WINDOWS UPDATE PROCESSING

Sarah Santos

August 10, 2022

WHY DO WE NEED WINDOWS UPDATES?

WHY DO WE NEED WINDOWS UPDATES?

Windows Updates (MSUs)

MSU is an update package

MEMORY FORENSICS
(e.g., Volatility)

End goal: **memory forensics**, which requires ^

WHY DO WE NEED WINDOWS UPDATES?

Windows Updates (MSUs)

MSU is an update package

Windows profile

MEMORY FORENSICS
(e.g., Volatility)

Right **profile** for Windows platform, which we build from ^

End goal: **memory forensics**, which requires ^

WHY DO WE NEED WINDOWS UPDATES?

Windows Updates (MSUs)

MSU is an update package

kernel symbols + data structs

↓

Windows profile

↓

MEMORY FORENSICS
(e.g., Volatility)

Symbols and **data structures**, which are contained in ^

Right **profile** for Windows platform, which we build from ^

End goal: **memory forensics**, which requires ^

WHY DO WE NEED WINDOWS UPDATES?

Windows Updates (MSUs)

MSU is an update package

kernel PDB files

↓

kernel symbols + data structs

↓

Windows profile

↓

MEMORY FORENSICS
(e.g., Volatility)

PDB = program database

Kernel PDB files, which involve info from ^

Symbols and **data structures**, which are contained in ^

Right **profile** for Windows platform, which we build from ^

End goal: **memory forensics**, which requires ^

WHY DO WE NEED WINDOWS UPDATES?

Windows Updates (MSUs)

MSU is an update package

kernel PE files (executables)

↓

kernel PDB files

↓

kernel symbols + data structs

↓

Windows profile

↓

MEMORY FORENSICS
(e.g., Volatility)

PE = portable executable

PDB = program database

Kernel PE files (executables), which we derive from ^

Kernel PDB files, which involve info from ^

Symbols and **data structures**, which are contained in ^

Right **profile** for Windows platform, which we build from ^

End goal: **memory forensics**, which requires ^

WHY DO WE NEED WINDOWS UPDATES?

Windows Updates (MSUs)

MSU is an update package

CAB files

CAB = Cabinet

CAB files, which are the meat of a ^

kernel PE files (executables)

PE = portable executable

Kernel PE files (executables), which we derive from ^

kernel PDB files

PDB = program database

Kernel PDB files, which involve info from ^

kernel symbols + data structs

Symbols and **data structures**, which are contained in ^

Windows profile

Right **profile** for Windows platform, which we build from ^

MEMORY FORENSICS (e.g., Volatility)

End goal: **memory forensics**, which requires ^

WHY DO WE NEED WINDOWS UPDATES?

Windows Updates (MSUs)

MSU is an update package

Windows Update!

CAB files

CAB = Cabinet

CAB files, which are the meat of a ^

kernel PE files (executables)

PE = portable executable

Kernel PE files (executables), which we derive from ^

kernel PDB files

PDB = program database

Kernel PDB files, which involve info from ^

kernel symbols + data structs

Symbols and **data structures**, which are contained in ^

Windows profile

Right **profile** for Windows platform, which we build from ^

MEMORY FORENSICS (e.g., Volatility)

End goal: **memory forensics**, which requires ^

This is "extraction stack". There's also a "collection stack" to get MSUs.

WHY DO WE NEED WINDOWS UPDATES?

Windows Updates (MSUs)

↓

CAB files

↓

kernel PE files (executables)

↓

kernel PDB files

↓

kernel symbols + data structs

↓

Windows profile

↓

MEMORY FORENSICS (e.g., Volatility)

MSU is an update package

CAB = Cabinet

PE = portable executable

PDB = program database

Windows Update!

CAB files, which are the meat of a ^

Kernel PE files (executables), which we derive from ^

Kernel PDB files, which involve info from ^

Symbols and **data structures**, which are contained in ^

Right **profile** for Windows platform, which we build from ^

End goal: **memory forensics**, which requires ^

WHY DO WE NEED WINDOWS UPDATES?

