

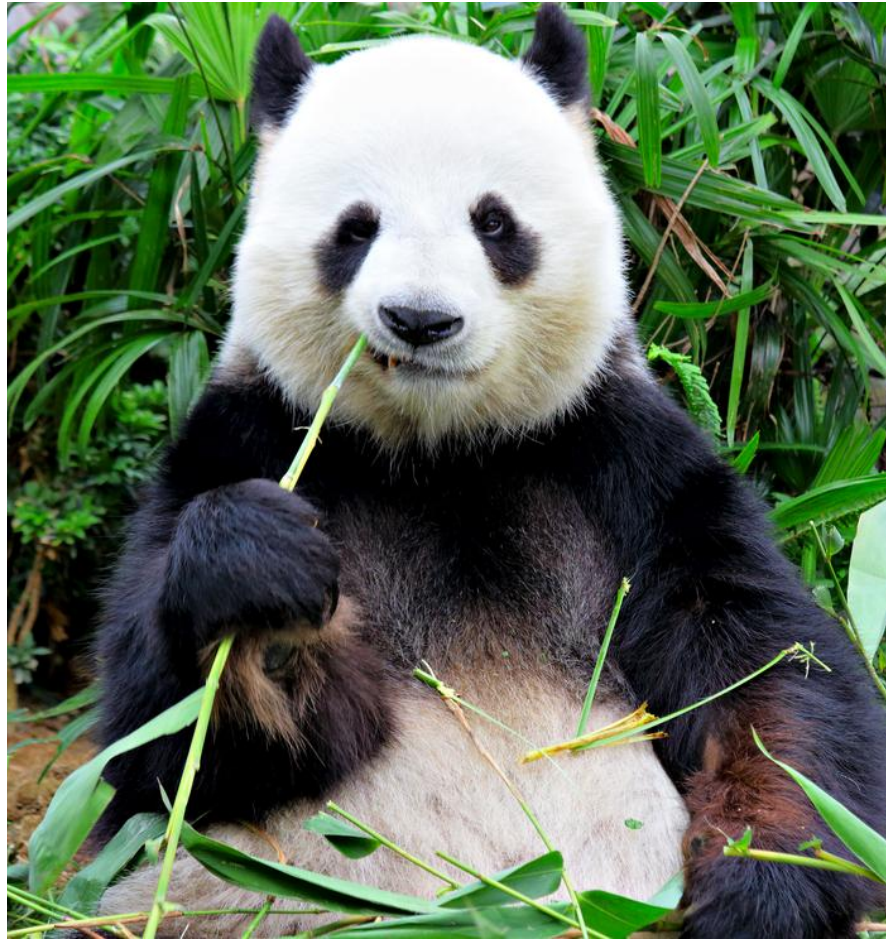


# Social Engineering the Windows Kernel

James Forshaw @tiraniddo  
Shakacon 2015

# Obligatory Background Slide

- Researcher in Google's Project Zero team
- Specialize in Windows
  - Especially local privilege escalation
- Never met a logical vulnerability I didn't like

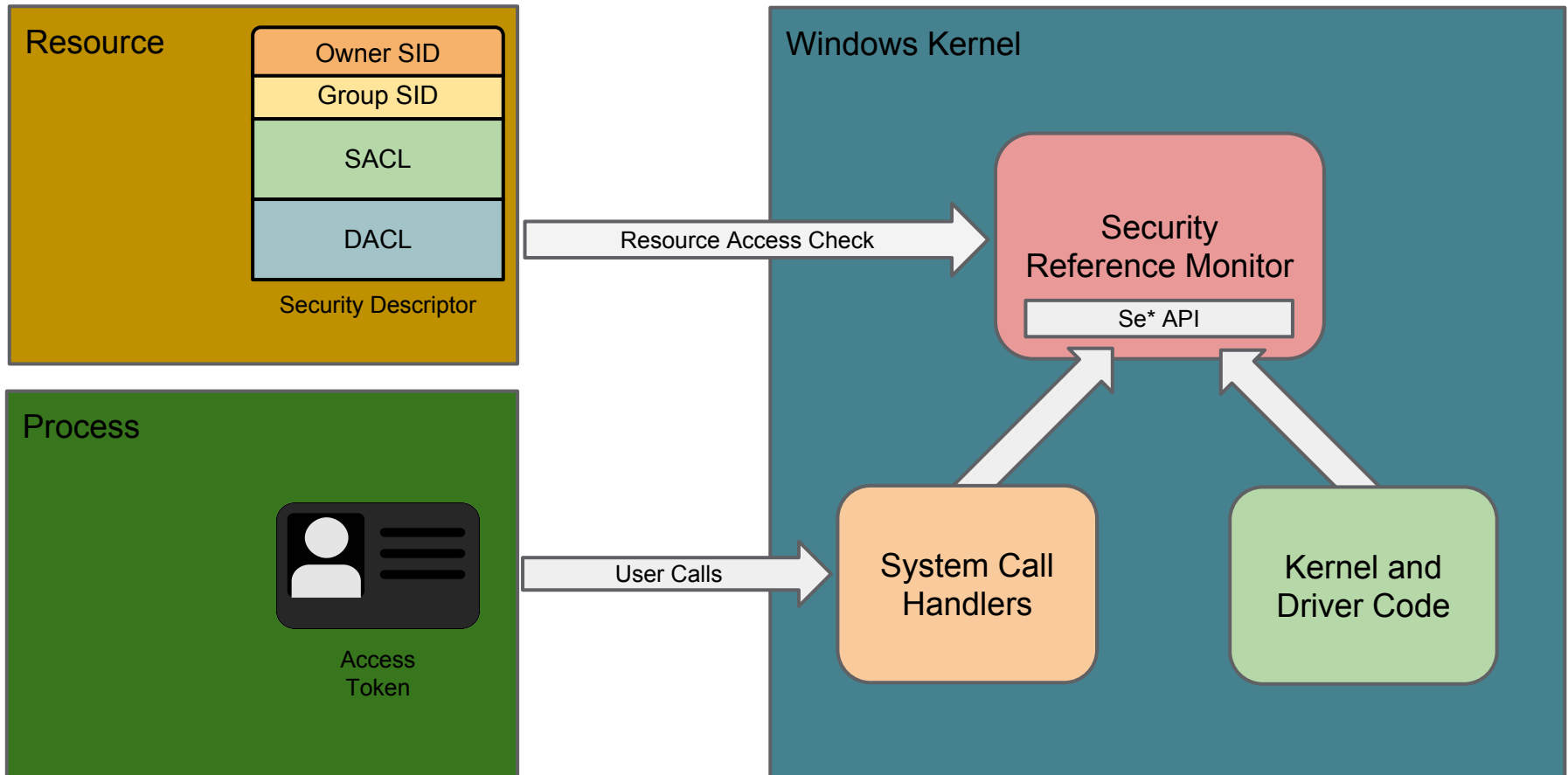


# What I'm Going to Talk About

- Windows Access Security
  - Security Reference Monitor
  - Access Tokens
  - Impersonation
- Kernel Token Handling Vulnerability Classes and Descriptions
- Kernel Token Handling Vulnerability Exploitation



