Windows 8 Security and ARM

Alex Ionescu
Chief Architect

BreakPoint 2012

@aionescu
alex@crowdstrike.com

Bio

- Reverse engineered Windows kernel since 1999
  - Previously lead kernel developer for ReactOS Project
- Interned at Apple for a few years (Core Platform Team)
- Co-author of Windows Internals 5th and 6th Edition
  - Also instructor and contributor to Windows Internals seminar for David Solomon Expert seminars
- Founded Winsider Seminars & Solutions Inc., to provide services and Windows Internals training for enterprise/government
- Now Chief Architect at CrowdStrike

Outline

- Mitigations & Hardening
  - Process Mitigation Policies
  - Kernel Hardening
  - User Hardening
  - General Hardening
- New Security Features
  - AppContainer / LowBox
  - Signing Levels
  - Early-Launch Anti Malware
  - Dynamic Access Control
- Windows on ARM (WoA)
- The Future & Conclusion

References and Recommended Reading

1. Matt Miller, Ken Johnson "*Exploit Mitigation Improvements in Windows 8*"
   - http://media.blackhat.com/bh-us-12/Briefings/M_Miller/BH_US_12_Miller_Exploit_Mitigation_Slides.pdf
2. Chris Valasek, Tarjei Mandt "*Windows 8 Heap Internals*"
   - https://media.blackhat.com/bh-us-12/Briefings/Valasek/BH_US_12_Valasek_Windows_8_Heap_Internals_Slides.pdf
3. Alex Ionescu "*New Security Assertions in Windows 8*"
   - http://www.alex-ionescu.com/?p=69
4. Tarjei Mandt "*Smashing the Atom*"
   - Tarjei http://mista.nu/research/smashing_the_atom.pdf
5. IEBlog "*Enhanced Memory Protections in IE10*"
   - http://blogs.msdn.com/b/ie/archive/2012/03/12/enhanced-memory-protections-in-ie10.aspx

Mitigations & Hardening

Process Mitigation Policies

- Available through documented API
  - Can be set with **PROC_THREAD_ATTRIBUTE_MITIGATION_POLICY** at creation time when using *CreateProcess*
  - Can be set at runtime with *SetProcessMitigationPolicy* (less effective)
- Also available through system-wide administration policy
  - K: CurrentControlSet\Control\Session Manager\Kernel V: “MitigationOptions”
  - Set in *PspSystemMitigationOptions* and inherited
- Some are available through opt-in
  - **/HIGHENTROPYVA**
- Usually inherited from parent
- Can be hardened through “hardening level”
  - Turns on certain opt-in mitigations for AppContainers & others

Legacy Mitigations

- Execution (Windows Vista+)
  - DEP
  - ATL Thunk Emulation
  - SEHOP
- Memory (Windows Vista+)
  - Stack Randomization (not available through API until Windows 8)
  - Heap Termination (not available through API until Windows 8)

New Mitigations

- Forced ASLR
  - Fakes an image base collision and performs relocation based on relocation records
  - Will load at fixed address if no relocation data present
- Forced ASLR w/ Required Relocations (Disallow Stripped Images)
  - Same as above, but will not load the image if there are no relocation records
- Bottom-up ASLR
  - Superset of older heap & stack randomization code
  - Now randomizes all bottom-up allocations (*VirtualAlloc*, *MapViewOfFile*…)
- High Entropy ASLR (HEASLR)
  - Enables 1TB variance if Bottom-up ASLR is enabled (64-bit only)
  - Image HEASLR is based on /HIGHENTROPYVA (available through API too)

New Mitigations (cont)

- Top-down ASLR
  - Randomizes locations of MEM_TOP_DOWN (PEB, TEB, NLS Data, etc…)
  - Enabled if /DYNAMICBASE is used, combines with HEASLR automatically
- Strict Handle Check
  - Causes exception if an invalid handle is used, instead of a failure code
  - Similar to having made the handle protected through API, but applies for all handles
- Win32k System Call Disable
  - Does not allow calling system calls in Table 1 (0x1000 and higher)
    - *PsConvertToGuiThread* returns STATUS_ACCESS_DENIED
- Extension Point Disable
  - No longer loads AppInit_DLL's
  - Disables *SetWindowHookEx* and accessibility/journal hooks

New Mitigations (cont)

- NULL Allocation Disable
  - Always enabled in Windows 8 unless image is a 16-bit VDM
    - x86 only
- Entropy improvements
  - ASLR entropies, even without HEASLR, increased
    - See [1] for full table of entropy changes
  - Stack cookie (GS) entropy increased by using RDTSC
- Registry Type Checking
  - Hardening to *RtlQueryRegistryValue* allows drivers to check for correct registry type
  - Additionally protects against reading values from “untrusted hives” (user-loaded)
  - Backported with a hotfix

Kernel Hardening

- Kernel ASLR has increased entropy, including boot stack and driver (including kernel, HAL, etc…) image load addresses
  - KSEG0 mapping no longer enforced on x64, configurable on x86, results in high entropy boot allocations and structure locations
- Pool Hardening
  - Numerous changes to harden make pool exploitation harder [2]
- NX Pool and MDL Mappings (*MmMapLockedPagesSpecifyCache*)
  - Forced on ARM
  - Opt-in/out-out model on other platforms with *ExInitializeDriverRuntime*
- SMEP (Intel "OS Guard") / PXN (ARM)
  - Available on Cortex-A processors and Ivy Bridge
  - On ARM, per-page bit. On Intel, CR4 setting uses "Owner" (user) bit

More Kernel Hardening

- PatchGuard Improvements
  - Better obfuscation and encryption leading to more complex static analysis
  - Protection of OBJECT_TYPE_INITIALIZER information in OBJECT_TYPE
    - Used commonly by “DKOM”-based rookits
    - Protection of Win32k.sys system calls and code
- ACPITABL.DAT is only loaded in debugging mode
  - Breaks attack surface I described at SyScan 2012 [3]
- Win32k.sys System Call information leak is plugged
  - Certain APIs would return USHORT and thus leak part of the EAX register
- HAL Heap is no longer executable
- Win32k.sys symbols are gone again ☺
- ABIOS Support removed