Windows Updates (MSUs)	MSU is an update package	**Windows Update!**
CAB files	CAB = Cabinet	**CAB files**, which are the meat of a ^
kernel PE files (executables)	PE = portable executable	**Kernel PE files** (executables), which we derive from ^
kernel PDB files	PDB = program database	**Kernel PDB files**, which involve info from ^
kernel symbols + data structs		**Symbols** and **data structures**, which are contained in ^
Windows profile		Right **profile** for Windows platform, which we build from ^
MEMORY FORENSICS (e.g., Volatility)		End goal: **memory forensics**, which requires ^

WHY DO WE NEED WINDOWS UPDATES?

TLDR: They have the new **kernel executables** we need to update/make Windows profiles.

WHY DO WE NEED WINDOWS UPDATES?

TLDR: They have the new **kernel executables** we need to update/make Windows profiles.

The big question then is *how to extract new kernel executables from* ***CAB*** *files* (the meat of a Windows update).

HOW DO WE CURRENTLY EXTRACT KERNEL EXECUTABLES FROM CABS?

(The Current Process)

HOW DO WE CURRENTLY EXTRACT KERNEL EXECUTABLES FROM CABS?

CAB

ntoskrnl.exe

- Unfortunately, we can't just directly extract a full kernel exe from a CAB file

HOW DO WE CURRENTLY EXTRACT KERNEL EXECUTABLES FROM CABS?

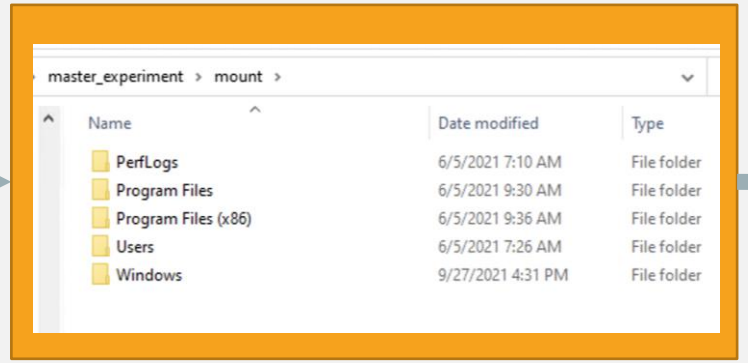
CAB

ntoskrnl.exe

- Unfortunately, we can't just directly extract a full kernel exe from a CAB file
- We could **apply the CAB** to a Windows machine to produce the new kernel executable

HOW DO WE CURRENTLY EXTRACT KERNEL EXECUTABLES FROM CABS?

CAB



WIM

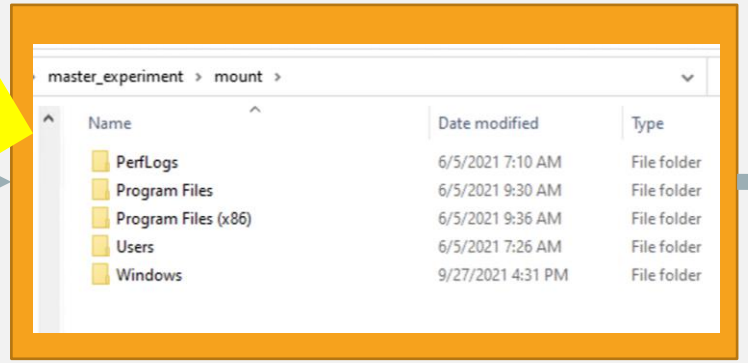
ntoskrnl.exe

- Unfortunately, we can't just directly extract a full kernel exe from a CAB file
- We could **apply the CAB** to a Windows machine to produce the new kernel executable
- Instead of an entire Windows machine, we use a WIM (Windows image)
  - WIM = "baby Windows file system" (to quote Jason)

HOW DO WE CURRENTLY EXTRACT KERNEL EXECUTABLES FROM CABS?

CAB

DISM



WIM

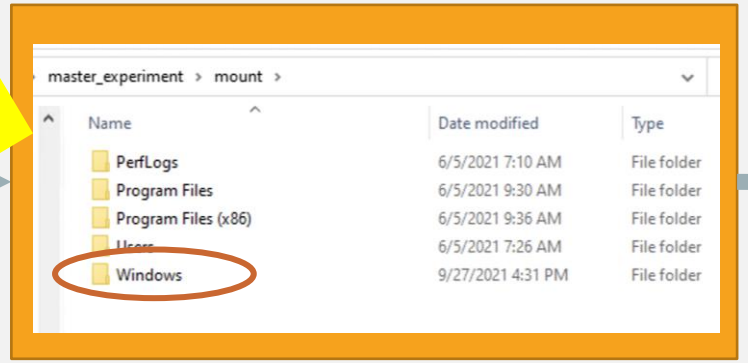
ntoskrnl.exe

- Unfortunately, we can't just directly extract a full kernel exe from a CAB file
- We could **apply the CAB** to a Windows machine to produce the new kernel executable
- Instead of an entire Windows machine, we use a WIM (Windows image)
  - WIM = "baby Windows file system" (to quote Jason)

HOW DO WE CURRENTLY EXTRACT KERNEL EXECUTABLES FROM CABS?

