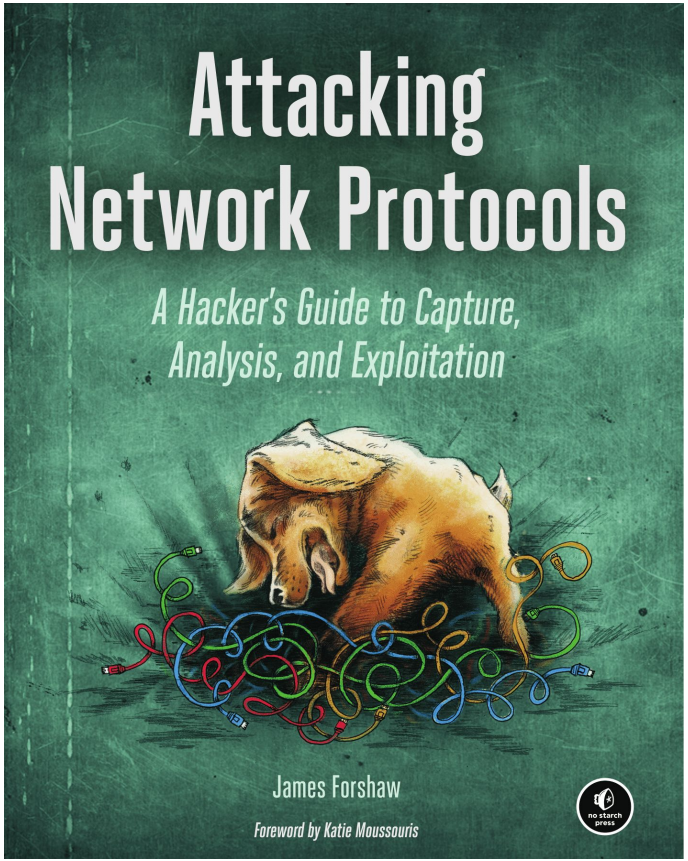

A BRIDGE TOO FAR
Exploiting Desktop Bridge on Windows
James Forshaw @tiraniddo
ZerOCon 2018
Photo by DAVID ILIFF. License: CC-BY-SA 3.0

Who am I?

- Researcher in Google's Project Zero
- Specialize in Windows
  - Especially local privilege escalation
  - Logical vulnerability specialist
- Author of a book on attacking network protocols
- @tiraniddo on Twitter.



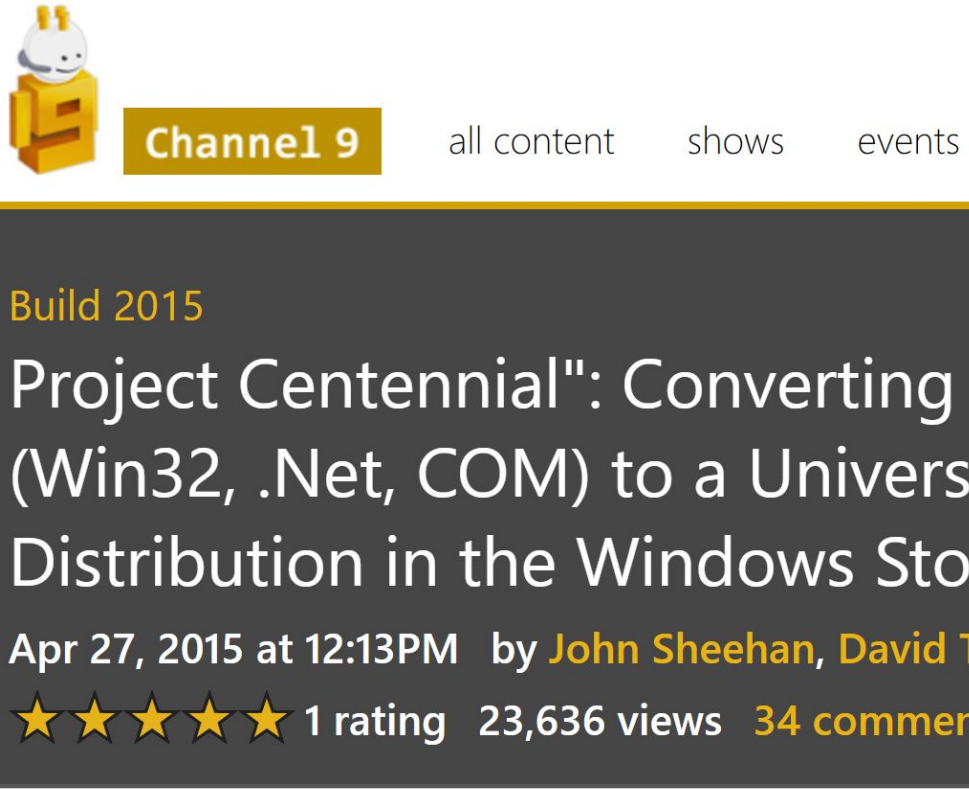
Agenda

- What is Desktop Bridge?
- Why do we care?
- How it works?
- How we can exploit it!

All research on Windows 10 RS2 + RS3. Not looked at RS4 yet.

https://www.jpl.nasa.gov/spaceimages/images/largesize/PIA18614_hires.jpg

Project Centennial to the Rescue?



https://channel9.msdn.com/Events/Build/2015/2-692?ocid=player

What Does Project Centennial Hope to Achieve?

Deploy Win32 Apps Through the Store

Simple Installation and Uninstallation

Convince Devs to Migrate to UWP

Desktop Bridge

Windows Dev Center

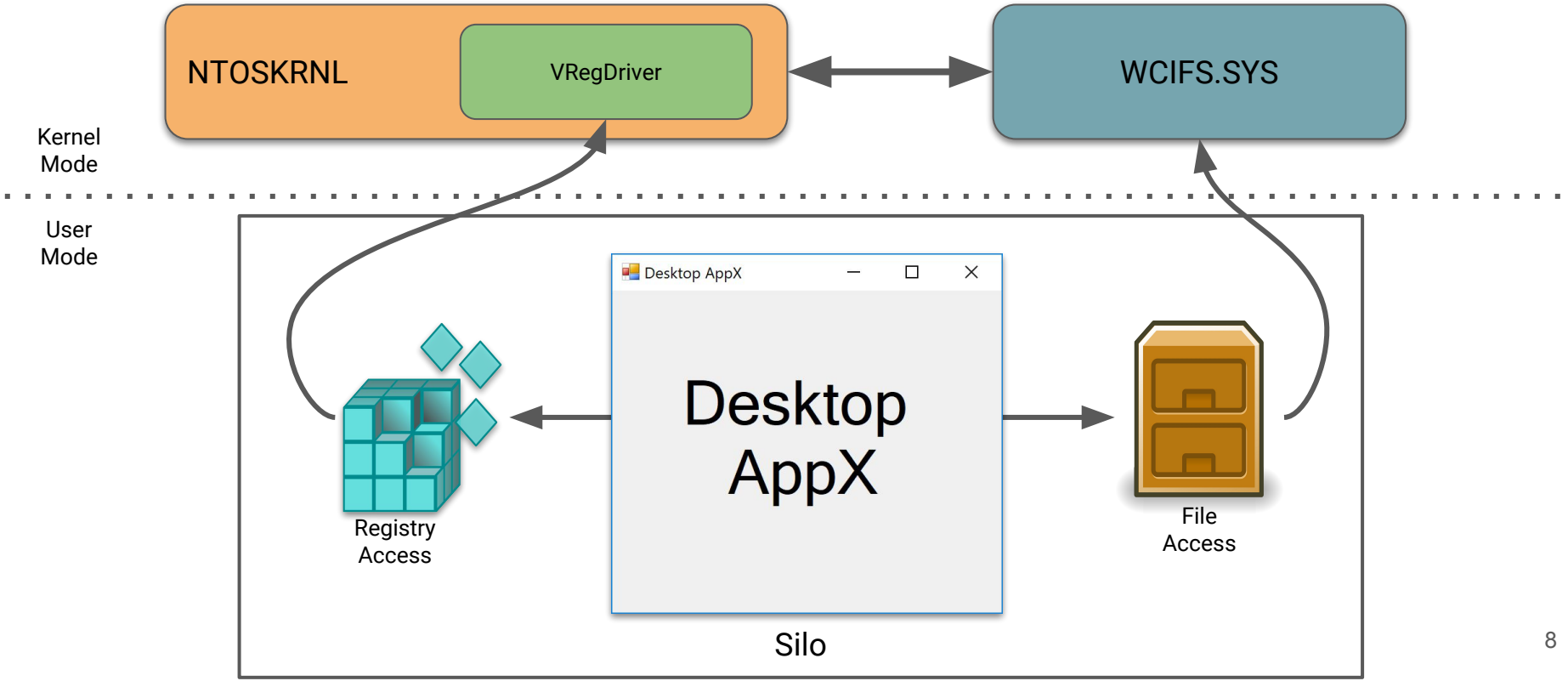
UWP apps > Windows Bridges > Desktop Bridge

Desktop Bridge

Bring existing apps and games to the Microsoft Store.

Learn more

Desktop Bridge Architecture



Silos

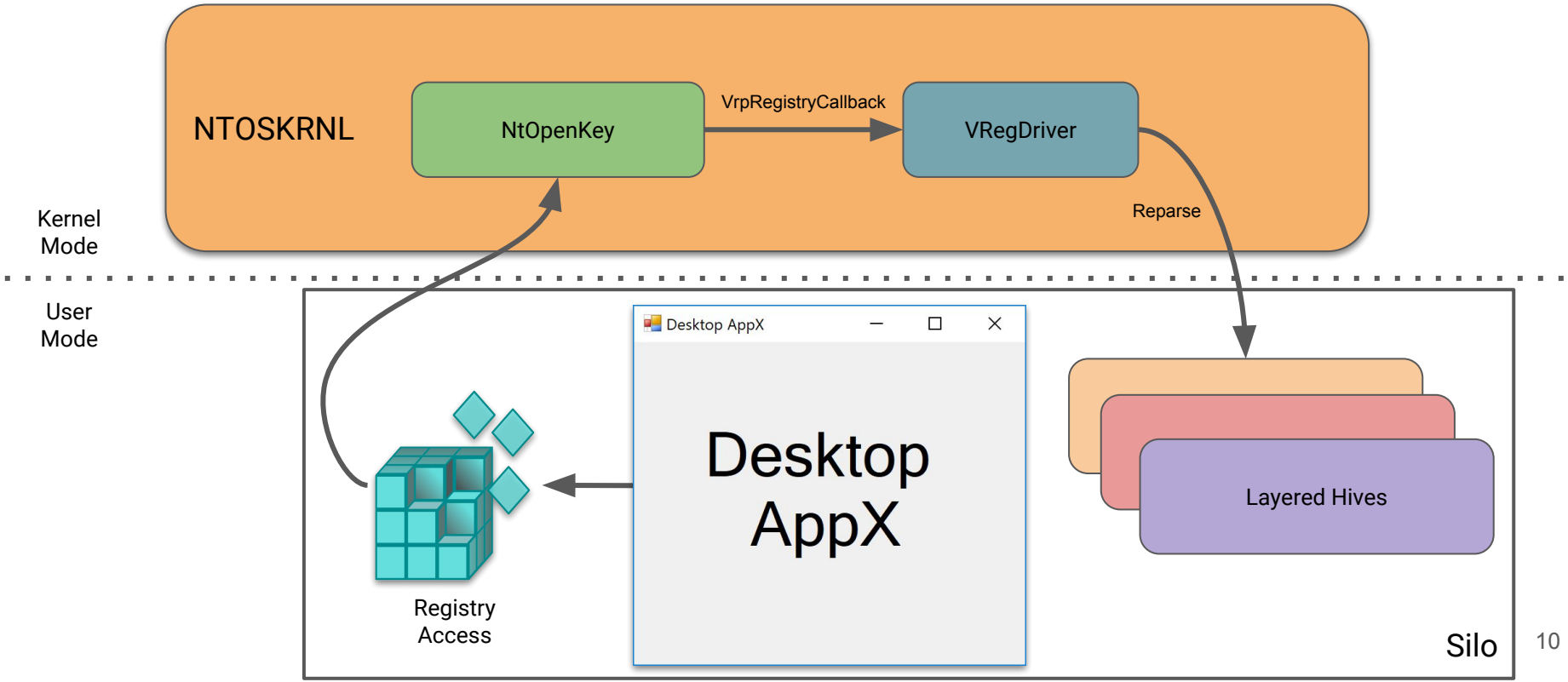
```
HANDLE hJob = CreateJobObject();
JOB_OBJECT_EXTENDED_EXTENDED_LIMIT_INFO info;
info.BasicLimitInformation.LimitFlags = 0x400000;

SetInformationJobObject(hJob,
                JobObjectExtendedLimitInformation,
                &info, sizeof(info));

SetInformationJobObject(hJob, JobObjectCreateSilo,
                                nullptr, 0);
```

