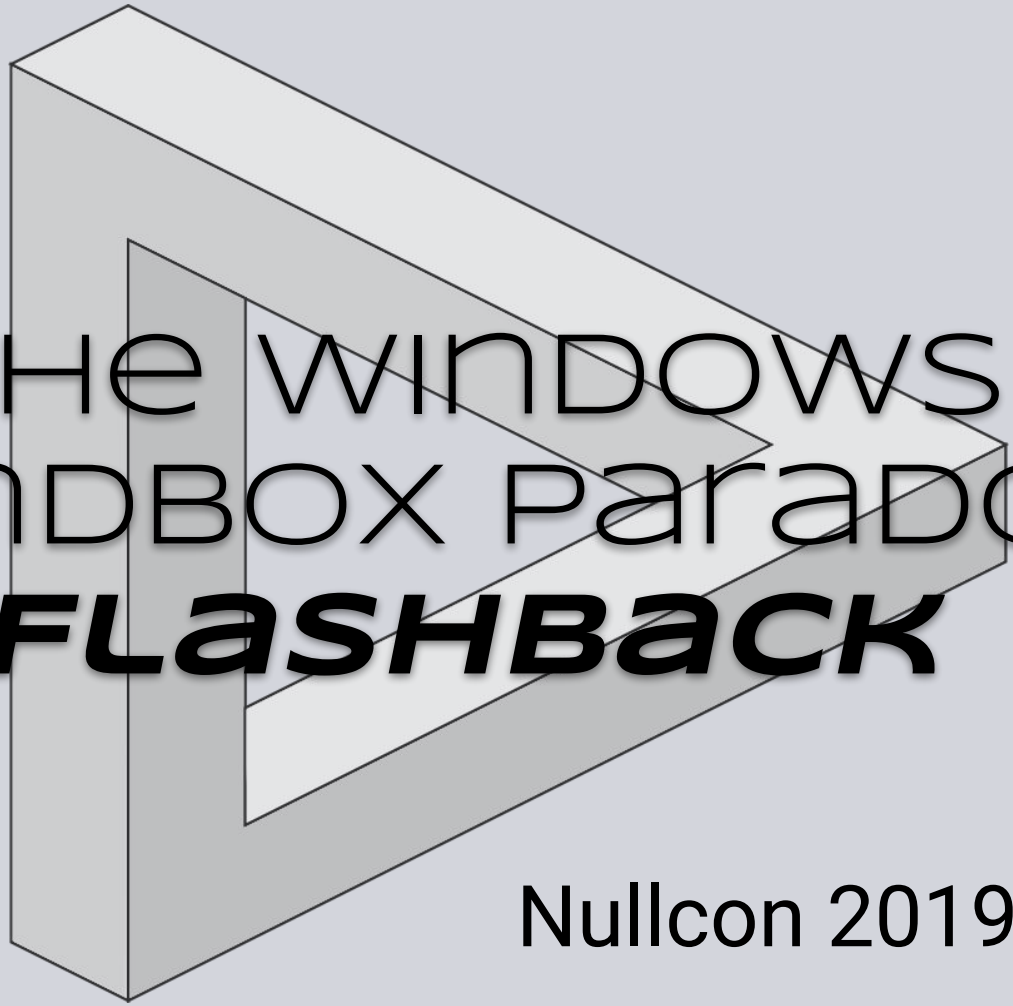


THE WINDOWS SANDBOX PARADOX

FLASHBACK

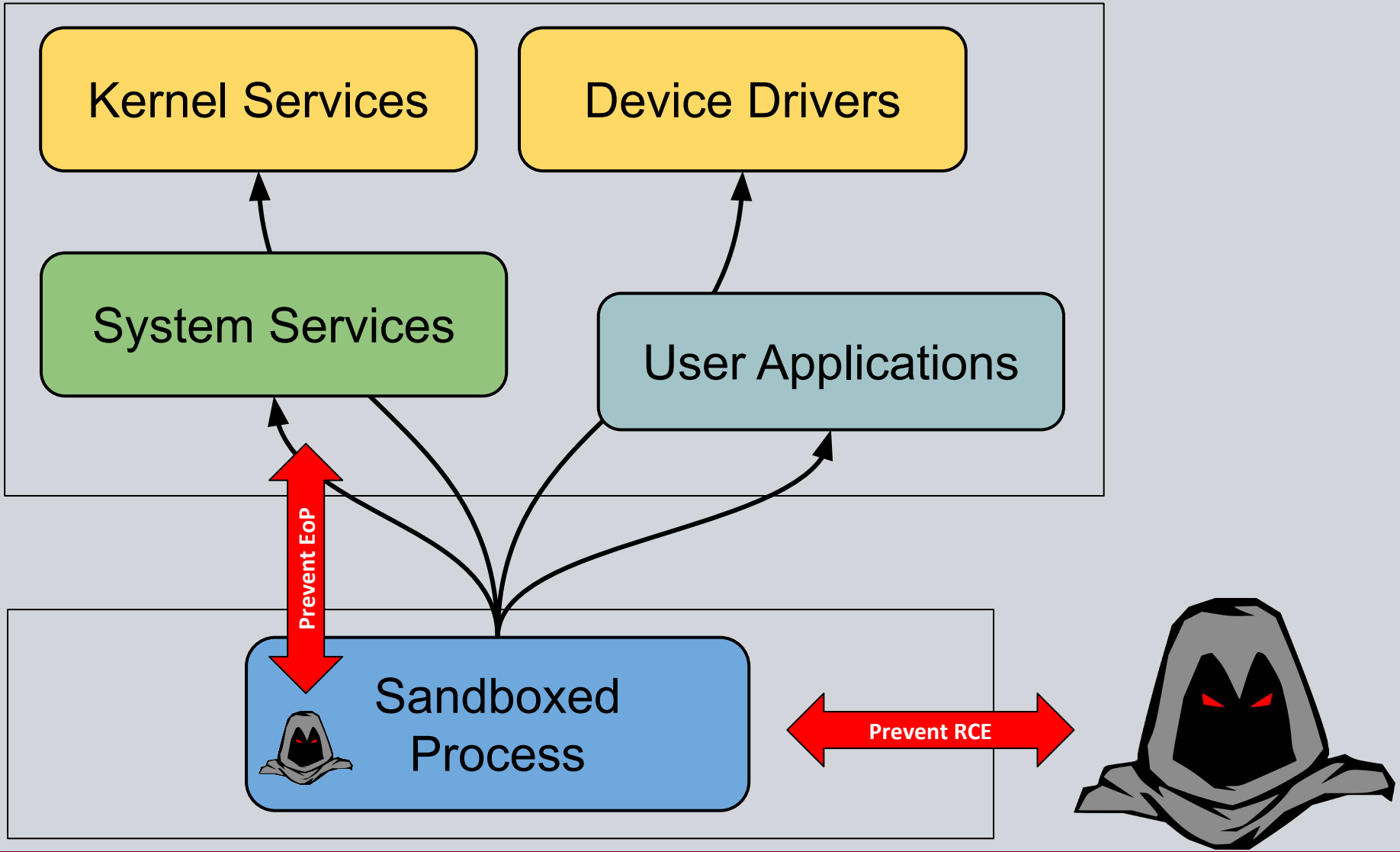
Nullcon 2019

James Forshaw @tiraniddo

Obligatory Background Slide

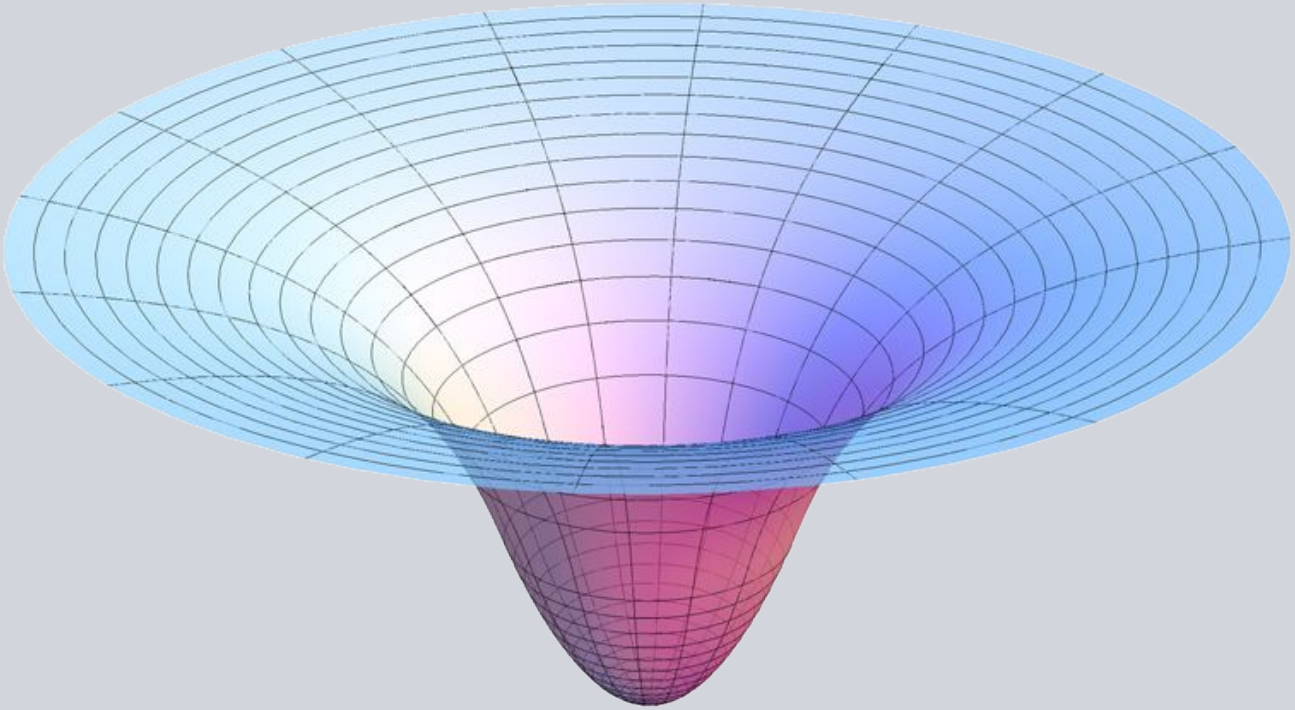
- Founder Member of Google's Project Zero
- 10+ Years of Windows Security Research
- Logical vulnerability specialist
- Chromium Windows Sandbox Owner
- "Attacking Network Protocols" Author
- @tiraniddo on Twitter.

What I'm Going to Talk About



Sandboxing Requirement #1

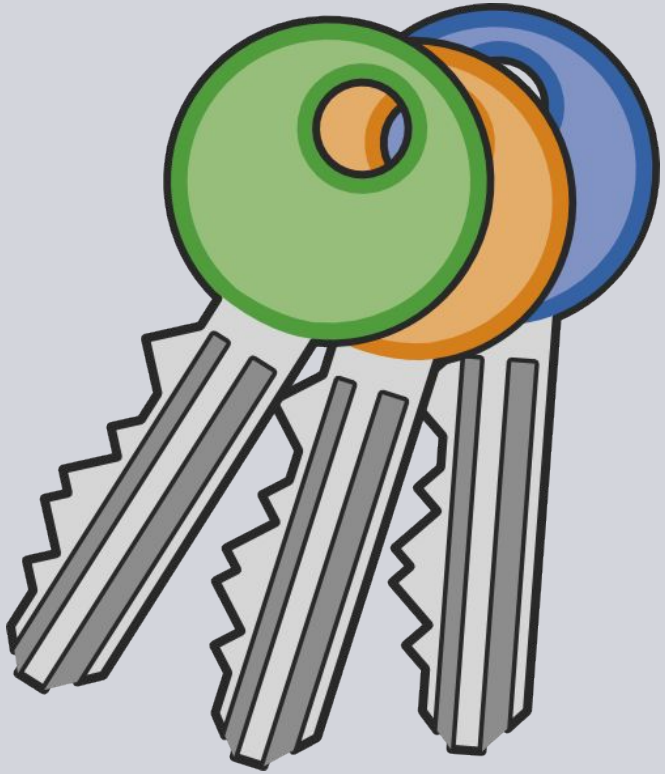
- Easy to get in, hard to get out



http://upload.wikimedia.org/wikipedia/commons/d/d9/GravityPotential.jpg

Sandboxing Requirement #2

- Protects the user's data from disclosure



https://openclipart.org/detail/190821/cles-de-serrure---lock-keys-by-enolynn-190821

Sandboxing Requirement #3

- Work within the limits of the OS

http://upload.wikimedia.org/wikipedia/commons/8/8b/MUTCD_R2-1.svg

Sandboxing Requirement #4

- Sandboxed application is usable

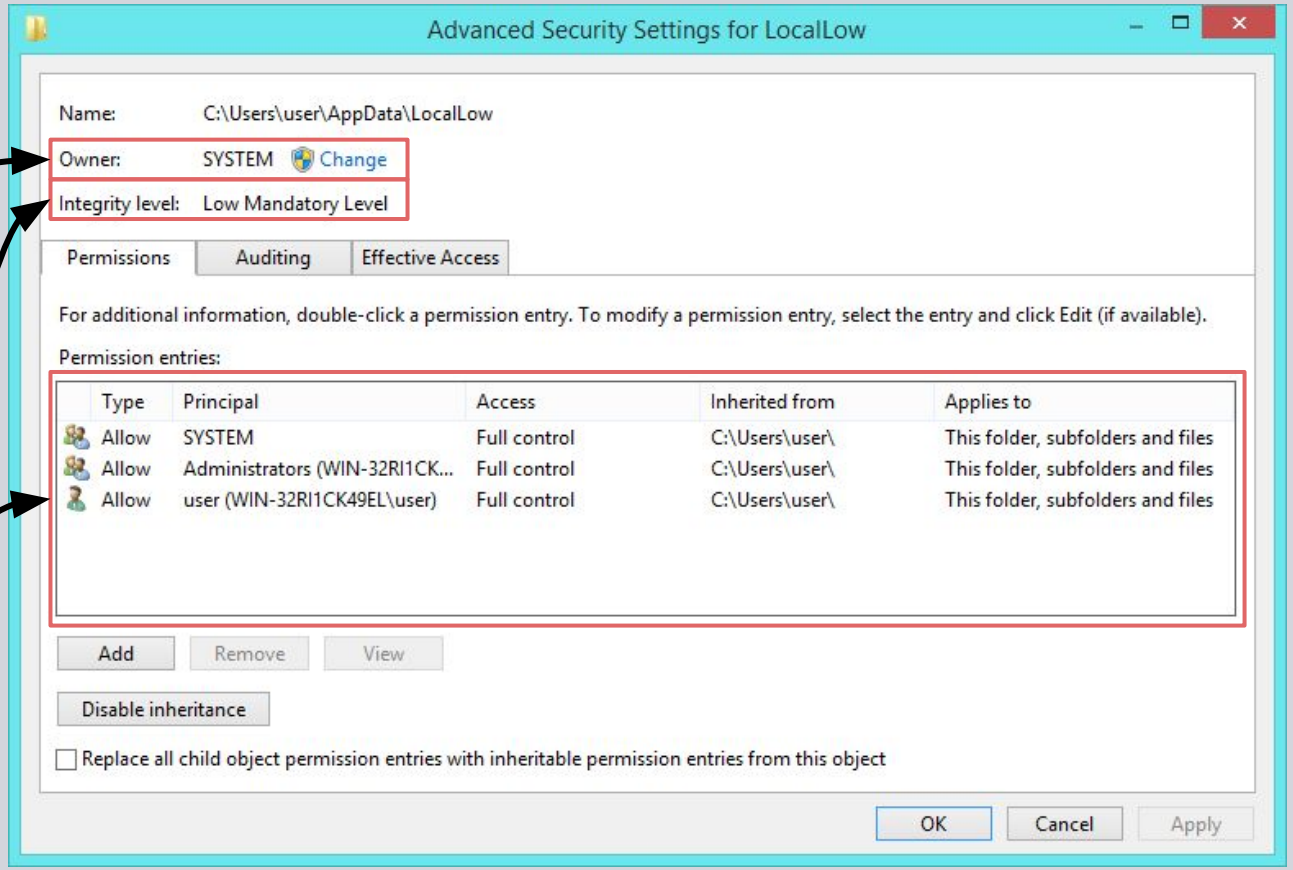
http://pixabay.com/p-305189/

Resource Security Descriptor

Owner of Secured Resource

Mandatory Integrity Label

Discretionary Access Control List (DACL)