CAB

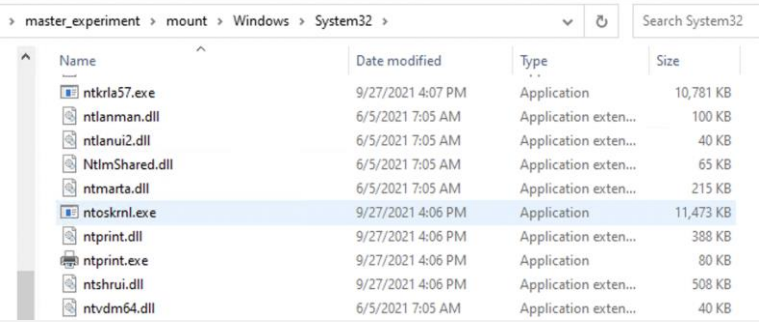
DISM



WIM

Windows\System32\ntoskrnl.exe

ntoskrnl.exe



- Unfortunately, we can't just directly extract a full kernel exe from a CAB file
- We could **apply the CAB** to a Windows machine to produce the new kernel executable
- Instead of an entire Windows machine, we use a WIM (Windows image)
  - WIM = "baby Windows file system" (to quote Jason)

APPLYING CABS WITH DISM

(to fake the full update process with a WIM)

DISM Overview

Article • 12/15/2021 • 3 minutes to read • 6 contributors

Deployment Image Servicing and Management (DISM.exe) is a command-line tool that can be used to service and prepare Windows images, including those used for Windows PE, Windows Recovery Environment (Windows RE) and Windows Setup. DISM can be used to service a Windows image (.wim) or a virtual hard disk (.vhd or .vhdx).

DISM comes built into Windows and is available through the command line or from Windows PowerShell. To learn more about using DISM with PowerShell, see Deployment Imaging Servicing Management (DISM) Cmdlets in Windows PowerShell.

APPLYING CABS WITH DISM

(to fake the full update process with a WIM)

DISM Overview

Article • 12/15/2021 • 3 minutes to read • 6 contributors

Deployment Image Servicing and Management (DISM.exe) is a command-line tool that can be used to service and prepare Windows images, including those used for Windows PE, Windows Recovery Environment (Windows RE) and Windows Setup. DISM can be used to service a Windows image (.wim) or a virtual hard disk (.vhd or .vhdx).

DISM comes built into Windows and is available through the command line or from Windows PowerShell. To learn more about using DISM with PowerShell, see Deployme
PowerShe

Administrator: Windows PowerShell

```
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Administrator> Dism /Mount-Image /ImageFile:D:\master_experiment\wims\win11-x64\install.wim
/Index:1 /MountDir:D:\master_experiment\mount

Deployment Image Servicing and Management tool
Version: 10.0.20348.681

Mounting image
[==========================100.0%==========================]
The operation completed successfully.
PS C:\Users\Administrator> Dism /Image:D:\master_experiment\mount /Add-Package /PackagePath:D:\master_ex
periment\cabs\1909\2019-11-12_windows10.0-kb4524570-x64_d9048d8efd3fda600e89c44808c8fcb5cfa2783c.cab

Deployment Image Servicing and Management tool
Version: 10.0.20348.681

Image Version: 10.0.22000.194

Processing 1 of 1 - Adding package Package_for_RollupFix~31bf3856ad364e35~amd64~~18362.476.1.8
[==========================100.0%==========================]

Error: 0x800f081e

The specified package is not applicable to this image.

The DISM log file can be found at C:\Windows\Logs\DISM\dism.log
PS C:\Users\Administrator>
```

CAN WE IMPROVE OUR DISM PROCESS?