User Mode Hardening

- Heap Hardening
  - Numerous changes, similar to pool, to make exploitation harder [1] [2]
- Loader Improvements
  - Many local attacks/shellcode use PEB (i.e.: fs:30h or gs:48h) to access PEB_LDR_DATA, which has the heads of the loader database's linked lists
    - Allows getting base address of NTDLL with a few instructions
    - Never made sense why Peb->Ldr exists in the first place (vs. using *ntdll!PebLdrData*)
  - Windows 8 has a new loader that uses Red-Black Trees instead
    - Faster for the OS, more complex to parse for the bad guys
    - Root node is not stored in PEB, but internal symbol of NTDLL
    - Legacy behavior maintained when LDR_DATA_TABLE_ENTRY.InLegacyLists is set
- Licensing System Improvements
  - VM Detection, Unique ID Generation, Warbird-protection of licensing code

More User Mode Hardening

- Shared User Data Information Leaks Plugged
    - No more pointers
        - System calls are now exclusively done through SYSENTER on x86
    - No more ASLR-bypass offset
- System Call Table “Randomization”
    - System calls are now sorted from Z->A instead of A->Z
- Dynamic Subsystem Heaps
    - CSRSS Heap no longer static (stored in PEB)
    - GDI and Desktop/Session Heaps no longer static (stored in Win32k structures)

General Hardening

- Fail Fast Assertions (__failfast, int 29h)
  - Safe List Functions [1, 3]
  - Virtual Table Validation [1, 5]
  - Range Check Failure [1, 5]
  - Untrusted Hive Access
  - Legacy: Stack Cookie Failures, Invalid Argument Failure, Fatal Application Exit
- RDRAND-based Cryptography and TPM Boot Entropy Seeding
  - Better random number generation in all parts of the system
- DWM Isolation
  - Desktop Window Manager no longer runs with elevated privileges
  - Each DWM has its own virtual session account
- VDM First Use Warning
  - Launching a VDM application requires Admin user confirmation

Overall Experience Improvements

- On Windows RT, all applications must be from the Windows Store, with all the improvements that the vetting process + the technical security improvements (TBD) will bring
- All applications leveraging new application model will receive benefits from the system
  - Especially given heavy use of Broker Infrastructure, which reduces attack exposure
- User-mode Driver Framework in Windows 8 enables true hardware drivers
  - I/O Mapping Access and Register Access
  - Interrupt Handling in User-Mode
  - Greater share of drivers can run in secured Ring 3 environment

Hardening & Mitigation Structure Artifacts

- Most process mitigation policies are in EPROCESS’ Flags2 bitfield
  - StackRandomizationDisabled
  - DisallowWin32kSystemCalls
  - DisallowStrippedImages
  - HighEntropyASLREnabled
  - ExtensionPointDisable
  - ForceRelocateImages
- Execution policies are in KPROCESS’ Flags field
  - ExecuteDisable
  - DisableThunkEmulation
  - ExecuteDispatchEnable
  - ImageDispatchEnable
  - DisableExceptionChainValidation
- Handle exceptions are in HANDLE_TABLE’ EnableHandleExceptions

New Security Features

App Containers

- Biggest overall change to Windows app model in Windows 8
- Multiple isolation layers (we’ll talk about each one)
  - Base Named Objects
  - Registry and File System
  - Atom Table
  - Handles
  - Windowing State
  - Network Policy / Firewall
- Also leverages Job Objects for additional limits
- Automatically used for all AppX packages
  - Link with /APPCONTAINER to have system create one
  - *NtCreateLowBoxToken* performs the token-level work

Package SID

- An AppContainer is identified by a unique name:
  - *“*To ensure uniqueness, it is recommended that this string contains the app name as well as the publisher. This string can be up to 64 characters in length. Further, it must fit into the pattern described by the regular expression "[-_. A-Za-z0-9]+".”
- The name is then converted into a Package SID
  - *AppContainerDriveSidFromMoniker*, called by *DeriveAppContainerSidFromAppContainerName*
  - The Package SID is nothing more than the S-15-2 Root SID with up to 8 additional RIDs
  - The RIDs are the SHA256 hash digest of the name
- ACLs can now be set based on the Package SID
  - “ALL APPLICATION PACKAGES” is SECURITY_BUILTIN_PACKAGE_ANY_PACKAGE (RID 1)