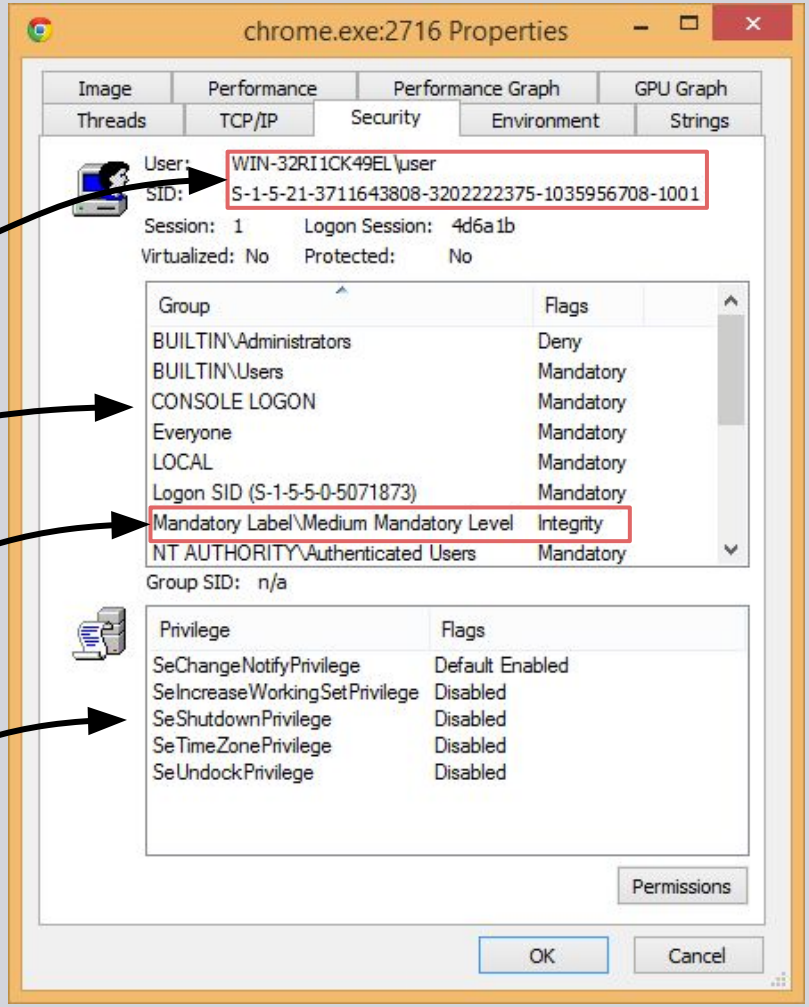
Access Tokens

User Security Identifier

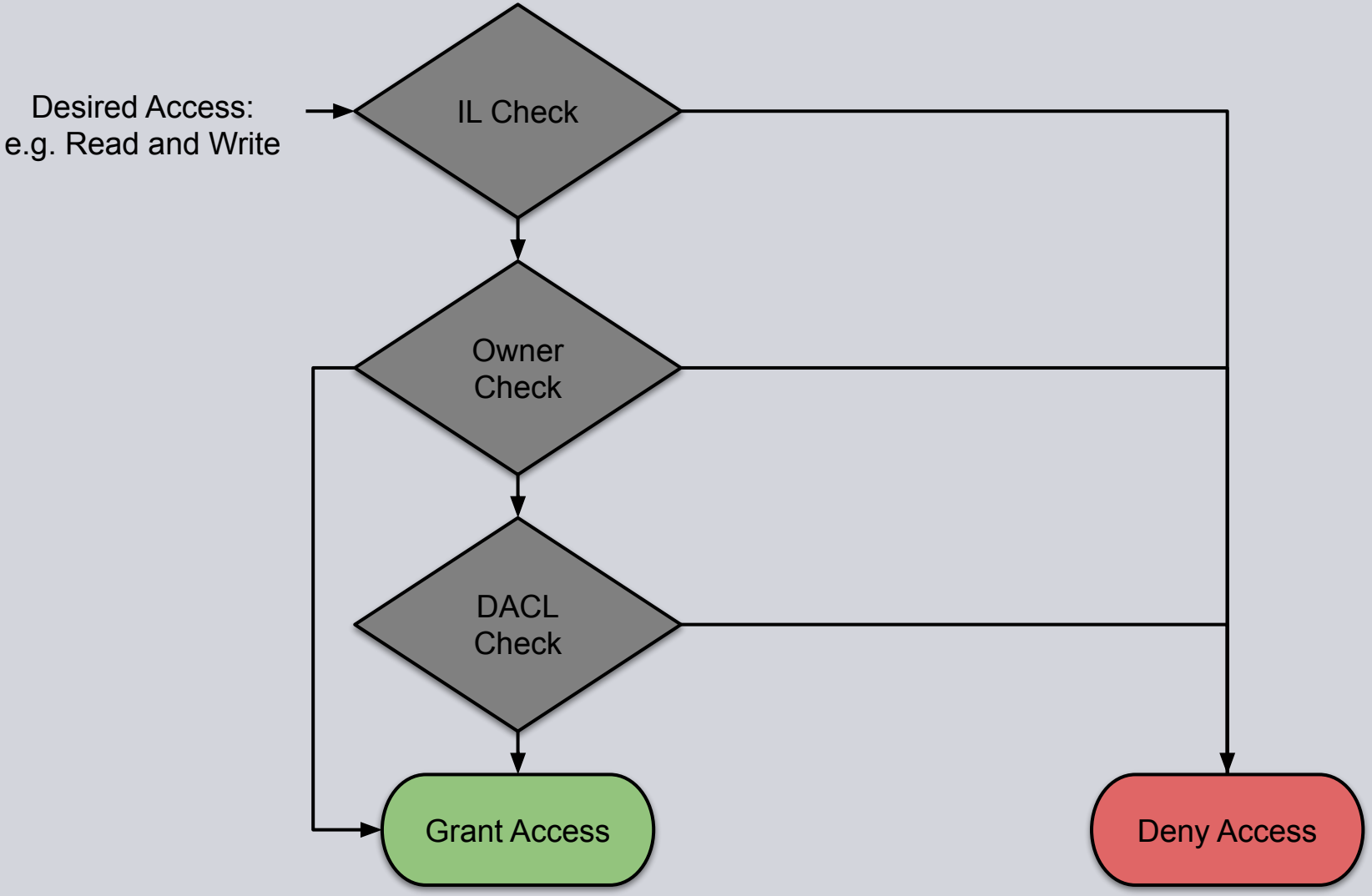
Groups

Mandatory Label

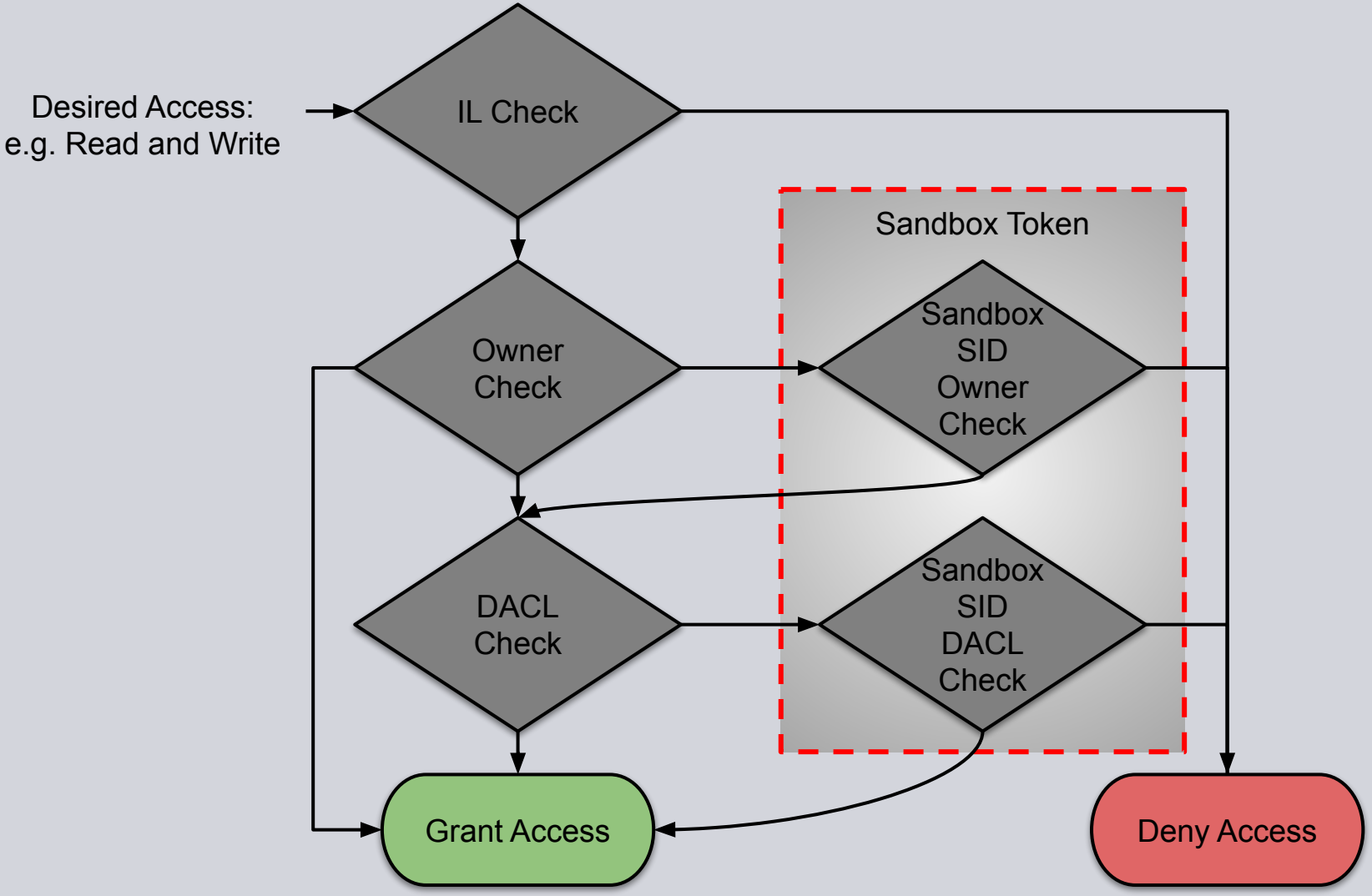
Privileges



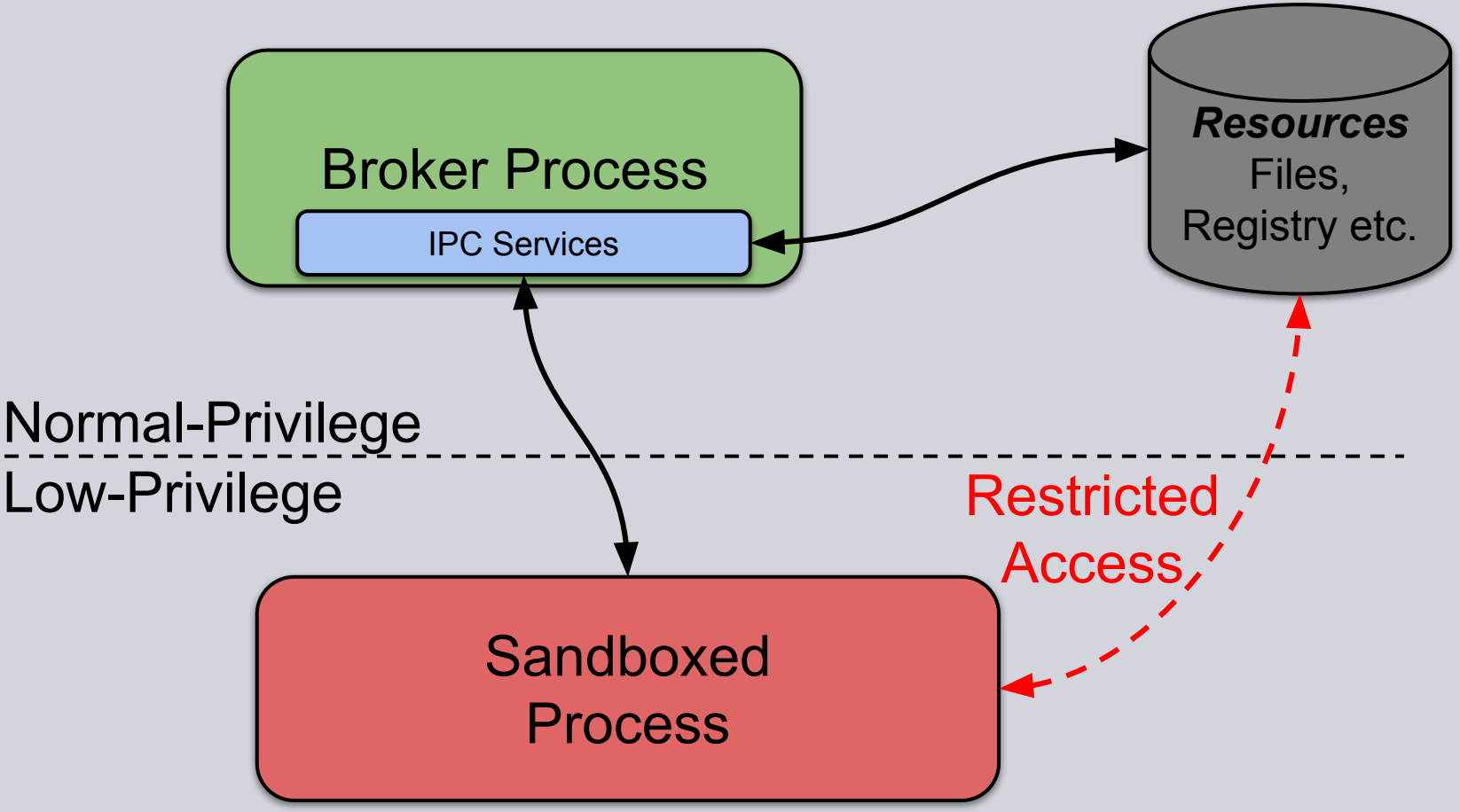
Access Check



Sandbox Access Check



Typical User-Mode Approach

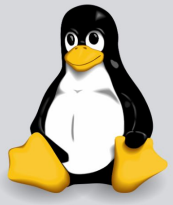


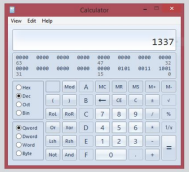
The Windows Sandbox Paradox

Kernel Attack Surface

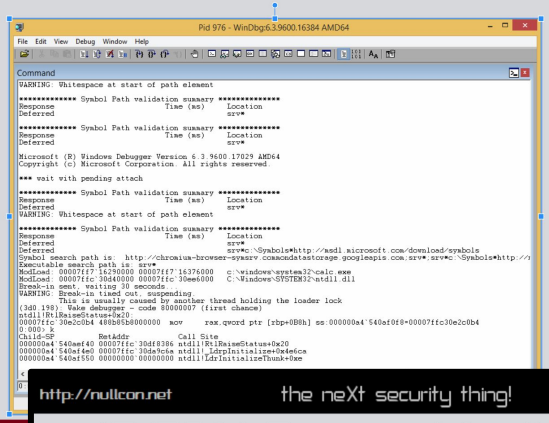
~300 Syscalls

~400 Syscalls + ~1000 Win32k

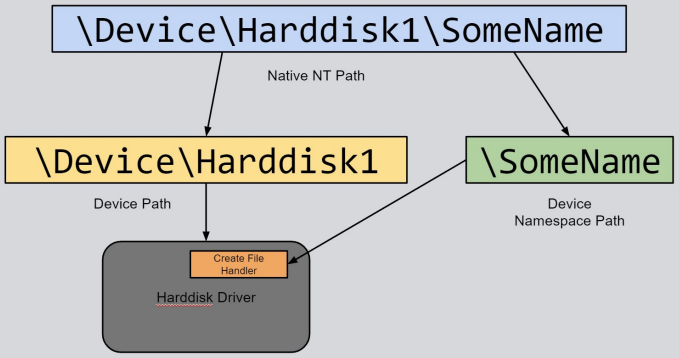




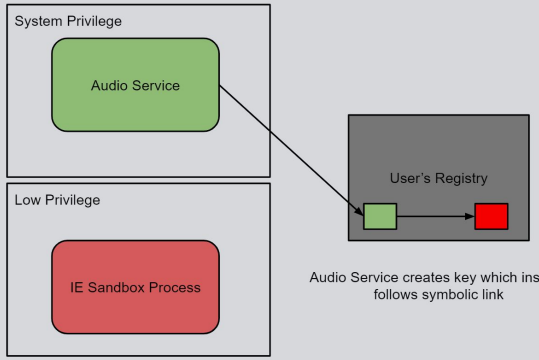
Crash!