Set new extra long limit info structure

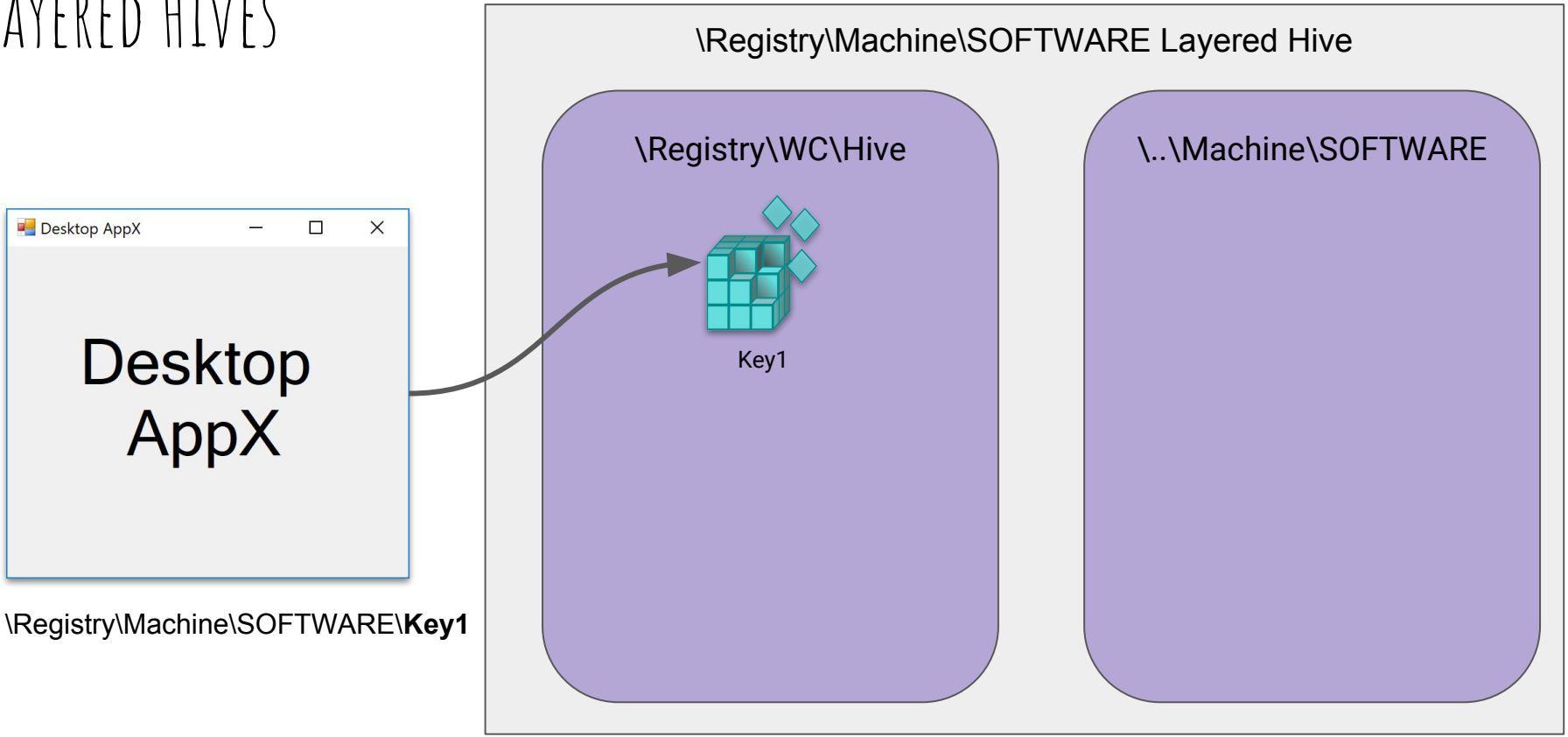
Undocumented "Application" Flag

Set Limit and Create Silo

Virtual Registry Driver

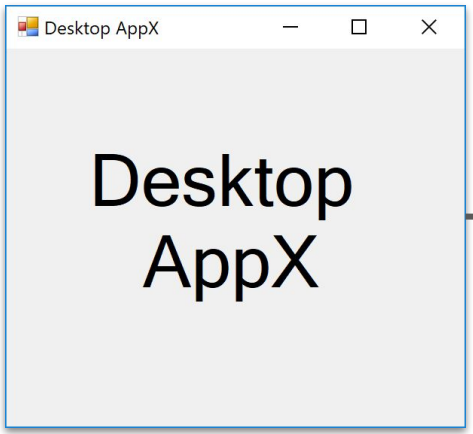


Layered Hives

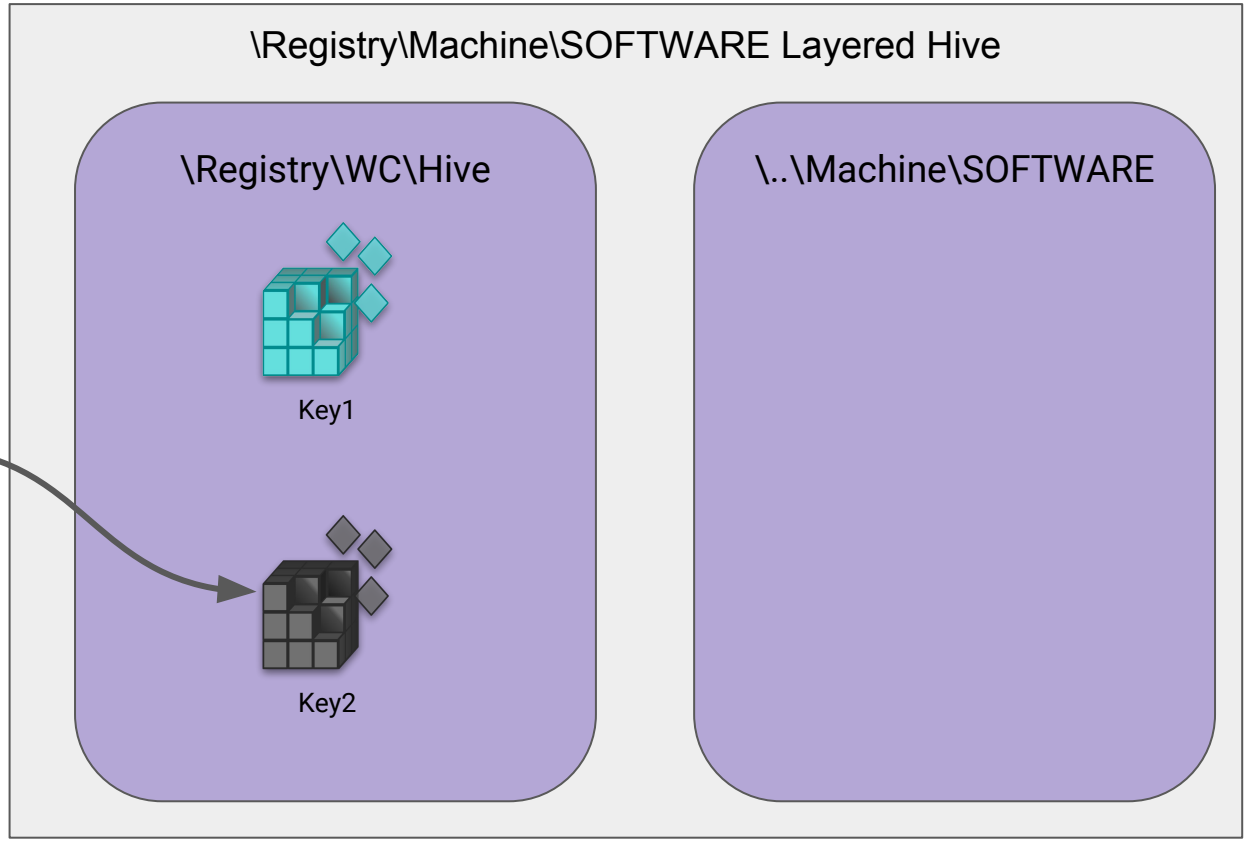


Open \Registry\Machine\SOFTWARE**Key1**

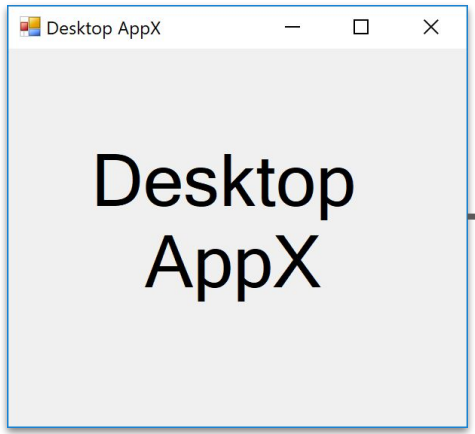
Layered Hives



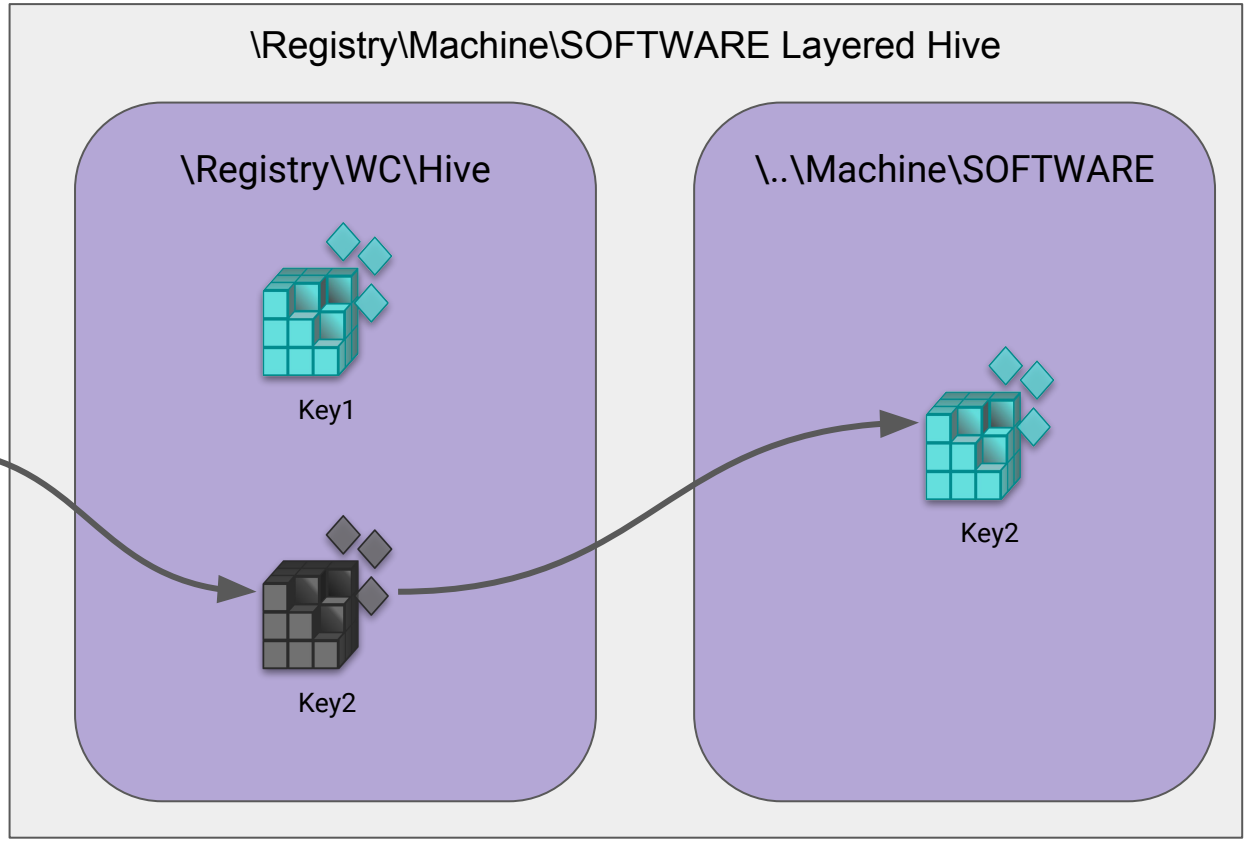
Open \Registry\Machine\SOFTWARE**Key2**



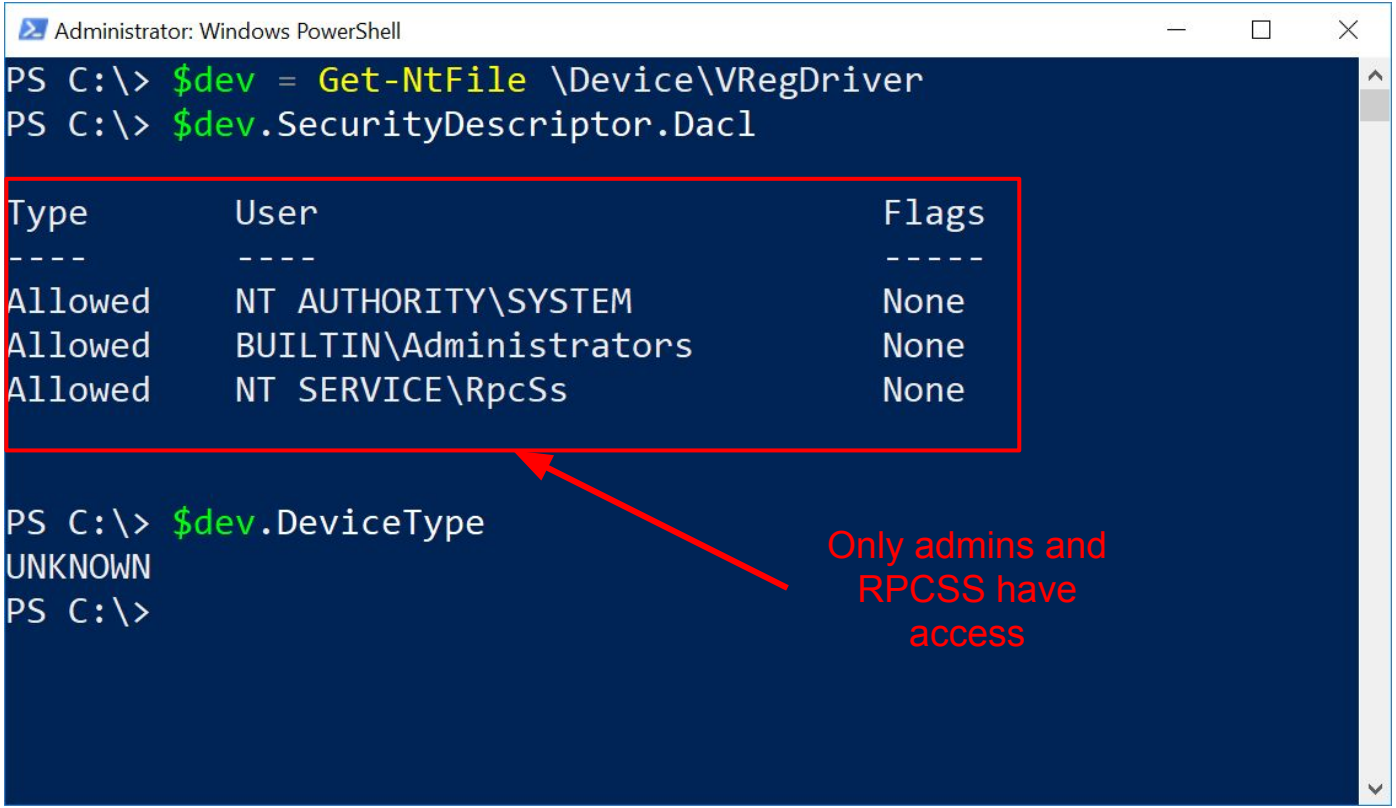
Layered Hives



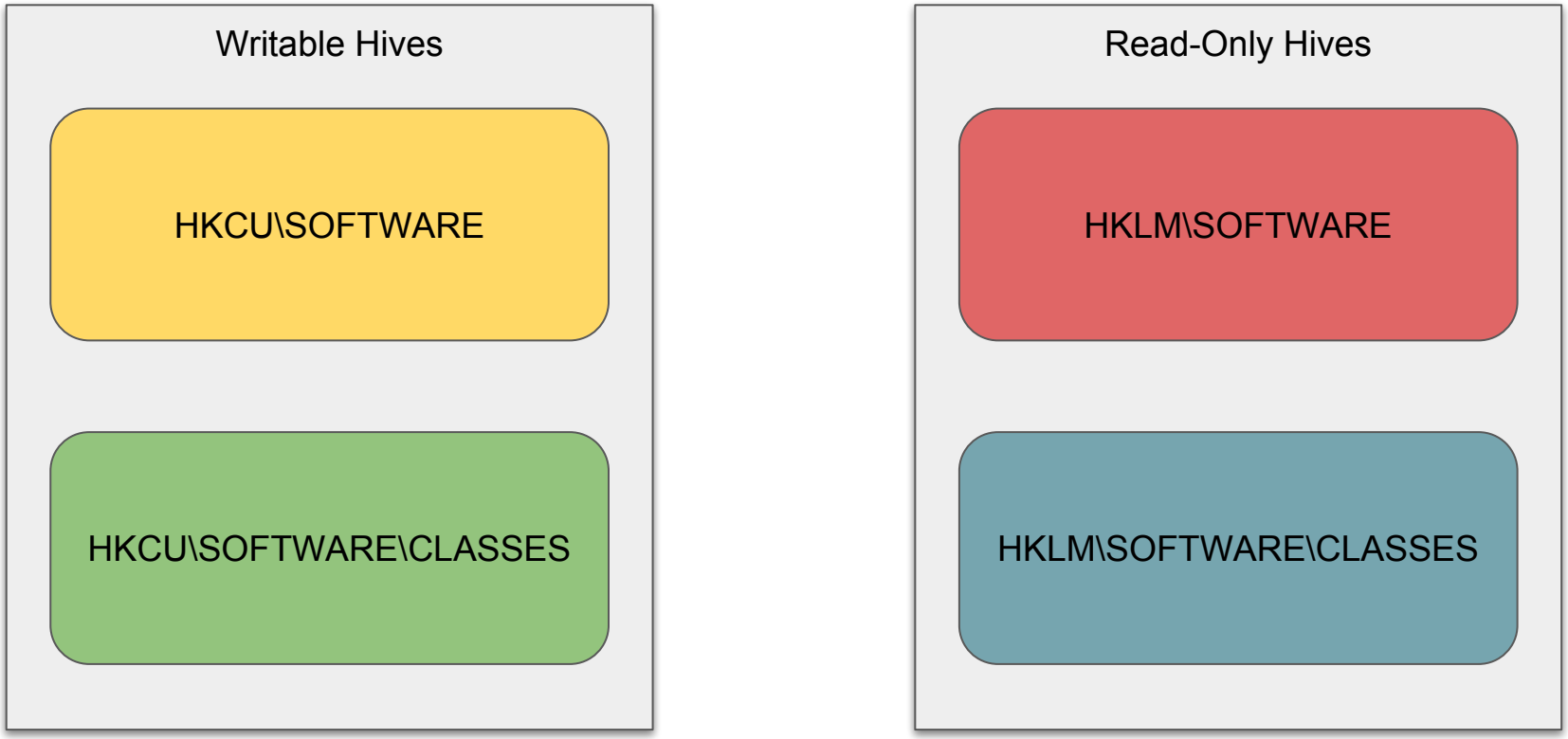
Open \Registry\Machine\SOFTWARE**Key2**



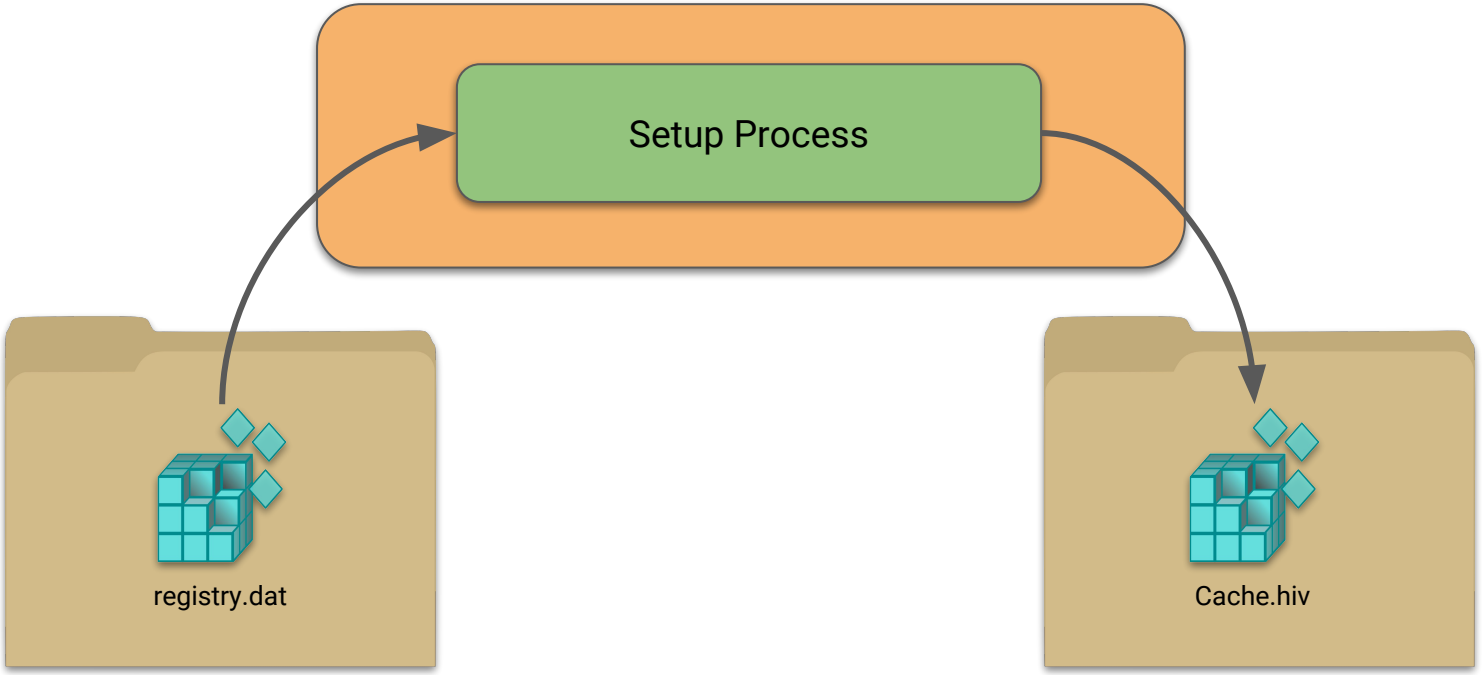
Virtual Registry Driver Device Object



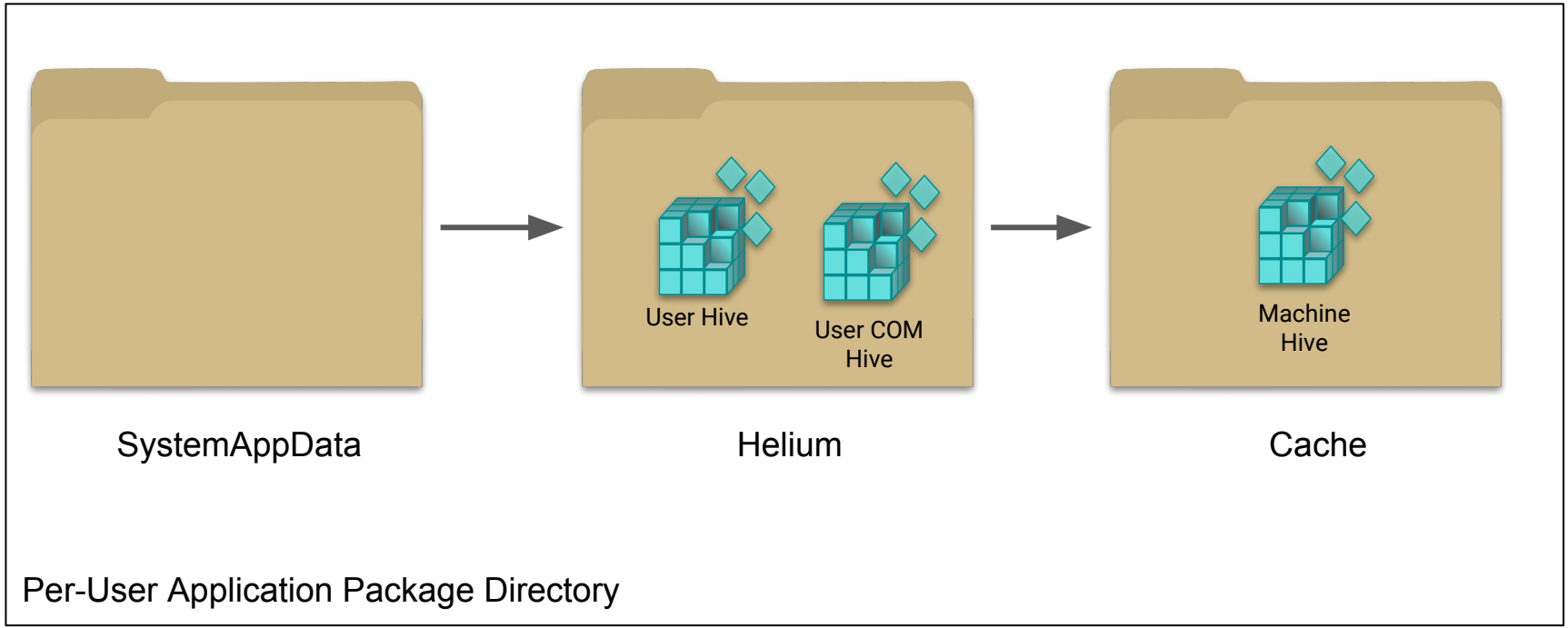
Differencing Hives



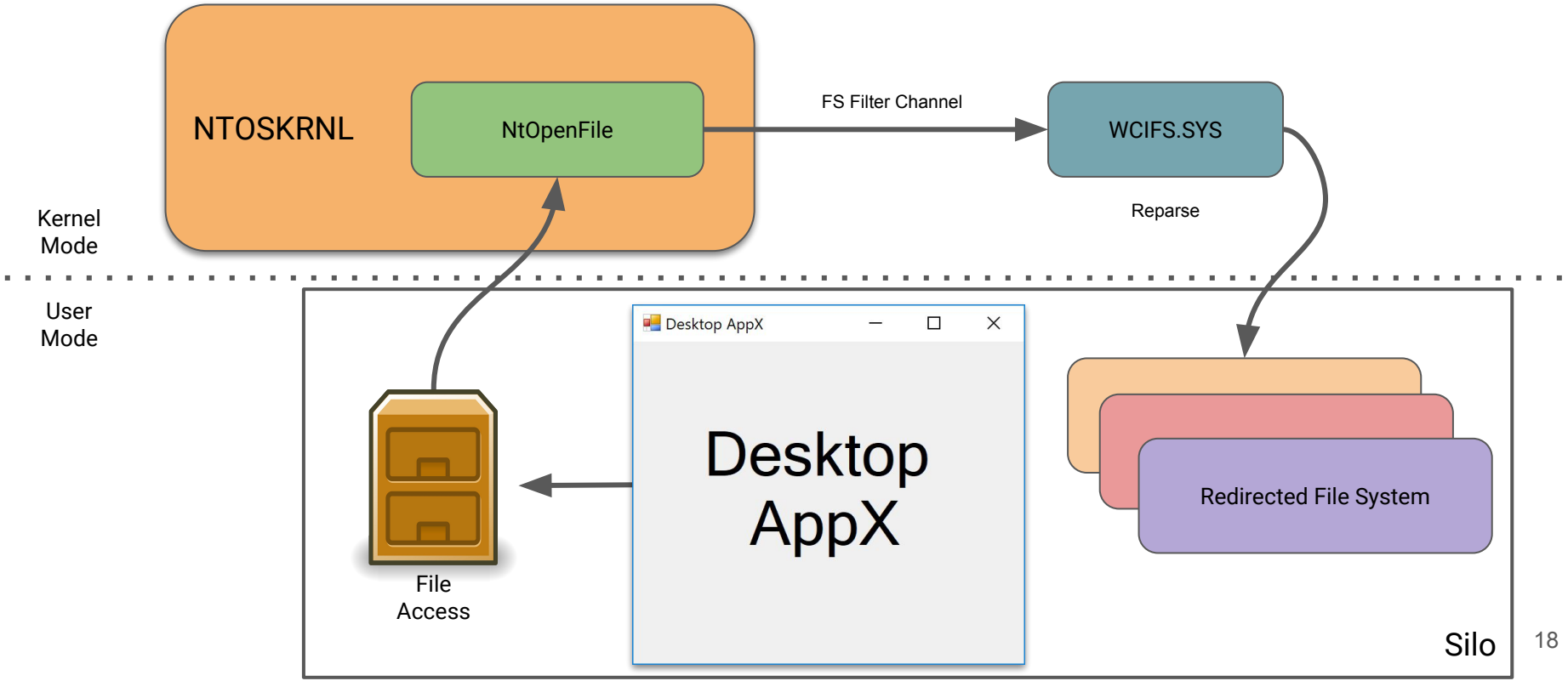