# Windows Security Components



# Security Reference Monitor

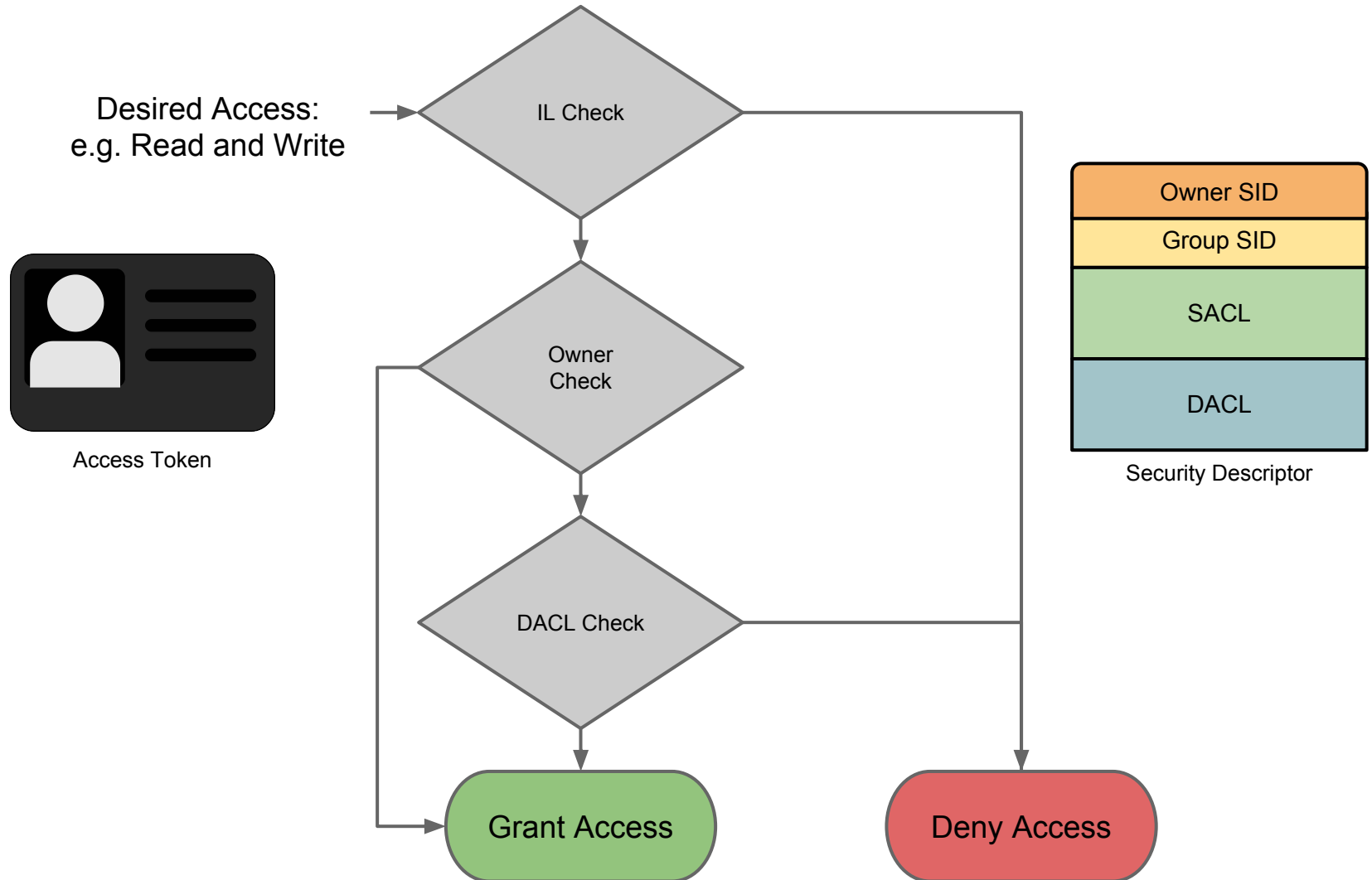
## Windows Kernel-Mode Security Reference Monitor

An increasingly important aspect of operating systems is security. Before an action can take place, the operating system must be sure that the action is not a violation of system policy. For example, a device may or may not be accessible to all requests. When creating a driver, you may want to allow some requests to succeed or fail, depending on the permission of the entity making the request.

Windows uses an access control list (ACL) to determine which objects have what security. The Windows kernel-mode security reference monitor provides routines for your driver to work with access control. For more information about the ACL, see [Access Control List](#).

Routines that provide a direct interface to the security reference monitor are prefixed with the letters "Se"; for example, `SeAccessCheck`. For a list of security reference monitor routines, see [Security Reference Monitor Routines](#).

# Security Access Check (SeAccessCheck)



# Access Token

User Security Identifier

Groups

Mandatory Label

Privileges

The screenshot shows the 'chrome.exe:2716 Properties' dialog box with the 'Security' tab selected. The 'User' field displays 'WIN-32RI1CK49EL\user' and the 'SID' field displays 'S-1-5-21-3711643808-3202222375-1035956708-1001'. The 'Groups' list includes 'BUILTIN\Administrators', 'BUILTIN\Users', 'CONSOLE LOGON', 'Everyone', 'LOCAL', 'Logon SID (S-1-5-5-0-5071873)', 'Mandatory Label\Medium Mandatory Level', and 'NT AUTHORITY\Authenticated Users'. The 'Privileges' list includes 'SeChangeNotifyPrivilege', 'SeIncreaseWorkingSetPrivilege', 'SeShutdownPrivilege', 'SeTimeZonePrivilege', and 'SeUndockPrivilege'. Red boxes highlight the 'User' and 'SID' fields, the 'Mandatory Label\Medium Mandatory Level' entry in the 'Groups' list, and the 'Privileges' list. Arrows point from the labels on the left to the corresponding elements in the dialog.

Group	Flags
BUILTIN\Administrators	Deny
BUILTIN\Users	Mandatory
CONSOLE LOGON	Mandatory
Everyone	Mandatory
LOCAL	Mandatory
Logon SID (S-1-5-5-0-5071873)	Mandatory
Mandatory Label\Medium Mandatory Level	Integrity
NT AUTHORITY\Authenticated Users	Mandatory

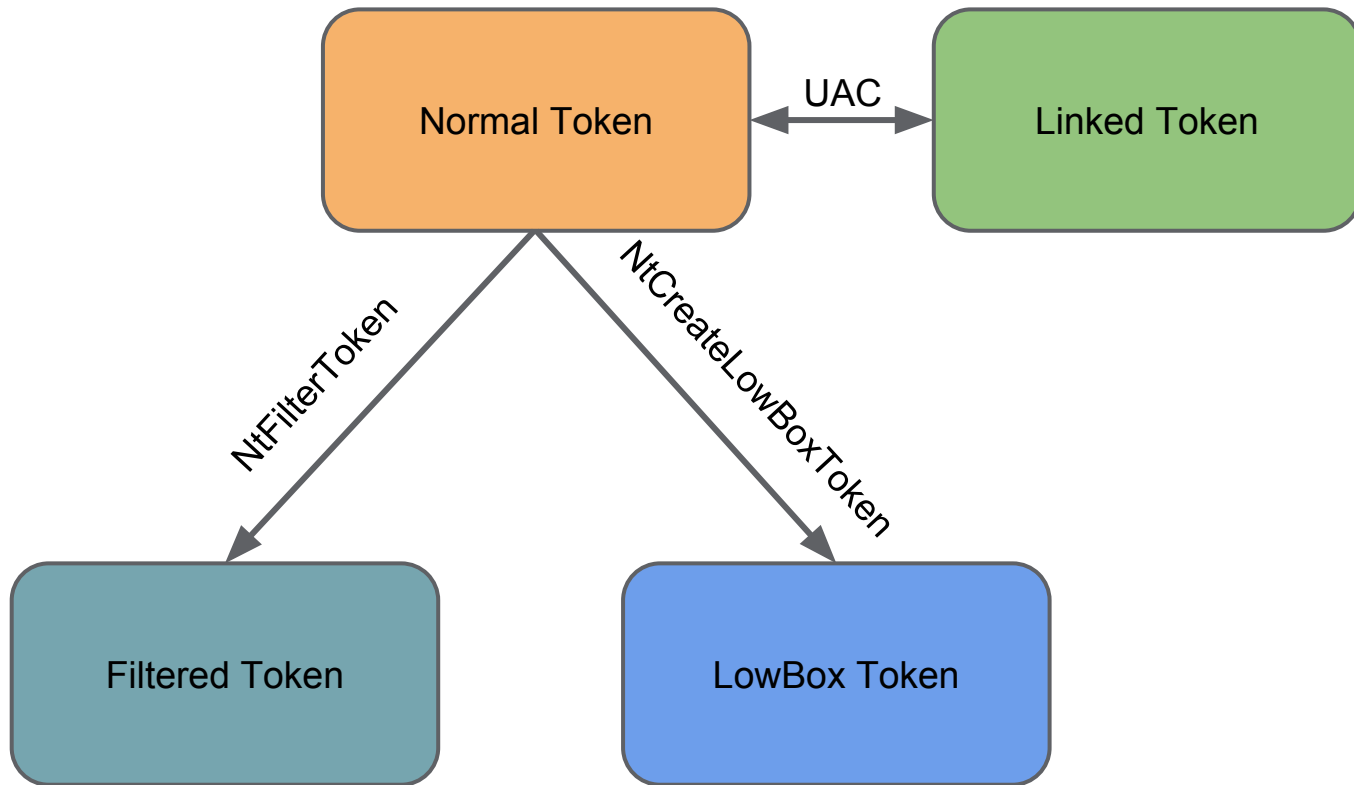
Privilege	Flags
SeChangeNotifyPrivilege	Default Enabled
SeIncreaseWorkingSetPrivilege	Disabled
SeShutdownPrivilege	Disabled
SeTimeZonePrivilege	Disabled
SeUndockPrivilege	Disabled

# Just Another Kernel Object

```
kd> dt nt!_TOKEN
+0x000 TokenSource      : _TOKEN_SOURCE
+0x010 TokenId          : _LUID
+0x018 AuthenticationId : _LUID
+0x020 ParentTokenId    : _LUID
+0x028 ExpirationTime   : _LARGE_INTEGER
+0x030 TokenLock        : Ptr32 _ERESOURCE
+0x034 ModifiedId       : _LUID
+0x040 Privileges       : _SEP_TOKEN_PRIVILEGES
+0x058 AuditPolicy      : _SEP_AUDIT_POLICY
+0x078 SessionId        : Uint4B
....
```



