Extraordinary String Based Attacks

# SMASHING THE ATOM

Tarjei Mandt

RECON 2012

# About Me

- Security Researcher at Azimuth Security
- Past presentations
  - Heaps of Doom (/w Chris Valasek)
  - Kernel Attacks Through User-Mode Callbacks
  - Kernel Pool Exploitation on Windows 7
- Generally interested in operating system internals and bug finding
- Recent focus on embedded platforms

# This Talk

- A rather unusual Windows bug class
  - Affects Windows atoms
  - 3 vulnerabilities patched 2 days ago in MS12-041
- Allows a non-privileged user to run code in the context of a privileged process
  - E.g. the Windows login manager (winlogon)
- No need to run arbitrary code in Ring 0
  - DEP/ASLR? SMEP? No problem!

# Previous Work

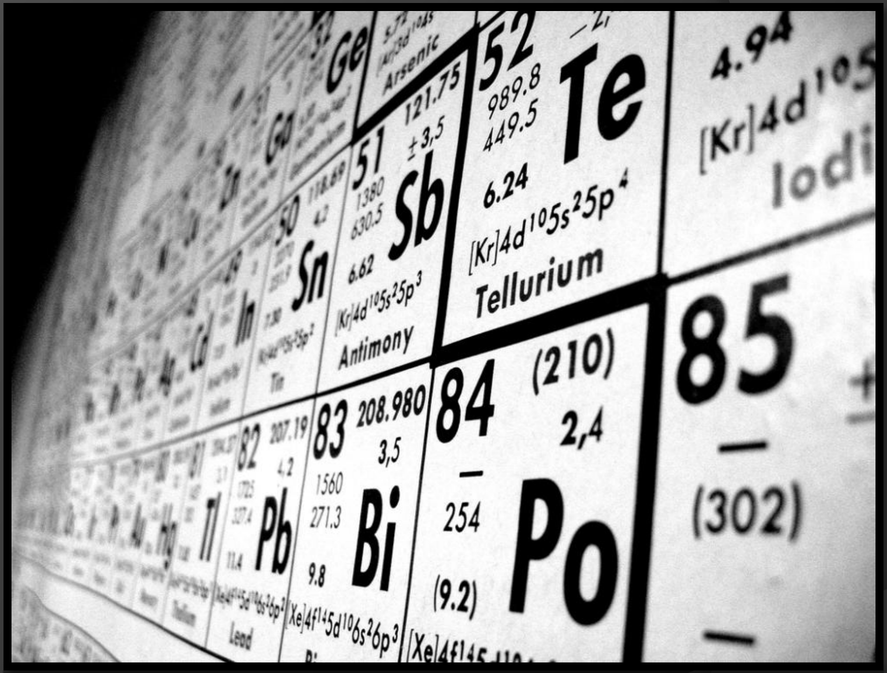
- Atoms briefly mentioned in Windows sandboxing literature
  - Stephen A. Ridley – Escaping the Sandbox
  - Tom Keetch – Practical Sandboxing on Windows
- Getadmin exploit (1997)
  - Exploited unchecked pointer in **NtAddAtom**
  - API issue – not specific to atom misuse

# Outline

- Atoms
- Vulnerabilities
- Attack Vectors
- Exploitation
- Windows 8
- Conclusion

Smashing the Atom

# Atoms



# Atoms

- A Windows data type used to store strings and integers
  - Referenced using 16-bit values
- Stored in a hash table known as an *atom table*
- Generally used to share information between processes
  - Initially designed to support Dynamic Data Exchange (DDE)
- Also used by the operating system

# Atom Tables

- Defined in the local (application) or global (system) scope
- Application defined tables are fully managed in user-mode
- System defined tables are managed by the kernel
  - Callouts to win32k where necessary
- Two common system tables
  - Global And User Atom Tables

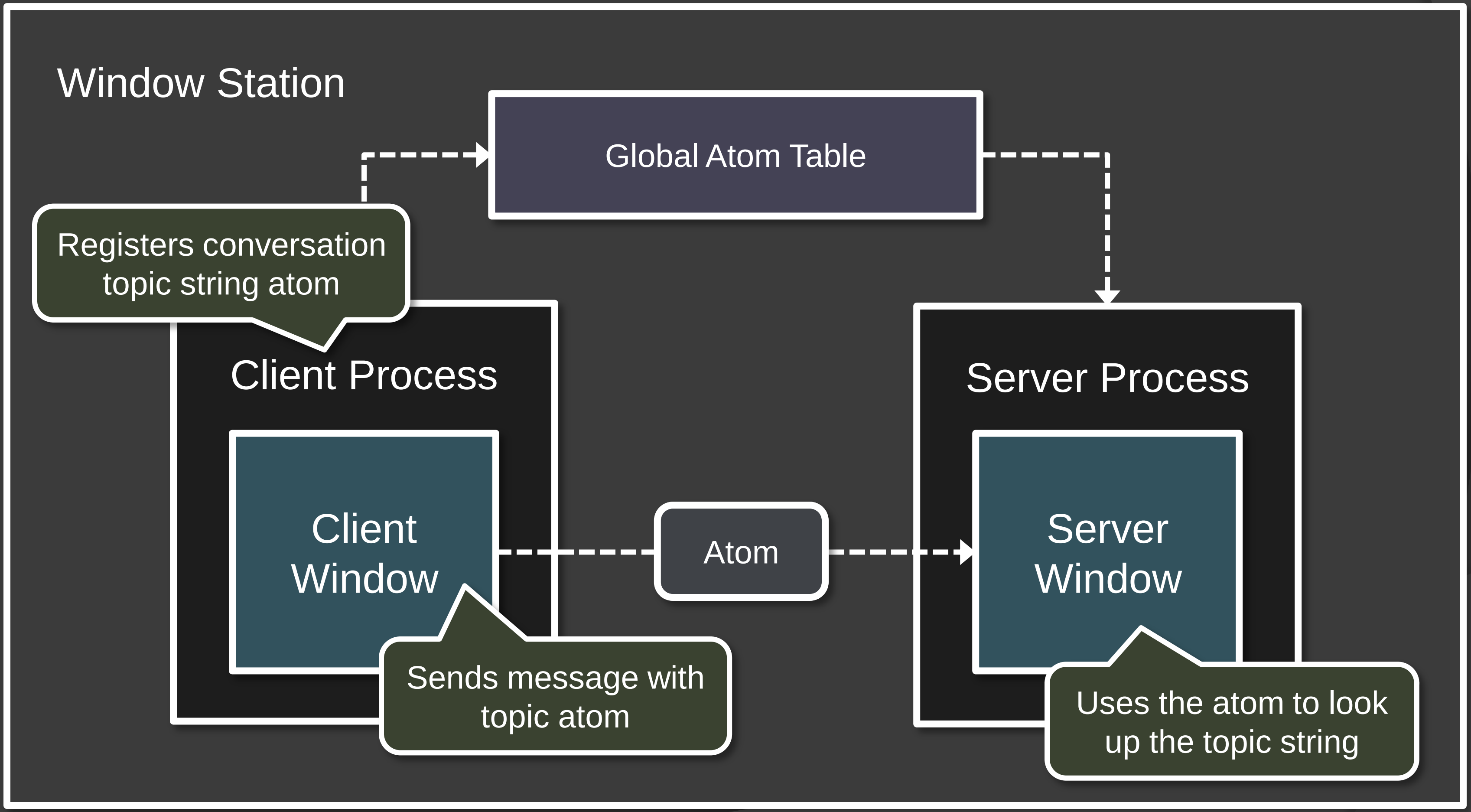
# Local Atom Table

- Defined per application
- Table initialization handled transparently to applications
- Exposed through an own set of APIs (kernel32)
  - **AddAtom**, **DeleteAtom**, **FindAtom**, …
- Actual implementation in runtime library (NTDLL)

# Global Atom Table

- Defined per window station
  - **win32k!CreateGlobalAtomTable**
- Accessible to any application in the same window station by default
- Can also be job specific if global atoms UI restrictions are enabled
- Exposed through an own set of APIs prefixed “Global”
  - **GlobalAddAtom**, **GlobalDeleteAtom**, …