Opening a Device Name



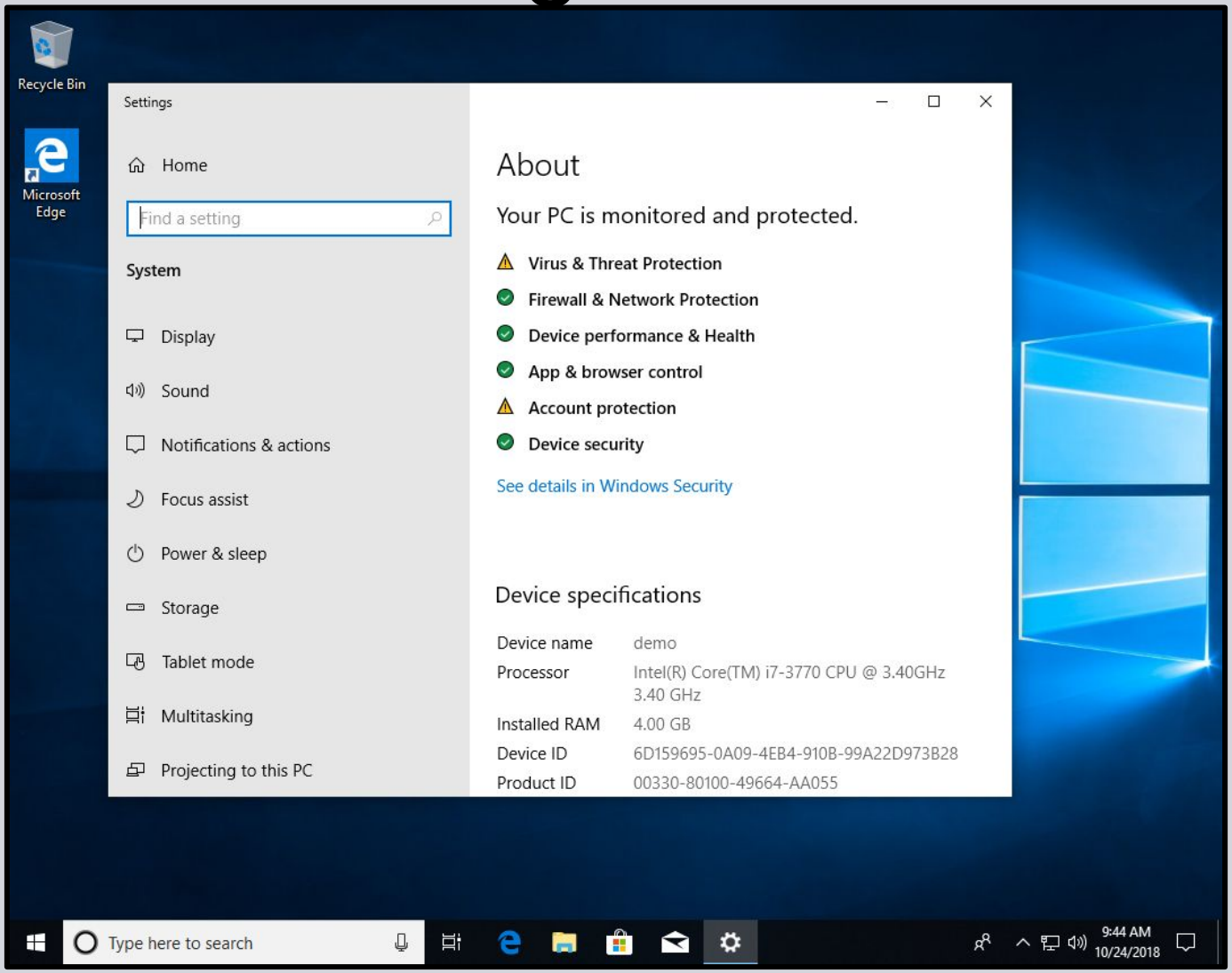
IE EPM Escape / Audio Server



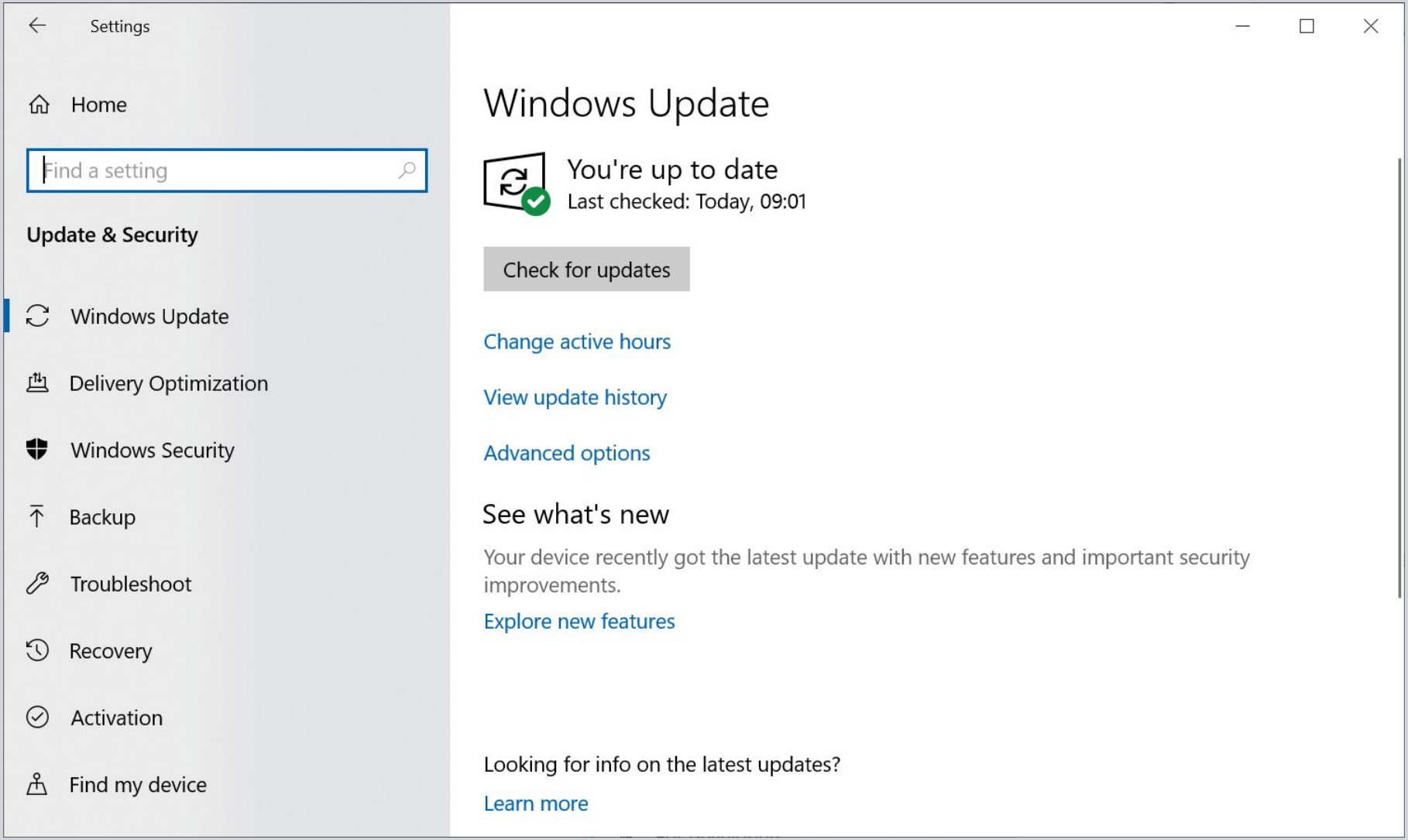
https://code.google.com/p/google-security-research/issues/detail?id=99