Changes to the TOKEN

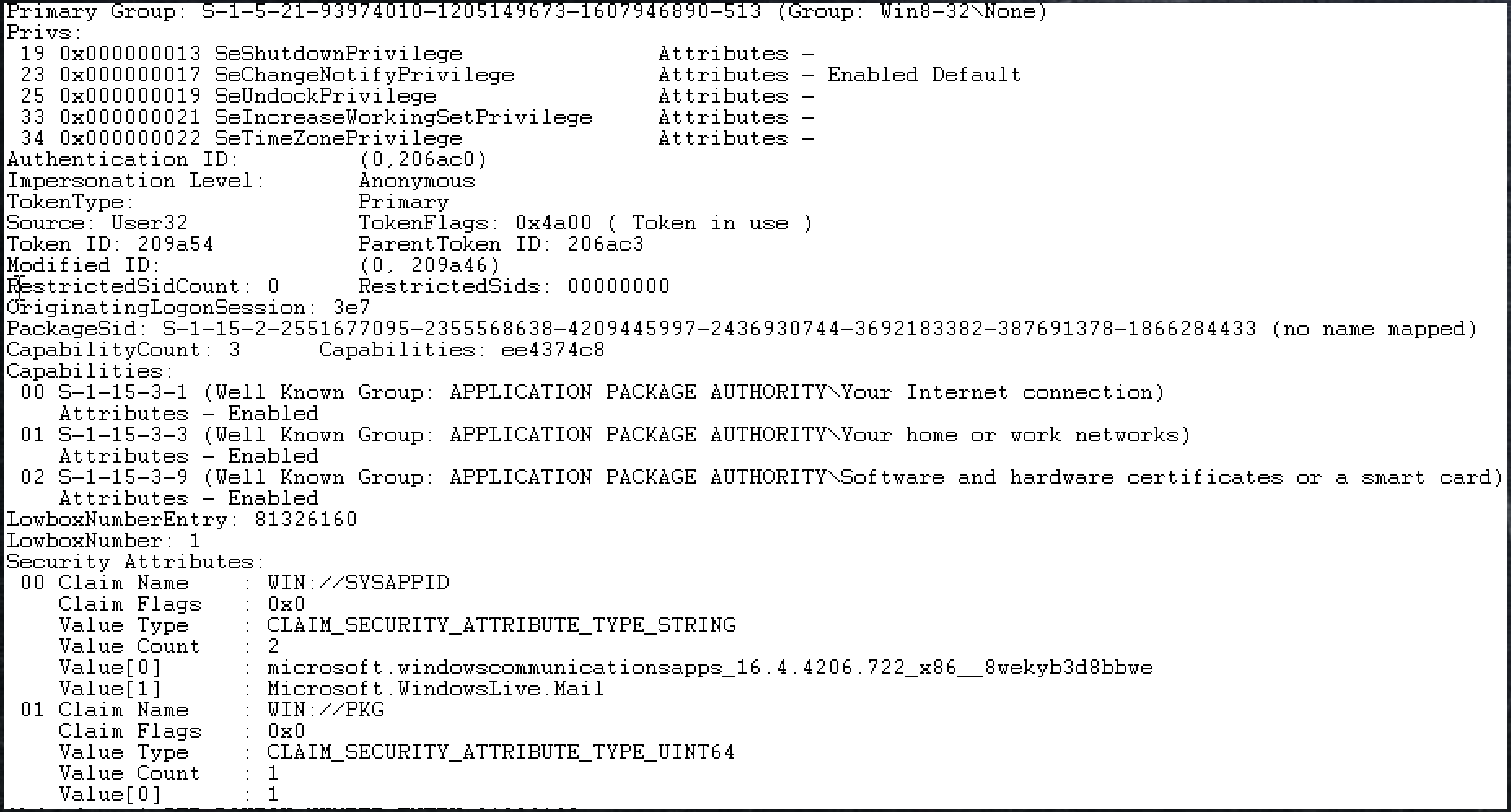
- The TOKEN is the one and only secure, kernel-managed, security state that a caller (process or thread) has
  - Stores the Privileges, User and Groups SIDs, Impersonation Level/Type
  - Windows 2000: Adds Restricted SIDs, Session ID
  - Vista: Adds the Integrity Level, Mandatory Policy
  - Windows 7: Adds Attributes (used only by AppLocker), now called Claims
  - Windows 8: Adds Capabilities, Package, Lowbox Number and Handle Entry
- TOKEN_LOWBOX (0x4000) is present in Token Flags
- New information classes (some examples):
  - TokenIsAppContainer (checks flags)
  - TokenAppContainerSid (returns Package SID)
  - TokenAppContainerNumber (returns LowBox number, see next slide)

Capability SIDs

- Describe the level of access (through brokers) that an app has
  - Start with S-1-15(AppPackage Authority)-3(Capability RID)
- Defined for Windows 8:
    - SECURITY_CAPABILITY_INTERNET_CLIENT
    - SECURITY_CAPABILITY_INTERNET_CLIENT_SERVER
    - SECURITY_CAPABILITY_PRIVATE_NETWORK_CLIENT_SERVER
    - SECURITY_CAPABILITY_PICTURES_LIBRARY
    - SECURITY_CAPABILITY_VIDEOS_LIBRARY
    - SECURITY_CAPABILITY_MUSIC_LIBRARY
    - SECURITY_CAPABILITY_DOCUMENTS_LIBRARY
    - SECURITY_CAPABILITY_ENTERPRISE_AUTHENTICATION
    - SECURITY_CAPABILITY_SHARED_USER_CERTIFICATES
    - SECURITY_CAPABILITY_REMOVABLE_STORAGE
    - SECURITY_CAPABILITY_INTERNET_EXPLORER
- Associated with “Intents” described by Metro App package