(My goal for the summer)

CAN WE IMPROVE OUR DISM PROCESS?

- Can we make it leaner, more efficient?
  - Run experiments to find out

CAN WE IMPROVE OUR DISM PROCESS?

- Can we make it leaner, more efficient?
  - Run experiments to find out
- Humble beginnings
  - Learning DISM
  - Automating DISM in Python

Step 4: Learn about Windows updates

Use the Settings to manually update your EC2 instance using Windows Update.

Determine which specific version of Windows you are running, then find the Windows Knowledge Base (KB) number that matches.

https://support.microsoft.com/en-us/topic/windows-10-update-history-857b8ccb-71e4-49e5-b3f6-7073197d98fb

For example, the latest Windows 10 21H2 update is version 10.0.19044.1706 and is KB5013942.

Download the Update associated with the KB for your platform from
https://www.catalog.update.microsoft.com/Home.aspx

Step 5: Locate the Windows Kernel

Find the ntoskrnl binary in c:\windows\system32 on your EC2 instance. What version is the kernel? How do you know?

Step 6: Use DISM to mount and unmount a WIM

Windows can be deployed using image management. We will use this capability to apply patches to mounted images and extract the kernel.

Here is a set of WIMs that we have extracted from various Windows installers:

https://software.research.volexity.com/windows/wims/

Copy one to your EC2 instance and use DISM to mount it

https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/what-is-dism?view=windows-10

https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/dism-image-management-command-line-options-s14?view=windows-10

https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/mount-and-modify-a-windows-image-using-dism?view=windows-11

Step 7: Locate and install a package in a mounted WIM

Find a KB that matches the WIM you mounted and apply it using DISM.

Step 8: Write a Python package that wraps DISM and performs the same operations

CAN WE IMPROVE OUR DISM PROCESS?

- Can we make it leaner, more efficient?
  - Run experiments to find out
- Humble beginnings
  - Learning DISM
  - Automating DISM in Python
- Eventually, kicked off **Experiment 0.0**
  - All while experiencing DISM's weak spots

June 23, 2022

Sarah Santos 6/20 7:11 PM 1

Experiment 0.0

Open to suggestions from more experienced eyeballs. Please rip apart with questions/concerns. Sorry this is so long. I can also jump on a call this week to go through it.

I ran a test experiment on 2 CABs x 25 images (*image* = index in a WIM). The 25 images came from 4 WIM files (win10-1903-x64 and win10-1909-x64, the boot.wim and install.wim in each). One of the CABs was from Nick's link in OneNote (but the x64 version), the other was randomly selected.

Step 4: Learn about Windows updates

Use the Settings to manually update your EC2 instance using Windows Update.

Determine which specific version of Windows you are running, then find the Windows Knowledge Base (KB) number that matches.

https://support.microsoft.com/en-us/topic/windows-10-update-history-857b8ccb-71e4-49e5-b3f6-7073197d98fb

For example, the latest Windows 10 21H2 update is version 10.0.19044.1706 and is KB5013942.

Download the Update associated with the KB for your platform from https://www.catalog.update.microsoft.com/Home.aspx

Step 5: Locate the Windows Kernel

Find the ntoskrnl binary in c:\windows\system32 on your EC2 instance. What version is the kernel? How do you know?

Step 6: Use DISM to mount and unmount a WIM

Windows can be deployed using image management. We will use this capability to apply patches to mounted images and extract the kernel.

Here is a set of WIMs that we have extracted from various Windows installers:

https://software.research.volexity.com/windows/wims/

Copy one to your EC2 instance and use DISM to mount it

https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/what-is-dism?view=windows-10

https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/dism-image-management-command-line-options-s14?view=windows-10

https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/mount-and-modify-a-windows-image-using-dism?view=windows-11

Step 7: Locate and install a package in a mounted WIM

Find a KB that matches the WIM you mounted and apply it using DISM.

Step 8: Write a Python package that wraps DISM and performs the same operations

CAN WE IMPROVE OUR DISM PROCESS?