# Token Categories



# Important Token Fields

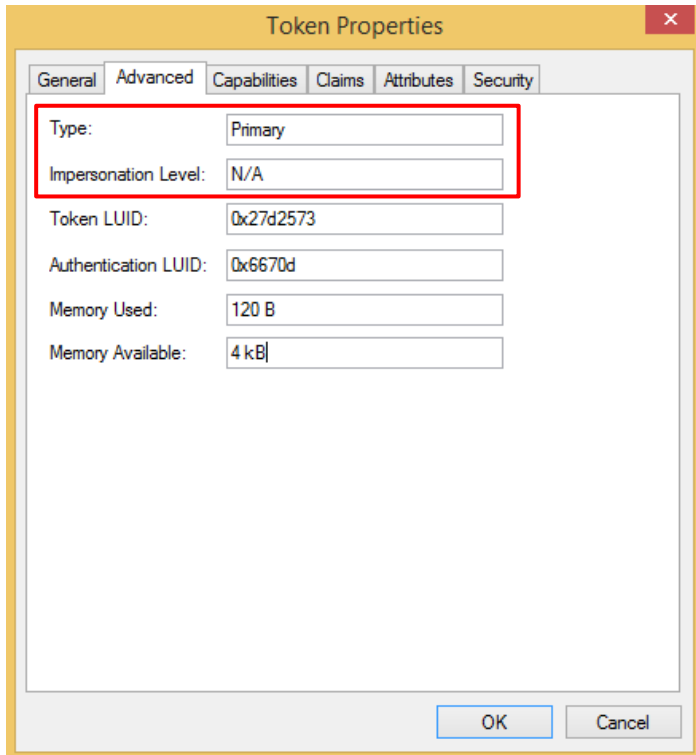
- Token User SID
- Token ID (unique 64 bit value assigned when token created)
- Parent Token ID (parent token ID when creating filtered tokens)
- Token Group SIDs
- Token Privileges
- Token Authentication ID (64 bit value ties it to a logon session)
- Token Flags (holds pre-tested privilege check values for speed)
- Restricted Token SIDs
- Lowbox Token SIDs

# Important Token Access Rights

- Tokens in user mode accessed via handles
- Can specify certain access rights based on DACL access check

TOKEN_DUPLICATE	Required to duplicate the token
TOKEN_ASSIGN_PRIMARY	Required to assign the token to a new process
TOKEN_IMPERSONATE	Required to impersonate the token
TOKEN_ADJUST_DEFAULT	Required to set the integrity level of a token
TOKEN_ADJUST_PRIVILEGES	Required to enable/disable privileges

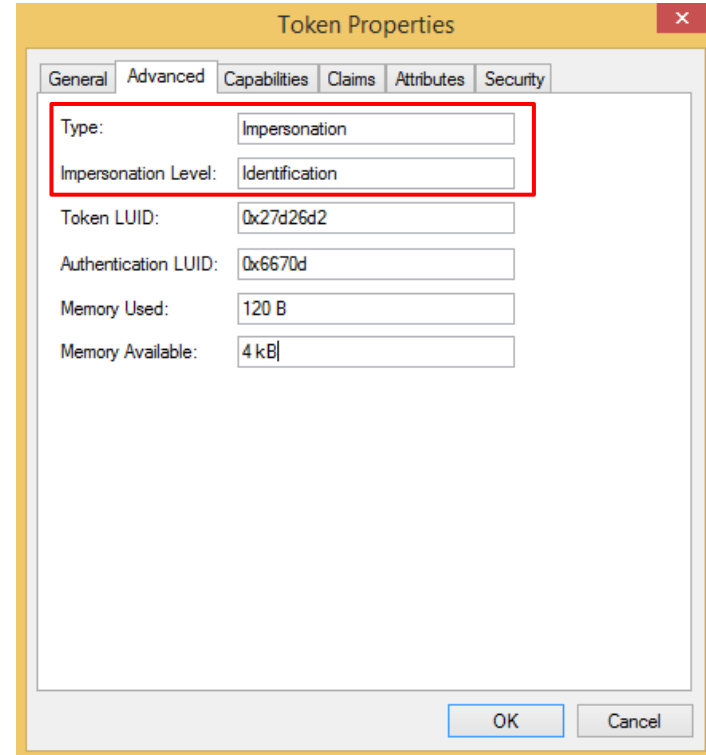
# Access Token Types



The image shows a 'Token Properties' dialog box with the 'General' tab selected. A red rectangle highlights the 'Type' and 'Impersonation Level' fields. The 'Type' field is set to 'Primary' and the 'Impersonation Level' field is set to 'N/A'. Other fields include 'Token LUID' (0x27d2573), 'Authentication LUID' (0x6670d), 'Memory Used' (120 B), and 'Memory Available' (4 kB). The 'OK' and 'Cancel' buttons are at the bottom right.

Field	Value
Type	Primary
Impersonation Level	N/A
Token LUID	0x27d2573
Authentication LUID	0x6670d
Memory Used	120 B
Memory Available	4 kB

Primary

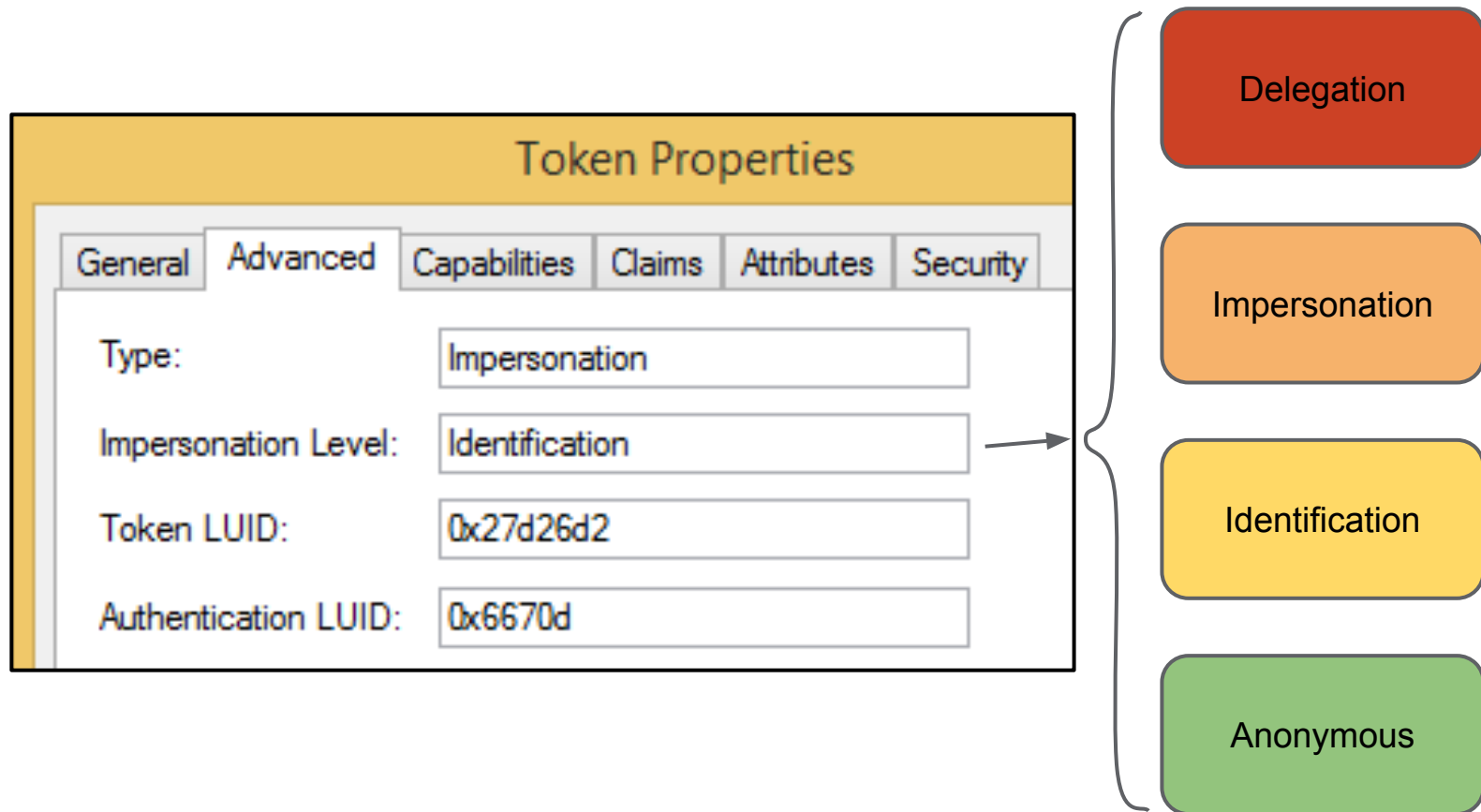


The image shows a 'Token Properties' dialog box with the 'General' tab selected. A red rectangle highlights the 'Type' and 'Impersonation Level' fields. The 'Type' field is set to 'Impersonation' and the 'Impersonation Level' field is set to 'Identification'. Other fields include 'Token LUID' (0x27d26d2), 'Authentication LUID' (0x6670d), 'Memory Used' (120 B), and 'Memory Available' (4 kB). The 'OK' and 'Cancel' buttons are at the bottom right.