Welcome to 2019

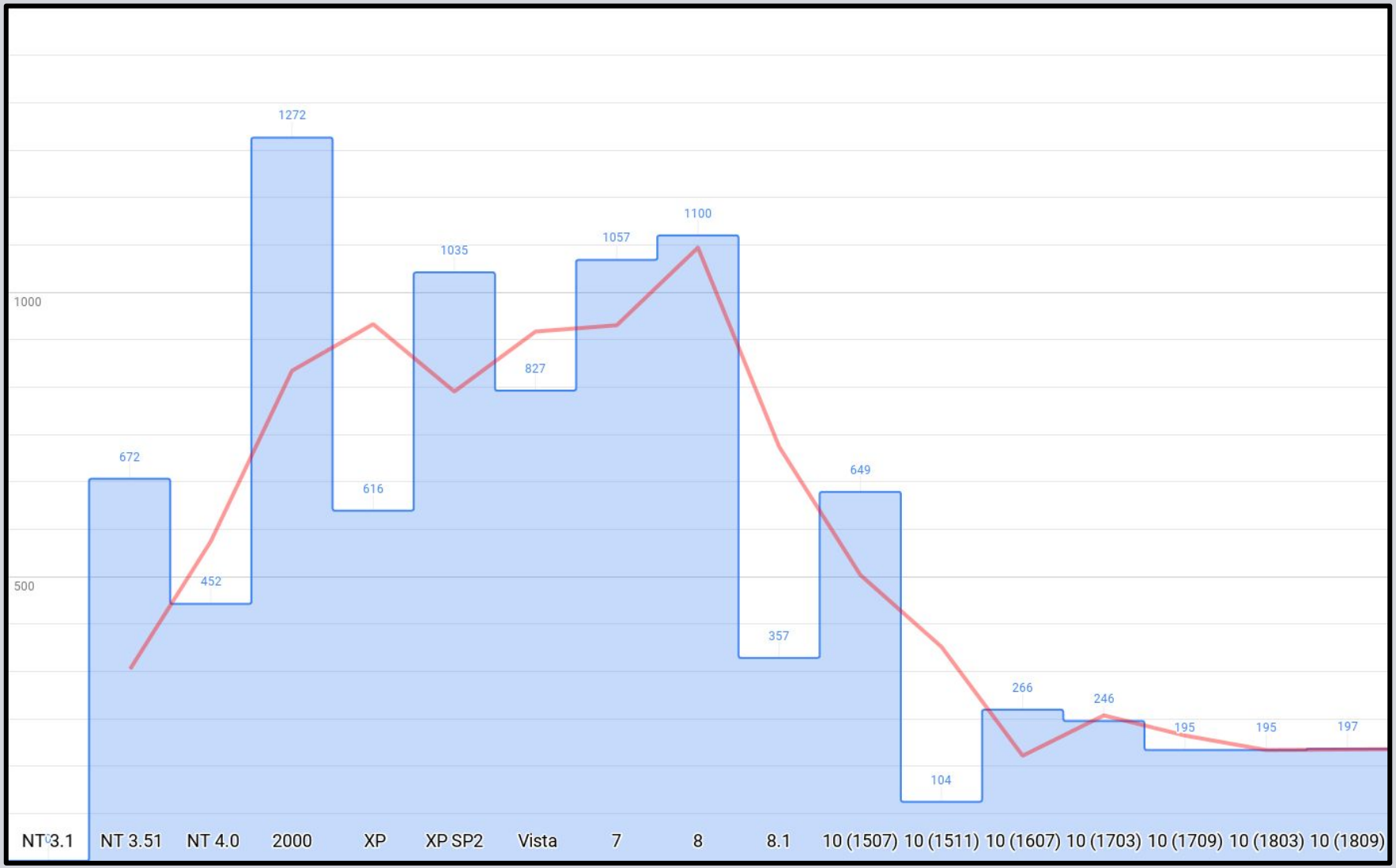
Introducing Windows 10



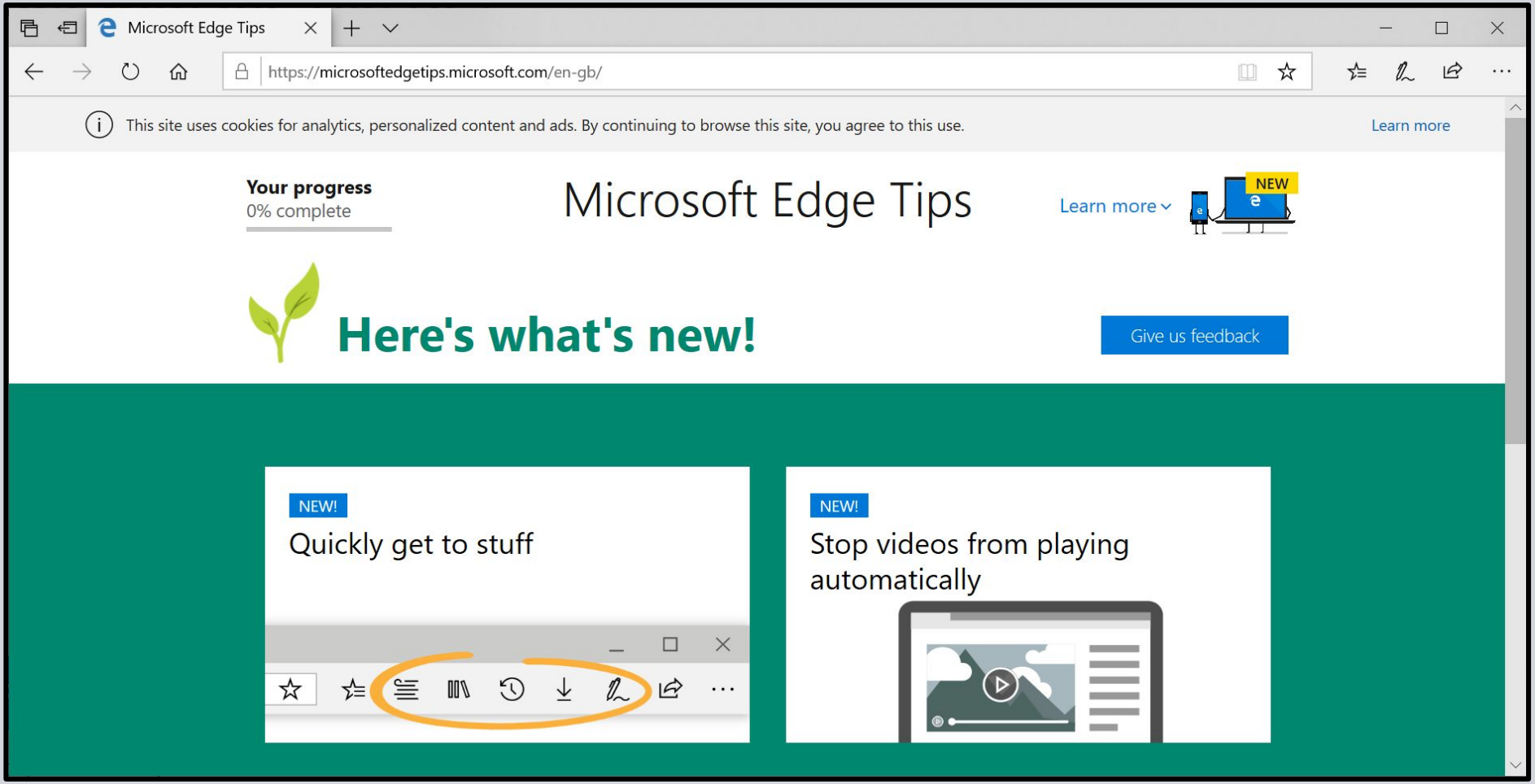
You're Going to Update



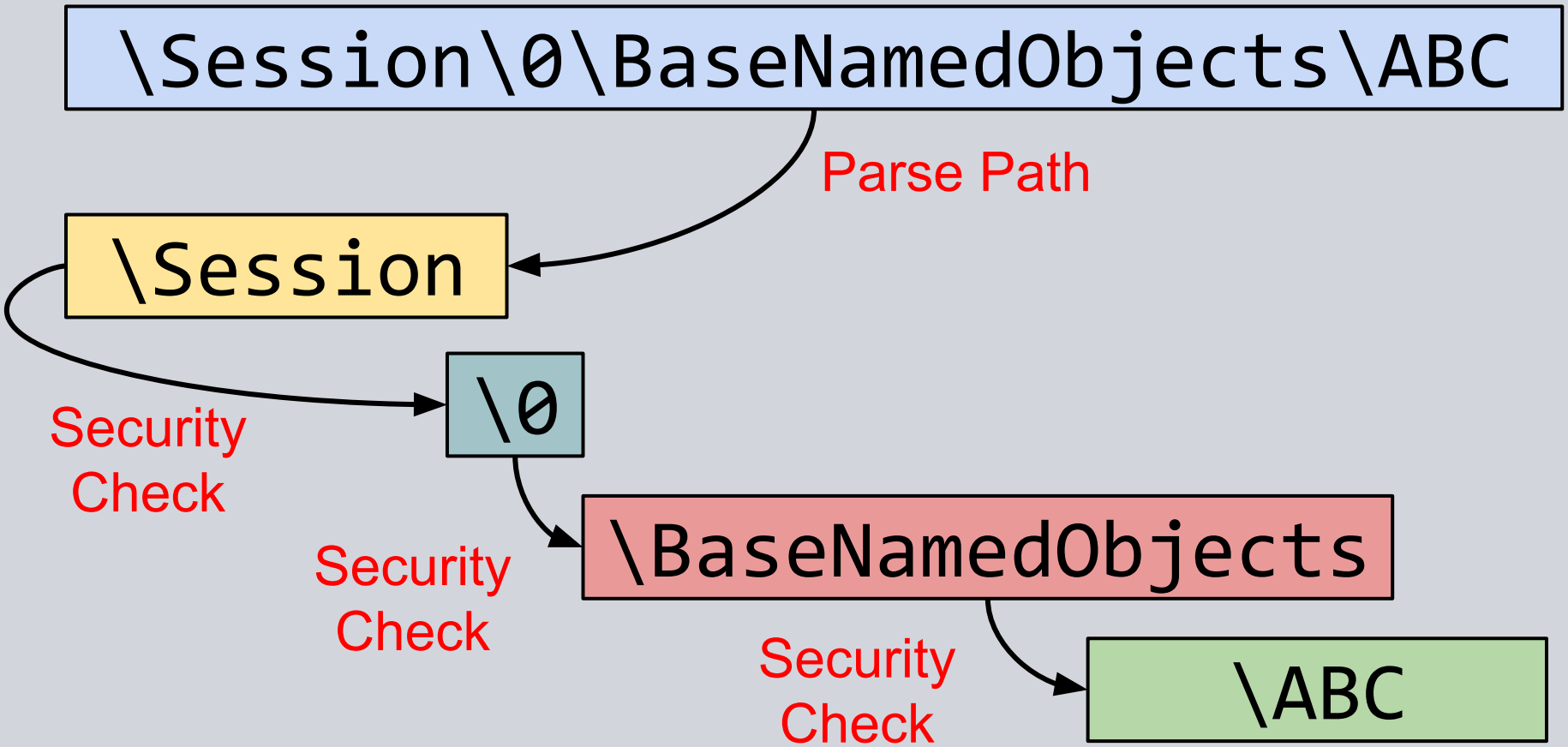
Time to Release is Shorter



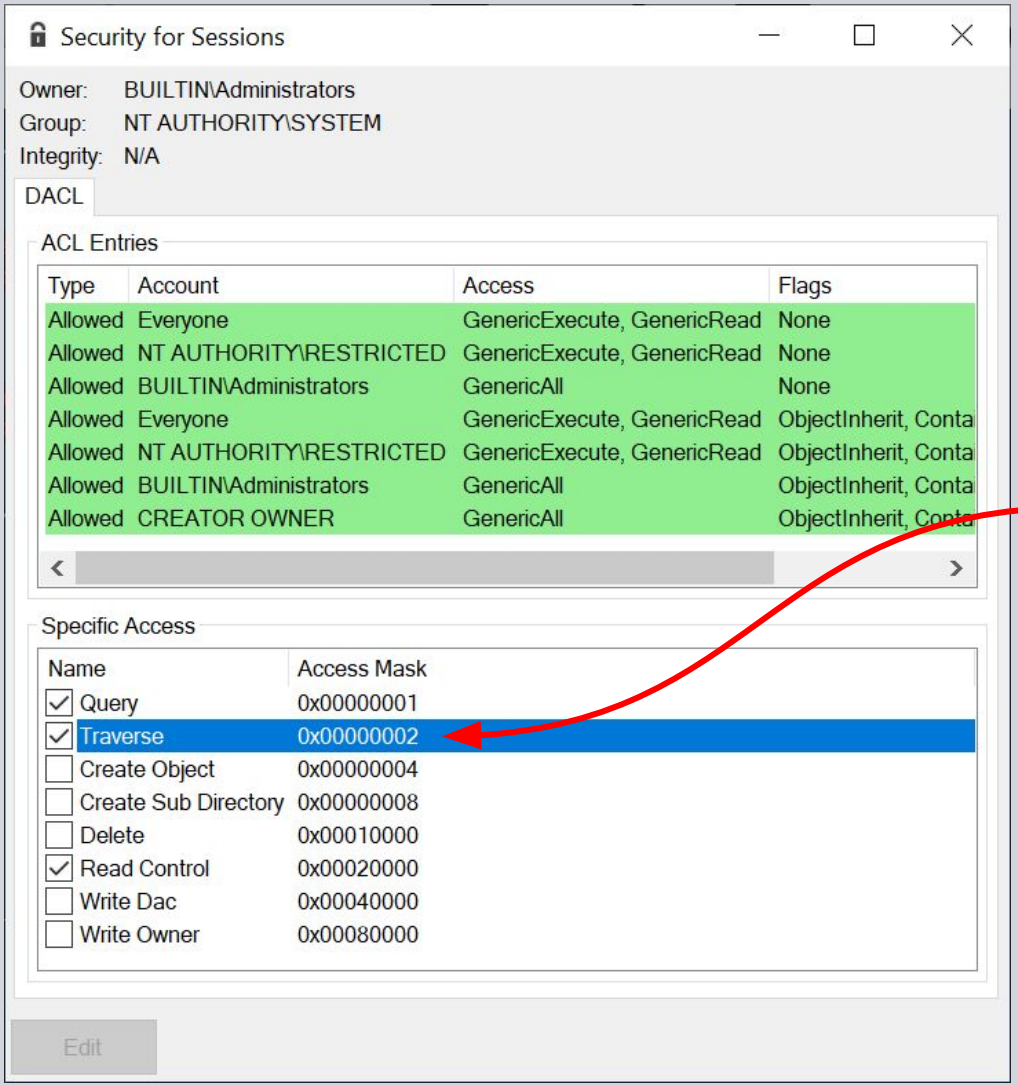
Microsoft Edge



Path Traversal Security

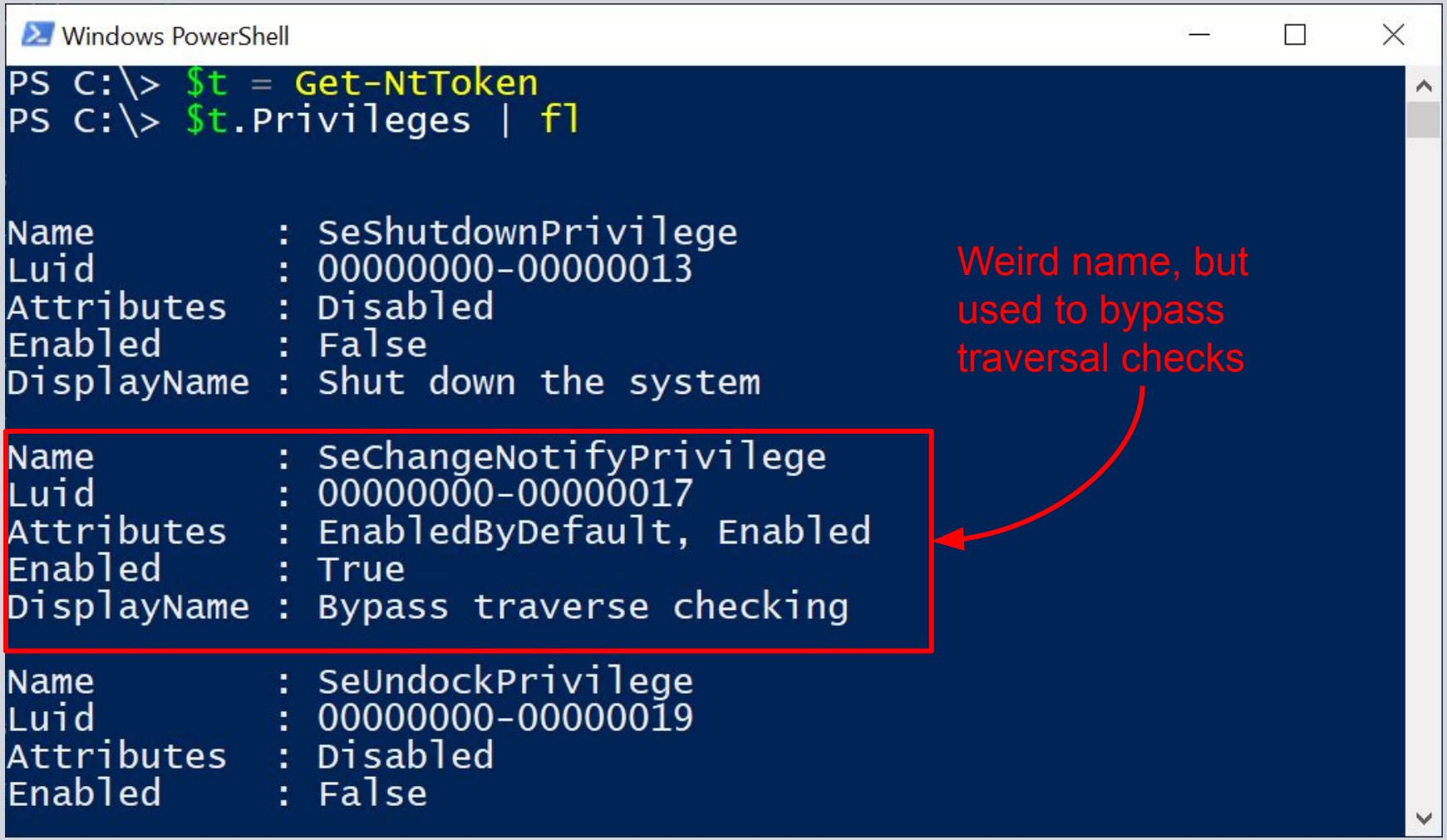


Traverse Access



Object Directory has specific “Traverse” access.

The Problem with Privileges



Remove Privilege?

```
NTSTATUS ObpLookupObjectName(
        OBJECT_ATTRIBUTES ObjectAttributes,
        PACCESS_STATE AccessState) {
  // ...
  if (AccessMode == KernelMode
   || AccessState->Flags & TOKEN_HAS_TRAVERSE_PRIVILEGE
   || ObpCheckTraverseAccess(Object, AccessState)) {
      // Continue traversal.
  } else {
    return STATUS_OBJECT_NAME_NOT_FOUND;
  }
}
```

Check for privilege

Do full traverse check

Full Traverse Check

```
BOOLEAN ObpCheckTraverseAccess(
            PVOID Object, PACCESS_STATE AccessState) {

  PSECURITY_DESCRIPTOR SecurityDescriptor;
  ObpGetObjectSecurity(Object, &SecurityDescriptor);

  if (SeFastTraverseCheck(SecurityDescriptor, AccessState,
                          DIRECTORY_TRAVERSE)) {
    return TRUE;
  } else {
      return SeAccessCheck(
             SecurityDescriptor,
             &AccessState->SubjectSecurityContext,
             DIRECTORY_TRAVERSE);
  }
}
```

Fast Traverse Check

```
BOOLEAN SeFastTraverseCheck(
                PSECURITY_DESCRIPTOR SecurityDescriptor,
                PACCESS_STATE AccessState,
                ACCESS_MASK DesiredAccess) {
  if (AccessState->Flags & TOKEN_IS_RESTRICTED) {
    return FALSE;
  }
  PACL dacl = SecurityDescriptor->Dacl;
  for(PACE ace = GetAce(dacl); ace; ace = GetAce(dacl)) {
      if (ace->AceType == ACCESS_ALLOWED_ACE_TYPE &&
          ace->Mask & DesiredAccess &&
          RtlEqualSid(SeWorldSid, ace->SidStart)) {
        return TRUE;
      }
  }
  return FALSE:
}
```

If a “Restricted” Token return immediately

Otherwise if ACL has World SID ACE then allow

Fast Traverse Check

```
BOOLEAN SeFastTraverseCheck(
                PSECURITY_DESCRIPTOR SecurityDescriptor,
                PACCESS_STATE AccessState,
                ACCESS_MASK DesiredAccess) {
  if (AccessState->Flags & TOKEN_IS_RESTRICTED) {
    return FALSE;
  }
  PACL dacl = SecurityDescriptor->Dacl;
  for(PACE ace = GetAce(dacl); ace; ace = GetAce(dacl)) {
      if (ace->AceType == ACCESS_ALLOWED_ACE_TYPE &&
          ace->Mask & DesiredAccess &&
          RtlEqualSid(SeWorldSid, ace->SidStart)) {
        return TRUE;
      }
  }
  return FALSE:
}
```

If a “Restricted” Token return immediately

Otherwise if ACL has World SID ACE then allow

Reporting the Problem

Issue 206: Windows: Limited Bypass of Traverse Permissions in Kernel Object Manager

Code

1 of 7 Next ›
Back to list

Reported by forshaw@google.com, Dec 2 2014 **Project Member**

Windows: Limited Bypass of Traverse Permissions in Kernel Object Manager
Platform: Windows 7 32/64 bit, Windows 8+
Class: Security Bypass

Windows has the concept of traversal permission which allows a user to open resources under a directory where they don't have read permissions on the directory itself, this can be expressed using the FILE_TRAVERSE permission or DIRECTORY_TRAVERSE permissions for the file system or object manager. Most tokens have the SeChangeNotifyPrivilege available which is used to always grant traverse permissions regardless of the actual DACL on the object.

When a token doesn't have this privilege (say the Anonymous token, or something like the heavily restricted Chrome sandbox token) it falls back to the access check. However there's some unusual behaviour in the SeFastTraverseCheck method which could grant traverse permissions to tokens which wouldn't normally be able to get it.

https://bugs.chromium.org/p/project-zero/issues/detail?id=206

Fixed in Windows 10

```
NTSTATUS SepCreateAccessStateFromSubjectContext(
      SECURITY_SUBJECT_CONTEXT *SubjectContext,
      PACCESS_STATE AccessState) {
  AccessState->SubjectSecurityContext = *SubjectContext;
  PTOKEN Token = SeQuerySubjectContextToken(SubjectContext);
  DWORD Flags = 0;

  if (Token->Privileges.Present & TRAVERSE_PRIVILEGE)
    Flags = TOKEN_HAS_TRAVERSE_PRIVILEGE;

  Flags |= Token->TokenFlags & TOKEN_IS_RESTRICTED;
  AccessState->Flags = Flags;

  // ...
}
```

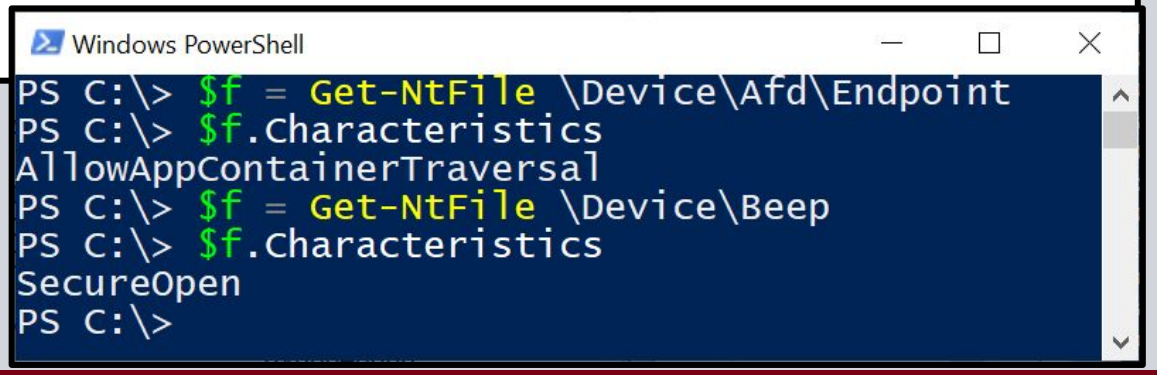
Added Flag

AC Device Attack Surface

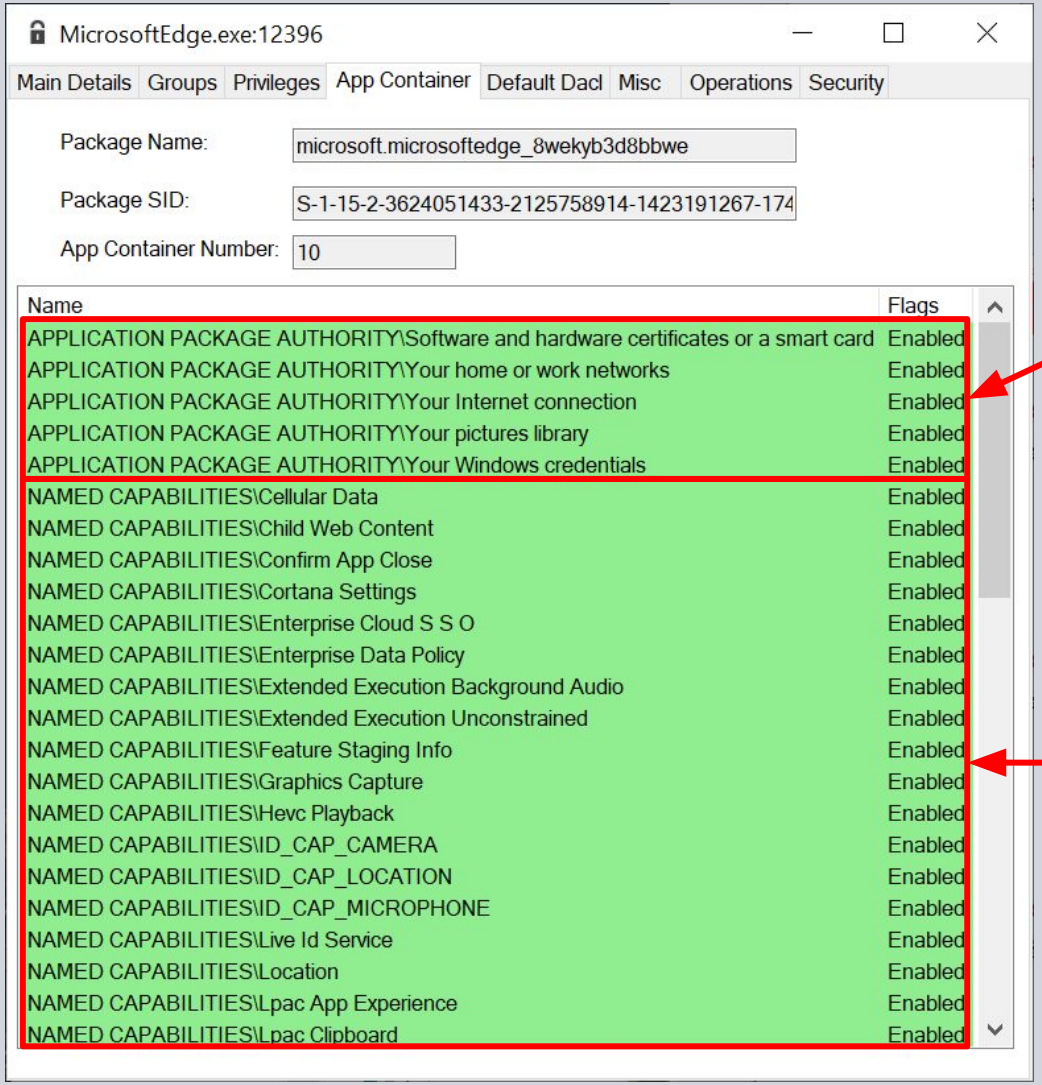
```
BOOLEAN IopDoFullTraverseCheck(PDEVICE_OBJECT Device,
        PSECURITY_SUBJECT_CONTEXT SubjectSecurityContext) {
  if (Device->Characteristics &
     (FILE_DEVICE_ALLOW_APPCONTAINER_TRAVERSAL |
           FILE_DEVICE_SECURE_OPEN)
      == FILE_DEVICE_ALLOW_APPCONTAINER_TRAVERSAL) {
    return FALSE;
  }
  BOOLEAN IsAppContainer;
  SeIsAppContainerOrIdentifyLevelContext(SubjectSecurityContext,
                                         &IsAppContainer);
  return IsAppContainer;
}
```

If AC Traversal flag in device then allow

Only allow if not AC.