- **PROBLEM:** a CAB doesn't always successfully apply to a WIM

```
PS C:\Users\Administrator> Dism /Image:D:\master_experiment\mount /Add-Package /PackagePath:D:\master_ex
periment\cabs\1909\2019-11-12_windows10.0-kb4524570-x64_d9048d8efd3fda600e89c44808c8fcb5cfa2783c.cab

Deployment Image Servicing and Management tool
Version: 10.0.20348.681

Image Version: 10.0.22000.194

Processing 1 of 1 - Adding package Package_for_RollupFix~31bf3856ad364e35~amd64~~18362.476.1.8
[==========================100.0%==========================]

Error: 0x800f081e

The specified package is not applicable to this image.

The DISM log file can be found at C:\Windows\Logs\DISM\dim.log
PS C:\Users\Administrator>
```

CAN WE IMPROVE OUR DISM PROCESS?

- **PROBLEM:** a CAB doesn't always successfully apply to a WIM - we don't know why/when

```
PS C:\Users\Administrator> Dism /Image:D:\master_experiment\mount /Add-Package /PackagePath:D:\master_ex
periment\cabs\1909\2019-11-12_windows10.0-kb4524570-x64_d9048d8efd3fda600e89c44808c8fcb5cfa2783c.cab

Deployment Image Servicing and Management tool
Version: 10.0.20348.681

Image Version: 10.0.22000.194

Processing 1 of 1 - Adding package Package_for_RollupFix~31bf3856ad364e35~amd64~~18362.476.1.8
[==========================100.0%==========================]

Error: 0x800f081e

The specified package is not applicable to this image.

The DISM log file can be found at C:\Windows\Logs\DISM\dim.log
PS C:\Users\Administrator>
```

CAN WE IMPROVE OUR DISM PROCESS?

- **PROBLEM:** a CAB doesn't always successfully apply to a WIM - we don't know why/when
  - Missing the logic for how to determine a successful CAB

```
PS C:\Users\Administrator> Dism /Image:D:\master_experiment\mount /Add-Package /PackagePath:D:\master_ex
periment\cabs\1909\2019-11-12_windows10.0-kb4524570-x64_d9048d8efd3fda600e89c44808c8fcb5cfa2783c.cab

Deployment Image Servicing and Management tool
Version: 10.0.20348.681

Image Version: 10.0.22000.194

Processing 1 of 1 - Adding package Package_for_RollupFix~31bf3856ad364e35~amd64~~18362.476.1.8
[==========================100.0%==========================]

Error: 0x800f081e

The specified package is not applicable to this image.

The DISM log file can be found at C:\Windows\Logs\DISM\dim.log
PS C:\Users\Administrator>
```

CAN WE IMPROVE OUR DISM PROCESS?

- **PROBLEM:** a CAB doesn't always successfully apply to a WIM - we don't know why/when
  - Missing the logic for how to determine a successful CAB
  - We use number of heuristics to approximate in current process

```
PS C:\Users\Administrator> Dism /Image:D:\master_experiment\mount /Add-Package /PackagePath:D:\master_ex
periment\cabs\1909\2019-11-12_windows10.0-kb4524570-x64_d9048d8efd3fda600e89c44808c8fcb5cfa2783c.cab

Deployment Image Servicing and Management tool
Version: 10.0.20348.681

Image Version: 10.0.22000.194

Processing 1 of 1 - Adding package Package_for_RollupFix~31bf3856ad364e35~amd64~~18362.476.1.8
[==========================100.0%==========================]

Error: 0x800f081e

The specified package is not applicable to this image.

The DISM log file can be found at C:\Windows\Logs\DISM\dism.log
PS C:\Users\Administrator>
```

CAN WE IMPROVE OUR DISM PROCESS?

- **PROBLEM:** a CAB doesn't always successfully apply to a WIM - we don't know why/when
  - Missing the logic for how to determine a successful CAB
  - We use number of heuristics to approximate in current process

```
PS C:\Users\Administrator> Dism /Image:D:\master_experiment\mount /Add-Package /PackagePath:D:\master_ex
periment\cabs\1909\2019-11-12_windows10.0-kb4524570-x64_d9048d8efd3fda600e89c44808c8fcb5cfa2783c.cab

Deployment Image Servicing and Management tool
Version: 10.0.20348.681

Image Version: 10.0.22000.194

Processing 1 of 1 - Adding package Package_for_RollupFix~31bf3856ad364e35~amd64~~18362.476.1.8
[==========================100.0%==========================]

Error: 0x800f081e

The specified package is not applicable to this image.

The DISM log file can be found at C:\Windows\Logs\DISM\dism.log
PS C:\Users\Administrator>
```

- **PROBLEM:** DISM is *dism*ally slow

CAN WE IMPROVE OUR DISM PROCESS?