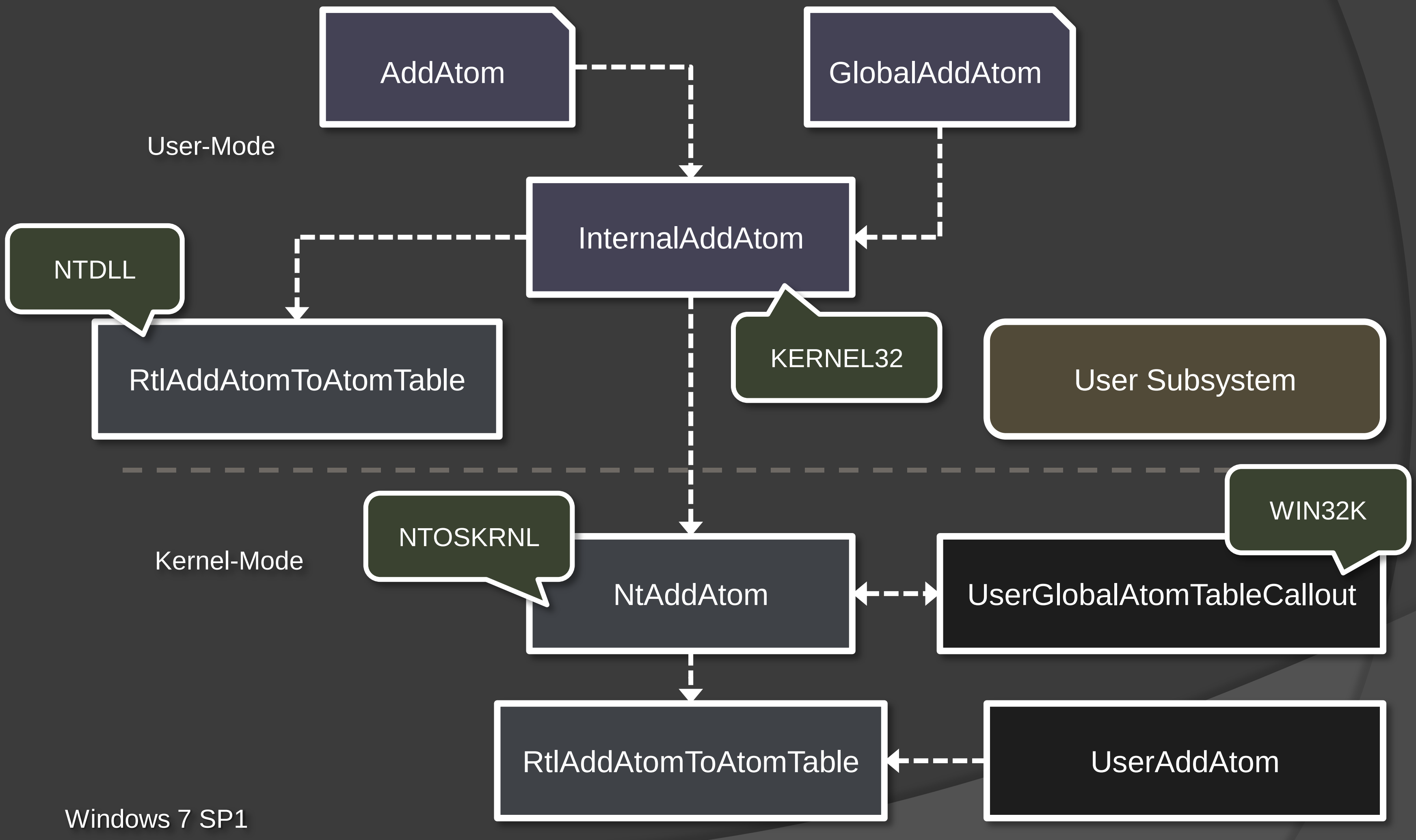
# Global Atom Table (DDE)


Window Station
Global Atom Table
Registers conversation topic string atom
Client Process
Client Window
Atom
Server Process
Server Window
Sends message with topic atom
Uses the atom to look up the topic string

# User Atom Table

- Defined per session
  - **win32k!UserRtlCreateAtomTable**
- Holds data used by the User subsystem
  - Window class names
  - Clipboard format names , …
- Not exposed to user applications directly
  - However, some APIs allow values to be inserted and queried
  - **RegisterWindowMessage**

# Atom Table Interaction



Windows 7 SP1

# Atom Types

- Two types of atoms
  - Strings and integers
- Both types are managed by the same atom table
  - Defined with separate atom value ranges
  - No type information needed
- Both types are handled using the same APIs

# String Atoms

- Registered upon passing a string to **RtlAddAtomToAtomTable**
- Assigned an atom value in the range 0xC001 through 0xFFFF
  - Subsequently used to look up the string
- Limits the string size to 255 bytes
- Reference counted to keep track of use
- Example: Window class names

# Integer Atoms

- Integer values map directly to the atom value
  - Never actually stored in the atom table
- Defined in the range 1 to 0xBFFF
  - Only stores decimal values up to 49151
- Only registered for the sake of consistency
- Example: Standard clipboard formats

# Atom Table Creation

- Created using **RtlCreateAtomTable**
- Initialized with an integer representing the number of hash buckets (default 37)
- A string atom is inserted into a bucket based on its string hash
  - Used for efficient lookup of string atoms
- The atom table itself is defined by the RTL_ATOM_TABLE structure

# Atom Table Structure

```
typedef struct _RTL_ATOM_TABLE
{
/*0x000*/     ULONG32      Signature;
/*0x004*/     struct _RTL_CRITICAL_SECTION CriticalSection;
/*0x01C*/     struct _RTL_HANDLE_TABLE RtlHandleTable;
/*0x03C*/     ULONG32      NumberOfBuckets;
/*0x040*/     struct _RTL_ATOM_TABLE_ENTRY* Buckets[1];
} RTL_ATOM_TABLE, *PRTL_ATOM_TABLE;
```

Windows 7 SP1 (x86)

# Atom Table Entries

- Each string atom is represented by an RTL_ATOM_TABLE_ENTRY structure
- Defines the atom value and string
- Reference counted to keep track of string (atom) use
  - Incremented whenever an identical string is added to the atom table
- Flags to indicate whether an atom has been *pinned*

# Atom Table Entry Structure

```
typedef struct _RTL_ATOM_TABLE_ENTRY
{
/*0x000*/    struct _RTL_ATOM_TABLE_ENTRY* HashLink;
/*0x004*/    UINT16     HandleIndex;
/*0x006*/    UINT16     Atom;
/*0x008*/    UINT16     ReferenceCount;
/*0x00A*/    UINT8      Flags;
/*0x00B*/    UINT8      NameLength;
/*0x00C*/    WCHAR      Name[1];
} RTL_ATOM_TABLE_ENTRY, *PRTL_ATOM_TABLE_ENTRY;
```

For handling string hash collisions

Used to generate atom values

Track atom use

Windows 7 SP1 (x86)

# Atom Pinning

- If the reference count of an atom overflows, the atom is pinned
  - Indicated by the RTL_ATOM_PINNED (1) flag
- A pinned atom is not freed until its atom table is destroyed
  - E.g. upon destroying a window station or logging out a user
- Windows also supports on-demand pinning
  - **RtlPinAtomInAtomTable**
  - Prevents atoms from being deliberately deleted

# Atom Value Assignment

- Atom tables use a separate handle table for string atom value assignment
  - Retrieved using **ExCreateHandle**
- Attempts to use a recently freed handle to optimize lookup
  - Otherwise performs exhaustive search
- Actual atom value is obtained by OR'ing the handle index with MAXINTATOM
  - Atom = ( Handle >> 2 ) | 0xC000

# System Atom Table Access

- System atom tables are generally available to all user processes
  - Designed for sharing information
- In a sandbox, we want to restrict access in the less privileged components
  - Prevent leaking of (sensitive) information
  - Prevent deletion of atoms used by other (e.g. more privileged) applications

