (Evaluation: <u>http://www.microsoft.com/en-us/download/details.aspx?id=19004</u>)
- Blue Hat v11 Technical Windows Pwn 7 OEM – Owned Every Mobile? (<u>http://labs.mwrinfosecurity.com/assets/128/mwri_wp7-bluehat-technical_2011-11-08.pdf</u>)
- "48 Windows Phone apps using native code Nanapho" (Japanese) (<u>http://nanapho.jp/archives/2012/02/only-48-windows-phone-apps-using-native-code/</u>)
- And more!



Thank you!

Updated and Detailed slides will be available at our website!

Tsukasa Oi | Research Engineer <<u>oi@fourteenforty.jp</u>>

Fourteenforty Research Institute, Inc. http://www.fourteenforty.jp/