!token in Windows 8

- Note Capabilities, Lowbox, and Security Attribute data

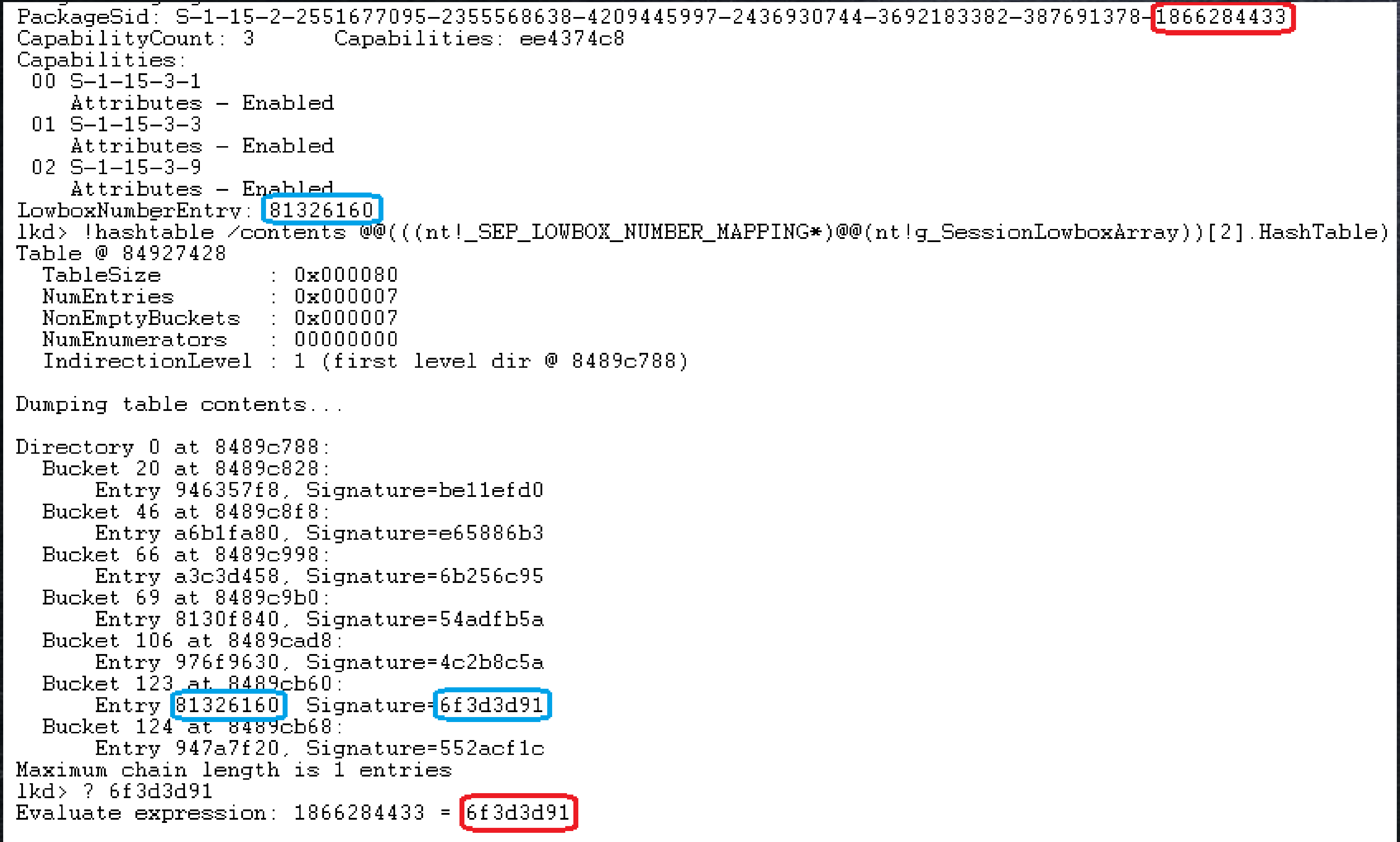


```
Primary Group: S-1-5-21-93974010-1205149673-1607946890-513 (Group: Win8-32\None)
Privs:
 19 0x000000013 SeShutdownPrivilege               Attributes -
 23 0x000000017 SeChangeNotifyPrivilege           Attributes - Enabled Default
 25 0x000000019 SeUndockPrivilege                 Attributes -
 33 0x000000021 SeIncreaseWorkingSetPrivilege     Attributes -
 34 0x000000022 SeTimeZonePrivilege               Attributes -
Authentication ID:          (0,206ac0)
Impersonation Level:        Anonymous
TokenType:                  Primary
Source: User32              TokenFlags: 0x4a00 ( Token in use )
Token ID: 209a54            ParentToken ID: 206ac3
Modified ID:                (0, 209a46)
RestrictedSidCount: 0       RestrictedSids: 00000000
OriginatingLogonSession: 3e7
PackageSid: S-1-15-2-2551677095-2355568638-4209445997-2436930744-3692183382-387691378-1866284433 (no name mapped)
CapabilityCount: 3      Capabilities: ee4374c8
Capabilities:
 00 S-1-15-3-1 (Well Known Group: APPLICATION PACKAGE AUTHORITY\Your Internet connection)
    Attributes - Enabled
 01 S-1-15-3-3 (Well Known Group: APPLICATION PACKAGE AUTHORITY\Your home or work networks)
    Attributes - Enabled
 02 S-1-15-3-9 (Well Known Group: APPLICATION PACKAGE AUTHORITY\Software and hardware certificates or a smart card)
    Attributes - Enabled
LowboxNumberEntry: 81326160
LowboxNumber: 1
Security Attributes:
 00 Claim Name    : WIN://SYSAPPID
    Claim Flags   : 0x0
    Value Type    : CLAIM_SECURITY_ATTRIBUTE_TYPE_STRING
    Value Count   : 2
    Value[0]      : microsoft.windowscommunicationsapps_16.4.4206.722_x86__8wekyb3d8bbwe
    Value[1]      : Microsoft.WindowsLive.Mail
 01 Claim Name    : WIN://PKG
    Claim Flags   : 0x0
    Value Type    : CLAIM_SECURITY_ATTRIBUTE_TYPE_UINT64
    Value Count   : 1
    Value[0]      : 1
```

Lowbox Number Entry

- Assigns a unique per-session “Lowbox Number” to each token associated with an AppContainer
- For the Session ID 0 to 5, stored in *g_SessionLowBoxArray* as array of SEP_LOWBOX_NUMBER_MAPPING structures
  - For higher sessions, *g_SessionLowBoxMap* is allocated and becomes a dynamically growing array (linked through LIST_ENTRY) of mapping structures
- To find the entry for a number, get the correct mapping for the Session ID
- Compute the “Package SID Signature”, defined as the last RID
- Enumerate the hash table looking for a bucket which matches the signature
  - This is the SEP_LOWBOX_NUMBER_ENTRY