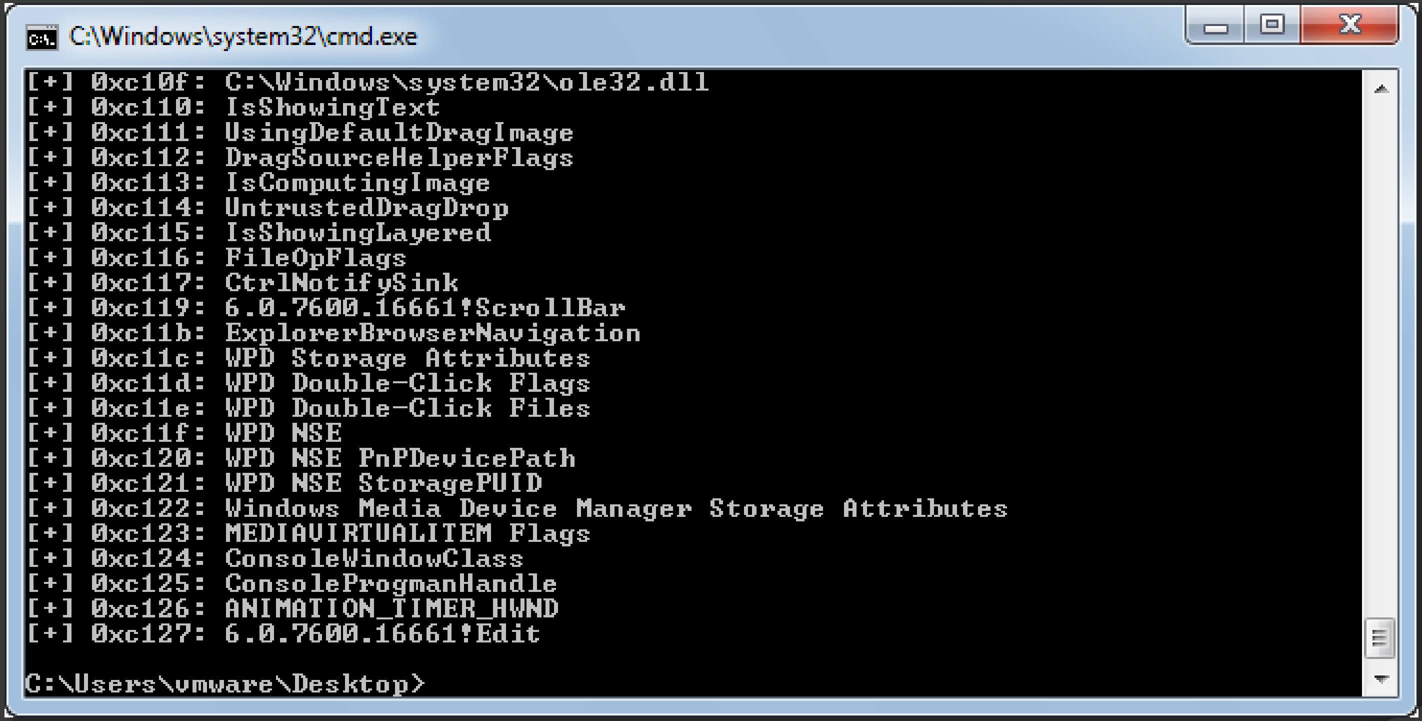
# Global Atom Table Access

- Access can be restricted using job object UI restrictions
  - JOB_OBJECT_UILIMIT_GLOBALATOMS
- When set, Windows creates a separate atom table and associates it with the job object
- The process of choosing the correct atom table is handled in **win32k!UserGlobalAtomTableCallout**
  - Checks the global atoms UI restriction flag by calling **nt!PsGetJobUIRestrictionsClass**

# User Atom Table Access

- In Windows 7, there's no practical isolation of the user atom table
  - More on Windows 8 later
- Accessible to any process running in the same session
  - E.g. using APIs which (indirectly) operate on it
- A process can query the values of any user atom using **GetClipboardFormatName**
  - No distinction made between clipboard format strings and other user atom strings

# Enumerating User Atoms



```
[+] 0xc10f: C:\Windows\system32\ole32.dll
[+] 0xc110: IsShowingText
[+] 0xc111: UsingDefaultDragImage
[+] 0xc112: DragSourceHelperFlags
[+] 0xc113: IsComputingImage
[+] 0xc114: UntrustedDragDrop
[+] 0xc115: IsShowingLayered
[+] 0xc116: FileOpFlags
[+] 0xc117: CtrlNotifySink
[+] 0xc119: 6.0.7600.16661!ScrollBar
[+] 0xc11b: ExplorerBrowserNavigation
[+] 0xc11c: WPD Storage Attributes
[+] 0xc11d: WPD Double-Click Flags
[+] 0xc11e: WPD Double-Click Files
[+] 0xc11f: WPD NSE
[+] 0xc120: WPD NSE PnPDevicePath
[+] 0xc121: WPD NSE StoragePUID
[+] 0xc122: Windows Media Device Manager Storage Attributes
[+] 0xc123: MEDIAVIRTUALITEM Flags
[+] 0xc124: ConsoleWindowClass
[+] 0xc125: ConsoleProgmanHandle
[+] 0xc126: ANIMATION_TIMER_HWND
[+] 0xc127: 6.0.7600.16661!Edit

C:\Users\vmware\Desktop>
```

Smashing the Atom

# Vulnerabilities

# Atom Handling Vulnerabilities

- 3 separate vulnerabilities in string atom handling
  - Register Class Name Handling Vulnerability
  - Set Class Name Handling Vulnerability
  - Clipboard Format Name Handling Vulnerability
- Addressed in MS12-041
  - http://technet.microsoft.com/en-us/security/bulletin/ms12-041
- Allows an attacker to take control over system managed string atoms
  - We discuss the implications of this later

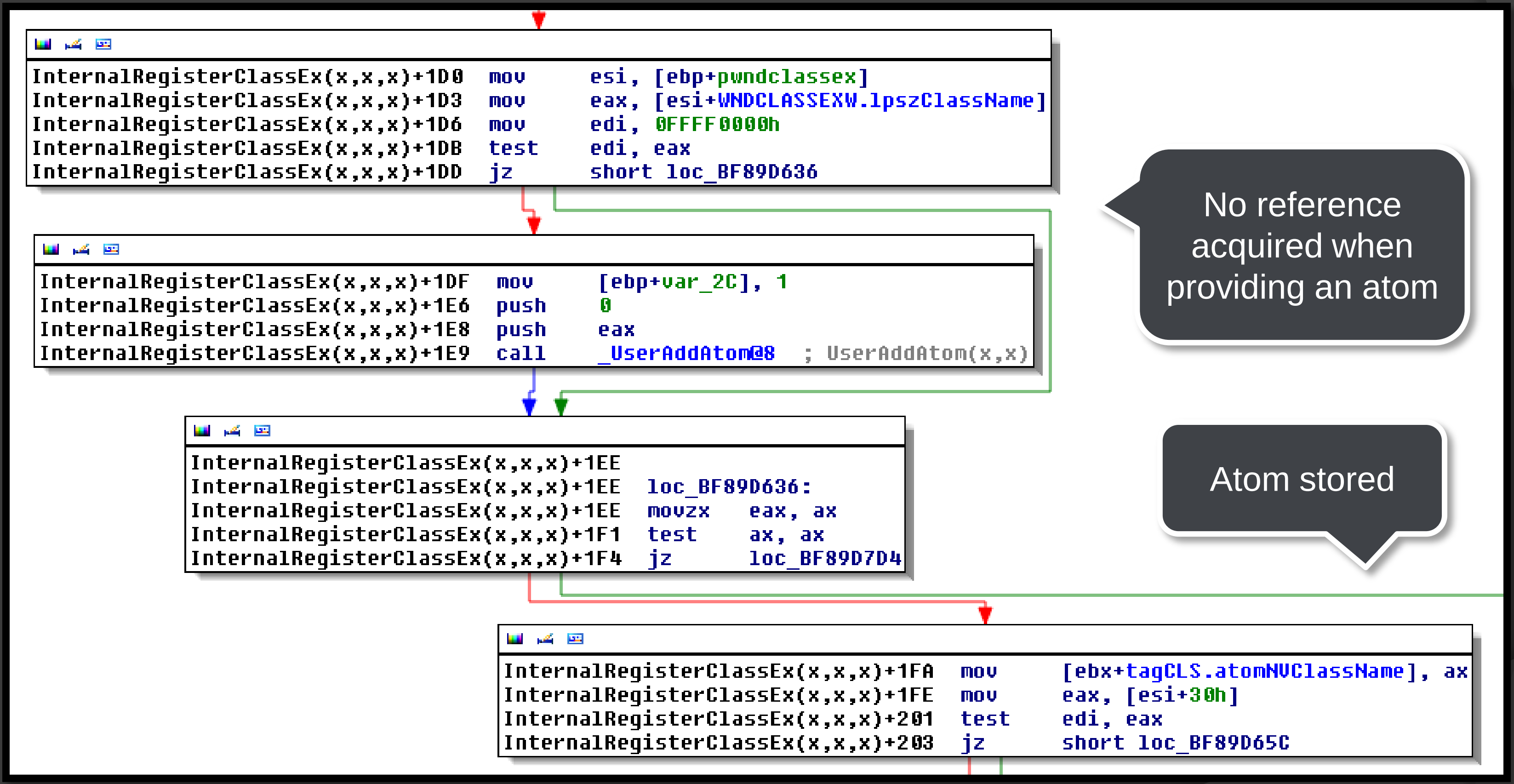
# Window Class

- An application describes a window's attributes using a window class
  - Defined by the WNDCLASS(EX) structure
- lpszClassName sets the class name
  - Can either be a string or an atom
- Win32k differs between the two internally by looking at the high 16-bits
  - If only lower 16-bits are set, it is handled as an atom