Field	Value
Type	Impersonation
Impersonation Level	Identification
Token LUID	0x27d26d2
Authentication LUID	0x6670d
Memory Used	120 B
Memory Available	4 kB

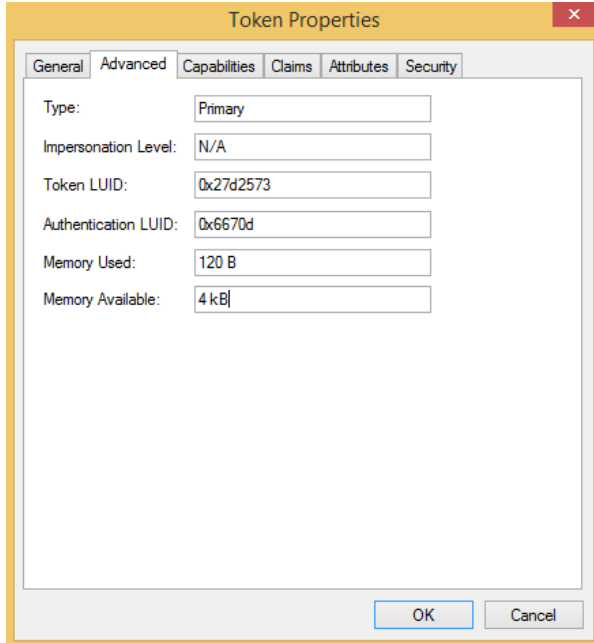
Impersonation

# Impersonation Security Level



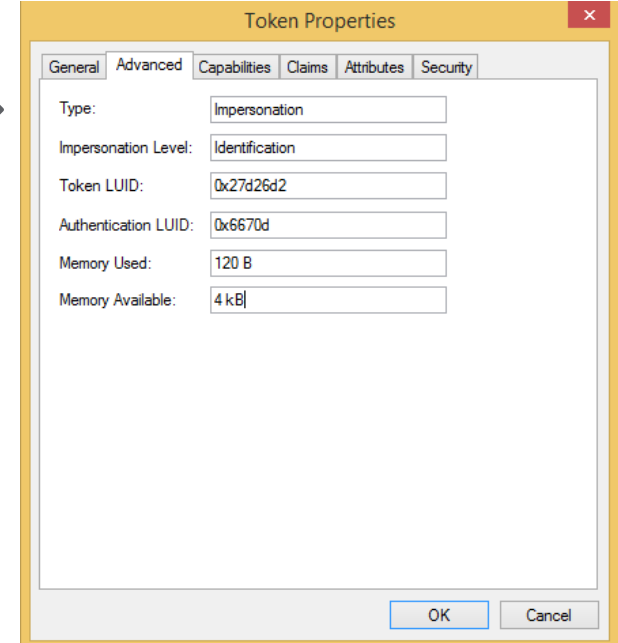
# Token Duplication

Can duplicate token's between types if have TOKEN\_DUPLICATE access on handle  
Duplicated Token can be referenced with any set of access rights



The 'Token Properties' dialog box for a Primary token. The 'General' tab is selected. The fields are: Type: Primary, Impersonation Level: N/A, Token LUID: 0x27d2573, Authentication LUID: 0x6670d, Memory Used: 120 B, and Memory Available: 4 kB. The dialog has OK and Cancel buttons at the bottom right.

Primary

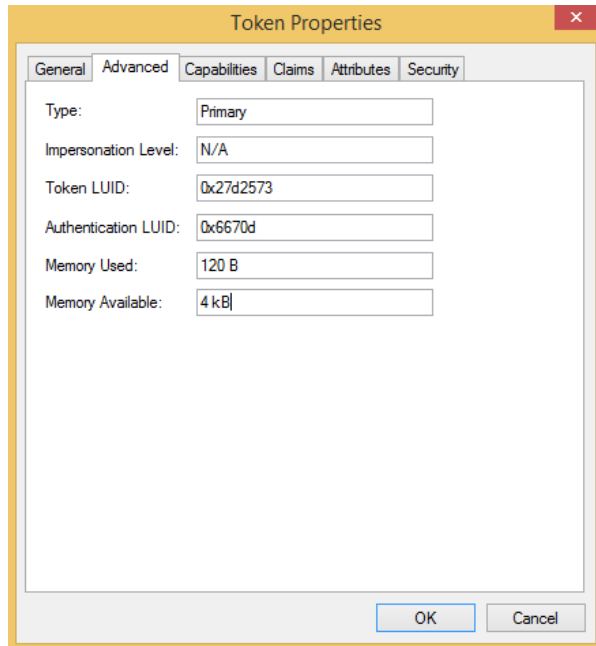


The 'Token Properties' dialog box for an Impersonation token. The 'General' tab is selected. The fields are: Type: Impersonation, Impersonation Level: Identification, Token LUID: 0x27d26d2, Authentication LUID: 0x6670d, Memory Used: 120 B, and Memory Available: 4 kB. The dialog has OK and Cancel buttons at the bottom right.

Impersonation

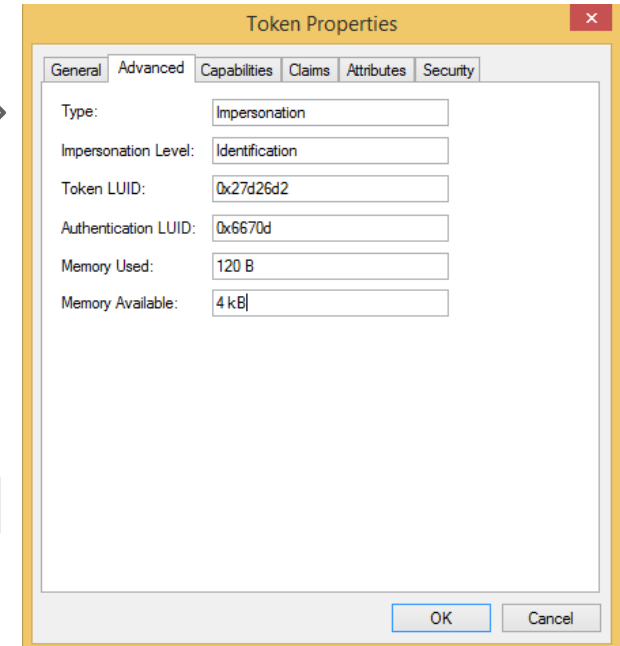
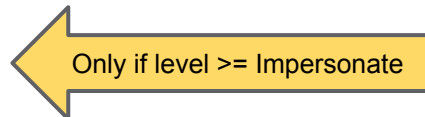
# Token Duplication

Can duplicate token's between types if have TOKEN\_DUPLICATE access on handle  
Duplicated Token can be referenced with any set of access rights



Token Properties dialog box, General tab. Fields: Type: Primary, Impersonation Level: N/A, Token LUID: 0x27d2573, Authentication LUID: 0x6670d, Memory Used: 120 B, Memory Available: 4 kB. Buttons: OK, Cancel.