Generic AC Capabilities

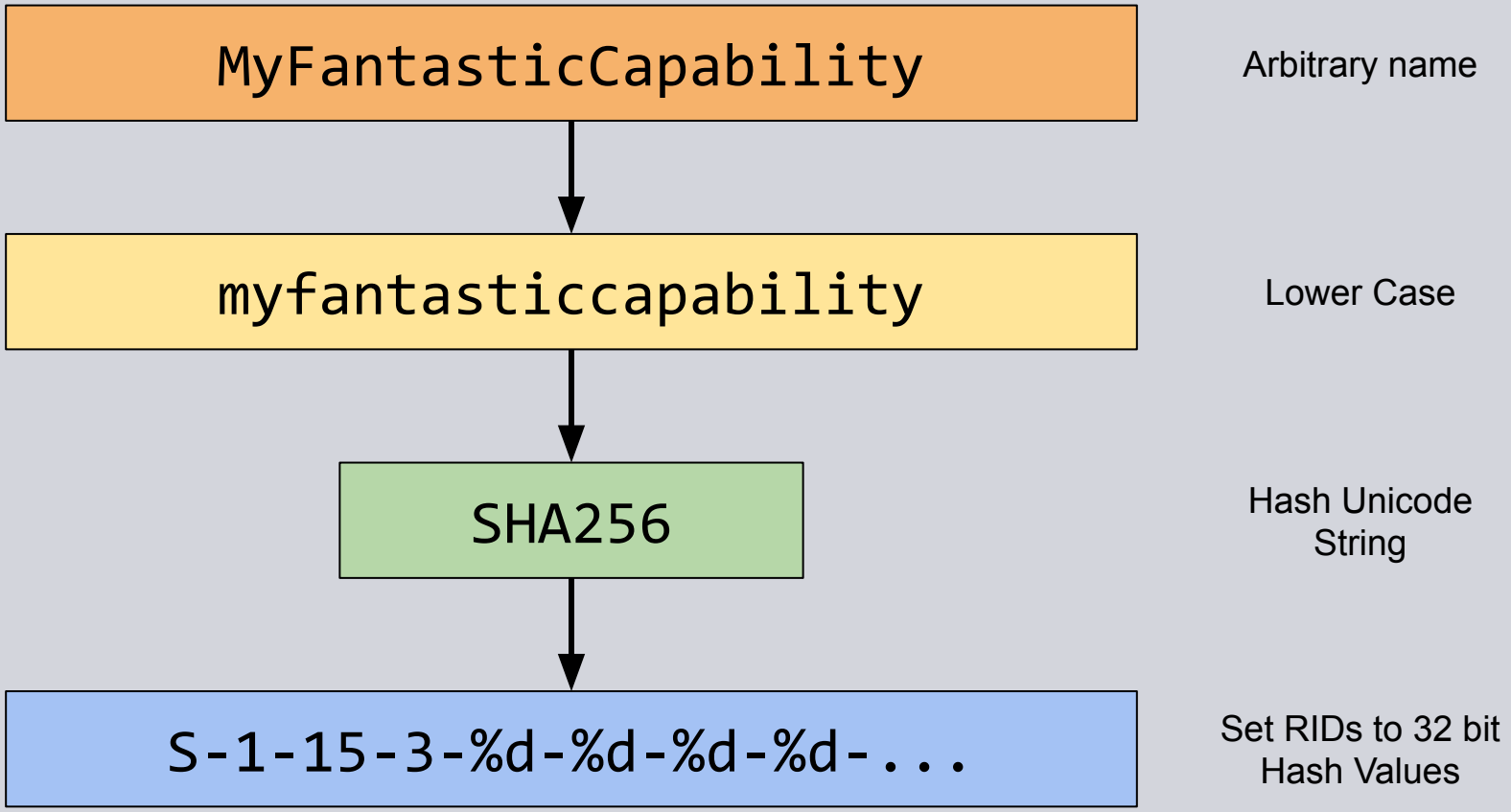


Fixed capabilities, introduced in Windows 8

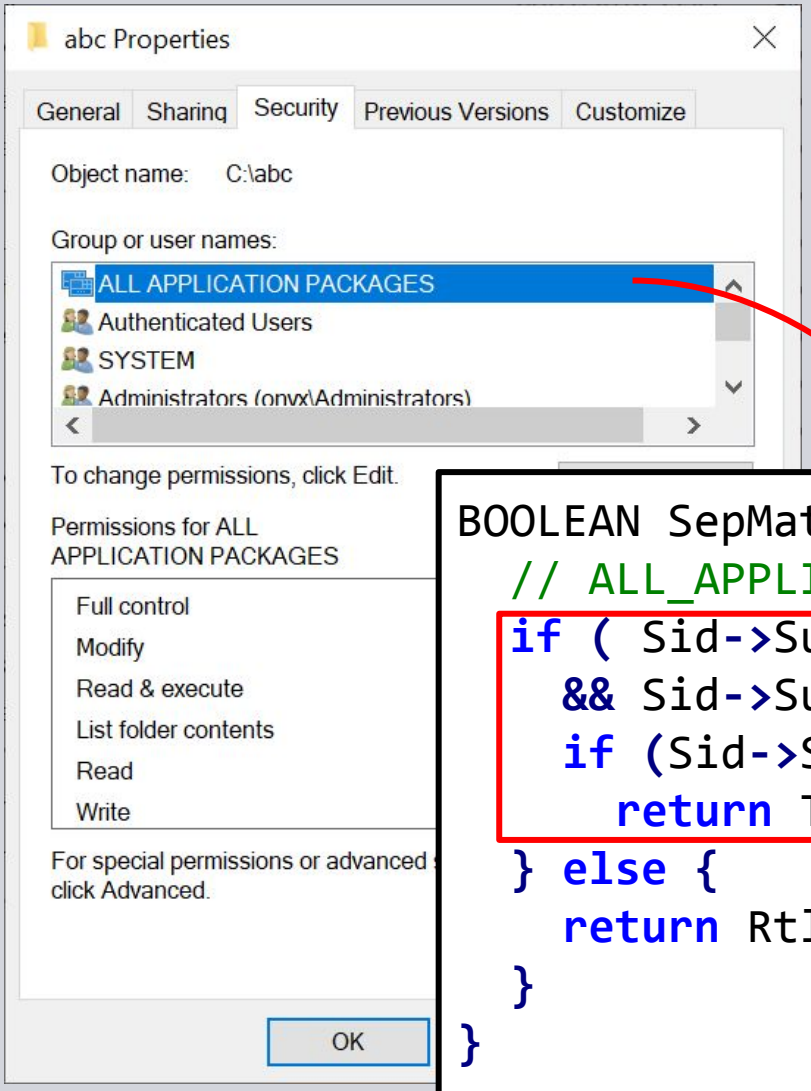
Generic capabilities, introduced in Windows 10

Capability String to SID

```
BOOL DeriveCapabilitySidsFromName(LPWSTR CapName, PSID **Sids)
```



All Application Packages



Hardcoded Group Check if an App Container

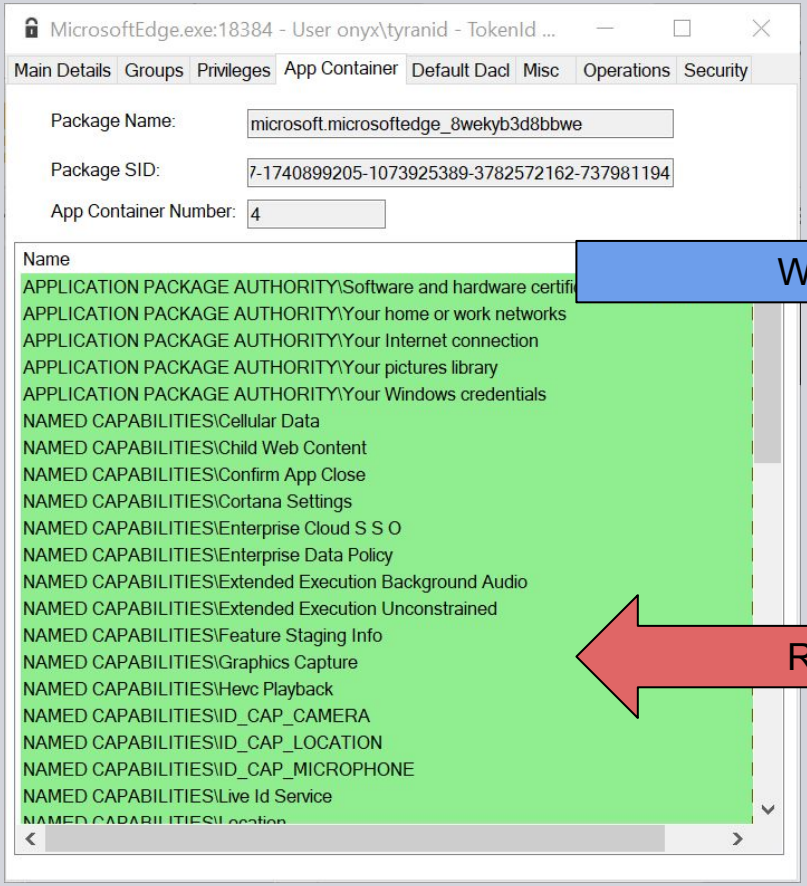
```
BOOLEAN SepMatchPackage(PTOKEN Token, PSID Sid) {
  // ALL_APPLICATION_PACKAGES is S-1-15-2-1
  if ( Sid->SubAuthority[0] == 2
    && Sid->SubAuthorityCount == 2 ) {
    if (Sid->SubAuthority[1] == 1)
      return TRUE;
  } else {
    return RtlEqualSid(Token->Package, Sid);
  }
}
```

Low Privilege App Container

```
BOOLEAN SepMatchPackage(PTOKEN Token, PSID Sid) {
  if ( Sid->SubAuthority[0] == 2 &&
       Sid->SubAuthorityCount == 2 ) {
    if (Sid->SubAuthority[1] == 1 &&
        SepCanTokenMatchAllPackageSid(Token))
      return TRUE;
    // ALL_RESTRICTED_APPLICATION_PACKAGES is S-1-15-2-2
    if (Sid->SubAuthority[1] == 2)
      return TRUE;
  } else {
    return R
  }
}
```

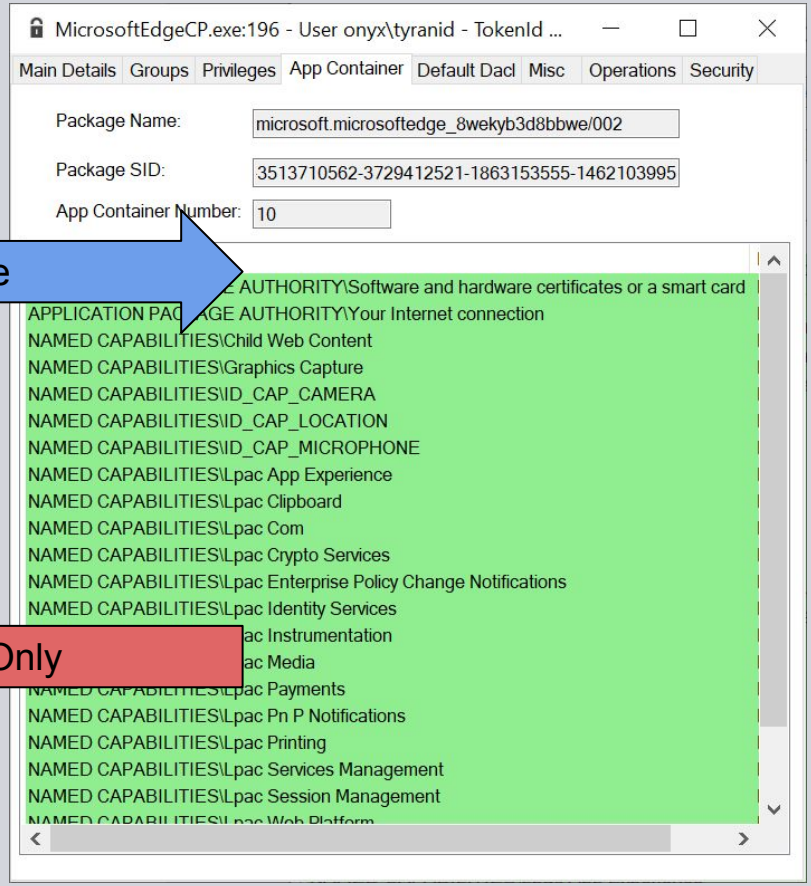
```
BOOLEAN SepCanTokenMatchAllPackageSid(PTOKEN Token) {
  int Policy;
  AuthzBasepQuerySecurityAttributeAndValues(
        L"WIN://NOALLAPPPKG", &Policy)

  return Policy == 0;
}
```

Child App Containers



Writable

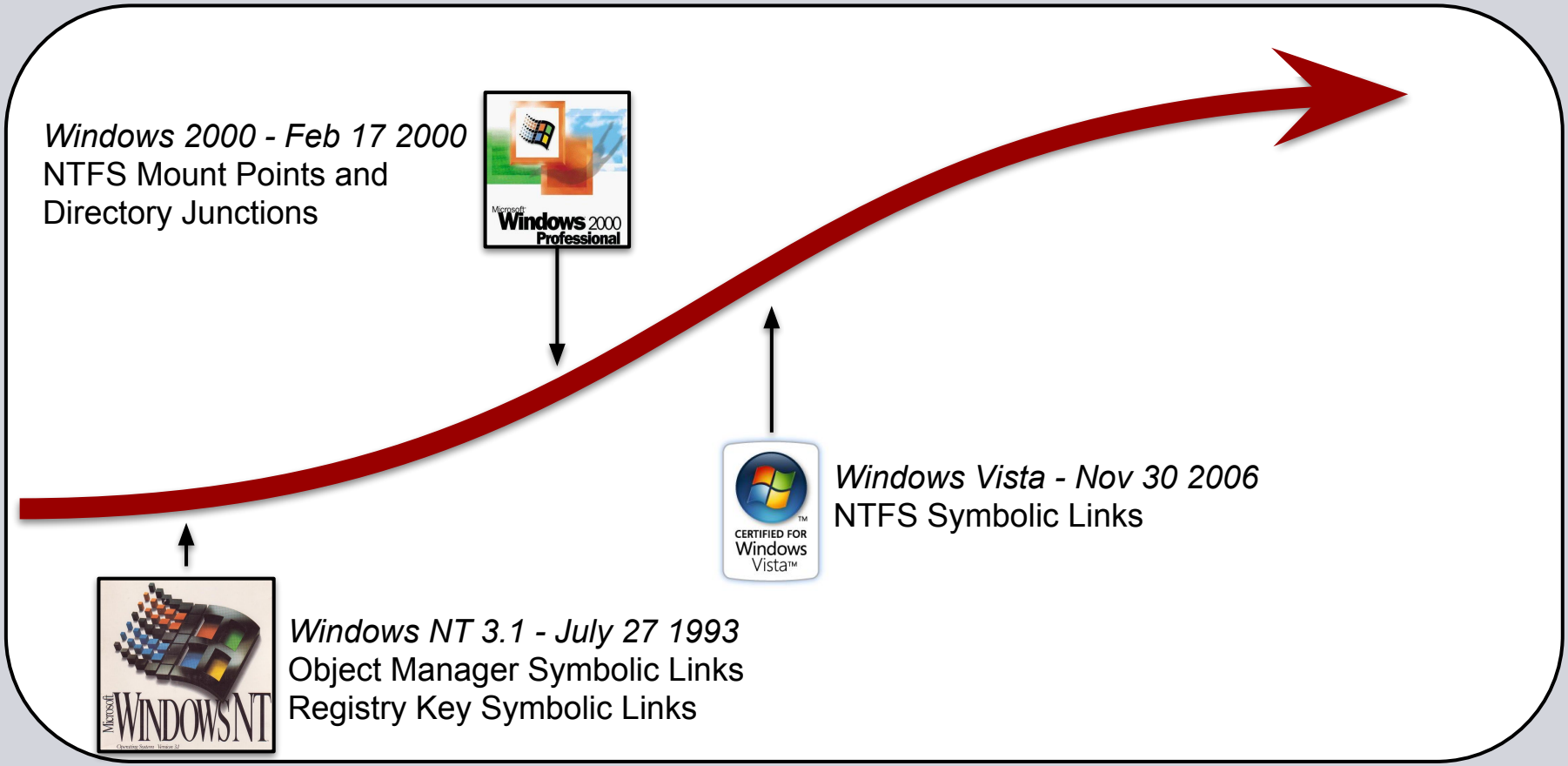
Read-Only



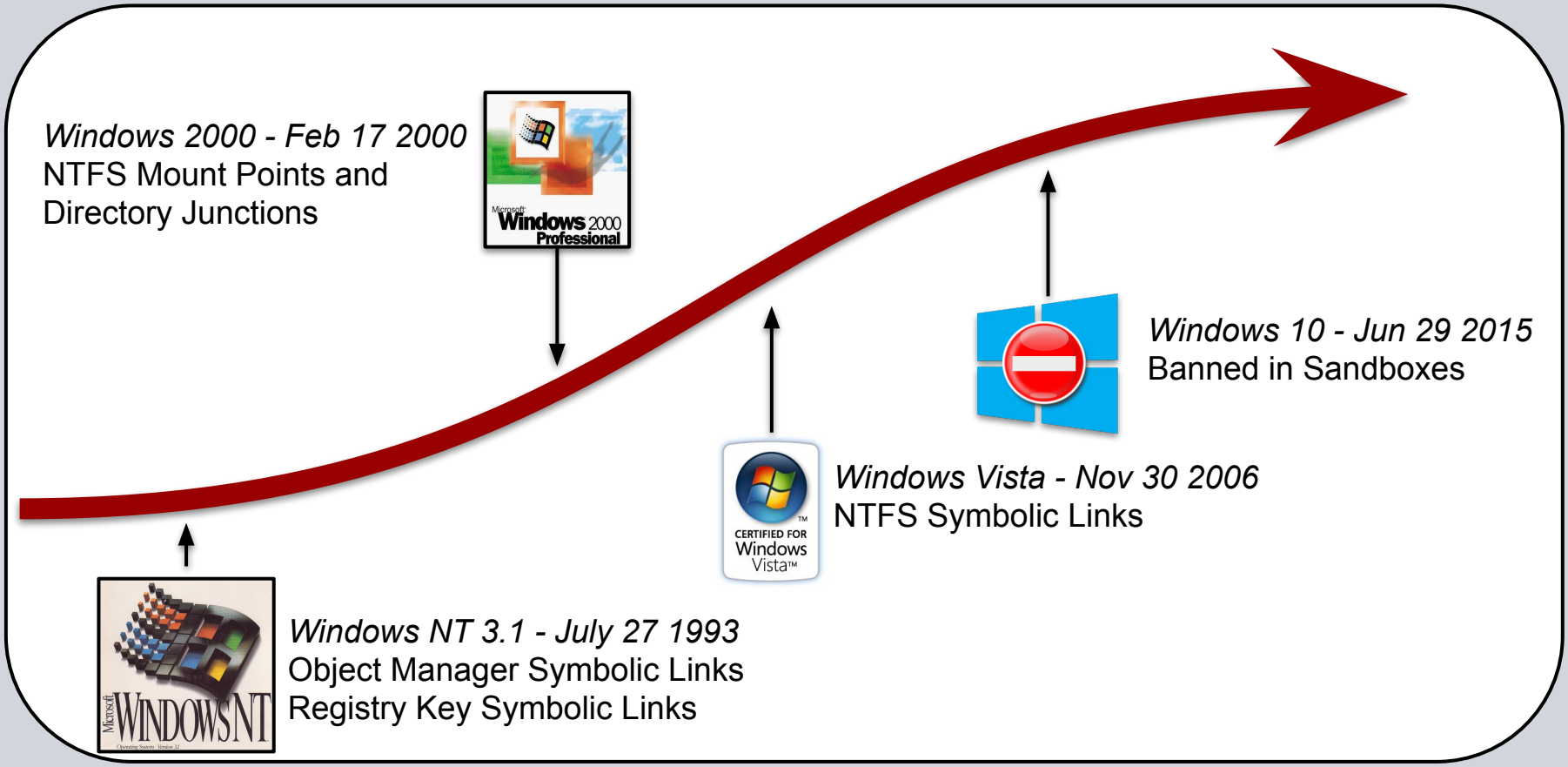
S-1-15-2-PARENT-RIDS

S-1-15-2-PARENT-RIDS-CHILD-RIDS

History of Symbolic Links



History of Symbolic Links



RtlIsSandboxedToken

- Introduced in Windows 10 but backported to Windows 7

```
BOOLEAN RtlIsSandboxedToken() {
  SECURITY_SUBJECT_CONTEXT SecurityContext;

  SeCaptureSubjectContext(&SecurityContext);
  return !SeAccessCheck(
         SeMediumDaclSd,
         SubjectSecurityContext,
         READ_CONTROL);
}
```

Must pass security check

Registry Key Symbolic Links

```
NTSTATUS CmpCheckCreateAccess(...) {
    BOOLEAN AccessGranted = SeAccessCheck(...);
    if (AccessGranted &&
        CreateOptions & REG_OPTION_CREATE_LINK &&
        RtlIsSandboxedToken()) {
        return STATUS_ACCESS_DENIED;
    }
}
```

Hard Ban!

```
NTSTATUS CmSetValueKey(...) {
  if(Type == REG_LINK &&
    RtlEqualUnicodeString(&CmSymbolicLinkValueName,
                          ValueName, TRUE) &&
    RtlIsSandboxedToken())
    return STATUS_ACCESS_DENIED;
}
```

Blocking NTFS Mount Points

```
NTSTATUS IopXxxControlFile(...) {
   if (ControlCode == FSCTL_SET_REPARSE_POINT &&
       RtlIsSandboxedToken()) {
    if (buffer.ReparseTag == IO_REPARSE_TAG_MOUNT_POINT) {
      InitializeObjectAttributes(&ObjAttr, buffer.PathBuffer);
      status = ZwOpenFile(&FileHandle, FILE_GENERIC_WRITE,
                &ObjAttr, ..., FILE_DIRECTORY_FILE);
        if (status < 0)
          return status;
        // Continue.
      }
    }
}
```

Checks target is a directory and writable

Bypassing the Mitigation

Issue 486: Windows: Sandboxed Mount Reparse Point Creation Mitigation Bypass

Code ‹ Prev 7 of 23 Next › Back to list

Reported by forshaw@google.com, Jul 22 2015 Project Member

Windows: Sandboxed Mount Reparse Point Creation Mitigation Bypass
Platform: Windows 10 (build 10240), earlier versions do not have the functionality
Class: Security Feature Bypass

Summary:
A mitigation added to Windows 10 to prevent NTFS Mount Reparse Points being created at integrity levels below medium can be bypassed.