# Class Name String Atom

- If a string is provided, win32k converts the string into an atom
  - Handled by **win32k!UserAddAtom**
  - Atom value stored in the win32k managed class data structure (win32k!tagCLS)
- If an atom is provided, the function simply copies its value to the class data structure
  - No atom validation or retaining of reference

# CVE-2012-1864



```
InternalRegisterClassEx(x,x,x)+1D0  mov     esi, [ebp+pwndclassex]
InternalRegisterClassEx(x,x,x)+1D3  mov     eax, [esi+WNDCLASSEXW.lpszClassName]
InternalRegisterClassEx(x,x,x)+1D6  mov     edi, 0FFFF0000h
InternalRegisterClassEx(x,x,x)+1DB  test    edi, eax
InternalRegisterClassEx(x,x,x)+1DD  jz      short loc_BF89D636
```

```
InternalRegisterClassEx(x,x,x)+1DF  mov     [ebp+var_2C], 1
InternalRegisterClassEx(x,x,x)+1E6  push    0
InternalRegisterClassEx(x,x,x)+1E8  push    eax
InternalRegisterClassEx(x,x,x)+1E9  call    _UserAddAtom@8   ; UserAddAtom(x,x)
```

```
InternalRegisterClassEx(x,x,x)+1EE
InternalRegisterClassEx(x,x,x)+1EE  loc_BF89D636:
InternalRegisterClassEx(x,x,x)+1EE  movzx   eax, ax
InternalRegisterClassEx(x,x,x)+1F1  test    ax, ax
InternalRegisterClassEx(x,x,x)+1F4  jz      loc_BF89D7D4
```

```
InternalRegisterClassEx(x,x,x)+1FA  mov     [ebx+tagCLS.atomNVClassName], ax
InternalRegisterClassEx(x,x,x)+1FE  mov     eax, [esi+30h]
InternalRegisterClassEx(x,x,x)+201  test    edi, eax
InternalRegisterClassEx(x,x,x)+203  jz      short loc_BF89D65C
```

No reference acquired when providing an atom

Atom stored

Windows 7 SP1 (x86)

# CVE-2012-1864

- When a class is unregistered, **win32k!DestroyClass** releases the atom reference
  - Even when no reference was acquired previously
- An attacker could register a class using an atom of a more privileged application
  - Could free and reregister the atom with a different string

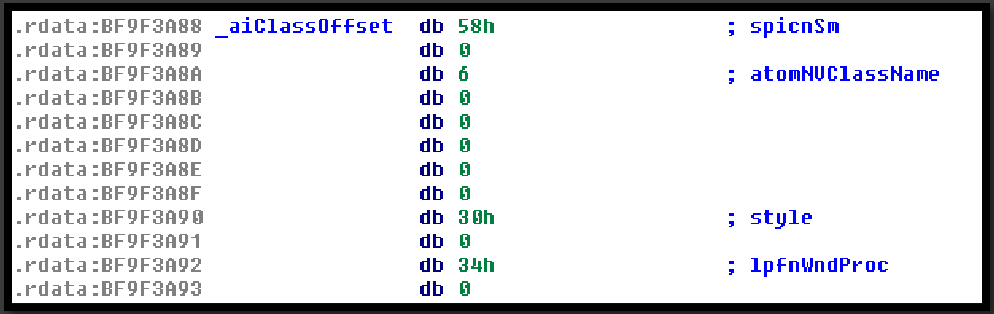
# Version Prefixed Class Name

- Since Windows XP, class objects define two class name atoms
  - atomClassName
  - atomNVClassName
- The former defines the base class name
  - Fixed once registered
- The latter prefixes the name with version specific information
  - 6.0.7600.16661!ScrollBar
  - Allows classes of the same name, but of different versions to be styled differently

# Updating Class Name Atom

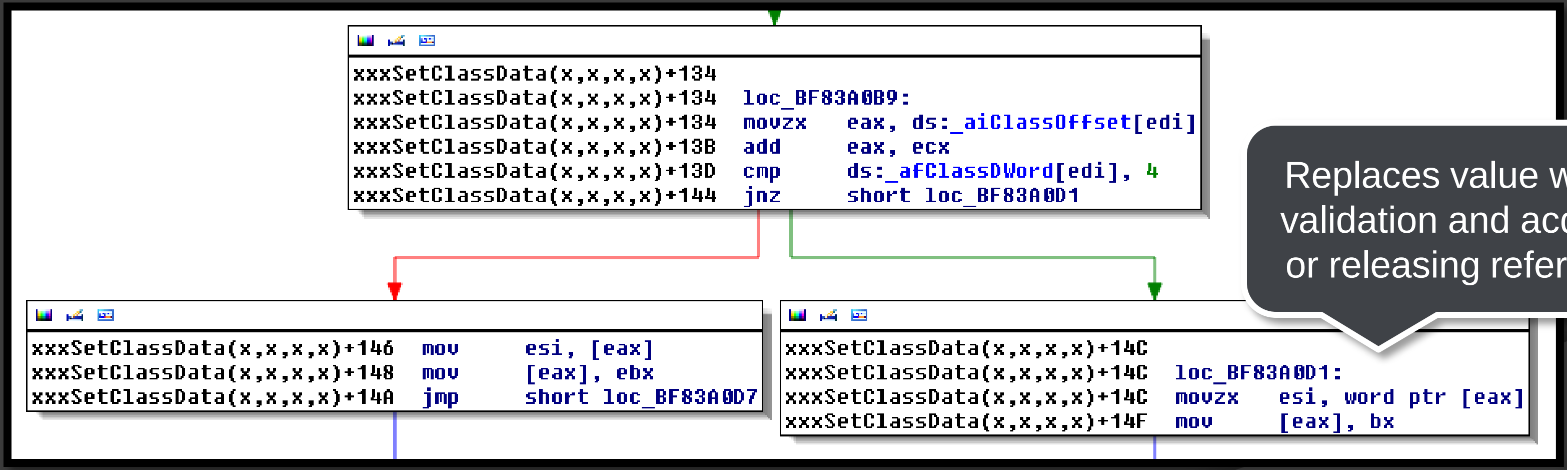
- An application can update the version prefixed name of a registered class
  - **SetClassLongPtr** using the GCW_ATOM (0xFFFFFFE0) index
- Internally, win32k looks up the index (adjusted) in an offset table
  - Finds the offset to the atom value in the class object structure
- In setting or replacing the version prefixed class name atom, no validation or referencing is performed

# CVE-2012-1865



```
.rdata:BF9F3A88 _aiClassOffset  db 58h                  ; spicnSm
.rdata:BF9F3A89                 db 0
.rdata:BF9F3A8A                 db 6                    ; atomNVClassName
.rdata:BF9F3A8B                 db 0
.rdata:BF9F3A8C                 db 0
.rdata:BF9F3A8D                 db 0
.rdata:BF9F3A8E                 db 0
.rdata:BF9F3A8F                 db 0
.rdata:BF9F3A90                 db 30h                  ; style
.rdata:BF9F3A91                 db 0
.rdata:BF9F3A92                 db 34h                  ; lpfnWndProc
.rdata:BF9F3A93                 db 0
```

Offset to version prefixed class name in the class data structure



```
xxxSetClassData(x,x,x,x)+134
xxxSetClassData(x,x,x,x)+134 loc_BF83A0B9:
xxxSetClassData(x,x,x,x)+134 movzx   eax, ds:_aiClassOffset[edi]
xxxSetClassData(x,x,x,x)+13B add     eax, ecx
xxxSetClassData(x,x,x,x)+13D cmp     ds:_afClassDWord[edi], 4
xxxSetClassData(x,x,x,x)+144 jnz     short loc_BF83A0D1
```

```
xxxSetClassData(x,x,x,x)+146 mov     esi, [eax]
xxxSetClassData(x,x,x,x)+148 mov     [eax], ebx
xxxSetClassData(x,x,x,x)+14A jmp     short loc_BF83A0D7
```

```
xxxSetClassData(x,x,x,x)+14C
xxxSetClassData(x,x,x,x)+14C loc_BF83A0D1:
xxxSetClassData(x,x,x,x)+14C movzx   esi, word ptr [eax]
xxxSetClassData(x,x,x,x)+14F mov     [eax], bx
```