Primary

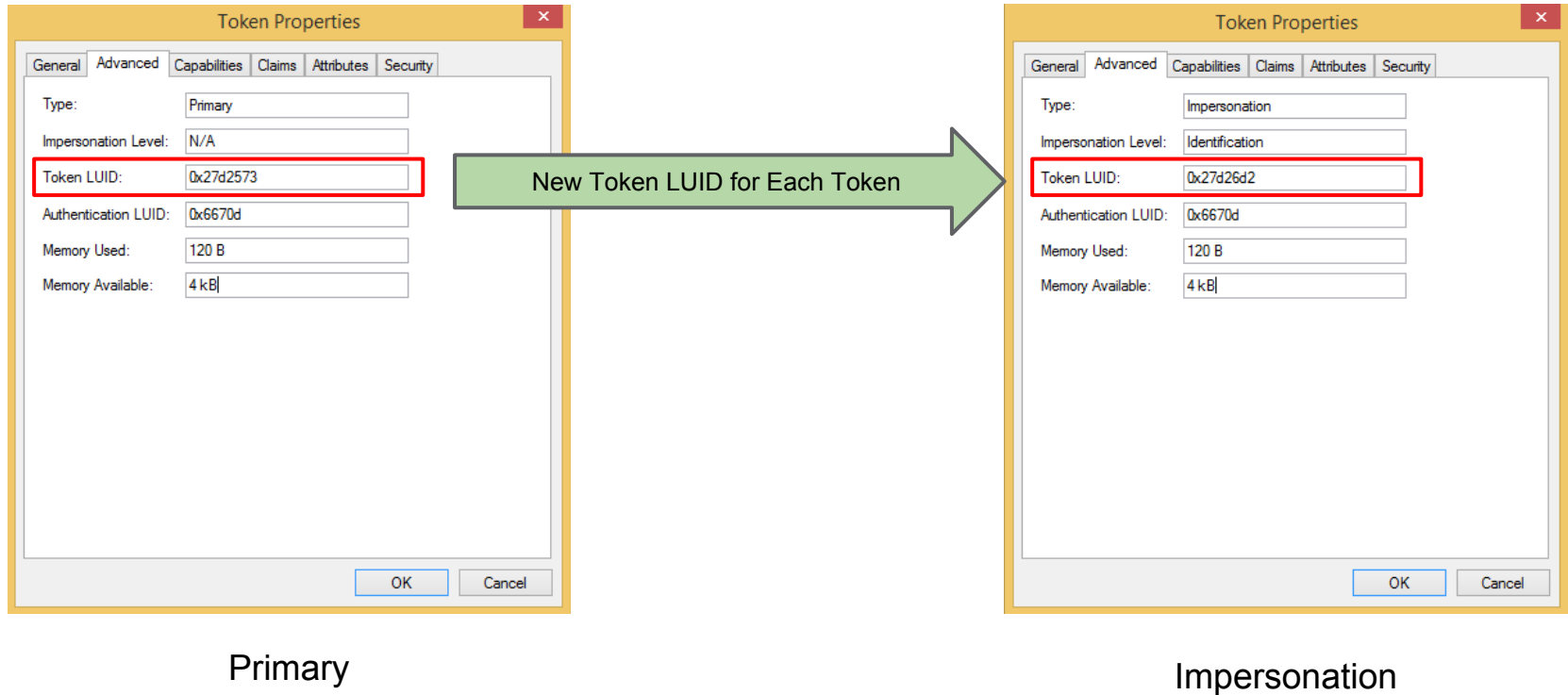


Token Properties dialog box, General tab. Fields: Type: Impersonation, Impersonation Level: Identification, Token LUID: 0x27d26d2, Authentication LUID: 0x6670d, Memory Used: 120 B, Memory Available: 4 kB. Buttons: OK, Cancel.

Impersonation

# Token Duplication

Can duplicate token's between types if have TOKEN\_DUPLICATE access on handle  
Duplicated Token can be referenced with any set of access rights





# Setting an Impersonation Token

- Impersonation assigns a token to a thread, replaced the token used in access checks for the majority of system calls

## *Direct Setting*

SetThreadToken()  
ImpersonateLoggedOnUser()  
NtSetInformationThread(...)

## *Indirect Setting*

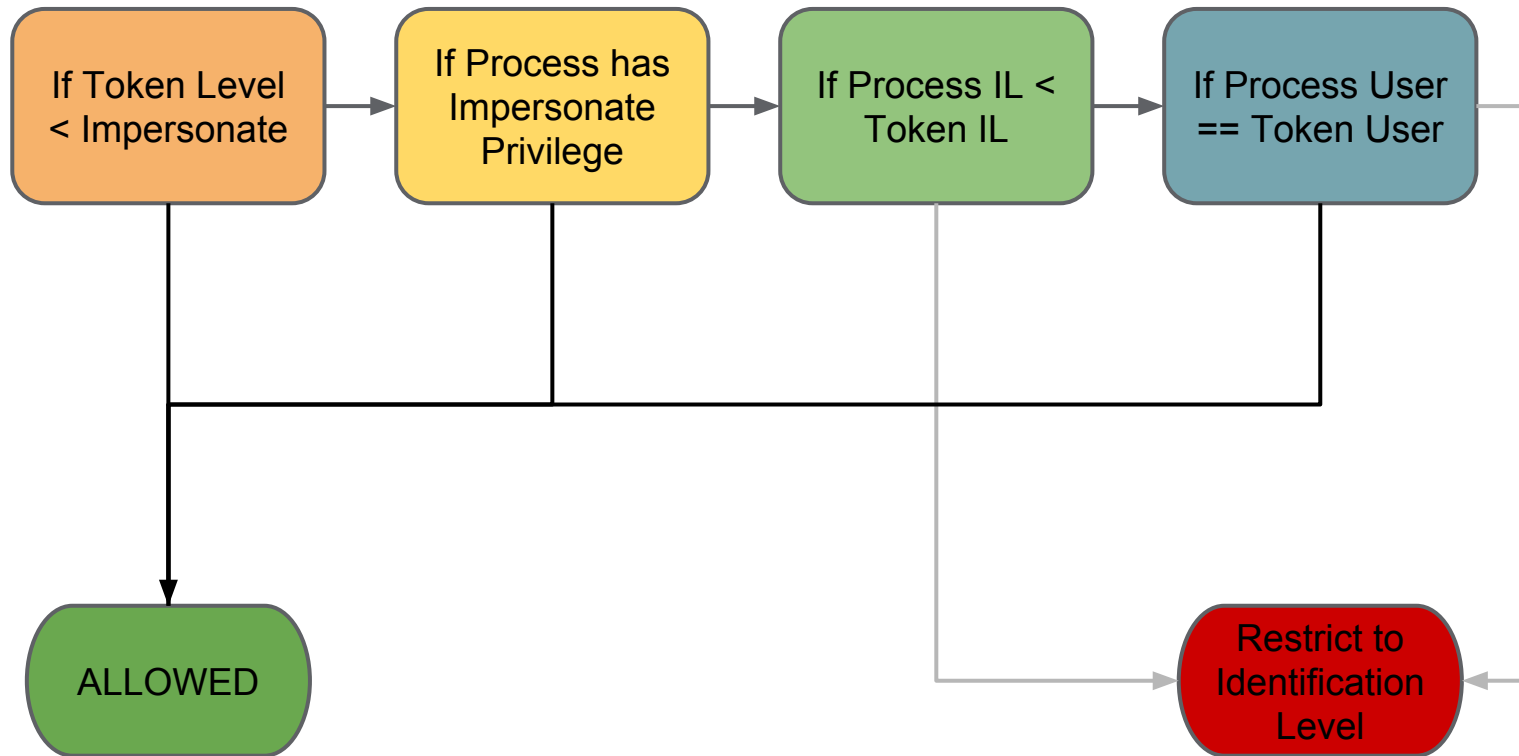
ImpersonateNamedPipeClient()  
RpcImpersonateClient()  
ColImpersonateClient()

## *Kernel Setting*

PsImpersonateClient()  
SeImpersonateClient/Ex()

# Impersonation Security

PsImpersonateClient(...) ► SeTokenCanImpersonate(...)



# NtAccessCheck versus SeAccessCheck

## *NtAccessCheck*

- User mode system call
- Takes an impersonation token handle
- Only verifies impersonation level  $\geq$  Identification

## *SeAccessCheck*

- Kernel mode only
- Takes a subject context
- Verifies impersonation level  $\geq$  Impersonation

# Identification Tokens == Fake IDs

- Would seem the biggest challenge is getting an impersonation token as a normal user
- Want to get hold of local system, or potentially arbitrary user



# Capturing Identification Tokens

# LogonUser

- Can be used to get a token if the user's password is known
- Returns a full impersonation token which is configured for the caller to impersonate with no additional privileges
- No need to play games

```
HANDLE hToken;  
LogonUser("UserName", ".", "Password",  
          LOGON32_LOGON_INTERACTIVE,  
          LOGON32_PROVIDER_DEFAULT, &hToken);  
  
ImpersonateLoggedOnUser(hToken);
```

# UAC Linked Tokens

- If split token admin get linked token using GetTokenInformation
- Without TCB privilege you only get back an identification token

```
TOKEN_LINKED_TOKEN linked_token;  
  
GetTokenInformation(hToken, TokenLinkedToken,  
                    &linked_token, ...);  
  
// Identification level token for the admin  
HANDLE hToken = linked_token.LinkationToken;
```

# Named Pipes

- Well known trick, get a system service to open a named pipe
- Call `ImpersonateNamedPipeClient` on pipe server handle
  - Normally needs pipe to be written to to get impersonation token so send over SMB (`\\localhost\pipe\mynamedpipe`)
- Get service like Anti-Virus to scan the path to capture token

```
HANDLE hPipe = CreateNamedPipe("\\\\.\\dummy", ...);  
  
// Start Windows defender to scan \\localhost\pipe\dummy  
  
ConnectNamedPipe(hPipe, NULL);  
ImpersonateNamedPipeClient(hPipe);
```



# RPC/DCOM

- Many DCOM services over RPC support impersonation
- Plenty of scope for callbacks to be made as privileged user

```
// Set callback in system COM object (e.g. BITS)

HRESULT ComCallback() {
    if(SUCCEEDED(CoImpersonateClient())) {
        HANDLE hToken;
        OpenThreadToken(GetCurrentThread(),
                        ..., &hToken);
    }
}
```

# NTLM Negotiation

- LSASS exposes APIs to do network authentication
- Can get localhost NTLM authentication using redirected WebDAV
- Creates an impersonation level token even for a normal user

```
// Start fake WebDAV service on localhost
// Init a AV scan to \\localhost\\fake\\fake
BYTE* type1 = GetType1();
BYTE* challenge;

AcceptSecurityContext(hContext type1, &challenge);
BYTE* result = SetChallengeAndGetResult(challenge);
AcceptSecurityContext(hContext, result, ...);

QuerySecurityContextToken(hContext, &hToken);
```