Setting up the Registry



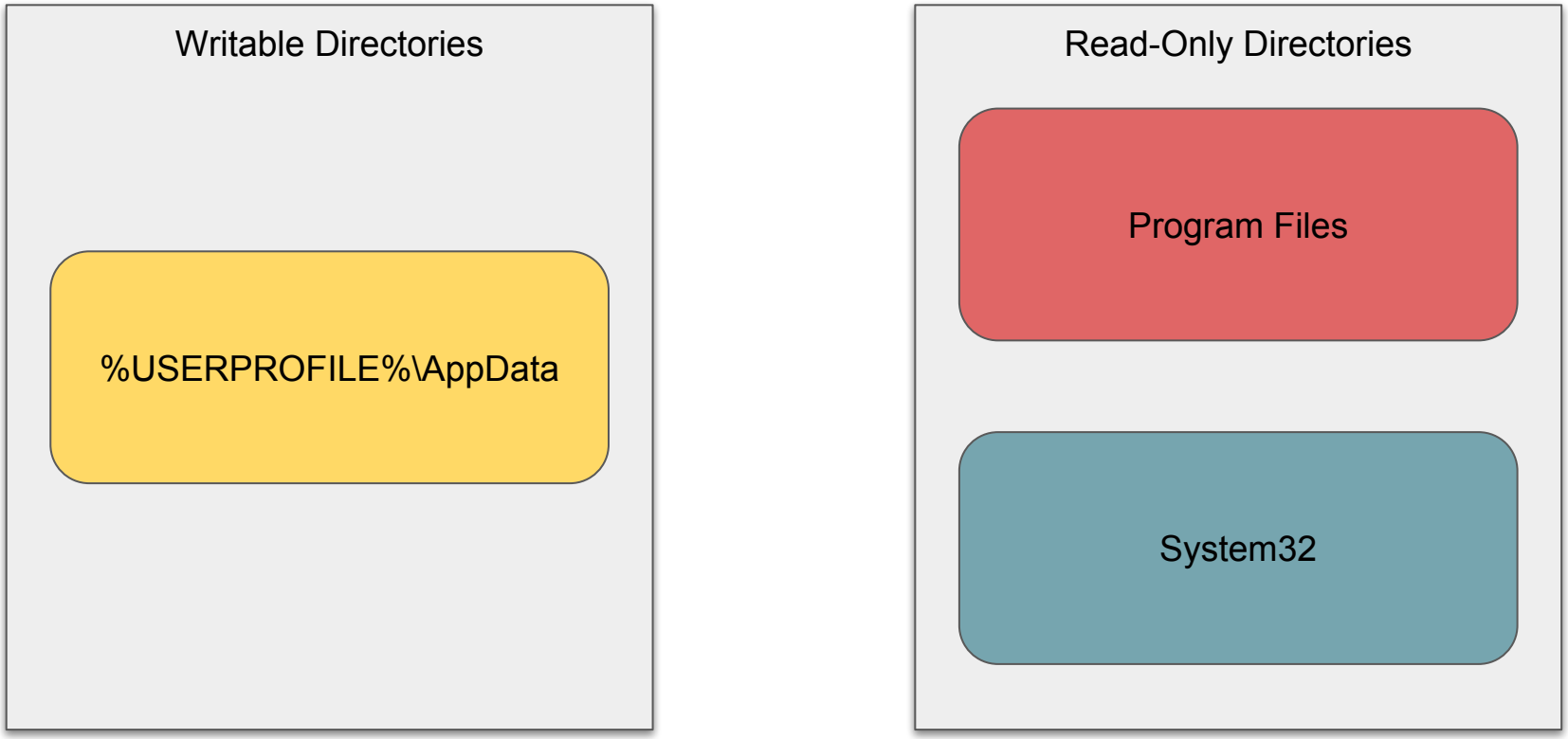
Registry Cache Files



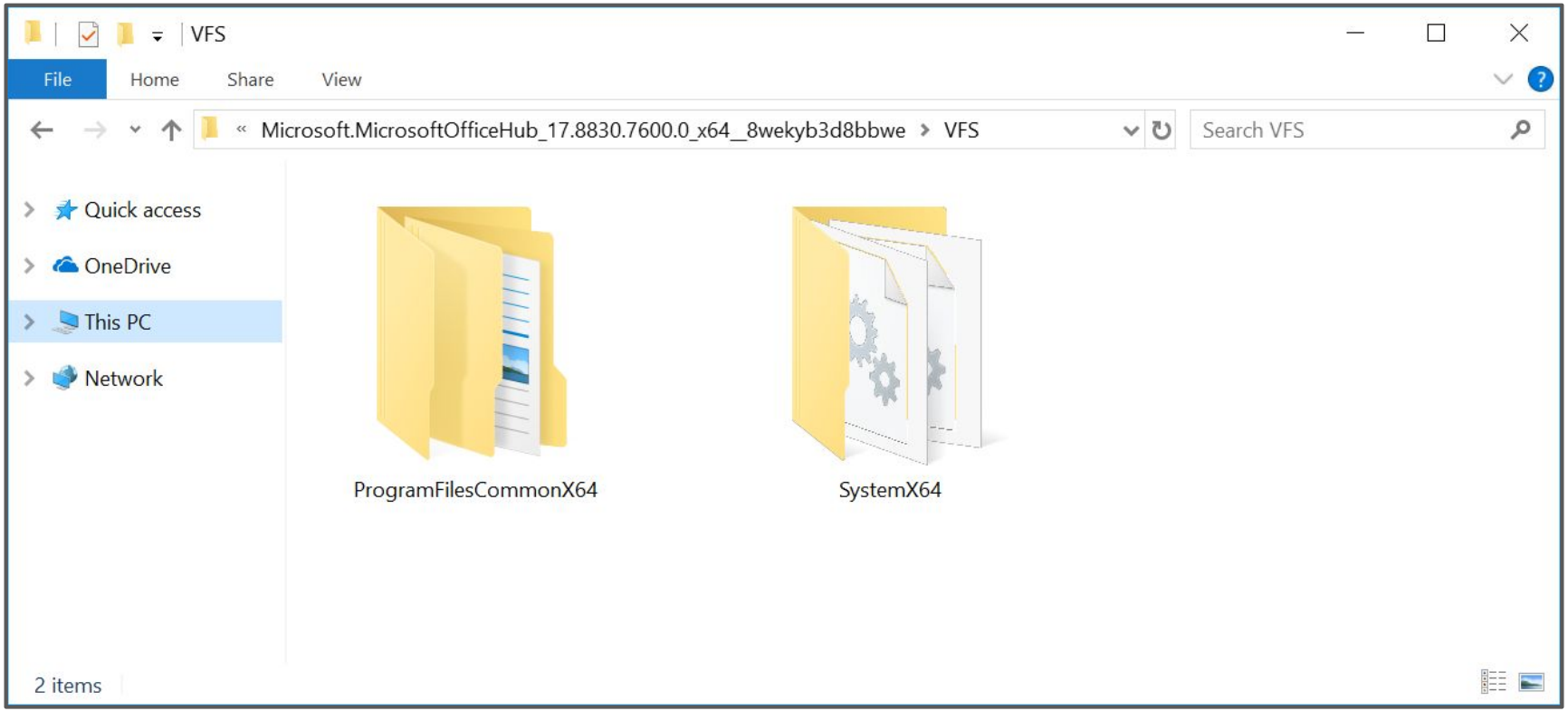
Windows Container Isolation Filesystem



Virtual Files



Windows Container Isolation Filesystem



Read-Only Virtual Filesystem Mappings

Known Folder	***VFS Directory***
FOLDERID_SystemX86	ApplicationDir\VFS\SystemX86
FOLDERID_System	ApplicationDir\VFS\SystemX64
FOLDERID_ProgramFilesX86	ApplicationDir\VFS\ProgramFilesX86
FOLDERID_ProgramFilesX64	ApplicationDir\VFS\ProgramFilesX64
FOLDERID_ProgramFilesCommonX86	ApplicationDir\VFS\ProgramFilesCommonX86
FOLDERID_ProgramFilesCommonX64	ApplicationDir\VFS\ProgramFilesCommonX64
FOLDERID_Windows	ApplicationDir\VFS\Windows
FOLDERID_ProgramData	ApplicationDir\VFS\Common AppData