Replaces value without validation and acquiring or releasing references

Windows 7 SP1 (x86)

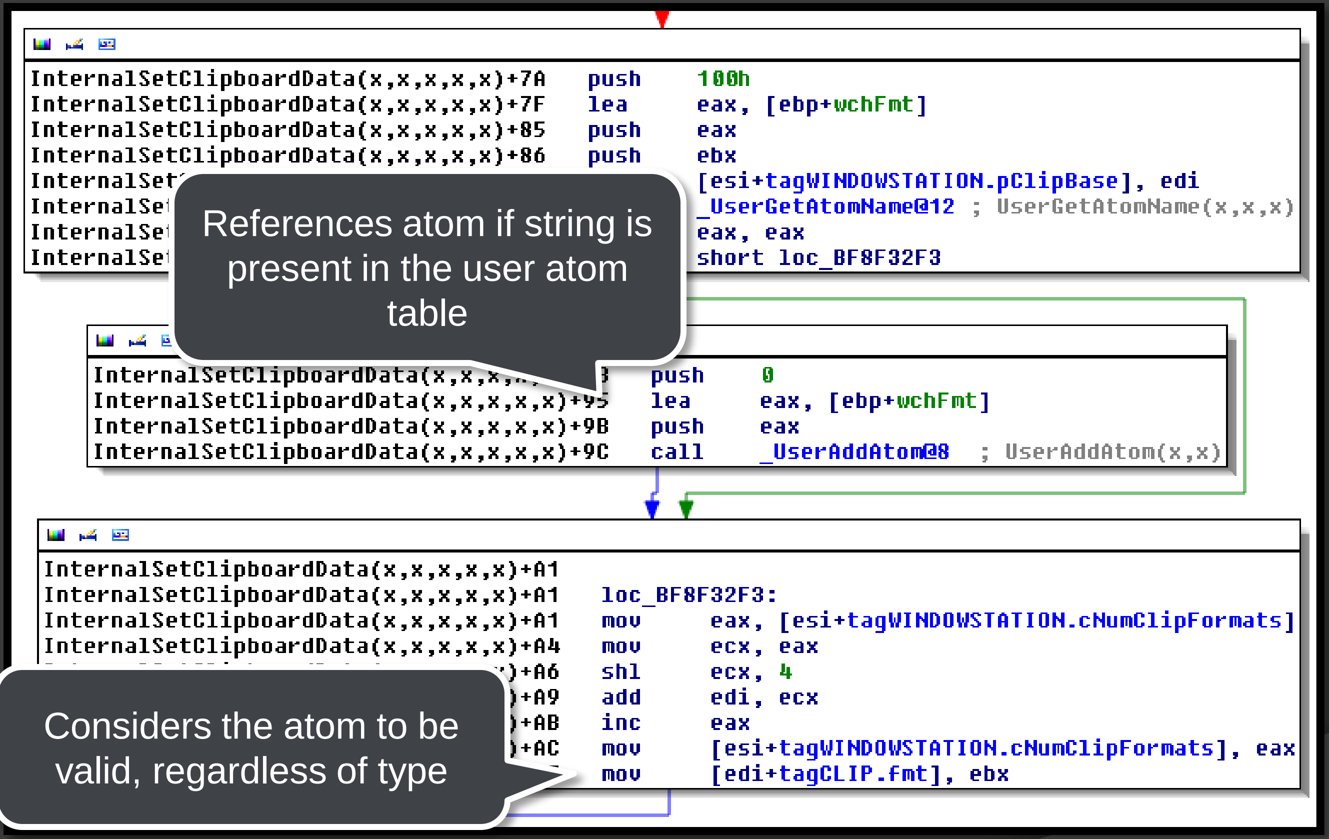
# Clipboard Formats

- Windows uses atoms to uniquely identify each clipboard format type
- Applications can also register their own clipboard formats
- **user32!RegisterClipboardFormat**
  - Registers the atom for the user provided format name string in the user atom table
- **user32!SetClipboardData**
  - Sets clipboard data of the particular type using the provided atom value

# InternalSetClipboardData

- Handles **SetClipboardData** requests
- Calls **win32k!UserGetAtomName** and **win32k!UserAddAtom** if the provided atom is present
  - Properly verifies and references the string atom
- If the atom is not present, the function still saves the data using the (invalid) atom
  - Considers the atom to be a default type (integer)
  - Fails to check if the atom is really an integer atom (i.e. below 0xC000)

# CVE-2012-1866



```
InternalSetClipboardData(x,x,x,x,x)+7A  push    100h
InternalSetClipboardData(x,x,x,x,x)+7F  lea     eax, [ebp+wchFmt]
InternalSetClipboardData(x,x,x,x,x)+85  push    eax
InternalSetClipboardData(x,x,x,x,x)+86  push    ebx
InternalSet                                     [esi+tagWINDOWSTATION.pClipBase], edi
InternalSe                                      _UserGetAtomName@12 ; UserGetAtomName(x,x,x)
InternalSe                                      eax, eax
InternalSe                                      short loc_BF8F32F3

InternalSetClipboardData(x,x,x,         3  push    0
InternalSetClipboardData(x,x,x,x,x)+95  lea     eax, [ebp+wchFmt]
InternalSetClipboardData(x,x,x,x,x)+9B  push    eax
InternalSetClipboardData(x,x,x,x,x)+9C  call    _UserAddAtom@8  ; UserAddAtom(x,x)

InternalSetClipboardData(x,x,x,x,x)+A1
InternalSetClipboardData(x,x,x,x,x)+A1  loc_BF8F32F3:
InternalSetClipboardData(x,x,x,x,x)+A1  mov     eax, [esi+tagWINDOWSTATION.cNumClipFormats]
InternalSetClipboardData(x,x,x,x,x)+A4  mov     ecx, eax
                                )+A6  shl     ecx, 4
                                )+A9  add     edi, ecx
                                )+AB  inc     eax
                                )+AC  mov     [esi+tagWINDOWSTATION.cNumClipFormats], eax
                                      mov     [edi+tagCLIP.fmt], ebx
```

References atom if string is present in the user atom table

Considers the atom to be valid, regardless of type

Windows 7 SP1 (x86)

Smashing the Atom

# Attack Vectors

# Enumerating Attack Vectors

- Look at how (string) atoms are used by the system
  - Registered window messages
  - Clipboard format names
  - Window class names
  - Cursor module paths
  - Hook module paths
- Evaluate how user input may affect string atom operations

# Registered Window Messages

- An application can register new window messages
  - **RegisterWindowMessage**
  - Stored as a string atom in the user atom table
- Typically used when messaging between two cooperating applications
  - If both register the same string, they receive the same message value

# Registered Window Messages

- Windows does not pin the string atom for the registered message
- An attacker may potentially free window message atoms registered by applications
  - Can cause desynchronization between two applications sending private messages
  - E.g. by freeing and re-registering messages in reverse-order

# Clipboard Format Names

- Applications can register their own clipboard formats
  - **RegisterClipboardFormat**
  - Identified as string atoms in the user atom table
- These atoms are not pinned, hence can be freed by an attacker
- However, clipboard data handling between privilege levels is subject to UIPI
  - List of exempt formats only contain standard (integer) clipboard formats

# Window Class Names

- Names of window classes are stored in the user atom table
  - Atom used by the class object to look up the class name string
- Windows does not pin the string atoms of non-system class objects
- An attacker could free the atom used by the system to identify class objects
  - Re-registering the string could cause lookups to resolve to the wrong object