# Services For User (S4U)

- LsaLogonUser has an S4U mode
- At least on Windows 8.1 can get you a identification token for any local and remote user without any privileges
- Can't get local system/local service/network service tokens
- Can craft token source which might be useful for certain things

# DEMO

Capturing Access Tokens

# Vulnerability Examples

# How the Kernel Code Interacts with Tokens

*Process/Thread Manager Calls (returns a ACCESS\_TOKEN pointer)*

- PsReferencePrimaryToken
- PsReferenceImpersonationToken
- PsReferenceEffectiveToken
- SeCaptureSubjectContext

*System Calls (returns a HANDLE)*

- ZwCreateToken(Ex)
- ZwOpenProcessToken(Ex)
- ZwOpenThreadToken(Ex)
- ZwFilterToken

*Direct Access (returns a TOKEN pointer)*

- EPROCESS::Token
- ETHREAD::ClientSecurity::ImpersonationToken
  - Only valid if ETHREAD::CrossThreadFlags bit 4 is set

*IO Manager*

- IRP\_MJ\_CREATE SecurityContext

# Not Checking Impersonation Level


- When checking the token it's important kernel code verifies impersonation level

```
struct SECURITY_SUBJECT_CONTEXT {
    PACCESS_TOKEN ClientToken;
    SECURITY_IMPERSONATION_LEVEL ImpersonationLevel;
    PACCESS_TOKEN PrimaryToken;
    PVOID ProcessAuditId;
} context;

#define SeQuerySubjectContextToken(c)
    (c->ClientToken ? c->ClientToken : c->PrimaryToken)

SeCaptureSubjectContext(&context);

// Get impersonation token or primary token
PACCESS_TOKEN token = SeQuerySubjectContextToken(&context)

// Do something with token  Bad!, no check on ImpersonationLevel
```

# Variants

```
SECURITY_IMPERSONATION_LEVEL imp_level;

PACCESS_TOKEN token = PsReferenceImpersonationToken(
    PsGetCurrentThread(),
    ..., &imp_level);

if (token == NULL) {
    token = PsReferencePrimaryToken(PsGetCurrentProcess());
}
// Do something with token
```

```
SECURITY_IMPERSONATION_LEVEL imp_level;
TOKEN_TYPE token_type;

PACCESS_TOKEN token = PsReferenceEffectiveToken(
    PsGetCurrentThread(),
    &token_type,
    ..., &imp_level);


// Do something with token
```



# Real Example CVE-2015-0002

```
BOOLEAN AhcVerifyAdminContext()
{
    TOKEN_USER *user;
    SECURITY_IMPERSONATION_LEVEL ImpersonationLevel;

    PACCESS_TOKEN token = PsReferenceImpersonationToken(
        PsCurrentThread(), ...,
        &ImpersonationLevel);
    if (token == NULL) {
        token = PsReferencePrimaryToken(PsCurrentProcess());
    }
    SeQueryInformationToken(token, TokenUser, &user);

    if(RtlEqualSid(user->User->Sid, LocalSystemSid) || SeTokenIsAdmin(token)) {
        return TRUE;  Bad!, no check on ImpersonationLevel
    }

    return FALSE;
}
```

# DEMO

CVE-2015-0002 AppHelp Cache Control  
EoP

# Is SeTokenIsAdmin Also Bad?

- CVE-2015-0002 was Windows 8+ only
- Similar bug existed on Windows 7, but could also bypass SeTokenIsAdmin as well

```
// Win7
BOOLEAN SeTokenIsAdmin(PACCESS_TOKEN Token) {
    return SepSidInToken(Token, SeExports->AdminGroupSid);
}

// Win8+
BOOLEAN SeTokenIsAdmin(PACCESS_TOKEN Token) {
    if (Token->TokenType == TokenImpersonation &&
        Token->SecurityLevel < SecurityImpersonation)
        return FALSE;
    return SepSidInToken(Token, SeExports->AdminGroupSid);
}
```

# Crafted Subject Context

```
PSECURITY_DESCRIPTOR sd = ...;
PACCESS_TOKEN primary_token =
    PsReferenceEffectiveToken(...);

SECURITY_SUBJECT_CONTEXT context = {0};

context->PrimaryToken = primary_token;

if (SeAccessCheck(sd, &context, ...) {
    // Do a privileged action
}
```

# Crafted Subject Context

```
PSECURITY_DESCRIPTOR sd = ...;
PACCESS_TOKEN primary_token =
    PsReferenceEffectiveToken(...);

SECURITY_SUBJECT_CONTEXT context = {0};

context->PrimaryToken = primary_token;
```

```
if (SeAccessCheck(...,
    // Do a pri
})

BOOLEAN SeAccessCheck(...,
    PSECURITY_SUBJECT_CONTEXT context, ...,
    PNTSTATUS AccessStatus) {

    // Some initialization
    if (context.ClientToken
        && context.ImpersonationLevel < SecurityImpersonation) {
        *AccessStatus = STATUS_ACCESS_DENIED;
        return FALSE;
    }

    // Do access check with token
}
```

# Real Example RtlpAllowsLowBoxAccess

```
NTSTATUS RtlpAllowsLowBoxAccess(wchar_t* atomname) {
    SubjectSecurityContext.PrimaryToken = PsReferenceEffectiveToken(
                                                PsGetCurrentThread(),
                                                ...);

    SubjectSecurityContext.ProcessAuditId =
        PsGetCurrentProcess()->UniqueProcessId;

    BOOLEAN result = SeAccessCheckWithHint(
        SeAtomSd,
        0,
        &SubjectSecurityContext,
        ...);

    // Handle result
}
```

# System Thread Impersonation

- Impersonation usually through `SeImpersonateClient`
- Can also use `PsImpersonateClient`
- If an explicit impersonation level is passed it will override what's in the token

```
VOID DoWorkThread(LPVOID arg) {
    PACCESS_TOKEN token = arg;

    PsImpersonateClient(PsCurrentThread(), token,
                        ..., SecurityImpersonation);
}

PACCESS_TOKEN token = PsReferenceImpersonationToken(...);

PsCreateSystemThread(&hThread, ..., DoWorkThread, token);
```

# Leaky Tokens

- Kernel code can capture access token objects just like any other kernel object and manipulate it
- Bad code could return the token to user mode as a handle
  - Only requires you pass the access check
- This goes against the leakage of tokens through impersonation as you can get hold of primary tokens from other processes
- Can be used to elevate privileges in restricted token scenarios

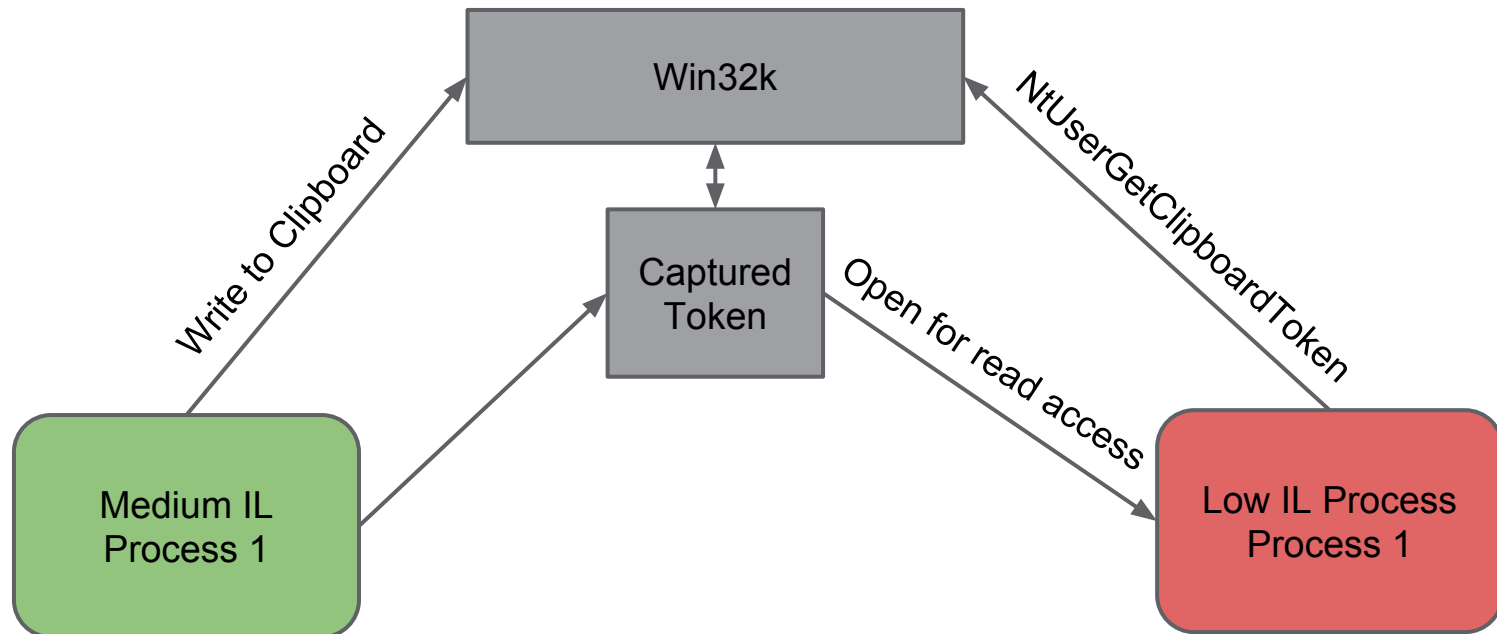
```
// Capture token in one process
PACCESS_TOKEN token = PsReferenceEffectiveToken();
CapturedAccessToken = token;

// Read in another
OpenCapturedAccessToken(PHANDLE TokenHandle, ACCESS_MASK Access) {
    ObOpenObjectByPointer(CapturedAccessToken, Access, TokenHandle);
}
```