WCIFS Filter Communication Port

```
HRESULT WciSetupFilter(...) {
  // ...
  WcpConstructContainerInfoMessage(&Message);
  FilterConnectCommunicationPort(
              L"\\WcifsPort", &hPort);
  FilterSendMessage(hPort, Message, Size);
  // ...
}
```

https://upload.wikimedia.org/wikipedia/commons/b/bb/Bridge_in_Seoul_-_night_time.JPG

Use the Converter

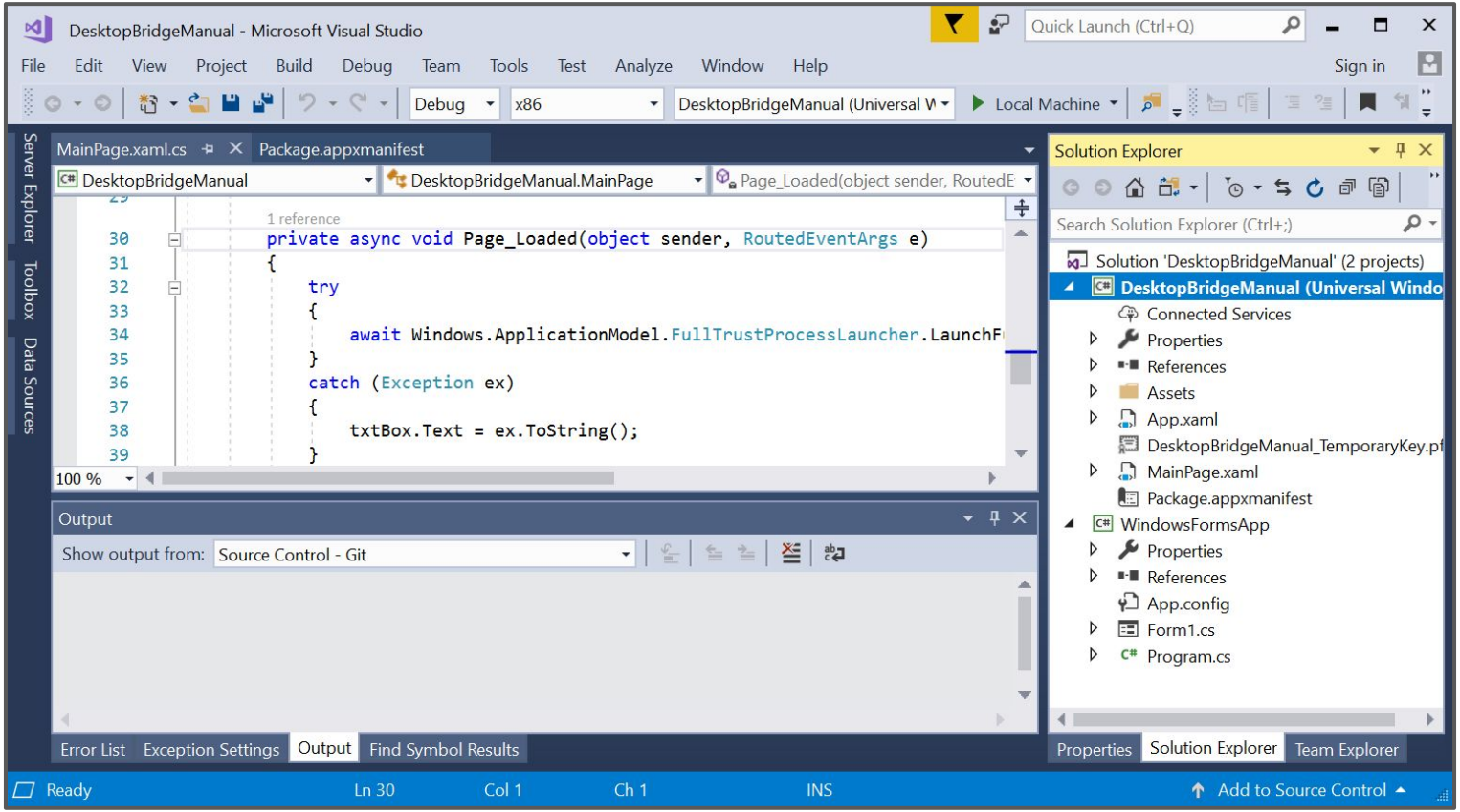
Package an app using the Desktop App Converter (Desktop Bridge)

09/18/2017 • 16 minutes to read • Contributors 

[Get the Desktop App Converter]

You can use the Desktop App Converter (DAC) to bring your desktop apps to the Universal Windows Platform (UWP). This includes Win32 apps and apps that you've created by using .NET 4.6.1.