Lowbox Number Entry Lookup



```
PackageSid: S-1-15-2-2551677095-2355568638-4209445997-2436930744-3692183382-387691378-1866284433
CapabilityCount: 3      Capabilities: ee4374c8
Capabilities:
 00 S-1-15-3-1
     Attributes - Enabled
 01 S-1-15-3-3
     Attributes - Enabled
 02 S-1-15-3-9
     Attributes - Enabled
LowboxNumberEntry: 81326160
lkd> !hashtable /contents @@(((nt!_SEP_LOWBOX_NUMBER_MAPPING*)@@(nt!g_SessionLowboxArray))[2].HashTable)
Table @ 84927428
   TableSize          : 0x000080
   NumEntries         : 0x000007
   NonEmptyBuckets    : 0x000007
   NumEnumerators     : 00000000
   IndirectionLevel   : 1 (first level dir @ 8489c788)

Dumping table contents...

Directory 0 at 8489c788:
   Bucket 20 at 8489c828:
         Entry 946357f8, Signature=be11efd0
   Bucket 46 at 8489c8f8:
         Entry a6b1fa80, Signature=e65886b3
   Bucket 66 at 8489c998:
         Entry a3c3d458, Signature=6b256c95
   Bucket 69 at 8489c9b0:
         Entry 8130f840, Signature=54adfb5a
   Bucket 106 at 8489cad8:
         Entry 976f9630, Signature=4c2b8c5a
   Bucket 123 at 8489cb60:
         Entry 81326160, Signature=6f3d3d91
   Bucket 124 at 8489cb68:
         Entry 947a7f20, Signature=552acf1c
Maximum chain length is 1 entries
lkd> ? 6f3d3d91
Evaluate expression: 1866284433 = 6f3d3d91
```

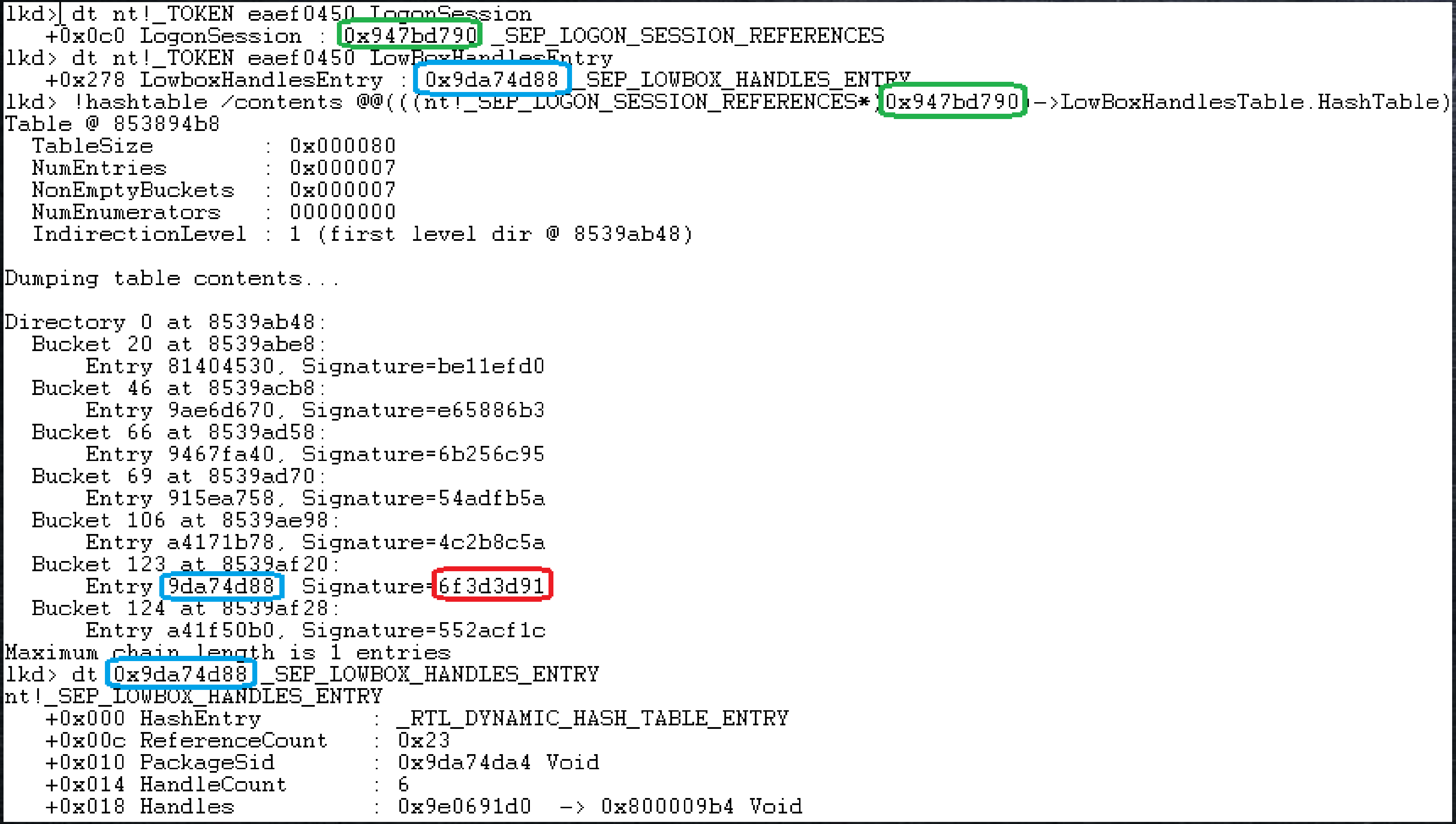
Lowbox Number Entry

- Binds a Package SID with a LowBox Number
- Used by Window Manager to determine if certain operations allowed
  - Some window messages/hooks only allowed within the same lowbox
- Contains pointer to LowBox Atom Table
  - Stores Atoms in a per-AppContainer location, only accessible by applications sharing the same lowbox
  - Also prevents global atom deletion by applications running in a lowbox
- Every atom now has a reference to a LowBox Number
- Mitigates against Tarjei’s Smashing Atom Attack
  - See [4] for more details

Lowbox Handle Entry

- When a LowBox Token is created from kernel32, a list of handles is passed in
- Token's LowBoxHandlesEntry contains a pointer
- Logon session contains table of all entries
- To do manual lookup:
  - Compute the "Package SID Signature", defined as the last RID
  - Get the logon session for the token, and dumb SEP_LOWBOX_HANDLE_TABLE
  - Enumerate the hash table looking for a bucket which matches the signature
  - This is the SEP_LOWBOX_HANDLE_ENTRY
- The list of handles is duplicated into the kernel
  - When the token is duplicated or filtered, references are tracked