Description:

Windows 10 has added some new mitigations to block the creation or change the behaviour of certain symbolic links when issued by a low integrity/sandboxed process. The presumed aim to to make it harder to abuse these types of tricks to break out of a sandbox.

https://bugs.chromium.org/p/project-zero/issues/detail?id=486

Bypassing the Mitigation

Issue 486: Windows: Sandboxed Mount Reparse Point Creation Mitigation Bypass

Code ‹ Prev 7 of 23 Next › Back to list

Reported by forshaw@google.com, Jul 22 2015 Project Member

Windows: Sandboxed Mount Reparse Point Creation Mitigation Bypass
Platform: Windows 10 (build 10240), earlier versions do not have the functionality
Class

Summ
A miti
mediu

Descr

Windo
when
tricks

```
NTSTATUS NtSetInformationProcess(...) {
    // ...

    case ProcessDeviceMap:
       HANDLE hDir = *(HANDLE*)Data;
       if (RtlIsSandboxedToken())
         return STATUS_ACCESS_DENIED;
       return ObSetDeviceMap(ProcessObject, hDir);

       // ...
}
```

Use Hardlinks

Issue 531: Windows: Creating Hardlinks Doesn't Require Write Permissions to the Target

Code

1 of 4

Back to list

Reported by forshaw@google.com, Sep 14 2015 **Project Member**

Microsoft requested I removed information from a public presentation that you can create NTFS hardlinks without needing write permissions on the target file. Their view is they want to fix this, at the least to prevent its abuse in sandboxed applications so a case has been set up to track the issue. It's still under the normal 90 day SLA.

This bug is subject to a 90 day disclosure deadline. If 90 days elapse without a broadly available patch, then the bug report will automatically become visible to the public.

https://bugs.chromium.org/p/project-zero/issues/detail?id=531

Use Hardlinks

Issue 531: Windows: Creating Hardlinks Doesn't Require Write Permissions to the Target

Code

1 of 4

Back to list

Reported by forshaw@google.com, Sep 14 2015 **Project Member**

Microsoft requested I removed information from a public presentation that you can create NTFS hardlinks without needing write permissions on the target file. Their view is they want to fix this, at the le
st
Th
w
be

```
NTSTATUS NtSetInformationFile(...) {
  case FileLinkInformation:
    ACCESS_MASK RequiredAccess = 0;
    if(RtlIsSandboxedToken()) {
      RequiredAccess |= FILE_WRITE_ATTRIBUTES;
    }

    ObReferenceObjectByHandle(FileHandle, RequiredAccess);
}
```

Setting an Mitigation Policy

SetProcessMitigationPolicy function

Sets the mitigation policy for the calling process.

Syntax

C++

```
BOOL WINAPI SetProcessMitigationPolicy(
  _In_ PROCESS_MITIGATION_POLICY MitigationPolicy,
  _In_ PVOID                     lpBuffer,
  _In_ SIZE_T                    dwLength
);
```

Policy Type

Accompanying Data

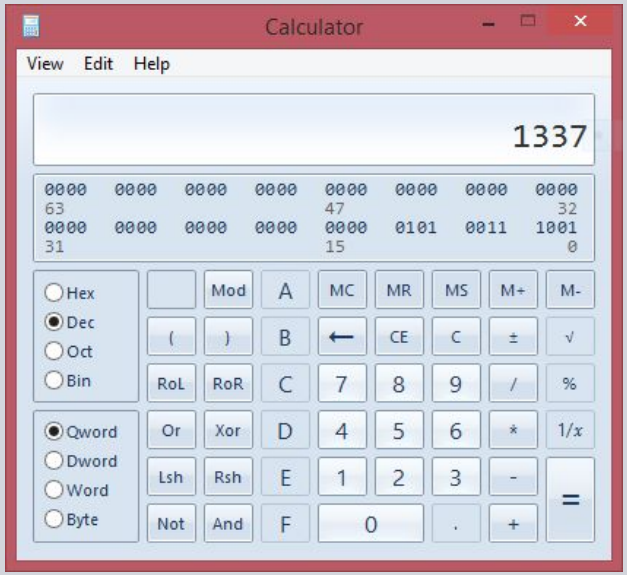
Available Policies

Policy	*Supported Win8.1 Update 2*	*Supported Win10 TH2*
ProcessDEPPolicy	Yes	Yes
ProcessASLRPolicy	Yes	Yes
ProcessDynamicCodePolicy	Yes	Yes
ProcessStrictHandleCheckPolicy	Yes	Yes
ProcessSystemCallDisablePolicy	Yes	Yes
ProcessMitigationOptionsMask	Invalid	Invalid
ProcessExtensionPointDisablePolicy	Yes	Yes
ProcessControlFlowGuardPolicy	Invalid	Invalid
ProcessSignaturePolicy	Yes*	Yes
ProcessFontDisablePolicy	No	Yes
ProcessImageLoadPolicy	No	Yes

* Not supported through SetProcessMitigationPolicy

Kernel Attack Surface

~400 Syscalls + ~1000 WIN3K

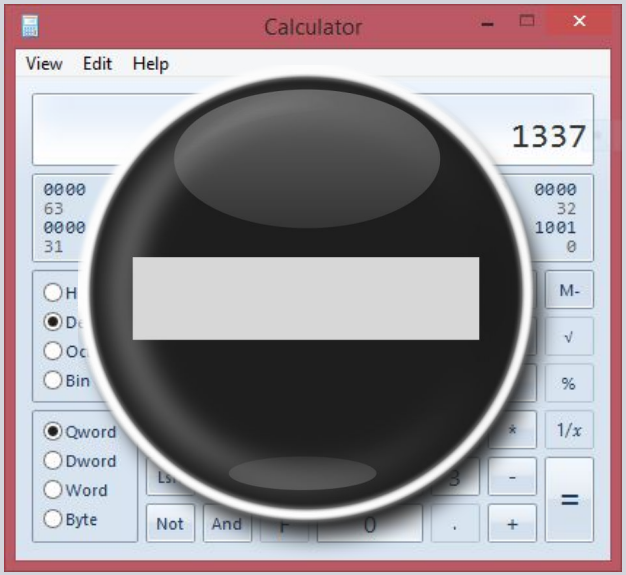


WIN32K System Call Disable

~400 Syscalls + ~0 WIN32K

WIN32K System Call Filter

~400 Syscalls + ~50 WIN32K



Dynamic Code Policy

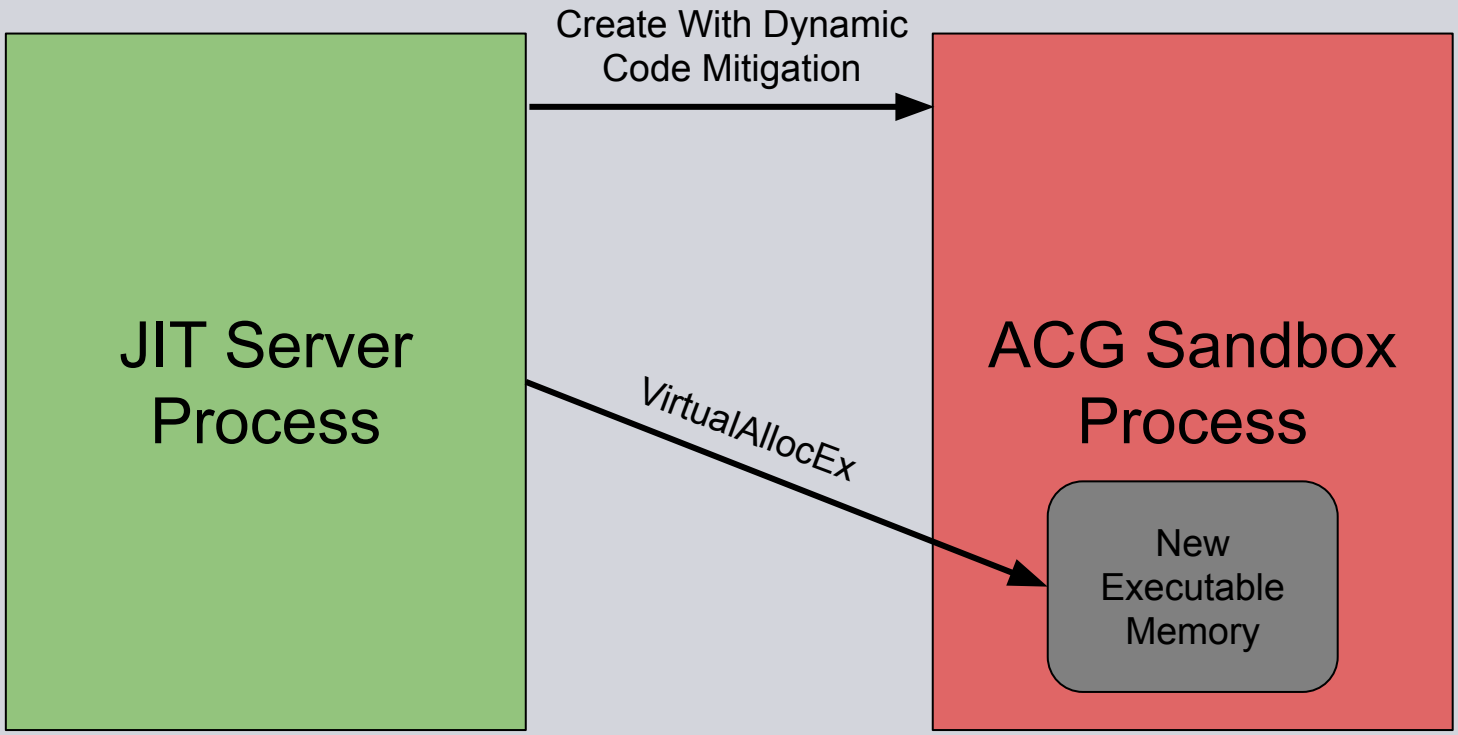
```
struct PROCESS_MITIGATION_DYNAMIC_CODE_POLICY {
    DWORD ProhibitDynamicCode : 1;
    DWORD AllowThreadOptOut : 1;
    DWORD AllowRemoteDowngrade : 1;
    DWORD AuditProhibitDynamicCode : 1;
};
```

Disables:

- VirtualAlloc with PAGE_EXECUTE_*
- MapViewOfFile with FILE_MAP_EXECUTE
- VirtualProtect with PAGE_EXECUTE_*

Called Arbitrary Code Guard (ACG) by Microsoft

Can Still JIT with a Helper




```
void InjectExecutableCode(DWORD dwPid, PBYTE pData, SIZE_T nSize) {
  HANDLE hProcess = OpenProcess(dwPid,
                        PROCESS_VM_WRITE | PROCESS_VM_OPERATION);
  LPVOID pMem = VirtualAllocEx(hProcess,
                        PAGE_EXECUTE_READWRITE, nSize);
  WriteProcessMemory(hProcess, pMem, pData, nSize);
}
```

Binary Signature Policy

```
struct PROCESS_MITIGATION_BINARY_SIGNATURE_POLICY
{
    DWORD MicrosoftSignedOnly : 1;
    DWORD StoreSignedOnly : 1;
    DWORD MitigationOptIn : 1;
    DWORD AuditMicrosoftSignedOnly : 1;
    DWORD AuditStoreSignedOnly : 1;
};
```

Blocks unsigned or non-Microsoft Store signed DLLs.

Called Code Integrity Guard by Microsoft

Bypassable Mitigation

Issue 1597: Windows: CiSetFileCache TOCTOU CVE-2017-11830 Variant WDAC Security Feature Bypass

Code

1 of 3 Next ›

Back to list

Reported by forshaw@google.com, Jun 18 2018 **Project Member**

Windows: CiSetFileCache TOCTOU CVE-2017-11830 Variant WDAC Security Feature Bypass
Platform: Windows 10 1803, 1709 (should include S-Mode but not tested)
Class: Security Feature Bypass

Summary:
While the TOCTOU attack against cache signing has been mitigated through NtSetCachedSigningLevel it's possible to reach the same code via NtCreateSection leading to circumventing WDAC policies and CIG/PPL.

https://bugs.chromium.org/p/project-zero/issues/detail?id=1597

Image Load Policy

```
struct PROCESS_MITIGATION_IMAGE_LOAD_POLICY
{
    DWORD NoRemoteImages : 1;
    DWORD NoLowMandatoryLabelImages : 1;
    DWORD PreferSystem32Images : 1;
    DWORD AuditNoRemoteImages : 1;
    DWORD AuditNoLowMandatoryLabelImages : 1;
};
```

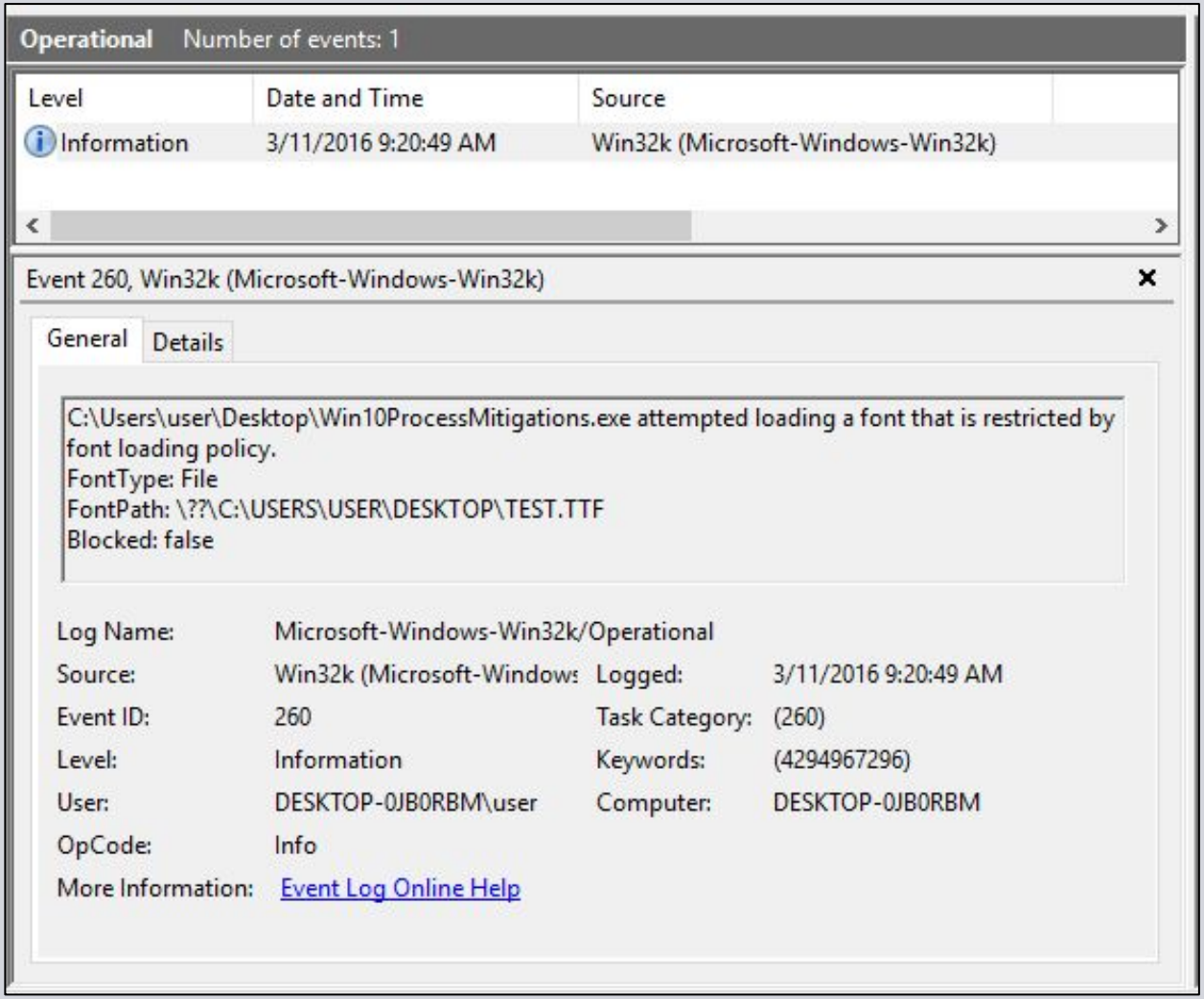
Disable DLL loading from UNC paths or with Low IL label.