# Real Example CVE-2015-0078

- Undocumented win32k system call NtUserGetClipboardToken introduced in Windows 8.1



# Why Is This a Problem?

- GENERIC\_READ access gives TOKEN\_DUPLICATE access right
- Duplicate to get a writable token then lower IL level
- As long as the same user allows us to assign the token.

```
TOKEN_MANDATORY_LABEL TokenMandatoryLabel;
```

```
TokenMandatoryLabel.Label.Attributes = SE_GROUP_INTEGRITY;  
TokenMandatoryLabel.Label.Sid = MediumMandatoryLevelSid;
```

```
SetTokenInformation(hNewToken,  
    TokenIntegrityLevel,  
    &TokenMandatoryLabel,  
    sizeof(TOKEN_MANDATORY_LABEL));
```

# DEMO

CVE-2015-0078  
Win32k Leaky Tokens

# Incorrect Token Duplication

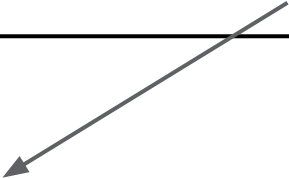
- External duplication APIs, NtDuplicateToken and SeDuplicateToken check for impersonation level < Impersonate
- Kernel code can also use internal duplication function SepDuplicatetoken
- No checks done on the original token that it's compatible for duplication

# Real Example CVE-2015-0078

- CreateProcessAsUser takes a token handle to assign as primary token
- Documented as requiring a Primary Token

## Parameters

*hToken* [in, optional]



A handle to the primary token that represents a user. The handle must have the **TOKEN\_QUERY**, **TOKEN\_DUPLICATE**, and **TOKEN\_ASSIGN\_PRIMARY** access rights. For more information, see [Access Rights for Access-Token Objects](#). The user represented by the token must have read and execute access to the application specified by the *lpApplicationName* or the *lpCommandLine* parameter.

To get a primary token that represents the specified user, call the [LogonUser](#) function. Alternatively, you can call the [DuplicateTokenEx](#) function to convert an impersonation token into a primary token. This allows a server application that is impersonating a client to create a process that has the security context of the client.

# Digging into NtCreateUserProcess

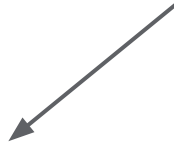
```
NTSTATUS PspReferenceTokenForNewProcess(  
    HANDLE token,  
    KPROCESSOR_MODE AccessMode,  
    TOKEN **outtok)  
{  
    result = ObReferenceObjectByHandle(token,  
        TOKEN_ASSIGN_PRIMARY,  
        SeTokenObjectType,  
        AccessMode, outtok);  
  
    return result;  
}
```

Only requires TOKEN\_ASSIGN\_PRIMARY privileges

# Token Creation

```
NSTATUS SeSubProcessToken(_EPROCESS *process,  
                        _TOKEN *token,  
                        _TOKEN **subtoken)  
{  
    OBJECT_ATTRIBUTES objattr;  
  
    InitializeObjectAttributes(&objattr);  
  
    SepDuplicateToken(token, &objattr, TokenPrimary,  
                    SecurityAnonymous, subtoken);  
  
    // Do rest of initialization  
}
```

Explicitly Duplicate as Primary



# SeAssignPrimaryTokenPrivilege

- Limit to general exploitability, caller process must have privilege or meet a set of criteria to assign token
- System services (such as IIS) have privilege so can be used as an EoP there, but what about user processes?

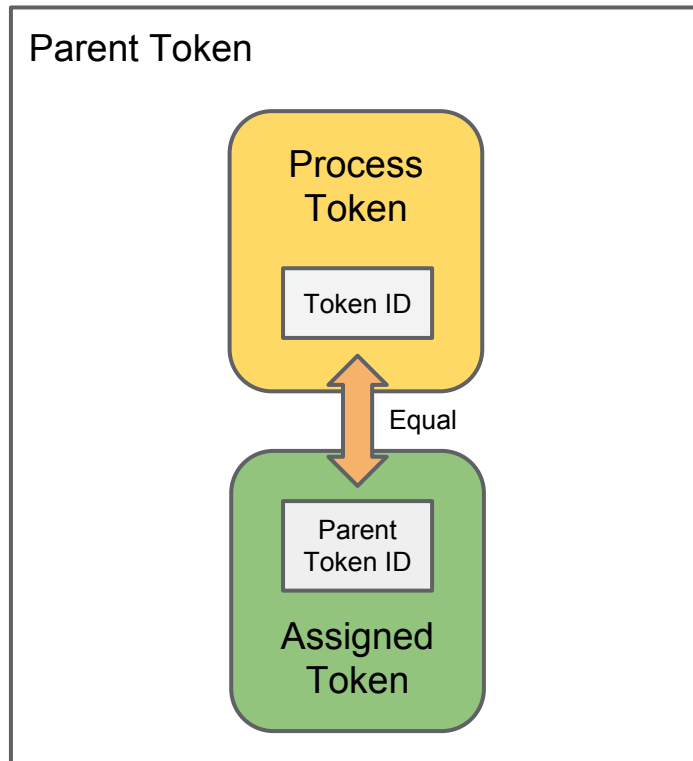
```
BOOLEAN has_assign_privilege = FALSE;
if (SeSinglePrivilegeCheck(SeAssignPrimaryTokenPrivilege))
    has_assign_privilege = TRUE;

BOOLEAN can_assign;
SeIsTokenAssignableToProcess(tokena, &can_assign);

if ( !can_assign && !has_assign_privilege )
    return STATUS_PRIVILEGE_NOT_HELD;
```

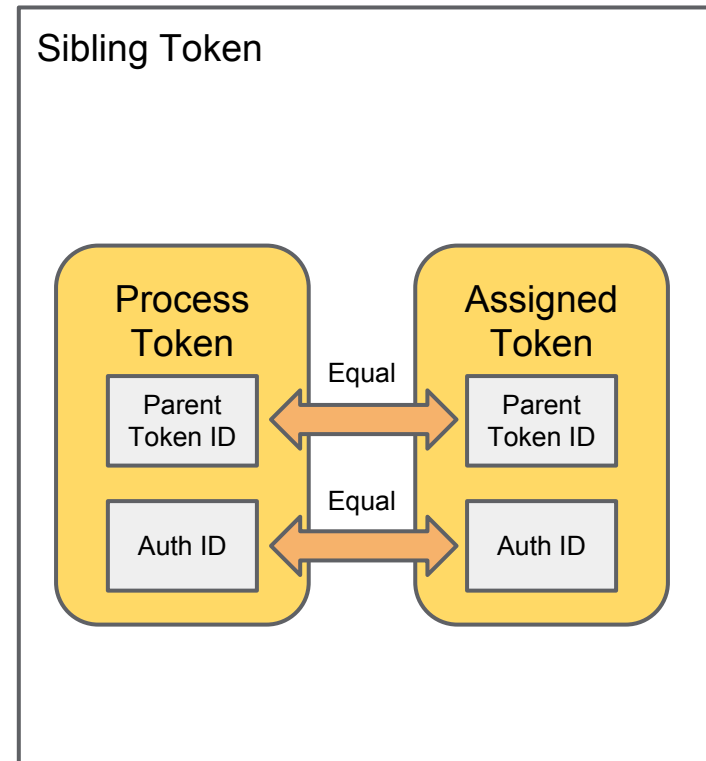


# SelsTokenAssignableToProcess



CreateRestrictedToken

OR



DuplicateToken  
NtCreateLowBoxToken

# LowBox Token Hierarchy

- LowBox Tokens used in things like IE EPM are siblings of the main user's token.
- Using this issue can capture an identification token from main user and use that to create a new process with elevated privileges
- Only restriction is must be at same IL, which for IE EPM is Low