Lowbox Handle Entry Lookup



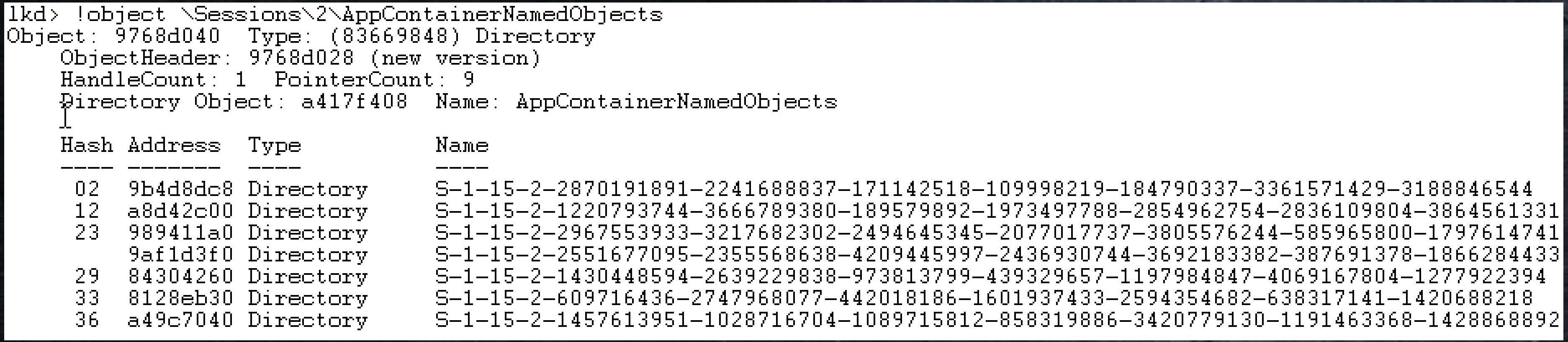
```
lkd> dt nt!_TOKEN eaef0450 LogonSession
   +0x0c0 LogonSession : 0x947bd790 _SEP_LOGON_SESSION_REFERENCES
lkd> dt nt!_TOKEN eaef0450 LowBoxHandlesEntry
   +0x278 LowboxHandlesEntry : 0x9da74d88 _SEP_LOWBOX_HANDLES_ENTRY
lkd> !hashtable /contents @@(((nt!_SEP_LOGON_SESSION_REFERENCES*)0x947bd790)->LowBoxHandlesTable.HashTable)
Table @ 853894b8
  TableSize         : 0x000080
  NumEntries        : 0x000007
  NonEmptyBuckets   : 0x000007
  NumEnumerators    : 00000000
  IndirectionLevel  : 1 (first level dir @ 8539ab48)

Dumping table contents...

Directory 0 at 8539ab48:
  Bucket 20 at 8539abe8:
      Entry 81404530, Signature=be11efd0
  Bucket 46 at 8539acb8:
      Entry 9ae6d670, Signature=e65886b3
  Bucket 66 at 8539ad58:
      Entry 9467fa40, Signature=6b256c95
  Bucket 69 at 8539ad70:
      Entry 915ea758, Signature=54adfb5a
  Bucket 106 at 8539ae98:
      Entry a4171b78, Signature=4c2b8c5a
  Bucket 123 at 8539af20:
      Entry 9da74d88, Signature=6f3d3d91
  Bucket 124 at 8539af28:
      Entry a41f50b0, Signature=552acf1c
Maximum chain length is 1 entries
lkd> dt 0x9da74d88 _SEP_LOWBOX_HANDLES_ENTRY
nt!_SEP_LOWBOX_HANDLES_ENTRY
   +0x000 HashEntry        : _RTL_DYNAMIC_HASH_TABLE_ENTRY
   +0x00c ReferenceCount   : 0x23
   +0x010 PackageSid       : 0x9da74da4 Void
   +0x014 HandleCount      : 6
   +0x018 Handles          : 0x9e0691d0  -> 0x800009b4 Void
```

Isolation Layers

- Applications in an AppContainer have their own application data
  - ~\AppData\Local\Packages\<Identifier>
  - Access to other locations requires Capability SID
  - *GetAppContainerFolderPath*
- They also have their own registry data
  - HKCR\Local Settings\<CurrentVersion>\AppContainer\Storage
  - *GetAppContainerRegistryLocation*
- Profile is also isolated
  - *Create/DeleteAppContainerProfile*
- Windows Firewall, Filtering Platform, and Network Isolation can track network/port access at the AppContainer level
- Win32k.sys protects UI in various ways with “Mosh Hardening”

Object Isolation

- Each AppContainer gets its own BaseNamedObjects directory now
  - \Sessions\n\AppContainerBaseNamedObjects



```
lkd> !object \Sessions\2\AppContainerNamedObjects
Object: 9768d040  Type: (83669848) Directory
    ObjectHeader: 9768d028 (new version)
    HandleCount: 1  PointerCount: 9
    Directory Object: a417f408  Name: AppContainerNamedObjects

    Hash Address  Type          Name
    ---- -------  ----          ----
     02  9b4d8dc8 Directory     S-1-15-2-2870191891-2241688837-171142518-109998219-184790337-3361571429-3188846544
     12  a8d42c00 Directory     S-1-15-2-1220793744-3666789380-189579892-1973497788-2854962754-2836109804-3864561331
     23  989411a0 Directory     S-1-15-2-2967553933-3217682302-2494645345-2077017737-3805576244-585965800-1797614741
         9af1d3f0 Directory     S-1-15-2-2551677095-2355568638-4209445997-2436930744-3692183382-387691378-1866284433
     29  84304260 Directory     S-1-15-2-1430448594-2639229838-973813799-439329657-1197984847-4069167804-1277922394
     33  8128eb30 Directory     S-1-15-2-609716436-2747968077-442018186-1601937433-2594354682-638317141-1420688218
     36  a49c7040 Directory     S-1-15-2-1457613951-1028716704-1089715812-858319886-3420779130-1191463368-1428868892
```