Font Disable Policy

```
struct PROCESS_MITIGATION_FONT_DISABLE_POLICY
{
    DWORD DisableNonSystemFonts      : 1;
    DWORD AuditNonSystemFontLoading : 1;
    DWORD ReservedFlags              : 30;
};
```

Disable fonts loaded from memory or outside of %WINDIR%\Fonts

Font Mitigation Auditing



Bypassable Mitigation

Issue 779: Windows: Custom Font Disable Policy Bypass

Code

‹ Prev 2 of 2

Back to list

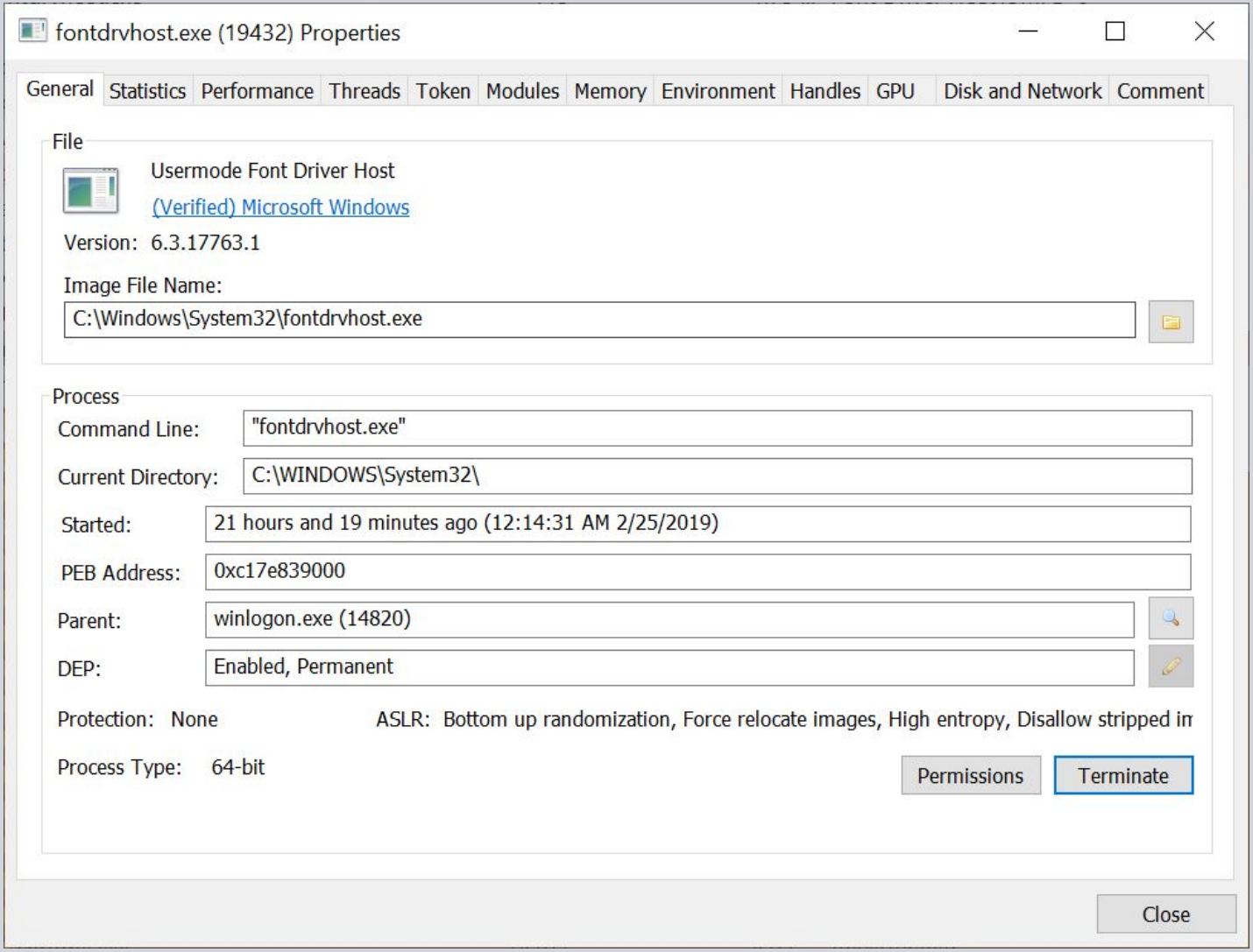
Reported by forshaw@google.com, Mar 24 2016 **Project Member**

Windows: Custom Font Disable Policy Bypass
Platform: Windows 10 Only
Class: Security Feature Bypass

Summary:
It's possible to bypass the ProcessFontDisablePolicy check in win32k to load a custom font from an arbitrary file on disk even in a sandbox. This might be used as part of a chain to elevate privileges. If anything this is really a useful demonstration that you probably really want to shutdown the object manager directory shadowing as part of the sandbox mitigations, even if you don't fix the explicit bypass.

https://bugs.chromium.org/p/project-zero/issues/detail?id=779

User Mode Font Driver



Bugs Bugs Bugs

Issue 468: Windows: User Mode Font Driver Thread Permissions EoP

Code

1 of 2 Next ›

Back to list

Reported by forshaw@google.com, Jun 30 2015 **Project Member**

Windows: User Mode Font Driver Thread Permissions EoP
Platform: Windows 10 Build 10130
Class: Elevation of Privilege

Summary:
The host process for the UMFD runs as a normal user but with a heavily restrictive process DACL. It's possible execute arbitrary code within the context of the process because it's possible to access the processes threads leading to local EoP.

Description:

NOTE: This was tested on the latest available build on Windows 10. I don't know if the final version will change the functionality to fix this vulnerability.

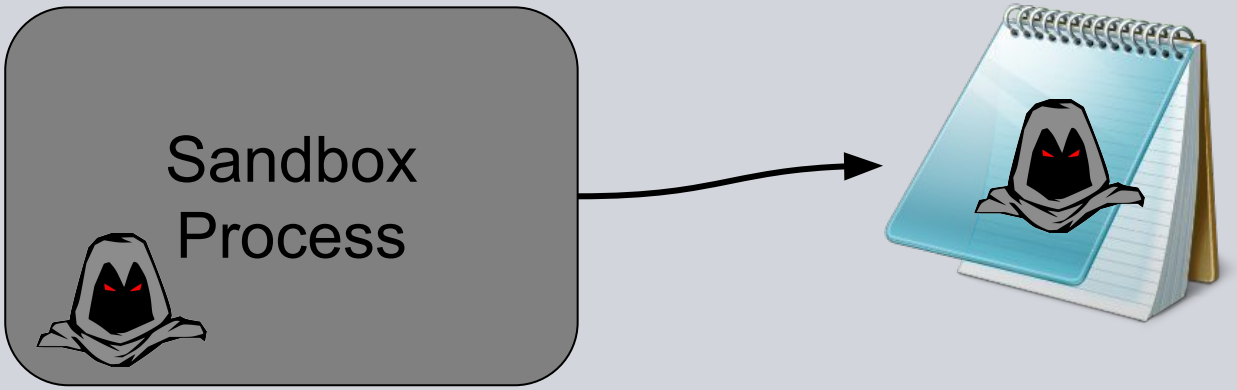
https://bugs.chromium.org/p/project-zero/issues/detail?id=468

Process Mitigations Inheritance

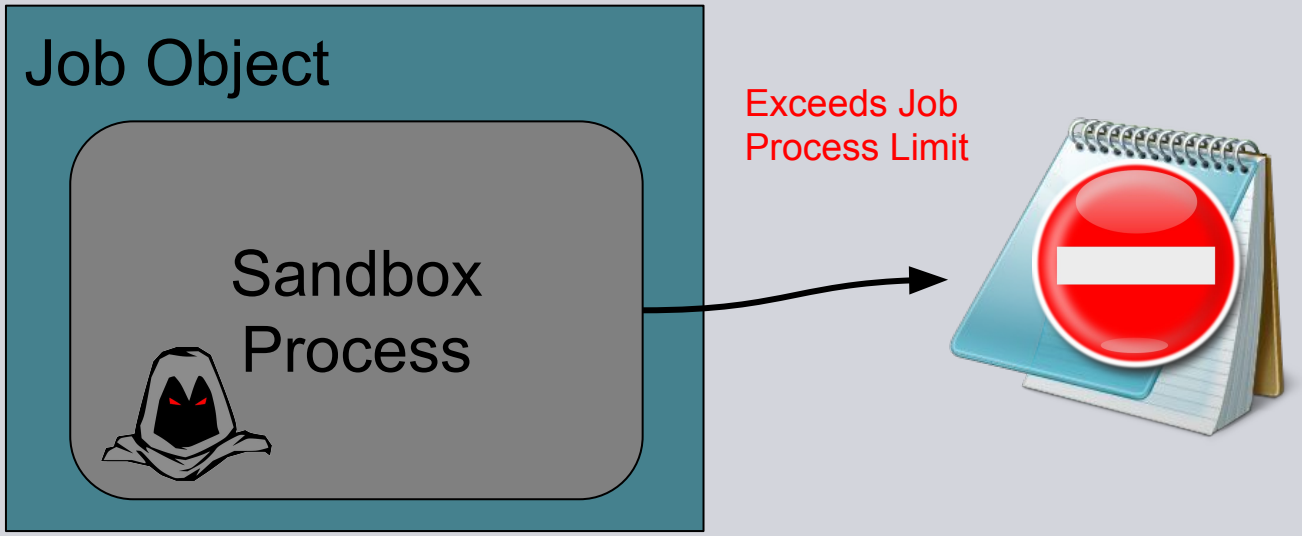
- No policies can be disabled once set in-process.
- However only a small subset of mitigations are inherited

Policy	*Inherited*
Dynamic Code	No
System Call Disable	Yes
Signature	No
Font Disable	No
Image Load	Yes

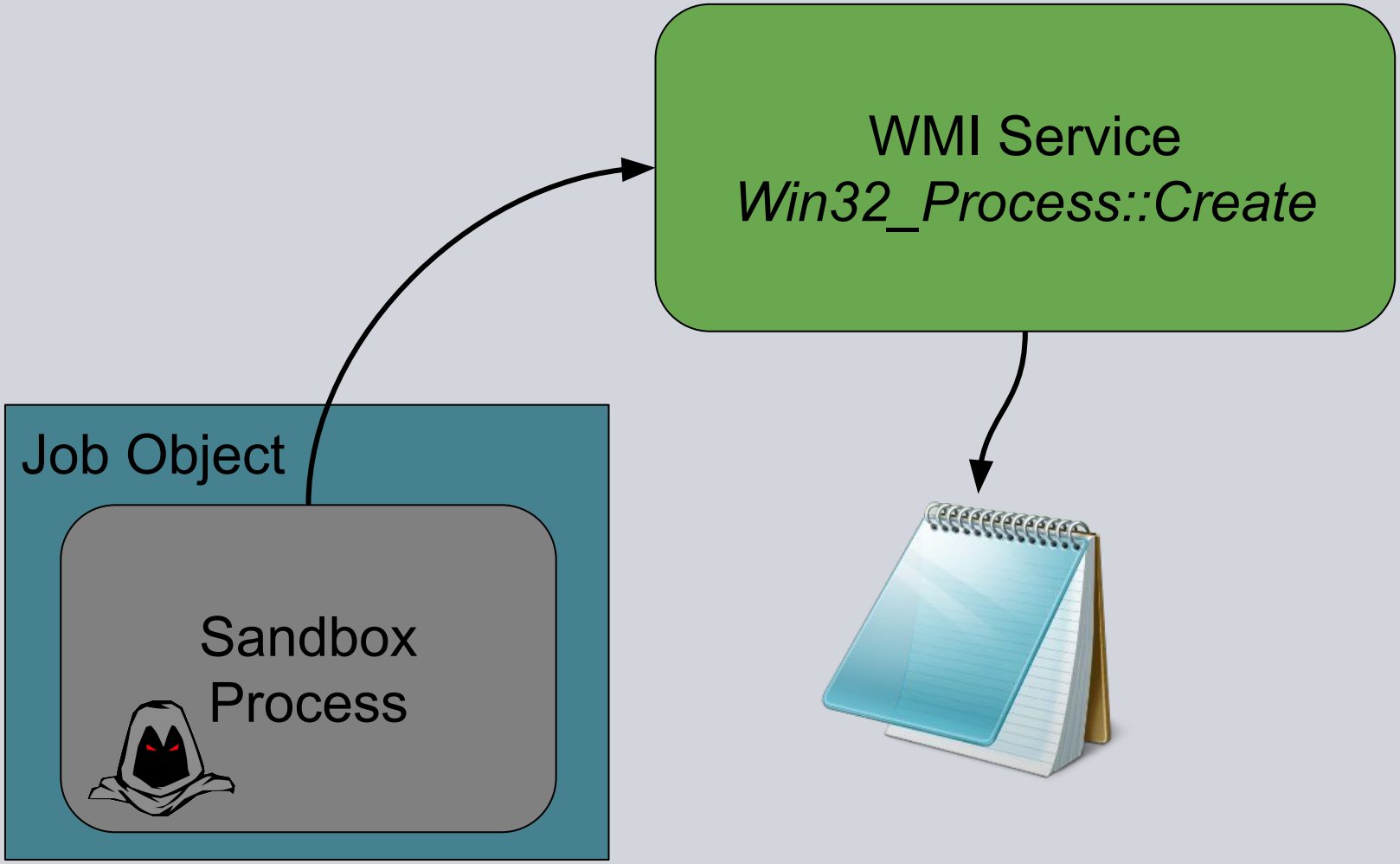
Migrate to a New Process



Restrict With Job Objects



Restrict With Job Objects



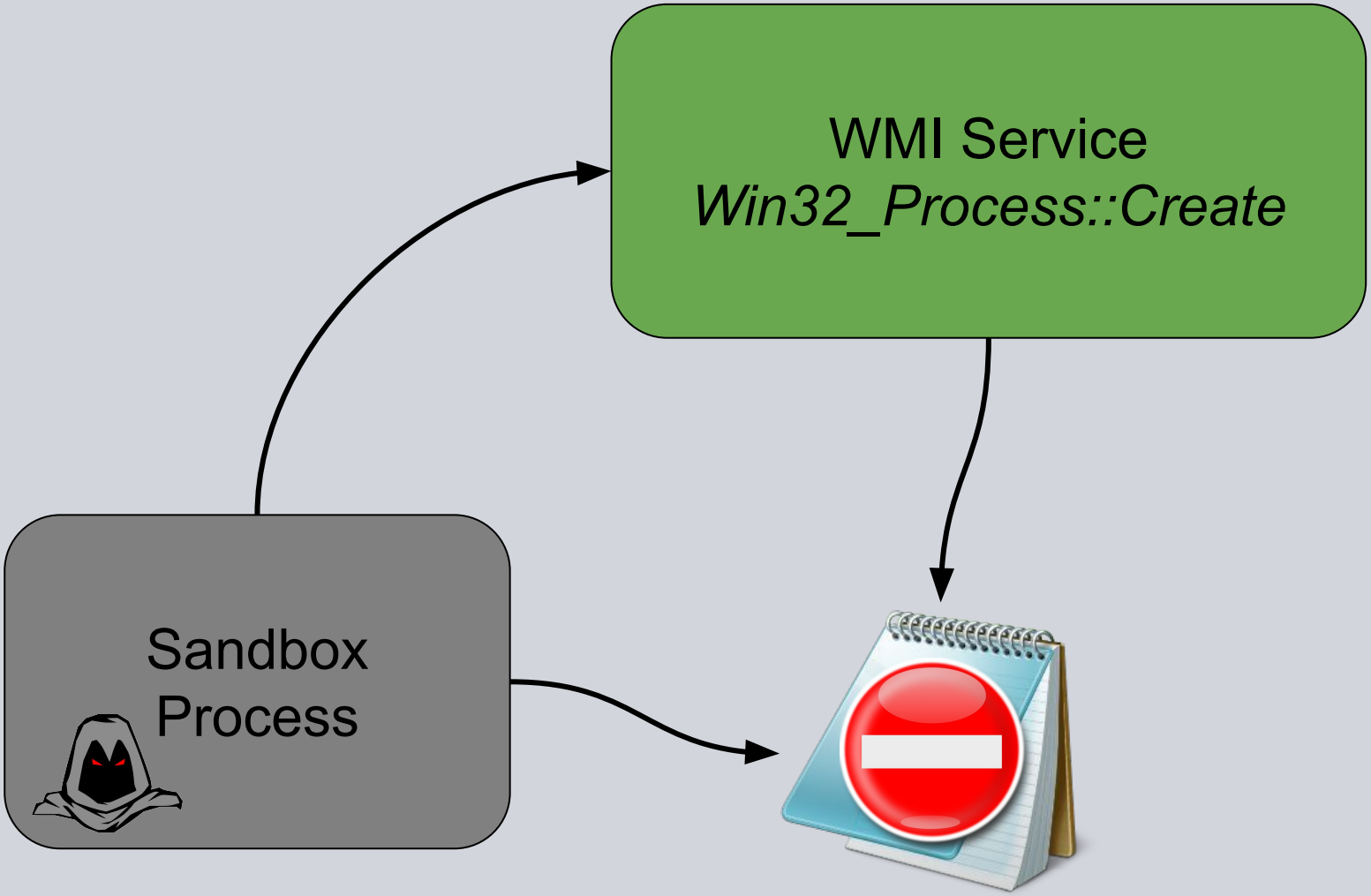
Inside NtCreateUserProcess

```
DWORD ChildProcessPolicyFlag = // From process attribute.
BOOLEAN ChildProcessAllowed = TokenObject->TokenFlags &
                                  CHILD_PROCESS_RESTRICTED;
if (!ChildProcessAllowed) {
  if (!ChildProcessPolicyFlag &
        PROCESS_CREATION_CHILD_PROCESS_OVERRIDE)
    || !SeSinglePrivilegeCheck(SeTcbPrivilege))
    return STATUS_ACCESS_DENIED;
}

SepDuplicateToken(TokenObject, ..., &NewTokenObject);
if (ChildProcessPolicyFlag &
    PROCESS_CREATION_CHILD_PROCESS_RESTRICTED)
      NewTokenObject->TokenFlags |= CHILD_PROCESS_RESTRICTED;
```

Block process with Token flag.

Set the flag on new Token

Effective Mitigation



Except When It's Not!