Use Visual Studio



Activation Through AppInfo Server

```
[
    uuid (0497B57D-2E66-424F-A0C6-157CD5D41700),
    version(1.0),
]
interface LaunchDesktopAppX {
    long RAiLaunchProcessWithIdentity(...);
    long RAiGetPackageActivationToken(...);
    long RAiFinishPackageActivation(...);
    long RAiEnsurePackageShutdown(...);
}
```

Activation from an AppContainer

```
<Application Id="App"
    Executable="app.exe"
    EntryPoint="App.App">
  <desktop:Extension
            Category="windows.fullTrustProcess"
            Executable="Win32.exe" />
</Application>
<Capabilities>
  <rescap:Capability Name="runFullTrust" />
</Capabilities>
```

Specify fullTrustProcess extension

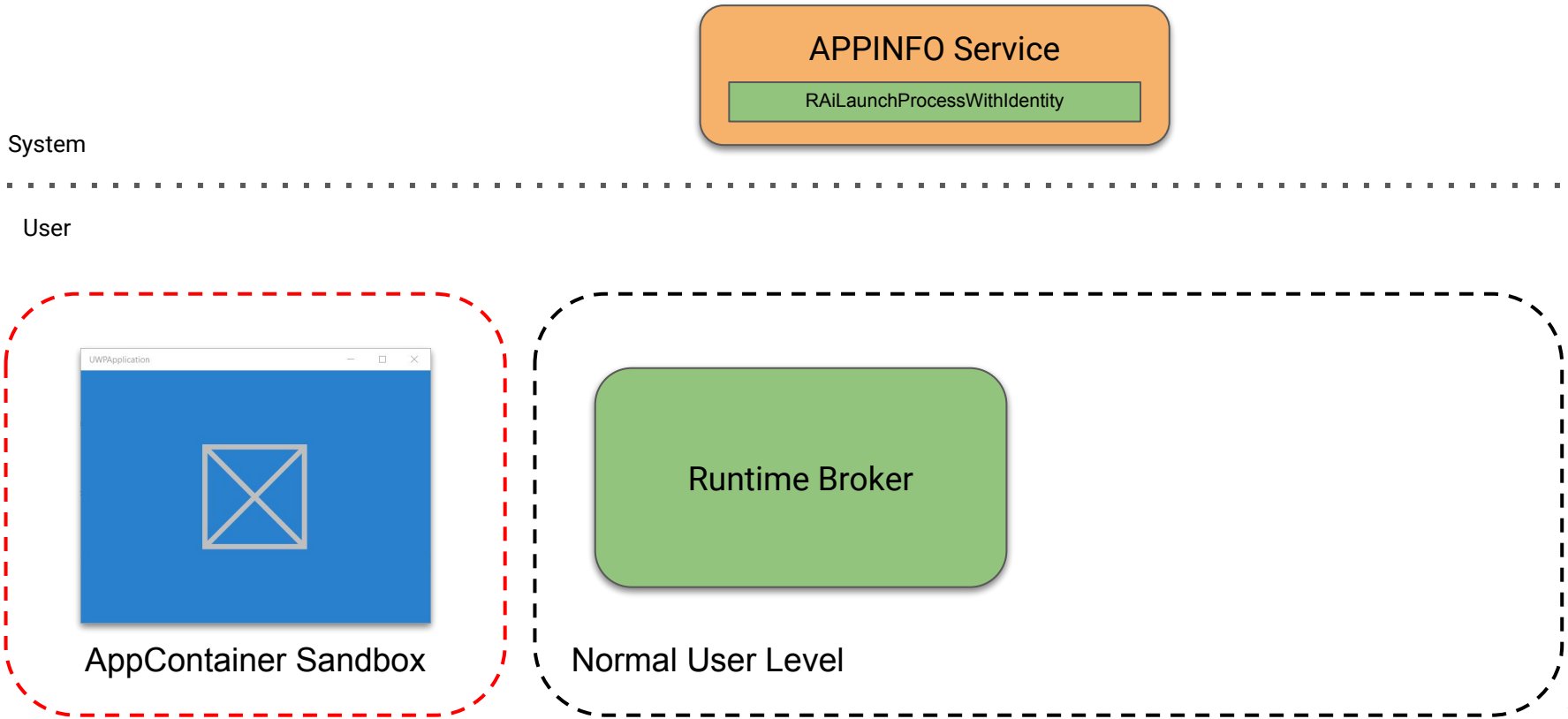
Path to executable in package

Specify runFullTrust capability

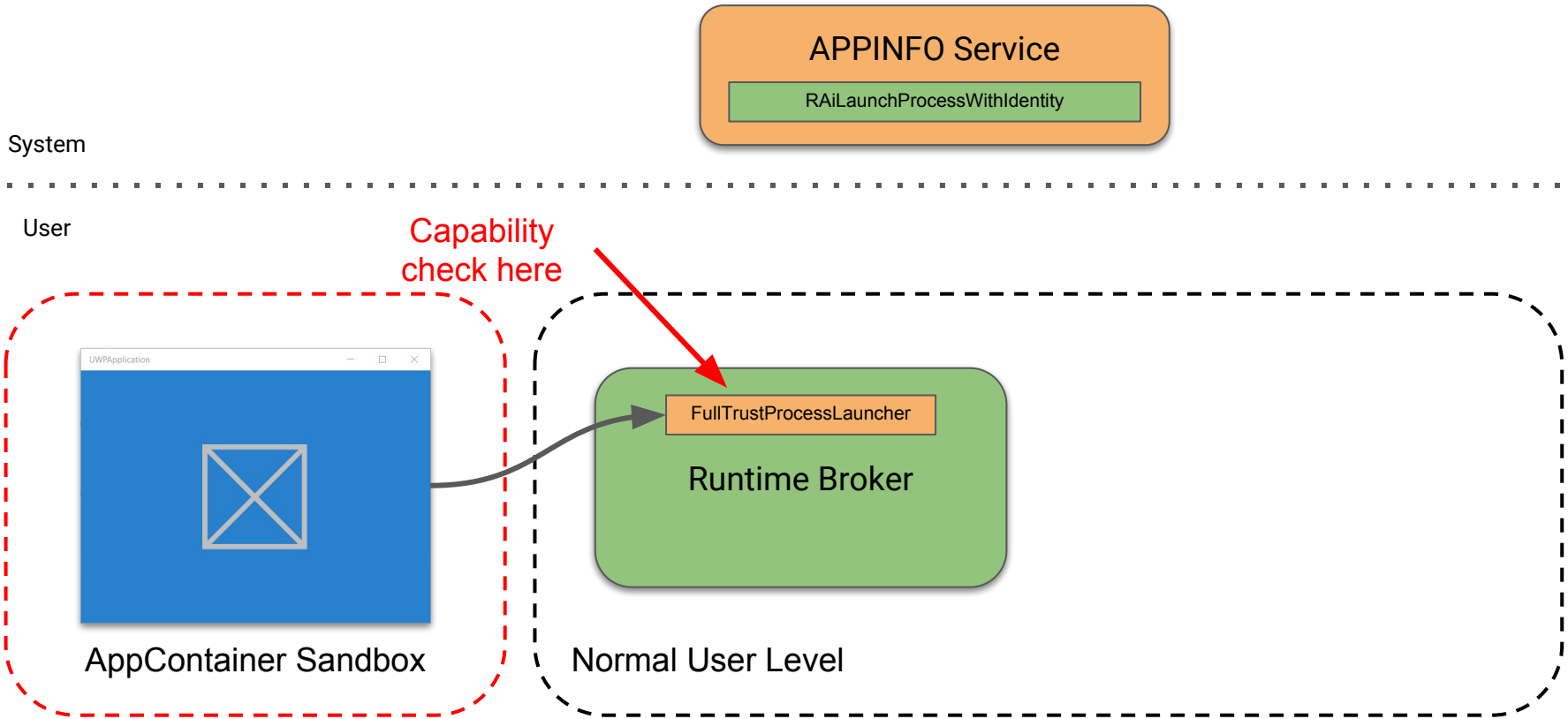
Activation from an AppContainer

```
async void StartDesktopBridgeProcess()
{
  await Windows.
        ApplicationModel.
        FullTrustProcessLauncher.
        LaunchFullTrustProcessForCurrentAppAsync();
}
```