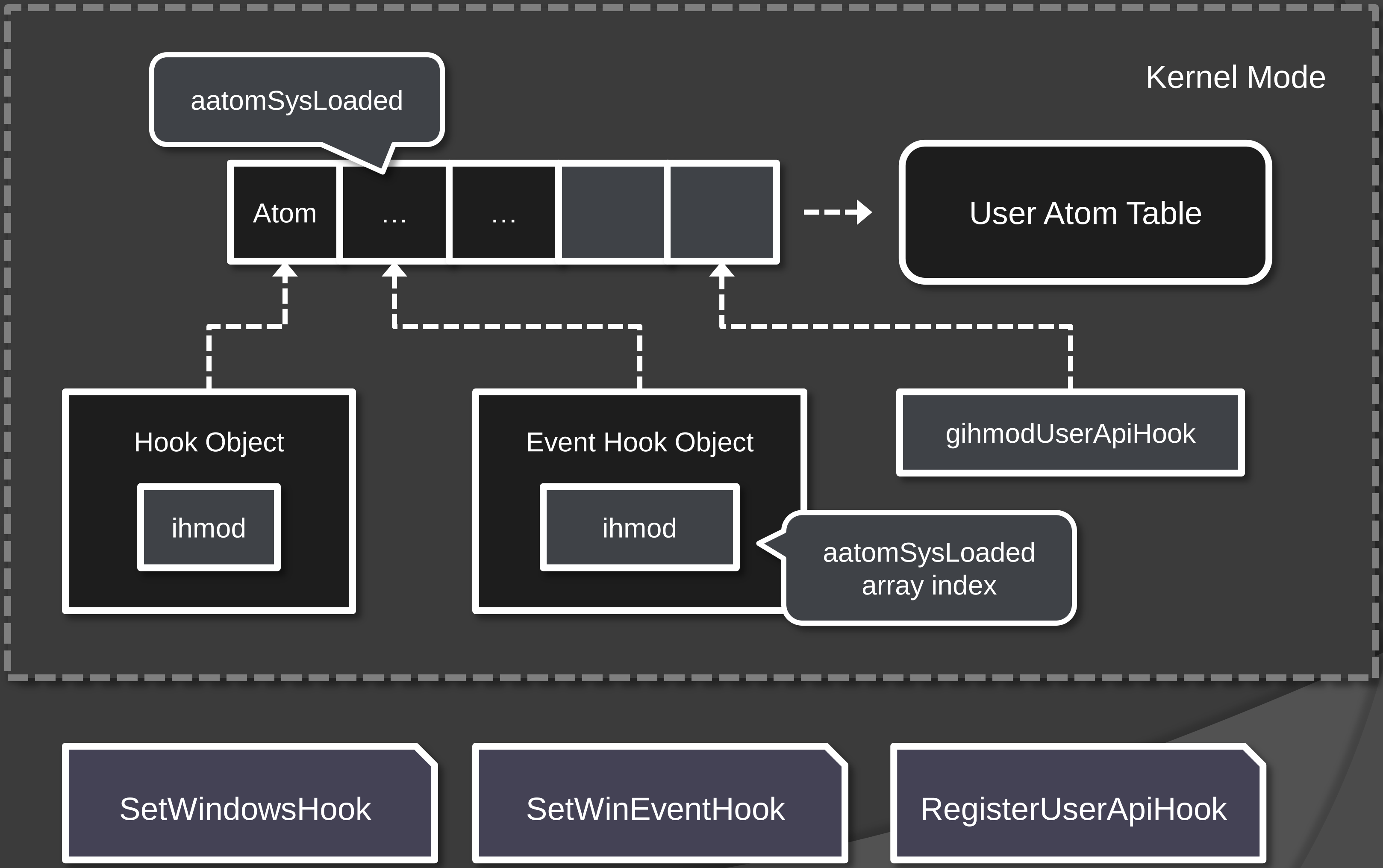
# Cursor Module Names

- Windows stores the module path of a loaded cursor as a string atom
  - atomModName field of the cursor object
- Used to determine if a cursor has already been loaded
  - **win32k!_FindExistingCursorIcon**
- Windows does not pin this atom
  - An attacker could potentially free its value
  - Minimal security impact

# Hook Module Paths

- Windows allows external modules to be used when setting windows hooks
  - **SetWindowsHookEx**
  - **SetWinEventHook**
  - **RegisterUserApiHook**
- The module path is stored as a string atom in the user atom table
  - Atom value stored at an index in the global **aatomSysLoaded** array

# Hook Module String Atoms



# Hook Module Loading

- Windows looks up the string atom upon loading an external module hook
  - Invokes a user-mode callback and passes the string to **LoadLibrary**
- An attacker who frees any such atom could possibly inject arbitrary modules
- Hooks play an integral part in Windows in providing application theming
  - Relies on the *user api hook*

# User Api Hook

- Special hooking mechanism introduced to support Windows themes
  - **RegisterUserApiHook**
- Can only be registered by privileged processes
  - Requires the TCB privilege
  - Caller must be running as SYSTEM
- Allows Windows to load a theme client module into every GUI application

Smashing the Atom

# Exploitation

# Theme Subsystem

- Introduced in Windows XP
  - Extended in Vista to support desktop composition (DWM)
- Hooks into USER32 in order to customize non-client region metrics
- Loads an instance of uxtheme.dll into every Windows application
  - Uses the user api hook registered by winlogon

# Theme Server

- Manages the theme subsystem
  - Runs in a service host process
  - Registers //ThemeApiPort
- Keeps track of the Windows theme configuration for all running sessions
- Each GUI (themed) process keeps an active connection with the theme server
  - Used to retrieve updated theme configurations

# Theme Api Port Connections

```
kd> !alpc /lpc 8701a458
8701a458('ThemeApiPort') 1, 10 connections
        85a17ae0 0 -> 85e53038 0 853c3790('winlogon.exe')
        872802f8 0 -> 863df540 0 853d8540('winlogon.exe')
        85289f00 0 -> 853e3038 0 853c3790('winlogon.exe')
        86464d18 0 -> 8538a928 0 853d8540('winlogon.exe')
        85be9038 0 -> 8533c2e0 0 853ea5c0('mmc.exe')
        87257980 0 -> 86fd6458 0 85e63030('explorer.exe')
        871fd038 0 -> 86f3db98 0 85dfc8a0('dwm.exe')
        85a53368 0 -> 8534f298 0 852eb030('explorer.exe')
        871c76a0 0 -> 8659ef00 0 852aa030('calc.exe')
        872bc8f8 0 -> 85e6b370 0 853a4388('procexp.exe')
```