  - Additionally, RPC endpoints are in an RPC Control sub-directory
    - See *RpcServerRegisterIf3* documentation (Interface<->Server must be in AppContainer)
- The initial objects created are the ones that will have their handles duplicated into the kernel by *NtCreateLowBoxToken* by using the LowBox Handle Entry

Other

- When a process is created under an AppContainer, SE_CHANGE_NOTIFY_PRIVILEGE is only privilege granted
  - Unless *AppContainerPrivilegesEnabled* API is present
- An AppContainer token can also have User and Device Claims, as well as Device Groups and Restricted Device Groups
  - Further research needed to determine use of Device Groups/Claims
  - Perhaps a mechanism to restrict hardware access to specific drivers?
- Various other parts of the system have been hardened around AppContainers
  - OLE, .NET, DCOM, Inbox Drivers, etc…
- No official documentation/spec exists (yet) on implementing a custom app container

Signing Levels

- Used to determine if an image was signed (test, production) and by whom (Microsoft Root, Microsoft DRM root, other roots)
- Partly defined by EKUs in image certificate:
  - 1.3.6.1.4.1.311.76.3.1 (Unknown)
  - 1.3.6.1.4.1.311.10.3.6 (NT5 Build Lab)
  - 1.3.6.1.4.1.311.76.5.1 (Unknown)
- Validated by Code Integrity (CI.DLL)
  - New to Windows 8, UMCI (User-Mode CI) checks signatures for Application Packages (AppX) and .NET Images as well
- Enforcement of signing level also driven by secure boot policy
  - NtQuerySystemInformation(SystemSecureBootPolicyInformation)
    - ARM: Always enforced (MUST NOT be configurable as per UEFI Specification)
    - Non-ARM: Configured by user (MUST be configurable as per UEFI specification)

Signing Level Usage

- *PspCreateProcess* responsible for which validating signing level
  - Uses *SeQuerySigningPolicy* to check signing level and policy to determine signing level and if the image should run
    - Checks for NGEN’d images (*SepIsNgenImage*)
    - Checks for Trusted images (*SepIsMinTCB*)
      - Smss.exe, spssvc.exe, werfaultsecure.exe, csrss.exe, lsass.exe, services.exe, userinit.exe, wininit.exe, winlogon.exe, autochk.exe, genvalobj.exe
      - Must be in system path
    - Checks if this is an AppX “packaged” application (WIN://PGK claim must be in token)
      - Queries the moniker to check if image is “strongly named” (WIN://SYSAPPID claim in token)
      - Next, checks licensing cache for application origin (Store vs. non-store Application)
      - Enforces ‘locked-down mode” (*SepIsLockedDown*)
        - WSLicensingService-LOBSideLoadingActivated licensing policy enforces LOB Side-Loading
  - On ARM/SecureBoot systems, uses *SeQuerySigningPolicyExt*
    - External policy in *ext_ms_win_ntos_ksigningpolicy_l1_1_0.dll,* uses Secure Boot Hashes

Signing Levels for Legacy Features

- Signing levels now enforce the Vista+ “protected process” mechanism
  - Instead of a hack using a bit in EPROCESS, signing level determines if DRM-signed Microsoft Image
  - *PspCheckForInvalidAccessBySignatureLevel* blocks attempts to open a process or thread across certain signing levels (for some rights), as well as attempts to attach/detach to a process and other user-mode debugging actions
- *MiCreateSection* now calls *SeGetImageRequiredSigningLevel* to check for image sections for protected processes and driver images and to see if the signature matches the signing level required for such images

Early Launch Anti Malware (ELAM)

- ELAM allows drivers to setup a “boot policy”
  - Can see hardware state/measured TPM information and can invalidate PCRs
  - Can see whenever a boot driver is being loaded and can veto
    - Some in-box drivers cannot be vetoed (“core drivers”)
- Access to signatures for white/blacklisting is done through registry
- ELAM drivers require a special signature from Microsoft
  - Requires 1+ year as a security company in good standing
  - Requires small footprint (< 128 KB) and small latency (< 0.5 ms)
  - Must unload after boot drivers have been started
- Implemented as Load Order Group: “Early-Launch”
- *IoRegister/UnregisterBootDriverCallback*

Dynamic Access Control

- Marketing name for Centralized Access Protection (ACL)
- Leverages Conditional ACEs (Windows XP) and Attributes/Claims (Windows 7) to perform access control checks
  - Allows for dynamic run-time rules that do not require using group hierarchies to enforce
- For example, the attribute "Clearance Level" can be set to "5" in Bob's token
  - And a file can have a Conditional Deny ACE of the type "CleranceLevel LessThan 6"
- Failure can also lead to custom user interaction for remediation
  - Such as sending the user to a web page in order to request a higher clearance level
- Attributes are really claims made through AD User Schema

Windows on ARM (WoA)

ARM Architecture

- 32-bit (soon: 64-bit) RISC processor with SMP support
- No ring levels
  - Different execution modes instead:
    - System (SVC), User (USR), Interrupt (IRQ), Fast Interrupt (FIQ), Abort (ABT)
    - Windows only uses two modes: SVC and USR
  - Each mode has its own set of *banked registers*
- “Thumb” mode is alternate 16-bit mode, Thumb-2 extends with 32-bit
  - Used for better instruction packing (at the cost of execution speed)
- No segmentation/TSS
  - OS goes through all the modes and sets initial state. CPU restores that state through the banked registers during transitions, similar to TSS switching
  - CPU thus has interrupt stack, exception stack, user stack, etc…
    - But Windows always switches to SVC

ARM Architecture (cont)