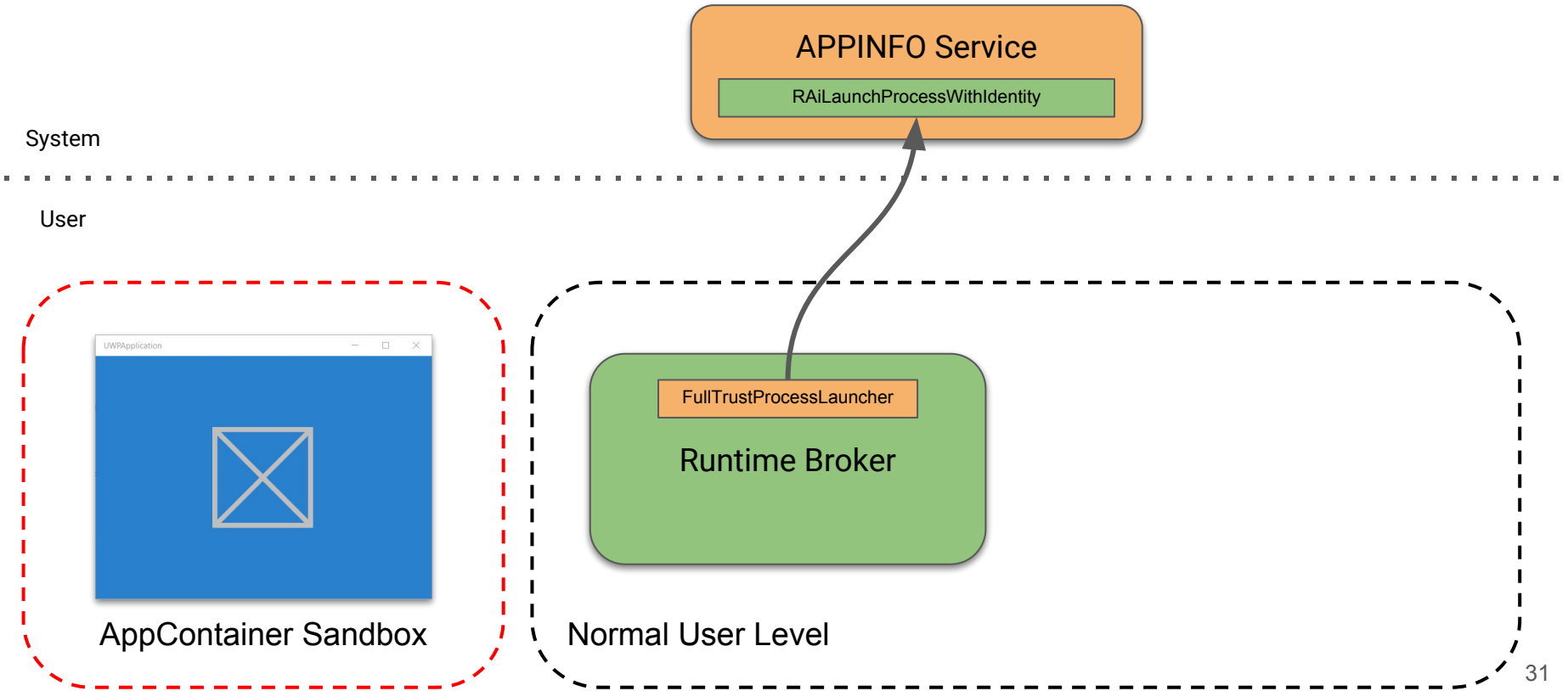
Activation from an AppContainer



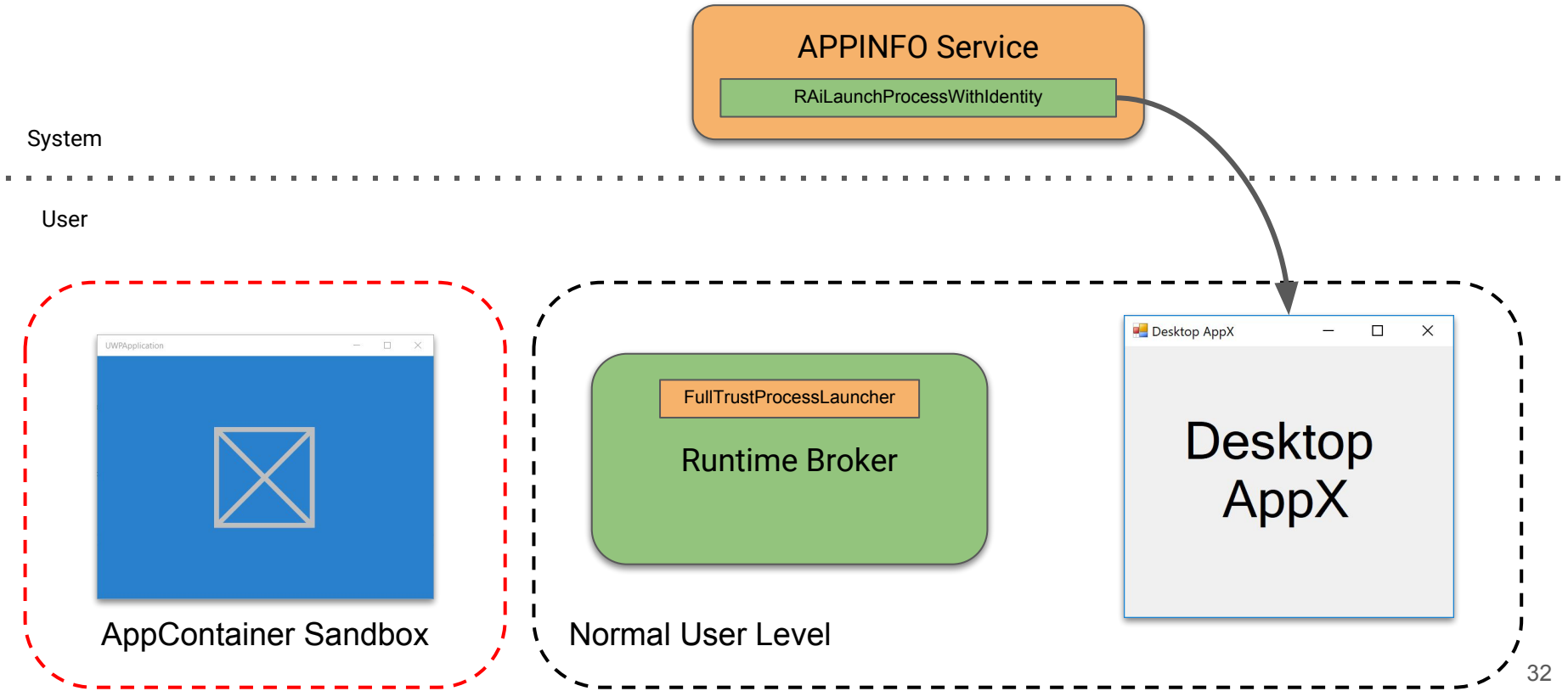
Activation from an AppContainer



Activation from an AppContainer



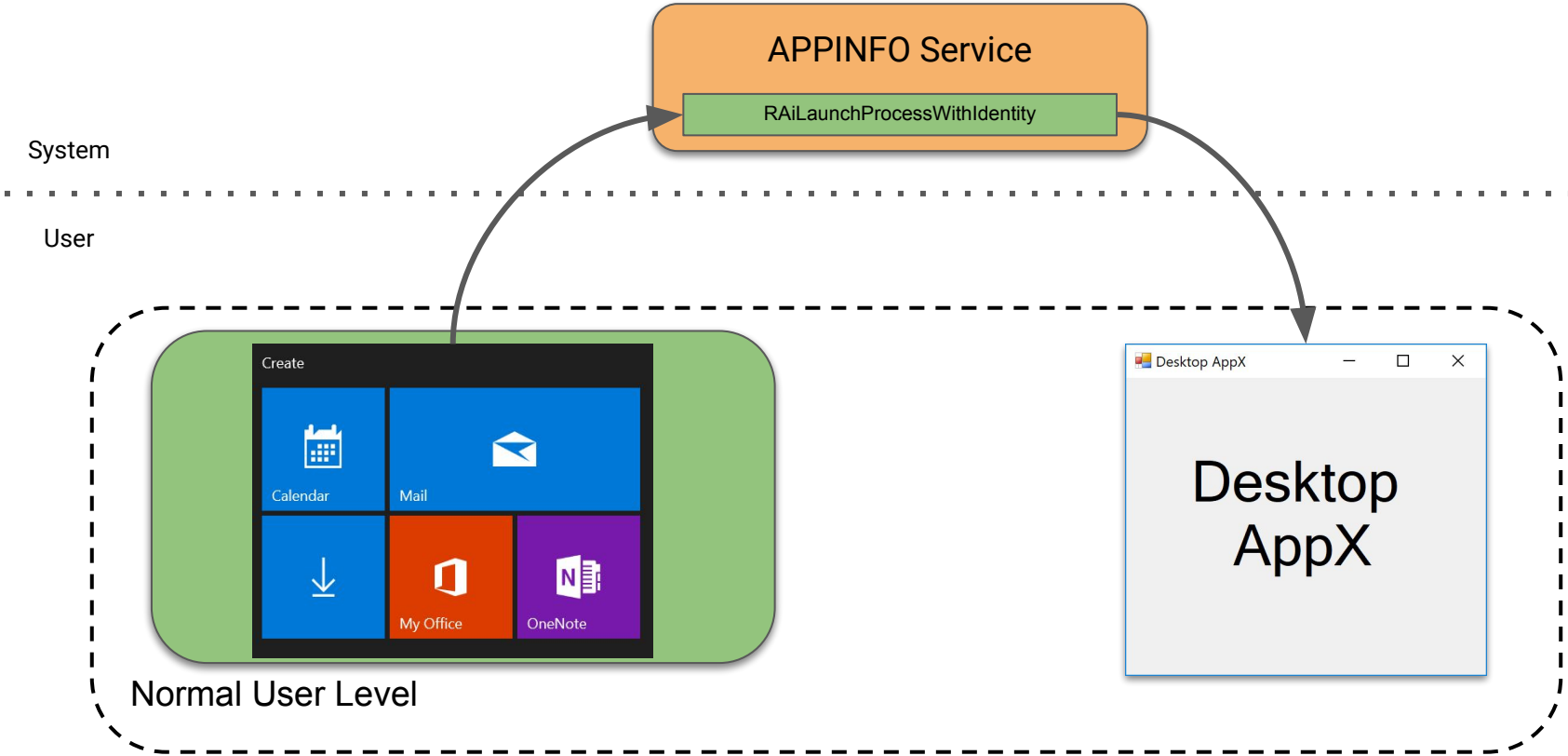
Activation from an AppContainer



Direct Activation

```
<Application Id="App"
        Executable="app.exe"
        EntryPoint="Windows.FullTrustApplication">
</Application>
<Capabilities>
    <rescap:Capability Name="runFullTrust" />
</Capabilities>
```

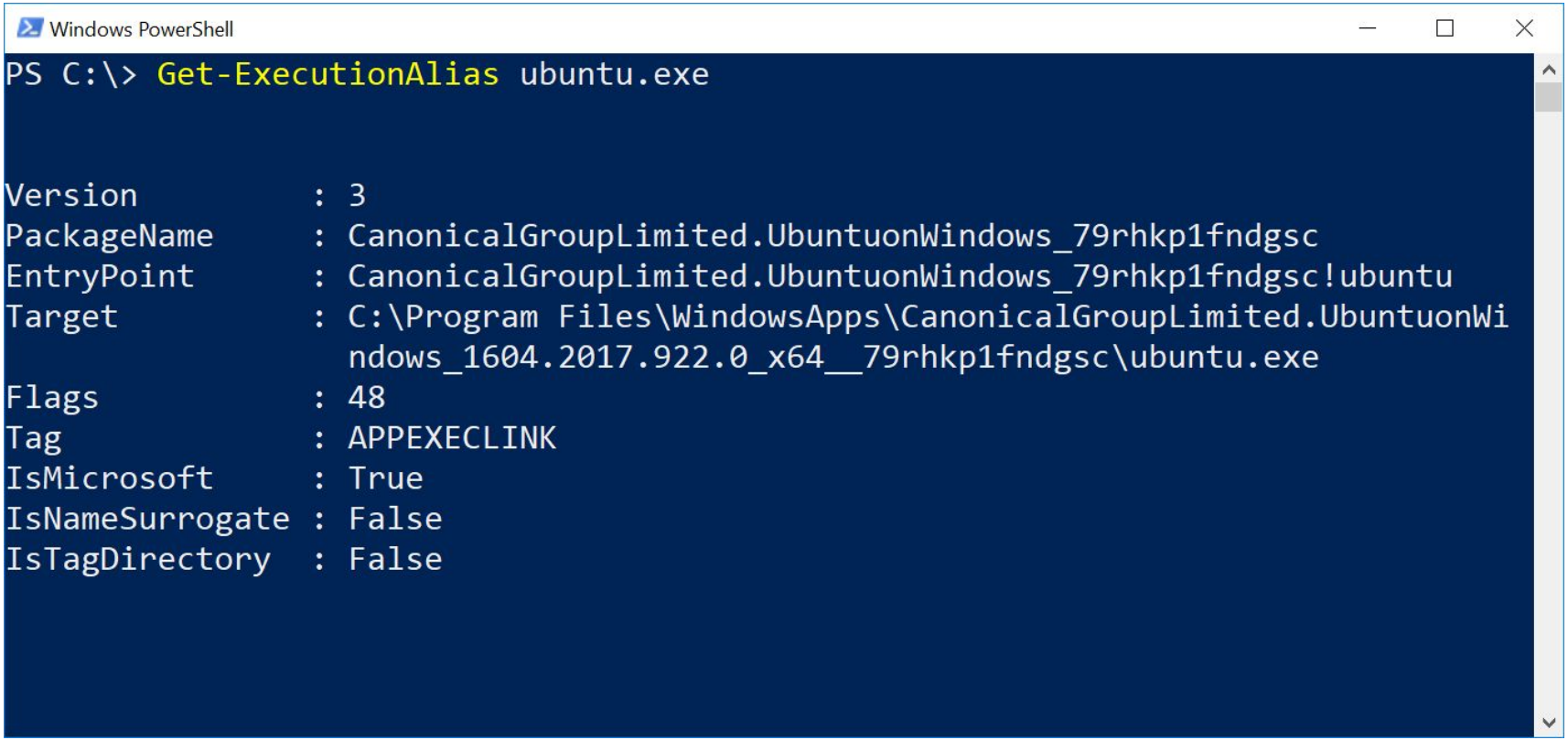
Direct Activation



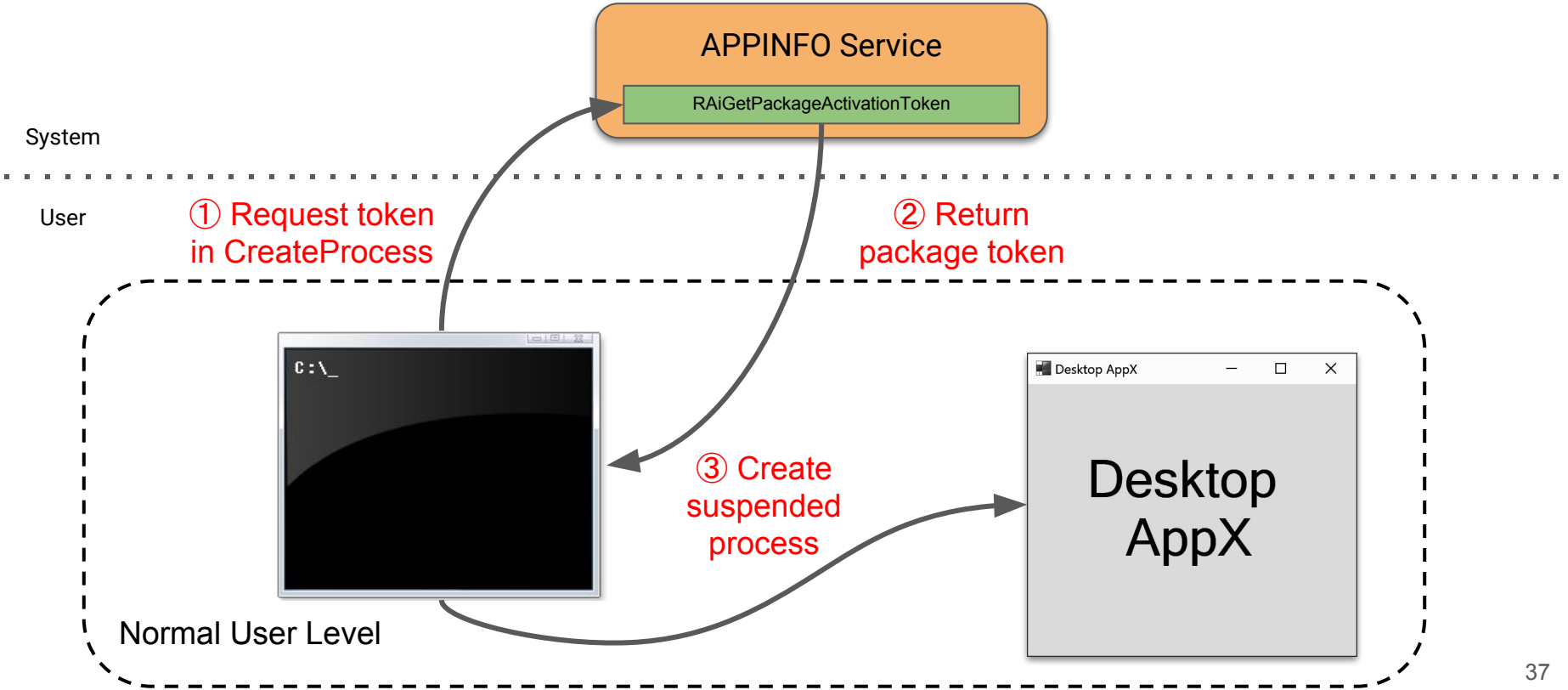
App Execution Alias

```
<Extension
  Category="windows.appExecutionAlias"
  Executable="app.exe"
  EntryPoint="Windows.FullTrustApplication">
 <AppExecutionAlias>
   <desktop:ExecutionAlias Alias="runapp.exe" />
 </AppExecutionAlias>
</Extension>
```