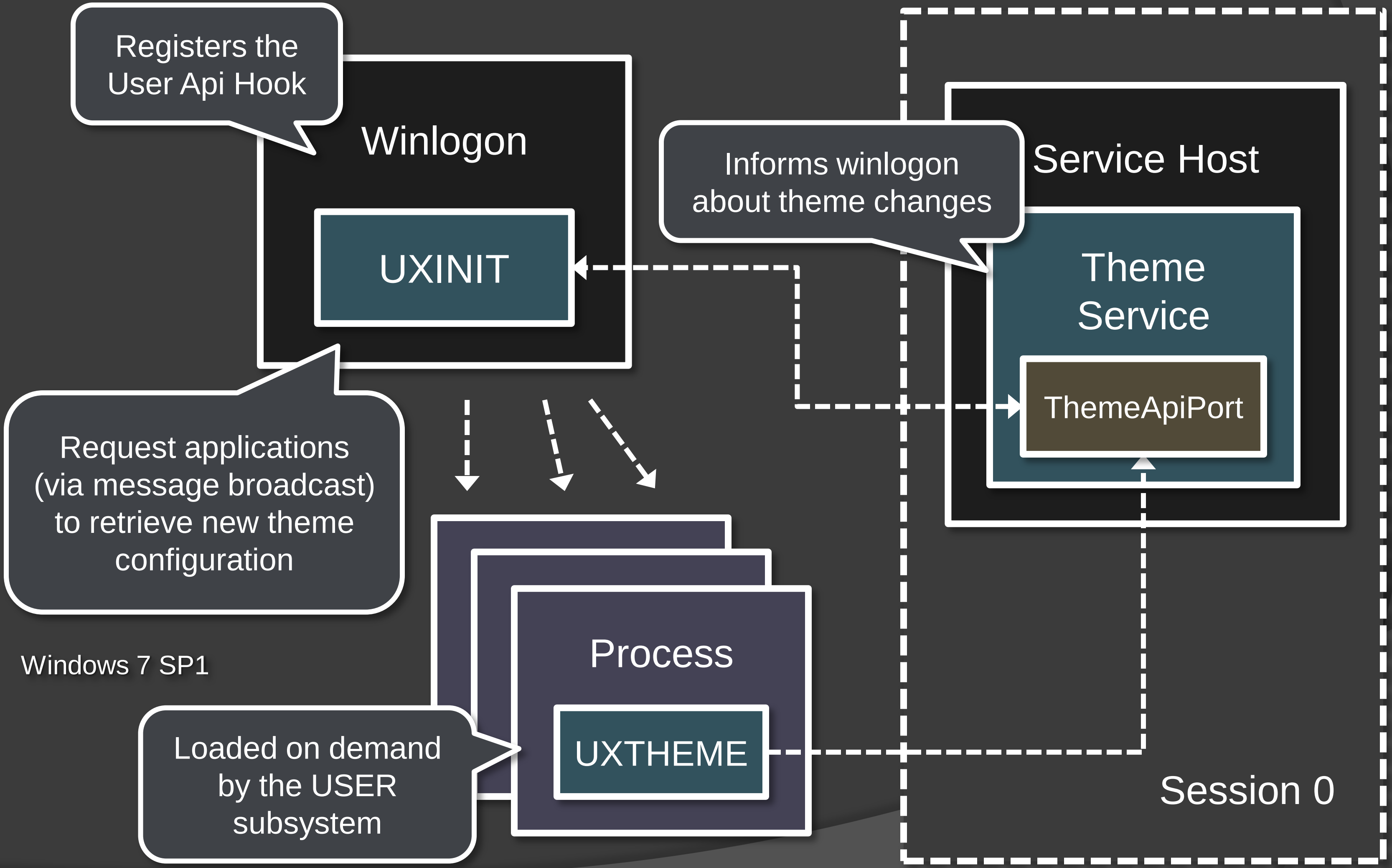
# Theme Session Initialization

- On each new session, Winlogon calls UXINIT to interface with the Theme Server
  - Acts as the theme server client
  - Sends a ThemeApiConnectionRequest packet to //ThemeApiPort over ALPC
- Once connected, Winlogon registers a set of callbacks
  - **CThemeServerClient::SessionCreate()**
  - Allows the theme server to load themes and install and remove theme hooks

# Theme Hooks Installation

- For installing hooks, the theme server service injects a thread into Winlogon
  - **UXINIT!Remote_ThemeHooksInstall**
- Winlogon (from UXINIT) subsequently calls **RegisterUserApiHook**
  - Takes a structure defining the library to load and the function (export) to execute
  - Library: %SystemRoot%/System32/uxtheme.dll
  - Function: **ThemeInitApiHook**

# Ux Theme Architecture



Registers the User Api Hook

Informs winlogon about theme changes

Request applications (via message broadcast) to retrieve new theme configuration

Loaded on demand by the USER subsystem

Windows 7 SP1

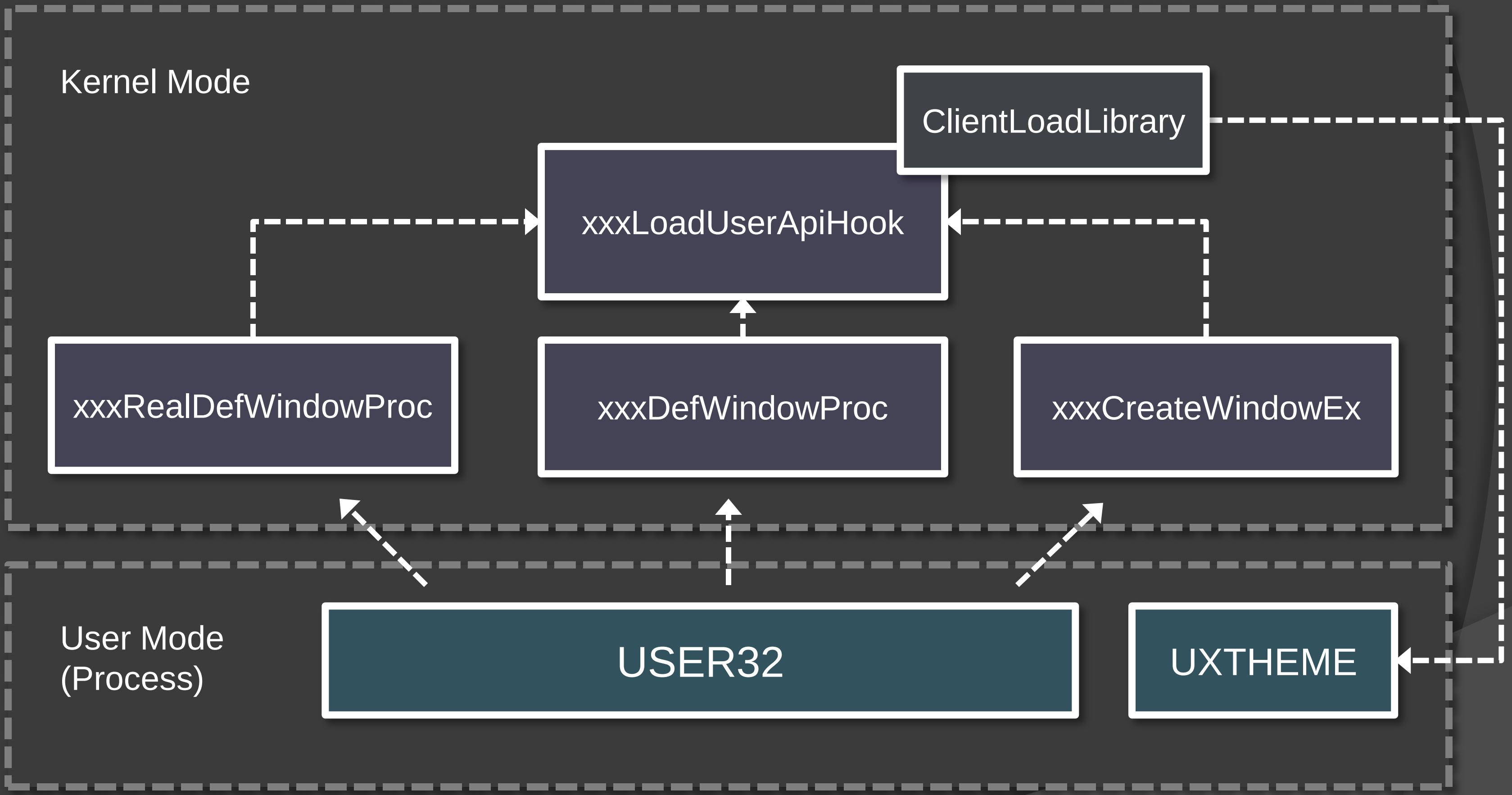
# RegisterUserApiHook

- Called by winlogon (UXINIT) to register the user api hook
  - **NtUserRegisterUserApiHook**
- Registers a string atom for the module path in the user atom table
  - Atom stored in win32k!aatomSysLoaded array
  - Array index stored in win32k!gihmodUserApiHook

# xxxLoadUserApiHook

- Retrieves the value of the UAH string atom held by **aatomSysLoaded**
  - Module (uxtheme.dll) path
- Calls **win32k!ClientLoadLibrary** to load the module in a user-mode callback
  - Client side calls **user32!InitUserApiHook** which hooks several user-mode functions
  - Subsequently called by USER32 to theme various aspects of the user interface

# UxTheme Loading



# Leveraging UxTheme

- Windows does not pin the string atom of the UxTheme library path
- An attacker could potentially free the atom and take control of the string
  - Atoms values used to perform lookups, i.e. no use-after-free of pointer values
- May cause subsequent processes to load the module of the specified string

# Plan of Attack

- Invoke an arbitrary module into a more privileged process
  - E.g. running as SYSTEM
- Requirements
  - Spawn a new (privileged) process
  - Running in the same session
  - Must invoke the USER subsystem (i.e. load user32.dll)

# System Processes

- Two SYSTEM processes in a typical user session
  - Client-Server Runtime SubSystem (CSRSS)
  - Windows Login Manager (winlogon)
- CSRSS manages the Windows subsystem
  - CSRSS and system worker threads are prevented from loading the user api hook
  - Checks in **win32k!xxxLoadUserApiHook**

# Winlogon and LogonUI

- Winlogon spawns a separate LogonUI process
  - Loads credential providers
  - Displays the Windows login interface
- Started on demand whenever Windows needs to present the login interface
- Runs on the Secure Desktop (/winlogon))
  - Only System processes can run on this desktop
  - Hence, LogonUI runs as System

# Targeting LogonUI

- Demo

Smashing the Atom

# Windows 8

# App Container

- A new application security boundary introduced in Windows 8
  - Not just specific to WinRT / metro applications
- Allows more granular access control
- Introduces the concept of capabilities
  - E.g. Internet access, music/picture/video libraries, removable storage, etc.
- Has its own namespace