# DEMO

CVE-2015-0078  
NtCreateUserProcess

# TOCTOU Issues

- Some bugs need to change the token between one call and the next
- Can be done from another thread
- Useful to bypass lower SeAccessCheck then subsequent additional checks

```
void DoUserAction()
{
    if (SeAccessCheck(...)) {
        PACCESS_TOKEN token = PsReferenceEffectiveToken();

        PSID sid = GetUserSid(token);

        // Do operations as that use
    }
}
```

# IO Manager TOCTOU

- IO Manager Only Captures Security during IRP\_MJ\_CREATE processing
- No subsequent IO calls are explicitly checked, so sometimes drivers do it manually

```
void UseDriver()
{
    // Security check made here
    HANDLE hDevice = CreateFile("\\\\.\\DeviceName", ...);

    ImpersonateLoggedOnUser(hFakeId);

    // No further security checks
    DeviceIoControl(hDevice, ...);
}
```

# Real Example CVE-2015-0011

- MRXDAV driver implements kernel mode component of WebDAV UNC path support
- Accessible through \Device\WebDavRedirector device, which does a security check
- Functionality implemented in Device IO Control commands, which does the following:

```
// Do Device IO Control
SeCaptureSubjectContext(&ctx)
SeQueryAuthenticationId(SeQuerySubjectContextToken(&ctx),
                        &luid);
if(luid == LocalSystemLUID)
    is_local_system = true;

if (is_local_system)
    // Do privileged operation
```

# Vulnerable Zw Calls

- Calls to ZwOpenThreadToken and ZwOpenProcessToken are deprecated, replaced with Ex Variants.
- This is because these calls were supposed to only be used from user mode and not the kernel
  - Didn't specify a handles flag option
  - Meant all handles were opened as user-mode handles
  - Potential for a user-mode process to get access to privileged handles

# Changes in Token Handling



# SeTokenIsAdmin Fixed in Windows 7

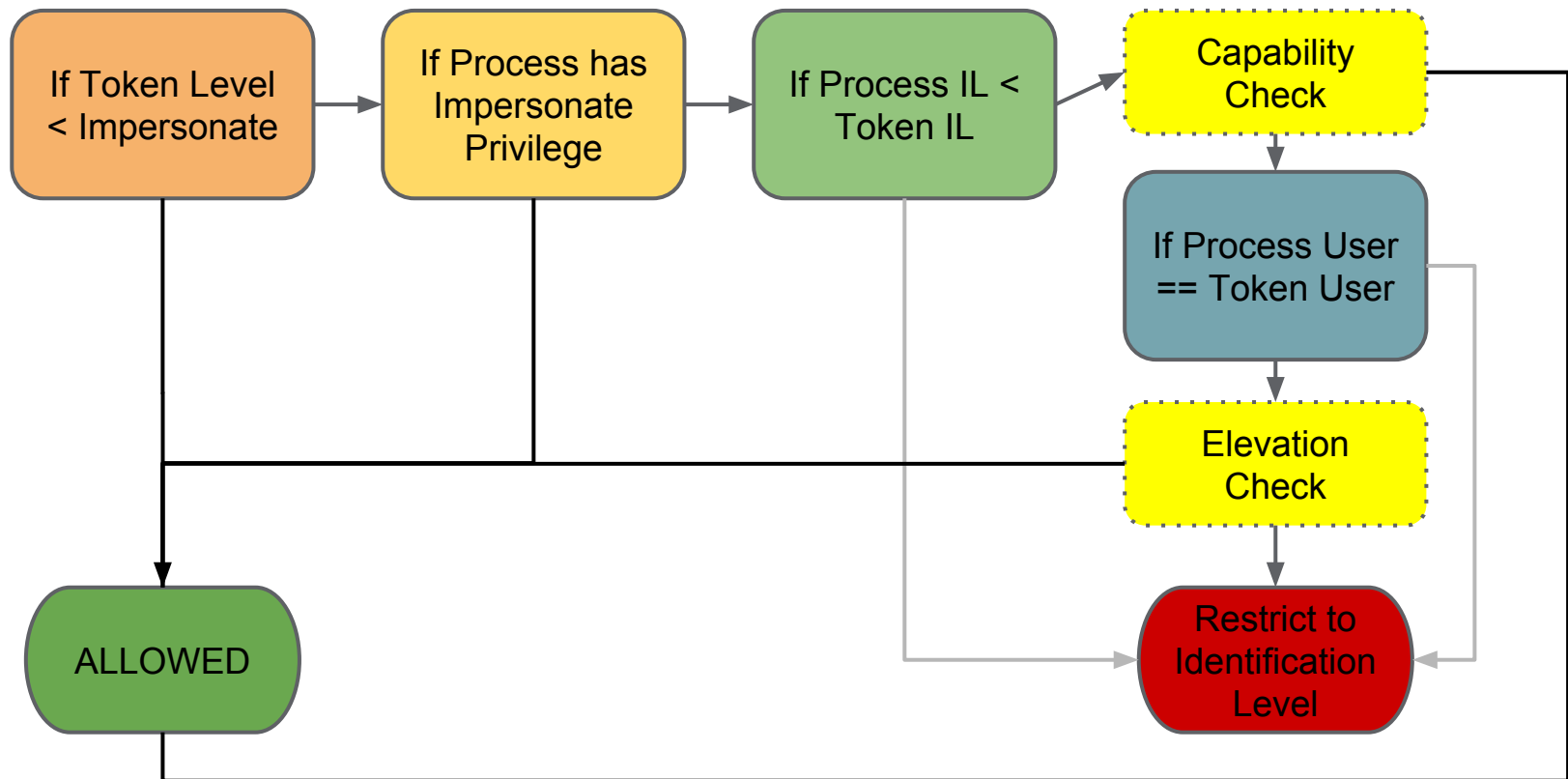
## **Impersonation Level Check Elevation of Privilege Vulnerability – CVE-2015-0075**

An elevation of privilege vulnerability exists when Windows fails to properly validate and enforce impersonation levels. An attacker who successfully exploited this vulnerability could bypass user account checks to gain elevated privileges.

To exploit this vulnerability, an attacker would first have to log on to the system. An attacker could then run a specially crafted application designed to increase privileges. The update addresses the vulnerability by correcting how Windows validates impersonation levels.

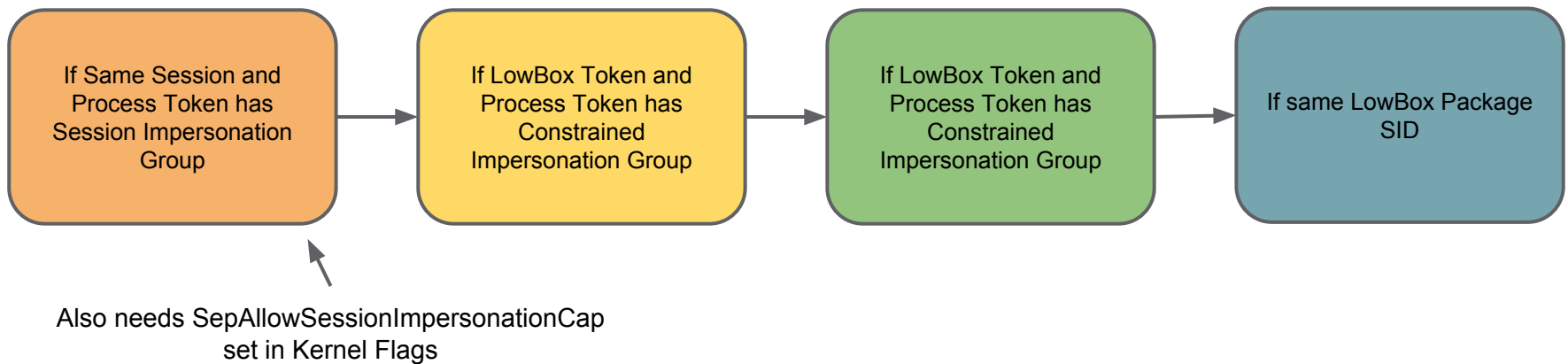
Microsoft received information about this vulnerability through coordinated vulnerability disclosure. When this security bulletin was originally issued Microsoft had not received any information to indicate that this vulnerability had been publicly used to attack customers.

# Windows 10 Changes (Build 10159)



# Windows 10 Impersonation Capability

- New Group and Capability SIDs to add impersonation capability
  - Session Impersonation Group
  - Constrained Impersonation Group
  - Constrained Impersonation Capability



# Windows 10 Elevated Token Impersonation

- Blocks impersonating an elevated token unless process token is also elevated
- Must be enabled in SeCompatFlags kernel flag
- Blocks trick used to elevate privileges with NtUserGetClipboardToken

```
if (SeTokenIsElevated(ImpersonationToken)) {  
    if ((SeCompatFlags & 1)  
        && !SeTokenIsElevated(ProcessToken)) {  
        return STATUS_PRIVILEGE_NOT_HELD;  
    }  
}
```

# Questions?