Issue 1544: Windows: Child Process Restriction Mitigation Bypass

Code ‹ Prev 79 of 94 Next › Back to list

Reported by forshaw@google.com, Mar 4 2018 **Project Member**

Windows: Child Process Restriction Mitigation Bypass
Platform: Windows 10 1709 (not tested other versions)
Class: Security Feature Bypass

Summary:

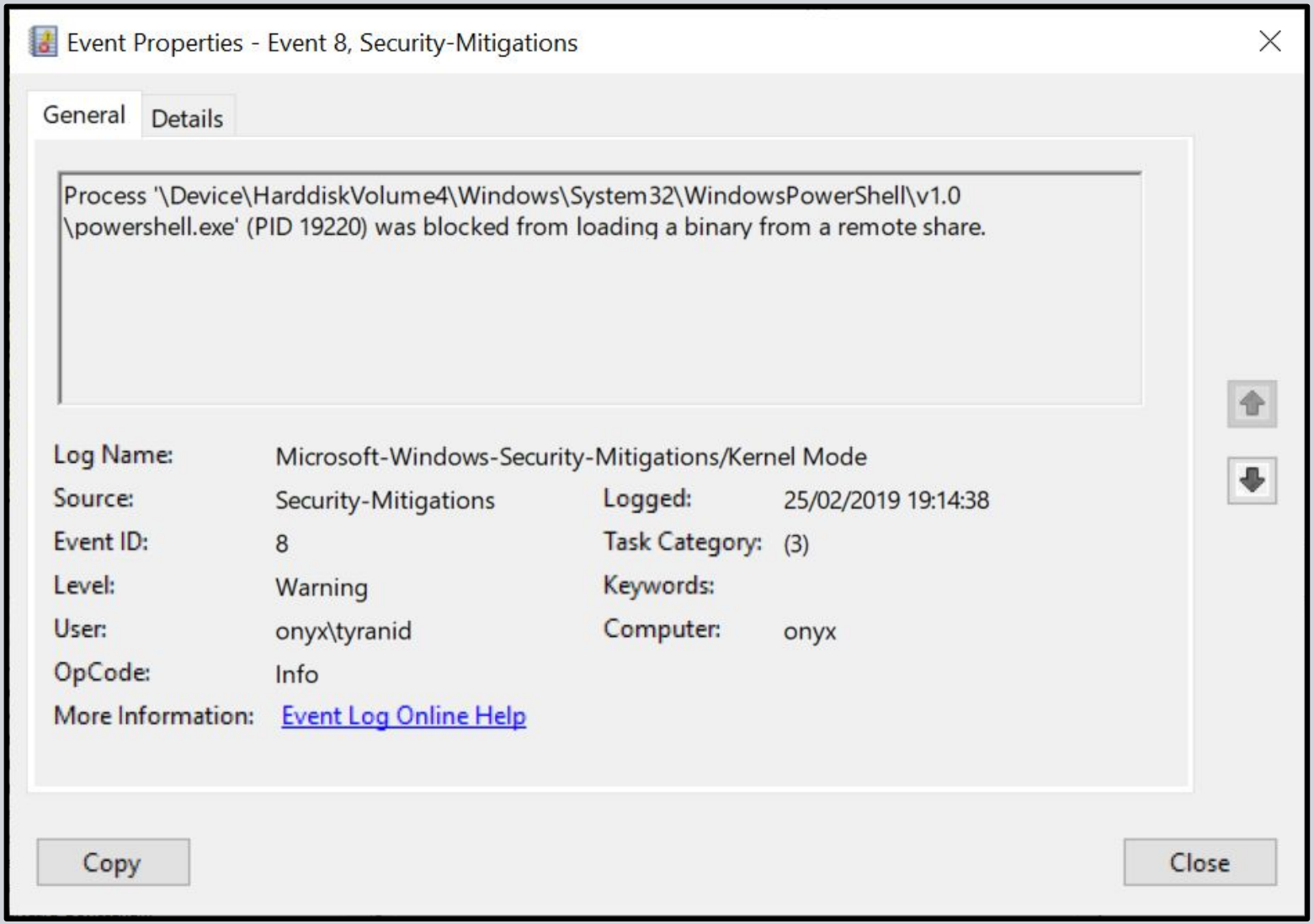
It's possible to bypass the child process restriction mitigation policy by impersonating the anonymous token leading to a security feature bypass.

Description:

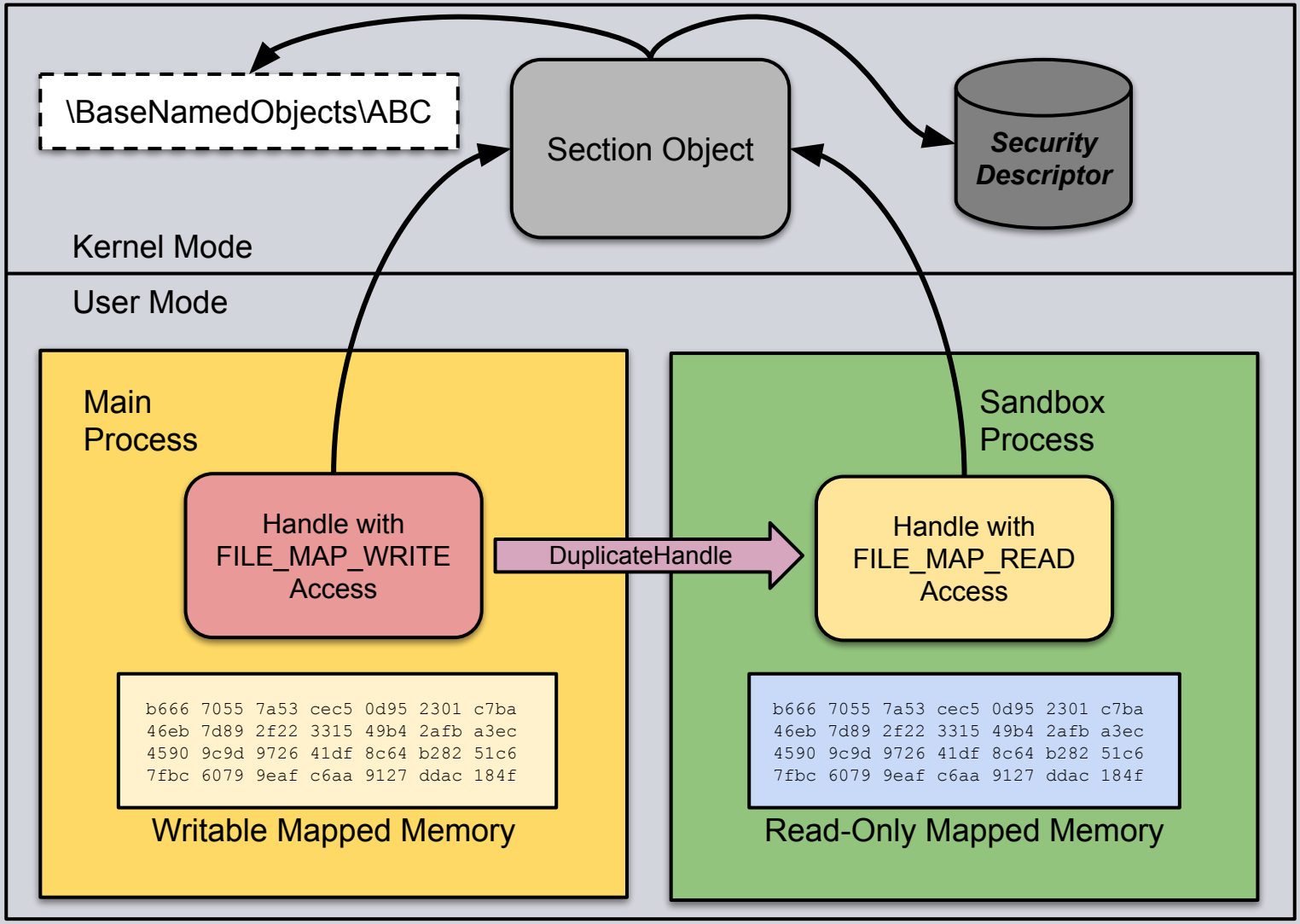
Windows 10 has a mitigation policy to restrict a process creating new child processes. I believe the main rationale is to prevent escaping some of the other mitigations which are not inherited across to new child processes as well as bugs which can only be exploiting from a fresh process. The policy is enforced as a flag in the token rather than on the process which allows the restriction to be passed across process boundaries during impersonation, which would also kill abusing WMI Win32_Process and similar.

https://bugs.chromium.org/p/project-zero/issues/detail?id=1544

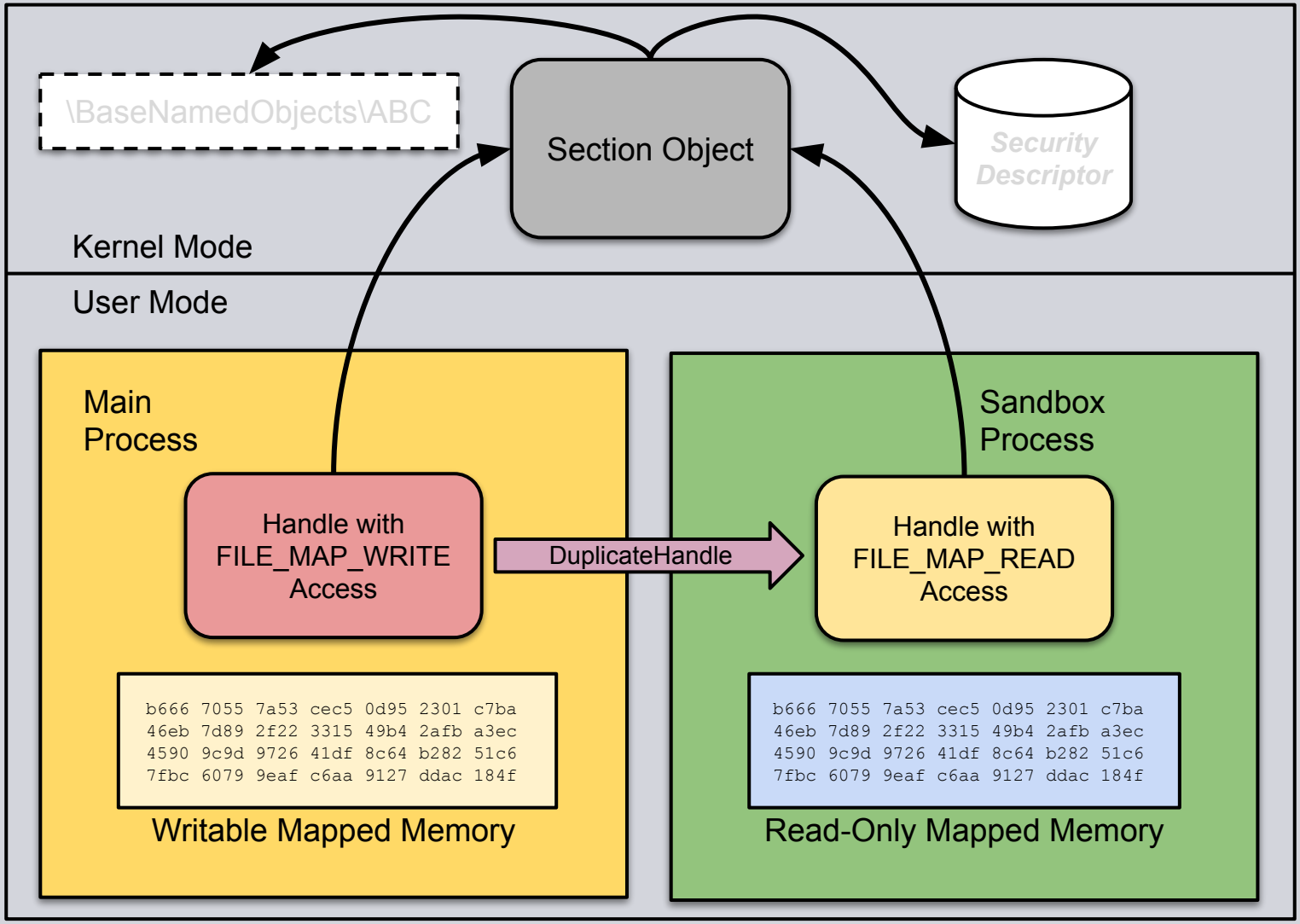
Logging for Mitigation



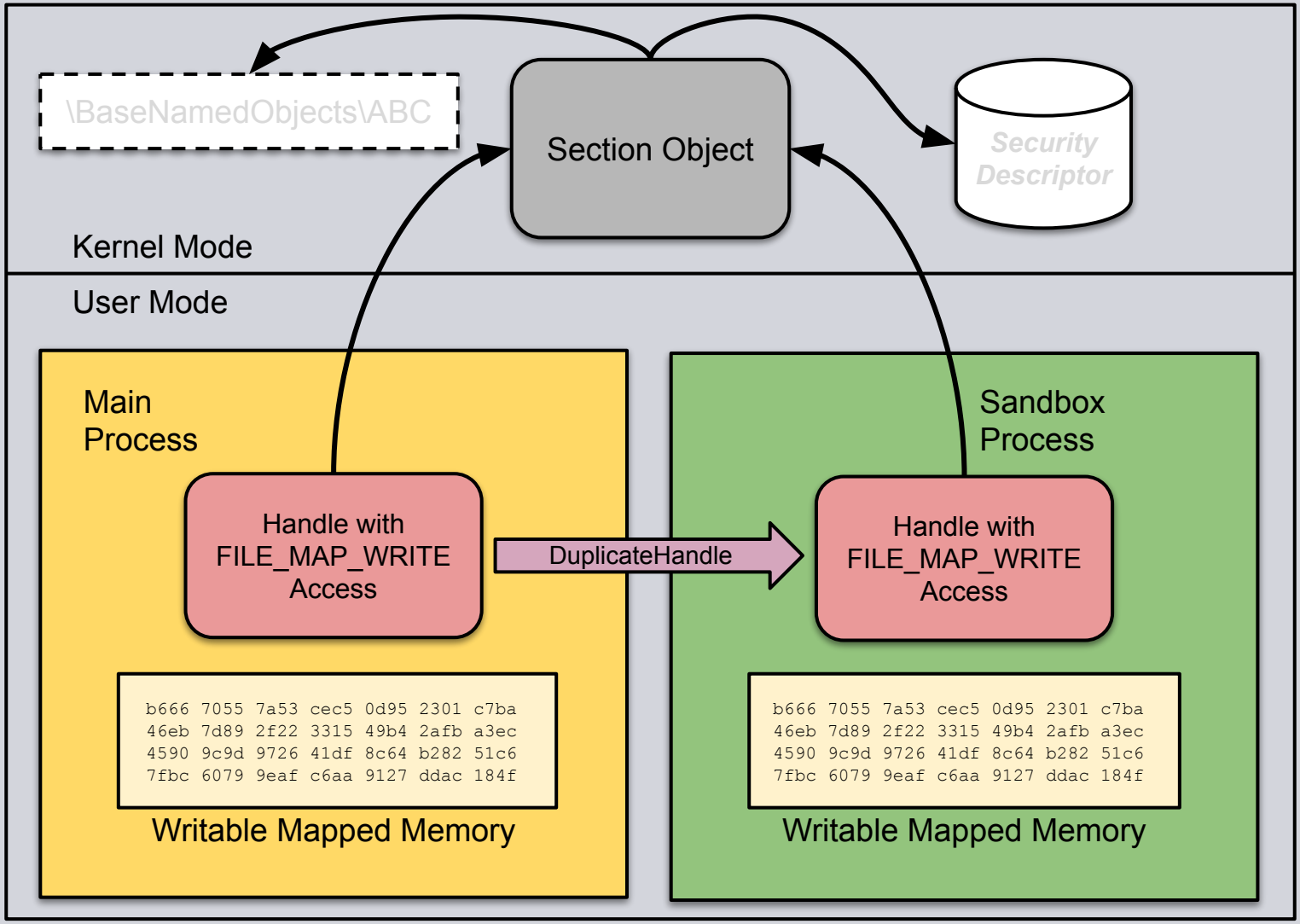
Sharing Sections



Sharing Sections

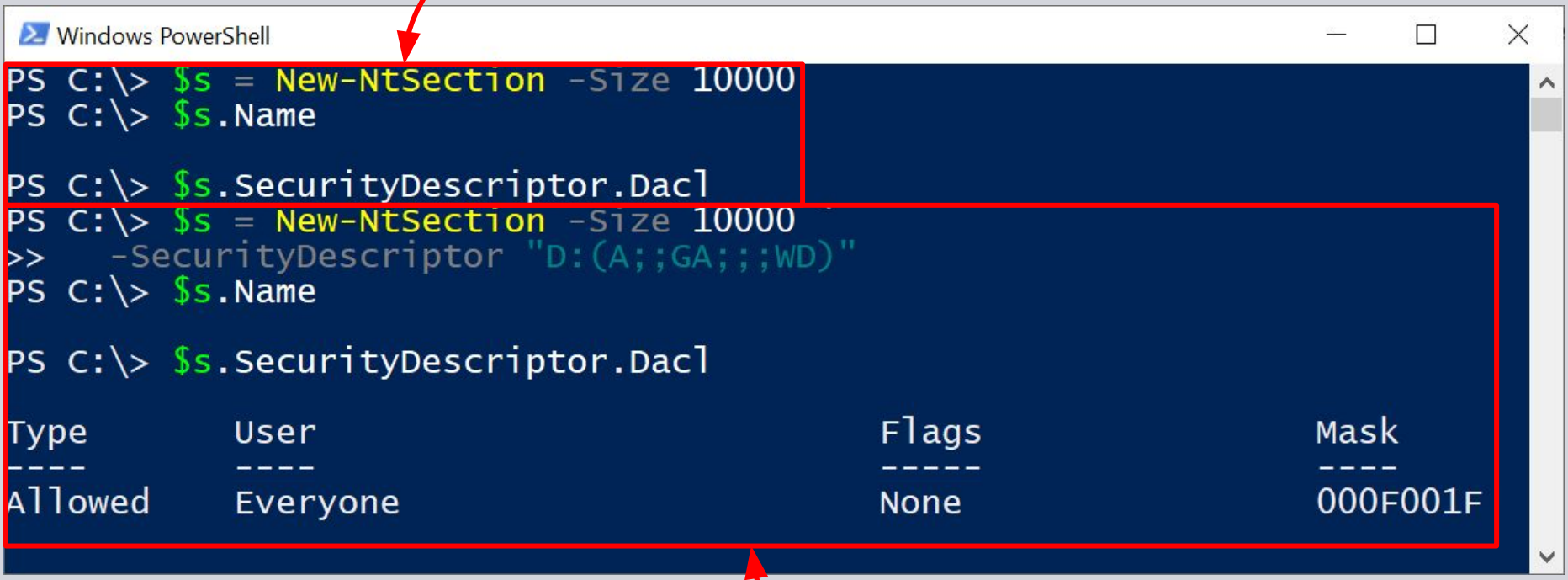


Sharing Sections



Name No Longer Needed

No Name - No SD



Specify SD to Create

Hyper-V Everywhere

- Windows Defender Application Guard
- Windows Sandbox (coming soon!)

Microsoft Hyper-v

PICO Process Available

root@onyx: ~

```
BITS 64

%define SYS_execve 59

_start:
  mov rax, SYS_execve
; Load the executable path
  lea rdi, [rel _exec_path]
; Load the executable path
  lea rsi, [rel _argument]
; Build argument array on stack = { _exec_path, _argument, NULL }
```

execve.asm 1,1 Top

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>

int main(int argc, char** argv) {
    struct sockaddr_un addr = {};
    int sock = -1;
```

afunix.c 1,1 Top

"afunix.c" 36L, 712C

DEMOS

All of the Above

Conclusions

- Introduction of Windows 10 Had Effect on Sandboxes
  - Edge gave Microsoft an excuse to innovate
  - Fast release cycle meant new mitigations could ship sooner
- Still plenty of things I'd like to see
  - Better system call filtering, NTOS as well as WIN32K
  - Improvements to Eliminate long standing warmup problems.

Thanks for Listening

Questions?