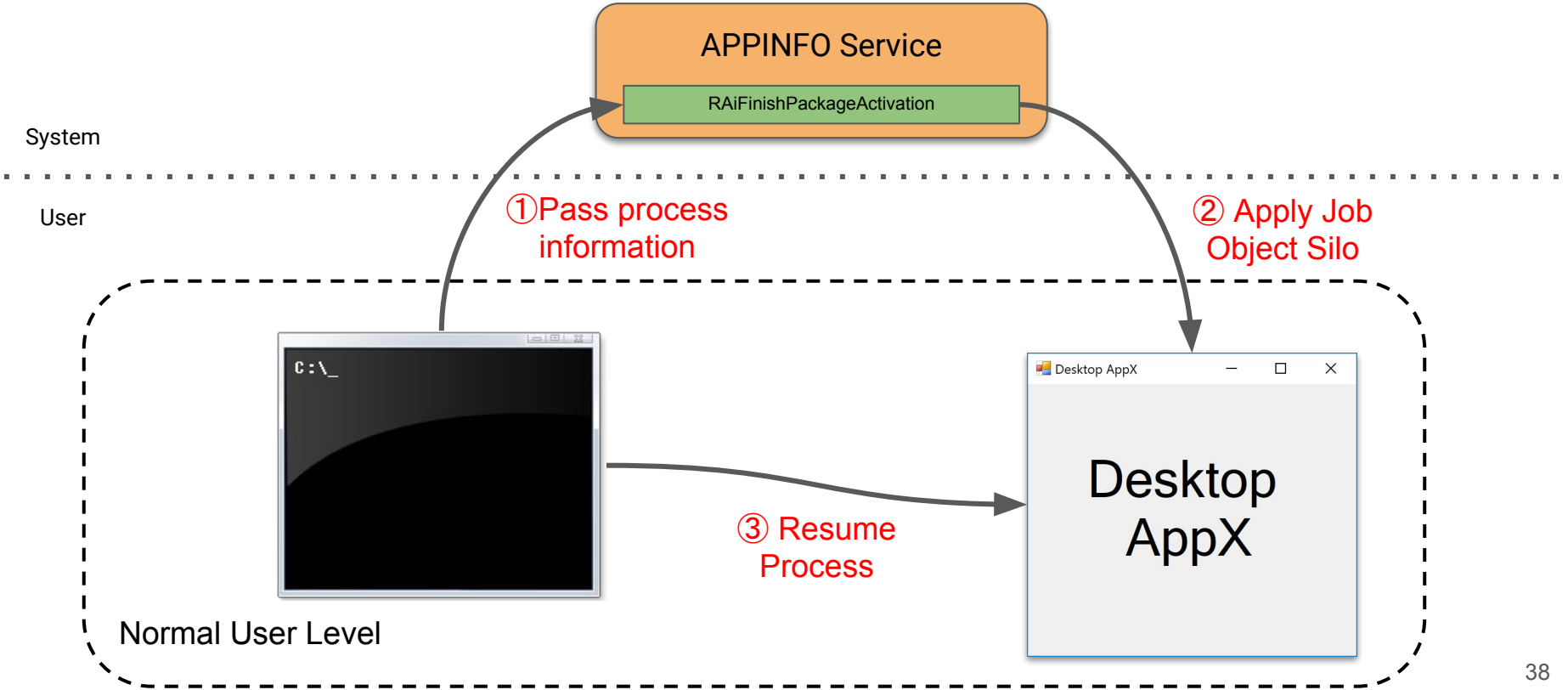
App Execution Alias



Start App Execution Alias Activation



Finish App Execution Alias Activation



Quick Activation for Testing

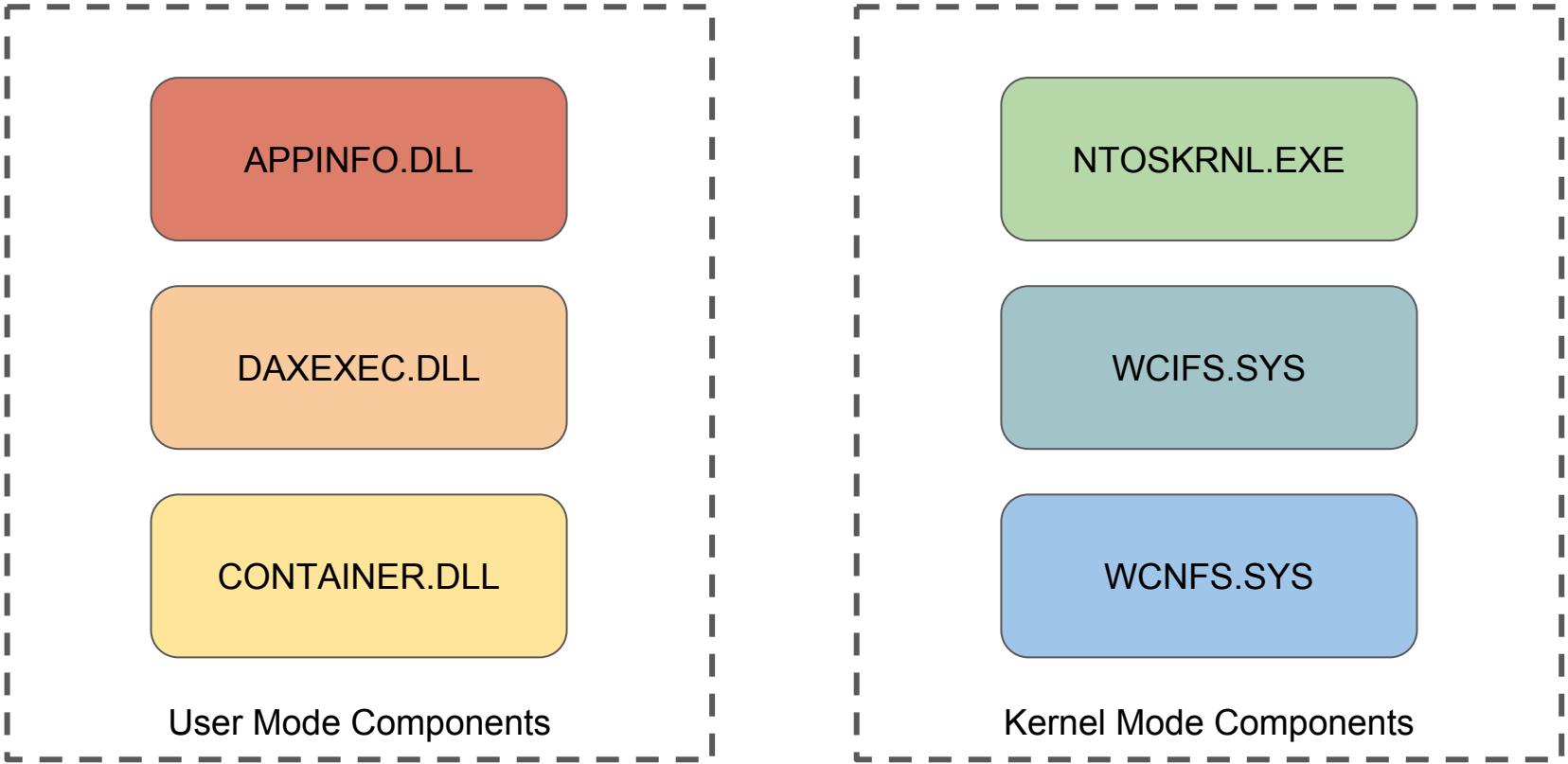
```
interface IDesktopAppXActivator {
 void Activate(string applicationUserModelId,
               string packageRelativeExecutable,
               string arguments,
               out IntPtr processHandle);
}

activator.Activate("package!App", "app.exe", ...)
```

https://upload.wikimedia.org/wikipedia/commons/7/74/Olympic_Bridge_on_Hangang_river_Seoul_Korea.jpg

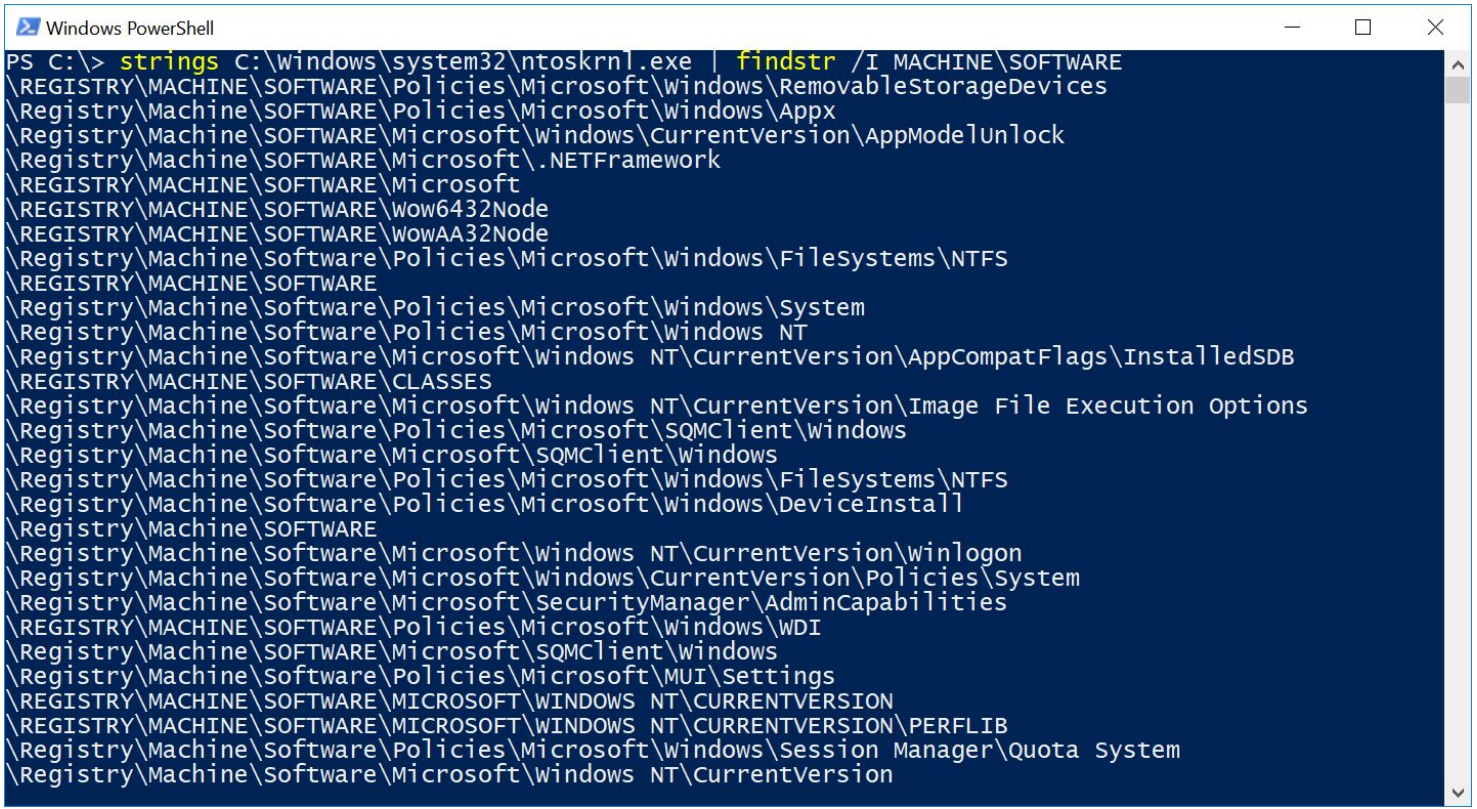
Abusing Desktop Bridge for EoP

https://commons.wikimedia.org/wiki/File:Wooden_bridge_over_So%C4%8Da_river.jpg

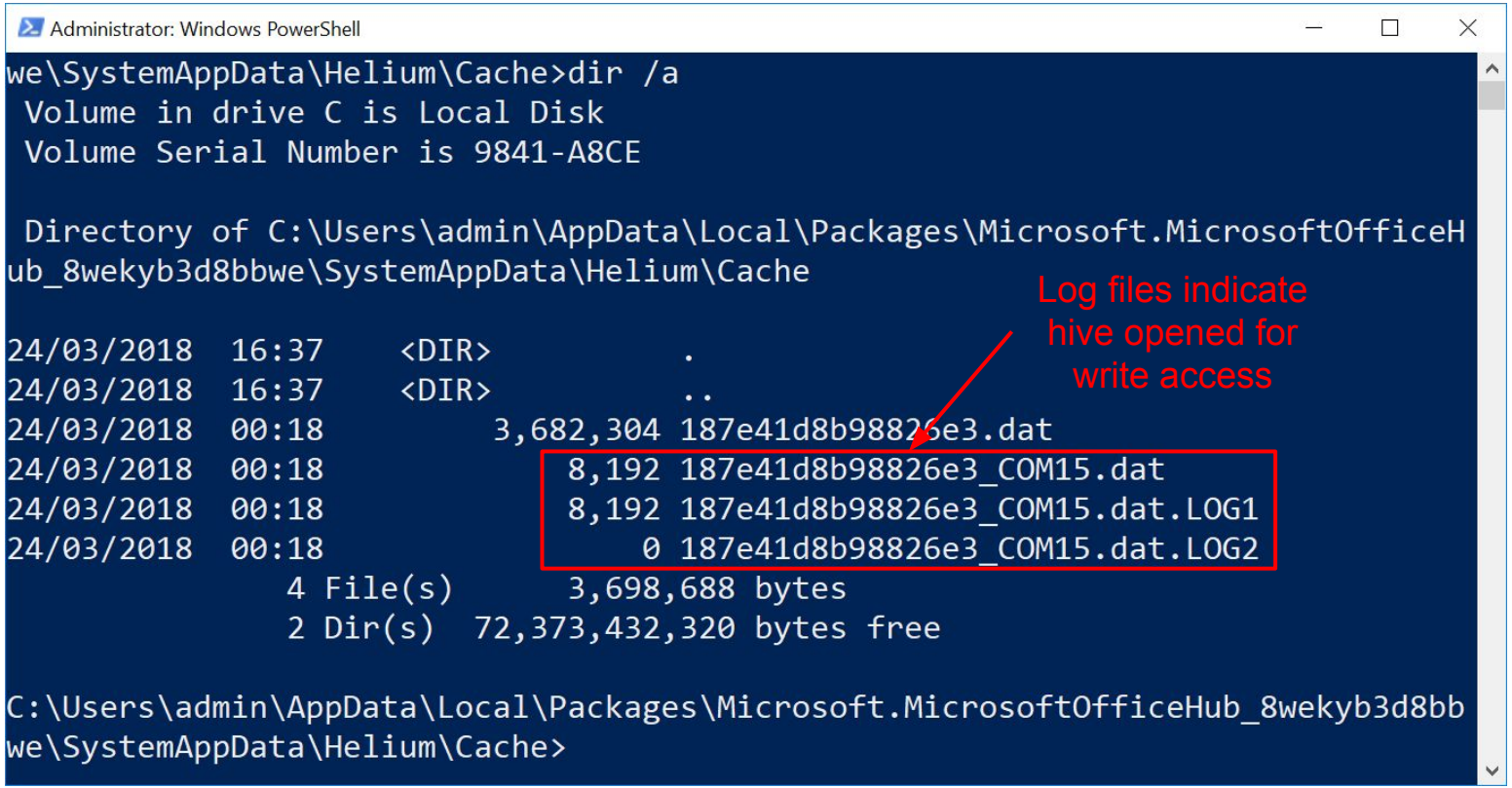
Things to Reverse Engineer



Kernel Reads from HKLM\SOFTWARE



Abusing Virtual Registry Creation



LOG Files Take Main Hive's Security

Windows: NtLoadKeyEx Read Only Hive Arbitrary File Write EoP

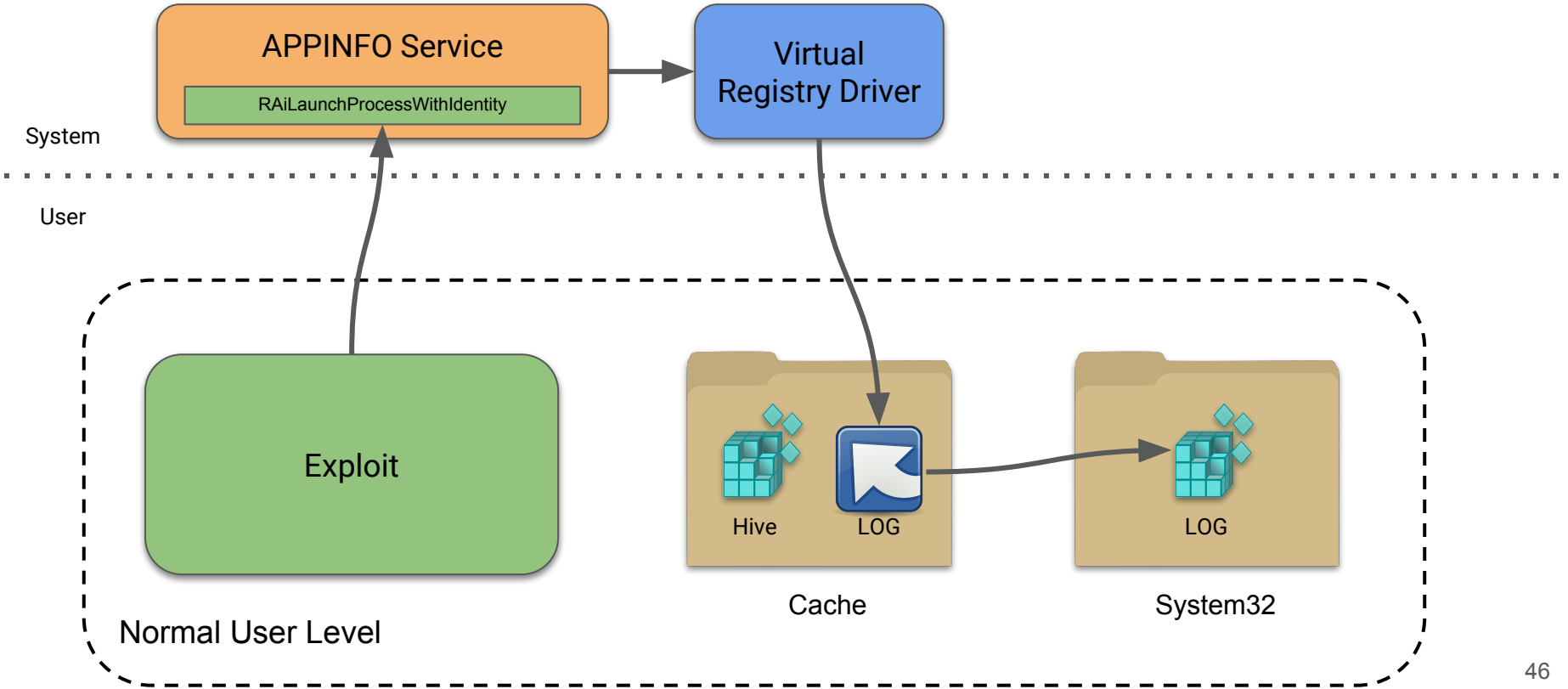
Project Member Reported by forshaw@google.com, Jul 14 2016