# App Container Launch

- **CreateProcess** allows processes to be run in app containers
  - E.g. used by IE 10 “Enhanced Protected Mode”
- Creates a *low box* token and assigns it to the created process
  - **BasepCreateLowBox**
- Sets up the namespace directories and Global, Local, and Session symlinks
  - /Sessions/<num>/AppContainerNamedObjects/<package-sid>
  - **BasepCreateLowBoxObjectDirectories**

# Low Box Token

- The crux of the app container
- Basically an extension of the token object (nt!_TOKEN)
  - TokenFlags defines whether a token is a low box token
  - #define TOKEN_NOT_LOW 0x2000
  - #define TOKEN_LOWBOX 0x4000
- Created by the kernel using a dedicated system call
  - **NtCreateLowBoxToken**

# NtCreateLowBoxToken

- Allows applications to arbitrarily create low box tokens
- Requires a base token
  - Must not be impersonating
  - Cannot already be a low box token
- Assigns capabilities (SIDs) to a token
- References a set of handles by duplicating them into the system process
  - Guarantees that objects (i.e. namespace) stay valid for the lifetime of the token

# NtCreateLowBoxToken

```
NTAPI
NTSTATUS
NtCreateLowBoxToken(
    OUT HANDLE * LowBoxTokenHandle,
    IN HANDLE TokenHandle,
    IN ACCESS_MASK DesiredAccess,
    IN OBJECT_ATTRIBUTES * ObjectAttributes OPTIONAL,
    IN PSID PackageSid,
    IN ULONG CapabilityCount OPTIONAL,
    IN PSID_AND_ATTRIBUTES Capabilities OPTIONAL,
    IN ULONG HandleCount OPTIONAL,
    IN HANDLE * Handles OPTIONAL
    );
```

# Low Box Number Entry

- Each low box token is assigned a low box number entry
  - Creates a hard link between the token and the package sid
  - nt!_SEP_LOWBOX_NUMBER_ENTRY
- Defines the low box (app container) id
  - Unique session specific numeric identifier
  - Retrieved from the session lowbox bitmap (nt!_SESSION_LOWBOX_MAP)

# Low Box Atoms

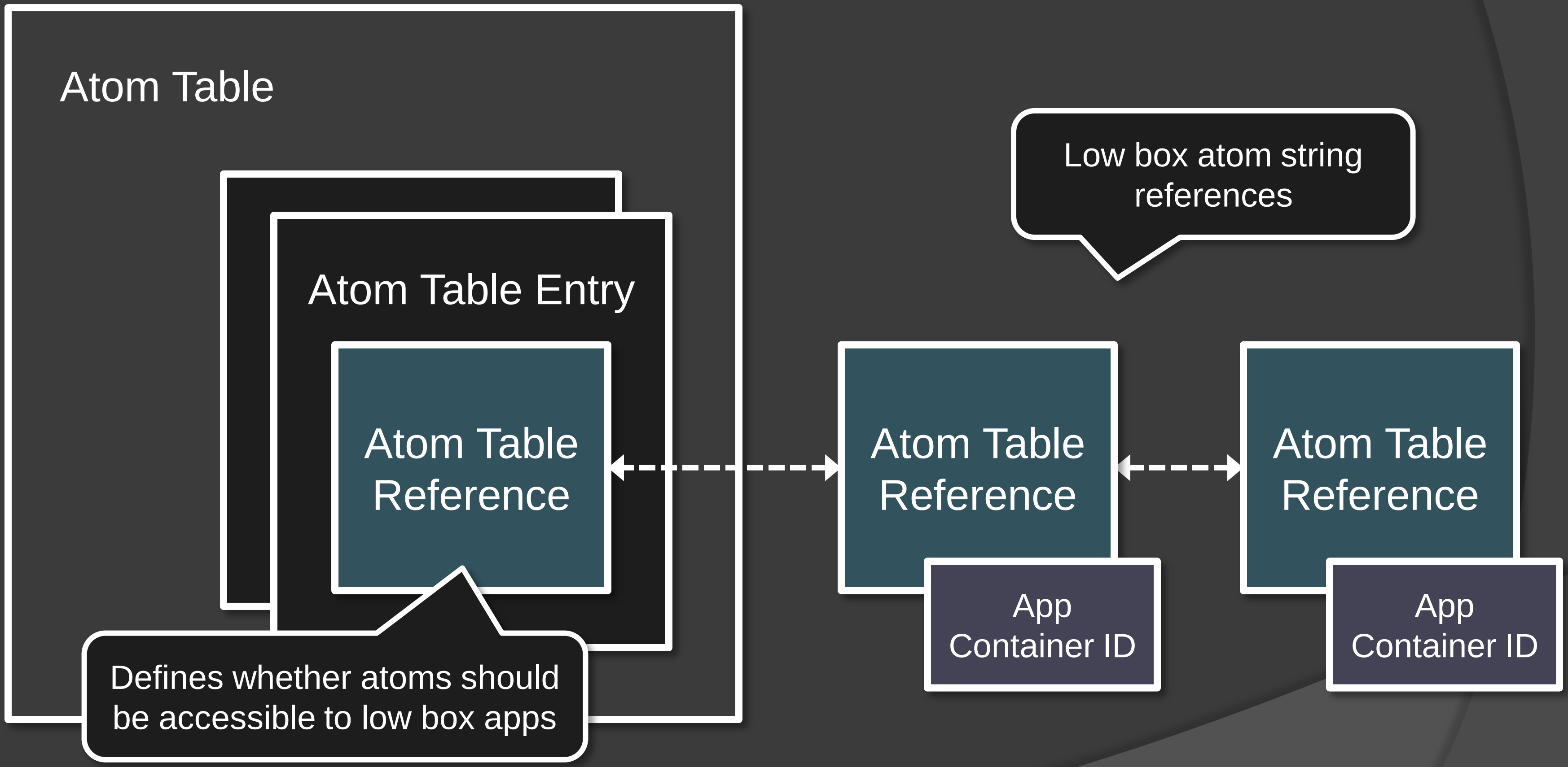
- Windows 8 introduces low box atoms
  - Implemented using a new atom table reference structure
- Allows atoms to be stored in the same table, while restricting access from other apps
- Prevents atoms from being deleted by low box (app container) applications

# Atom Reference Structure

- Embedded by the atom table entry structure
- Creates a link between the atom and the low box id
- Flags field indicates whether the atom should be shared globally
  - #define ATOM_FLAG_GLOBAL 0x2
  - Can be set using the new **AddAtomEx** API

```
kd> dt nt!_RTL_ATOM_TABLE_REFERENCE
      +0x000 LowBoxList          : _LIST_ENTRY
      +0x010 LowBoxID            : Uint4B
      +0x014 ReferenceCount      : Uint2B
      +0x016 Flags               : Uint2B
```

# Atoms in Windows 8



Low box atom string references

Defines whether atoms should be accessible to low box apps

# RtlpLookupLowBox

- Called when querying, deleting, or pinning an atom
  - Calls **RtlpQueryLowBoxId** to determine whether a low box token is active
- Returns the atom table entry if
  - The entry belongs to the current low box id
  - The entry permits access from low box apps
    - Flags & ATOM_FLAG_GLOBAL
- Can optionally override (set by argument) the entry and always deny low box access
  - Used by **RtlDeleteAtomFromAtomTable**

# Demo

- run_lowbox

Smashing the Atom

# Conclusion

# Developer Advice

- Always reference atoms on use
- Be cautious about trusting information held by the global atom table
  - Avoiding it is probably best
- Use job objects to restrict global atom table access on untrusted processes
- Windows 8: Use the low box token for added security
  - Intra-table atom access restriction

# System Hardening

- Not all kernel vulnerabilities involve semantically invalid memory access
  - Mitigations may be less effective
- OS hardening generally helps limit the impact of such vulnerabilities
- Code signing (page hashing) can address rogue module injection
  - Already used by Apple in iOS

# Thanks!

- Questions
  - @kernelpool
  - kernelpool@gmail.com
- Greetz
  - redpantz, aionescu, meder, mdowd, hzon, endrazine, msuiche, taviso, djrbliss, jono, mxatone, cesarcer, beist, ++
  - REcon

# References

- http://msdn.microsoft.com/en-us/library/windows/desktop/ms649053(v=vs.85).aspx
- http://technet.microsoft.com/en-us/security/bulletin/ms12-041