- ARM had a “vector table” at either 0x00000000 or 0xFFFF0000
  - New processors allow setting an ASLR’d value through the CP15 coprocessor
  - Contains handlers for CPU Traps
    - Page faults/exceptions are “aborts” if on data, “prefetch” if on execution
    - Undefined instructions have their own vector/mode
    - All interrupts come through the “interrupt” vector
- CP15 “Coprocessor 15” stores many configuration registers
  - “CR3” is CP15, c2, c0 (TTBR, Translation Table Base Register)
  - TEB is in CP15, c3, TPIDRURW (User RW)
  - KTHREAD is in CP15, c3, TPIDRURO (User RO) **Maybe**
  - PCR is in CP15, c3, TPIDRPRW (Privileged RW)
- ARM is forced-alignment architecture, but Windows-compatible SoCs must be ARMv7 (supports alignment fixups)

ARM Architecture (last)

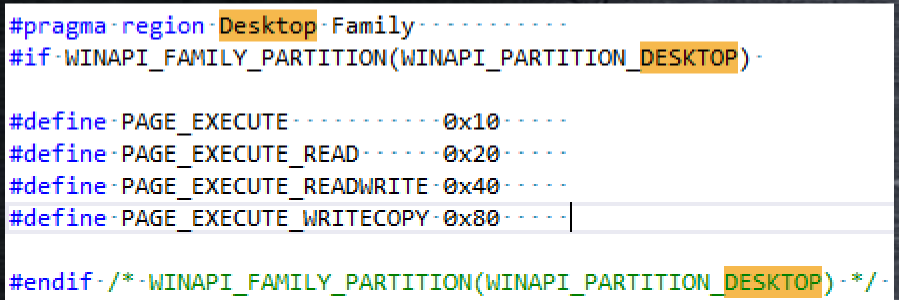
- System calls can be done in a variety of ways, but on ARMv7 most popular mechanism is SVC instruction
  - Uses r12 to pass system call ID
- KUSER_SHARED_DATA is still a fixed address
  - 0x7FFE0000 in user-mode (unchanged)
  - 0xFFFF9000 in kernel-mode (different from x86)
- ARM, like x64, supports RIP (relative instruction pointer) addressing
  - System Call Table is “compacted” similar to x64
- GIC is the Generic Interrupt Controller (similar to APIC), but secondary GPIO controllers can exist
  - Interrupt vectors start at 0x1000
  - No need for code inside the KINTERRUPT structure

Windows on ARM

- Many of the optional/user-configurable security feature/mitigations are always on
  - See Matt & Ken’s talk ([1]) which has a table of ARM Mitigation Defaults
- The only way to load applications is through the Windows Store
    - Or LOB Sideloading on Enterprise Licenses
  - Thus enforcing all the AppContainer-based policies
- Code signing is enforced
  - Also, “IL Signing Policy” can be used to force Medium/High IL code signing
- Secure Boot is enforced (limiting debug access and code execution)
- Most of the x64 security/mitigation mechanisms that are not on x86 due to compatibility concerns, are likely to be on ARM
  - High chance there’s an ARM PatchGuard protecting things like Vector Table

Windows on ARM (cont)

- API header files limit which APIs can be compiled by separating between “Desktop” and “App” families
  - For example, looks like executable anonymous memory won’t be allowed



```
#pragma region Desktop Family
#if WINAPI_FAMILY_PARTITION(WINAPI_PARTITION_DESKTOP)

#define PAGE_EXECUTE           0x10
#define PAGE_EXECUTE_READ      0x20
#define PAGE_EXECUTE_READWRITE 0x40
#define PAGE_EXECUTE_WRITECOPY 0x80

#endif /* WINAPI_FAMILY_PARTITION(WINAPI_PARTITION_DESKTOP) */
```

  - However, this is compiler-level enforcement!
    - No such check appears to exist in the kernel
    - Prevention of bypass (through manual definition) is probably done during Windows Store vetting process
- RWX memory can be allocated and executed, regardless of code signing

Game Not Over

What’s Next?

- This won’t be the end of Windows exploitation
  - But most of the tricks used in the past few year’s MS11/MS10 bugs are now dead
- There are still some avenues of attack
  - Many of the new mitigations require policy to be set by user (through ELAM) or by developer (through APIs and/or linker settings)
    - Developers continue to still fail at enabling ASLR on their DLLs. Will they remember to set Forced ASLR on their processes?
  - Injecting into 32-bit VDM process allows bypassing NULL-page mitigation
  - ARM Vector table will always be a hardcoded virtual address with executable code
- AppContainers are ***not security boundaries***
  - There will be bugs in the Broker Infrastructure model and sandbox escapes

What’s Next?

- The HAL Heap is still at a fixed address, and on Windows 8 has function pointers to key code in the HAL
  - Leveraged this as part of an exploit at SyScan 2012
- Some overrides seem to exist
  - *NtUserSetProcessRestrictionExemption*
  - Many registry keys checked by CI.DLL
  - New BCD options
  - Many of these check for a “developer license”
    - *ExQueryFastCacheDevLicense*
- There’s an Info Leak Party In Ring 0 (TBD ☺)

Conclusion

Key Takeaways

- Windows 8 was the subject of the most intensive and well-thought-out exploit mitigation and security hardening process ever attempted by Microsoft
  - And it delivered
  - *“If you can’t beat them, hire them”*
    - Anyone else miss uninformed.org Windows hacking articles? ☺
- On top of that, Windows 8 has a completely new application model (granted, Apple’s done it first)
  - Including key OS features to increase the security of TIFKAM applications
  - As well as, on ARM devices, enforced Windows Store vetting
- Enterprise security features and increased firmware/TPM security seal the deal
  - Along with greater access to boot state/decisions for security vendors

Is anyone listening?

- Doing this research uncovered a plethora of codenames for the new application model
  - “Immersive App”
  - “Modern App”
  - “Packaged App”
  - “Lowboxed App”
  - “Appcontained App”
  - “Windows 8 UI-Style App”
  - “Metro App”
- Microsoft, can you *please* pick a name? ☺
  - And what on Earth is “Mosh” Hardening?
    - 10/17/2012 UPDATE -- “Modern Shell”

QA

- Greetz/shouts to: Matt Miller, Ken Johnson, Bruce Dang, Matthieu Suiche, Tarjei Mandt, and the organizers!

CrowdStrike