‹ Prev 24 of 50 Next ›

Back to list

```
Windows: NtLoadKeyEx Read Only Hive Arbitrary File Write EoP
Platform: Windows 10 10586 not tested 8.1 Update 2 or Windows 7
Class: Elevation of Privilege

Summary:
NtLoadKeyEx takes a flag to open a registry hive read only, if one of the hive files cannot be opened for read access it
will revert to write mode and also impersonate the calling process. This can leading to EoP if a user controlled hive is
opened in a system service.

Description:

One of the flags to NtLoadKeyEx is to open a registry hive with read only access. When this flag is passed the main hive
file is opened for Read only, and no log files are opened or created. However there's a bug in the kernel function
CmpCmdHiveOpen, initially it calls CmpInitHiveFromFile passing flag 1 in the second parameter which means open read-
only. However if this fails with a number of error codes, including STATUS_ACCESS_DENIED it will recall the
initialization function while impersonating the calling process, but it forgets to pass the read only flag. This means
if the initial access fails, it will instead open the hive in write mode which will create the log files etc.
```

Exploitation



Exploitation

Windows: Diagnostics Hub DLL Load EoP

Project Member Reported by forshaw@google.com, Aug 7 2016

‹ Prev 74 of 111 Next ›

Back to list

```
Windows: Diagnostics Hub DLL Load EoP
Platform: Windows 10 10586, not tested 8.1 Update 2 or Windows 7
Class: Elevation of Privilege

Summary:
The fix for CVE-2016-3231 is insufficient to prevent a normal user specifying an insecure agent path leading to
arbitrary DLL loading at system privileges.

Description:

CVE-2016-3231 was an issue caused by passing a relative agent path name which allowed the DLL path loaded for the agent
DLL to be redirected to another file. This seems to have been fixed and as far as I can tell this issue is no longer
exploitable from a sandbox. However the problem is there's an assumption that it's not possible to write a file to the
system32 directory, which technically is true but practically for this exploit false.

As I've blogged about before, and also submitted bugs (for example MSRC-21233) a normal user can created named streams
on directories as long as they have FILE_ADD_FILE access right to the directory. When you do this you create what looks
from a path perspective to be in the parent. For example the system32\tasks folder is writable by a normal user, so you
can copy a DLL to system32\tasks:abc.dll and when GetFullPathName is called the filename returned is tasks:abc.dll. When
the GetValidAgentPath is called it checks if this file is in system32 by using GetFileAttributes, which succeeds and the
service will proceed to load the file.
```

Running Arbitrary Code in Desktop Bridge

The **DWORD** pointed to by *lpValue* can be one or more of the following values when you specify **PROC_THREAD_ATTRIBUTE_DESKTOP_APP_POLICY** for the *Attribute* parameter:

PROCESS_CREATION_DESKTOP_APP_BREAKAWAY_ENABLE_PROCESS_TREE 0x01

The process being created will create any child processes outside of the desktop app runtime environment. This behavior is the default for processes for which no policy has been set.

PROCESS_CREATION_DESKTOP_APP_BREAKAWAY_DISABLE_PROCESS_TREE 0x02

The process being created will create any child processes inside of the desktop app runtime environment. This policy is inherited by the descendant processes until it is overridden by creating a process with **PROCESS_CREATION_DESKTOP_APP_BREAKAWAY_ENABLE_PROCESS_TREE**.

PROCESS_CREATION_DESKTOP_APP_BREAKAWAY_OVERRIDE 0x04

The process being created will run inside the desktop app runtime environment. This policy applies only to the process being created, not its descendants..

Running Arbitrary Code in Desktop Bridge

```
$config = New-Win32ProcessConfig $CommandLine
$config.ParentProcess = $Process
$config.DesktopAppBreakaway = "Override"

$token = Get-NtToken -Primary -Process $Process
Invoke-NtToken $token {
    New-Win32Process $config
}
```

Specify override and the parent process

Create process under impersonation

https://commons.wikimedia.org/wiki/File:Jindo_Bridge.jpg

Exploiting System32 File Reads

Project Zero

News and updates from the Project Zero team at Google

Tuesday, August 8, 2017

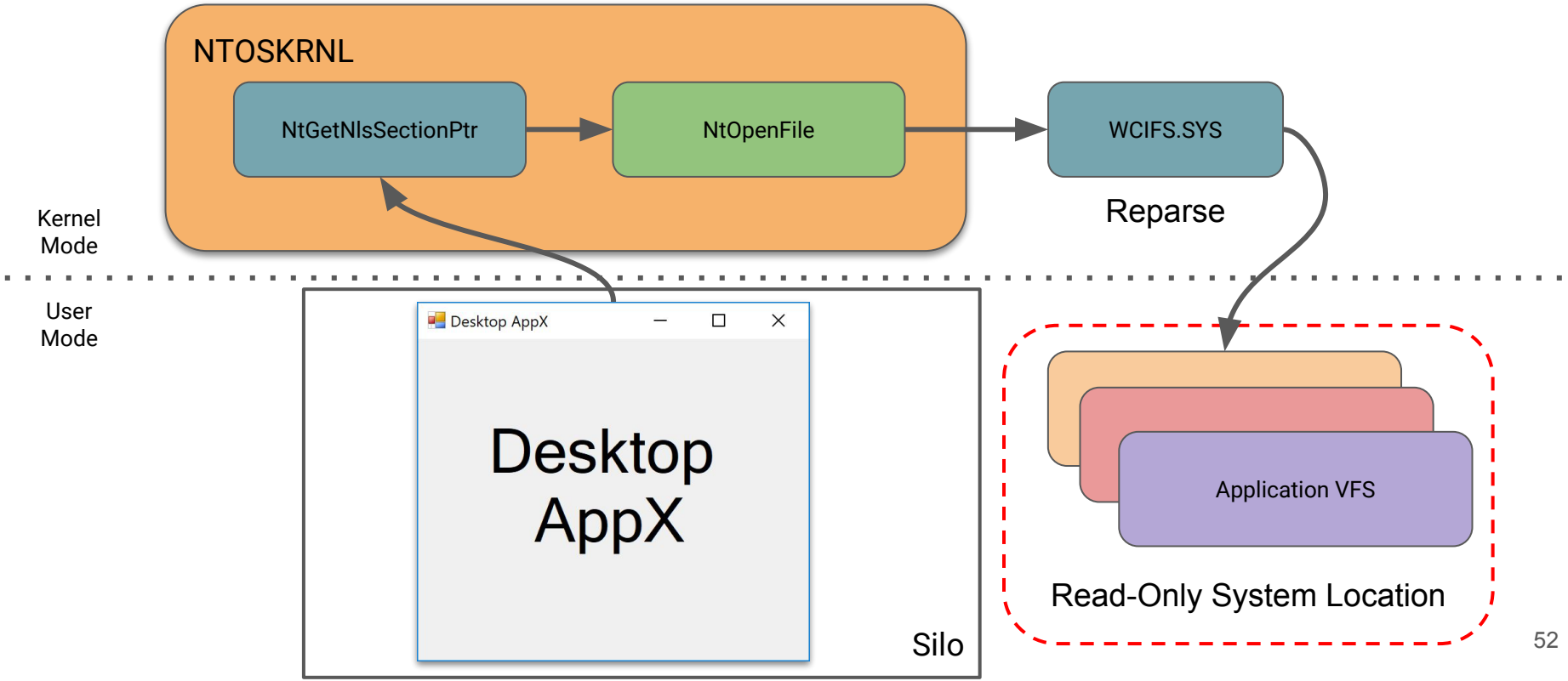
Windows Exploitation Tricks: Arbitrary Directory Creation to Arbitrary File Read

Posted by James Forshaw, Project Zero

For the past couple of months I've been presenting my "Introduction to Windows Logical Privilege Escalation Workshop" at a few conferences. The restriction of a 2 hour slot fails to do the topic justice and some interesting tips and tricks I would like to present have to be cut out. So as the likelihood of a full training course any time soon is pretty low, I thought I'd put together an irregular series of blog posts which detail small, self contained exploitation tricks which you can put to use if you find similar security vulnerabilities in Windows.

https://googleprojectzero.blogspot.co.uk/2017/08/windows-exploitation-tricks-arbitrary.html

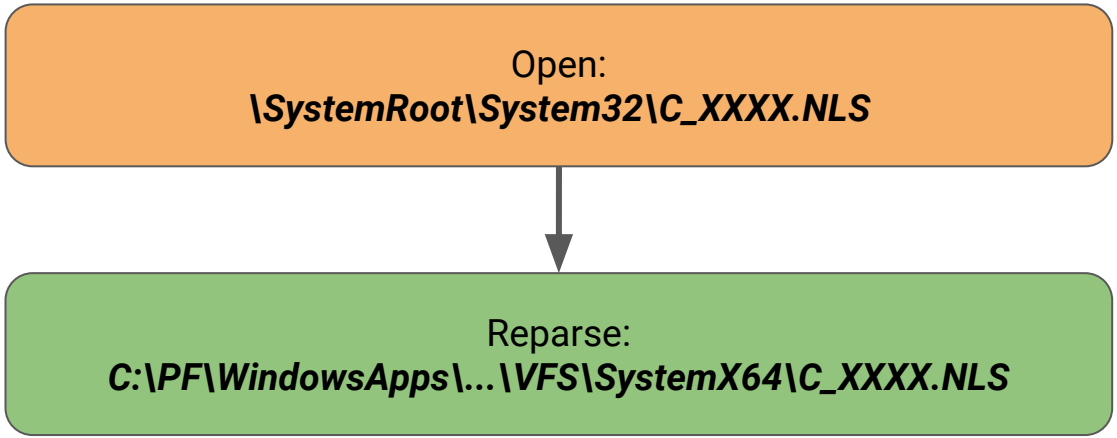
Windows Container Isolation Filesystem



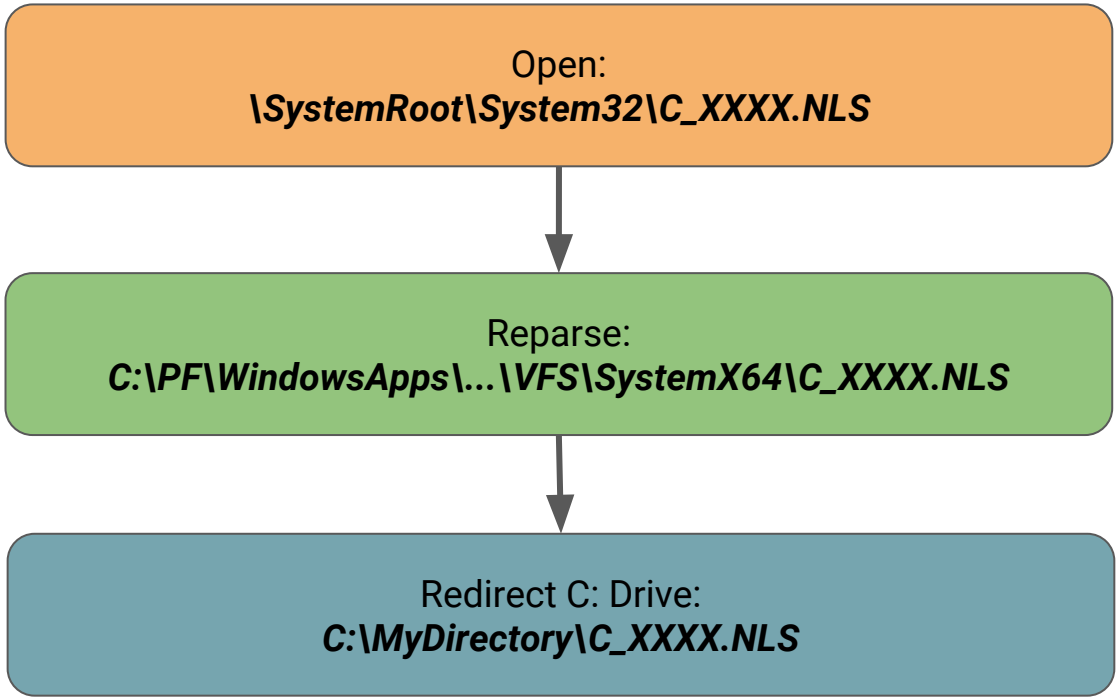
Tricking Reparse Operation

Open:
\SystemRoot\System32\C_XXXX.NLS

Tricking Reparse Operation



Tricking Reparse Operation



https://commons.wikimedia.org/wiki/File:Seongsu_Bridge_and_N_Seoul_Tower.jpg

Further Research Work

- COM Object Registration
  - In and out of process registration supported
  - File extensions and shell integration
  - Handled internal to the COM activation implementation
- Automatic startup at login of desktop appx (new execution points)
- Fully document IOCTLs for VRegDriver and filter calls for Virtual File System

References

https://docs.microsoft.com/en-us/windows/uwp/porting/desktop-to-uwp-behind-the-scenes

https://docs.microsoft.com/en-us/windows/uwp/porting/desktop-to-uwp-extensions

https://googleprojectzero.blogspot.co.uk/2017/08/windows-exploitation-tricks-arbitrary.html

https://upload.wikimedia.org/wikipedia/commons/b/bb/Bridge_in_Seoul_-_night_time.JPG