- **PROBLEM:** a CAB doesn't always successfully apply to a WIM - we don't know why/when
  - Missing the logic for how to determine a successful CAB
  - We use number of heuristics to approximate in current process

```
PS C:\Users\Administrator> Dism /Image:D:\master_experiment\mount /Add-Package /PackagePath:D:\master_ex
periment\cabs\1909\2019-11-12_windows10.0-kb4524570-x64_d9048d8efd3fda600e89c44808c8fcb5cfa2783c.cab

Deployment Image Servicing and Management tool
Version: 10.0.20348.681

Image Version: 10.0.22000.194

Processing 1 of 1 - Adding package Package_for_RollupFix~31bf3856ad364e35~amd64~~18362.476.1.8
[==========================100.0%==========================]

Error: 0x800f081e

The specified package is not applicable to this image.

The DISM log file can be found at C:\Windows\Logs\DISM\dism.log
PS C:\Users\Administrator>
```

- **PROBLEM:** DISM is *dism*ally slow
  - Requires mounting and unmounting a WIM

CAN WE IMPROVE OUR DISM PROCESS?

- **PROBLEM:** a CAB doesn't always successfully apply to a WIM - we don't know why/when
  - Missing the logic for how to determine a successful CAB
  - We use number of heuristics to approximate in current process

```
PS C:\Users\Administrator> Dism /Image:D:\master_experiment\mount /Add-Package /PackagePath:D:\master_ex
periment\cabs\1909\2019-11-12_windows10.0-kb4524570-x64_d9048d8efd3fda600e89c44808c8fcb5cfa2783c.cab

Deployment Image Servicing and Management tool
Version: 10.0.20348.681

Image Version: 10.0.22000.194

Processing 1 of 1 - Adding package Package_for_RollupFix~31bf3856ad364e35~amd64~~18362.476.1.8
[==========================100.0%==========================]

Error: 0x800f081e

The specified package is not applicable to this image.

The DISM log file can be found at C:\Windows\Logs\DISM\dism.log
PS C:\Users\Administrator>
```

- **PROBLEM:** DISM is *dism*ally slow
  - Requires mounting and unmounting a WIM
  - /Add-Package can take a while
    - Applies the *entire* CAB file, which has a bunch of other things besides the kernel update

EXPERIMENT 0.0

DISCOVERY #1

Multiple images in a single WIM file

```
PS C:\Users\Administrator> Dism /Get-ImageInfo /ImageFile:D:\master_experiment\wims\win10-1909-x64\insta
ll.wim

Deployment Image Servicing and Management tool
Version: 10.0.20348.681

Details for image : D:\master_experiment\wims\win10-1909-x64\install.wim

Index : 1
Name : Windows 10 Education
Description : Windows 10 Education
Size : 14,780,927,379 bytes

Index : 2
Name : Windows 10 Education N
Description : Windows 10 Education N
Size : 13,958,508,090 bytes

Index : 3
Name : Windows 10 Enterprise
Description : Windows 10 Enterprise
Size : 14,781,260,269 bytes

Index : 4
Name : Windows 10 Enterprise N
Description : Windows 10 Enterprise N
Size : 13,958,401,617 bytes

Index : 5
Name : Windows 10 Pro
Description : Windows 10 Pro
Size : 14,782,419,696 bytes

Index : 6
Name : Windows 10 Pro N
Description : Windows 10 Pro N
Size : 13,975,944,782 bytes

Index : 7
```

EXPERIMENT 0.0

DISCOVERY #1

Multiple images in a single WIM file

- Different editions of same Windows version, but same kernel

```
PS C:\Users\Administrator> Dism /Get-ImageInfo /ImageFile:D:\master_experiment\wims\win10-1909-x64\insta
ll.wim

Deployment Image Servicing and Management tool
Version: 10.0.20348.681

Details for image : D:\master_experiment\wims\win10-1909-x64\install.wim

Index : 1
Name : Windows 10 Education
Description : Windows 10 Education
Size : 14,780,927,379 bytes

Index : 2
Name : Windows 10 Education N
Description : Windows 10 Education N
Size : 13,958,508,090 bytes

Index : 3
Name : Windows 10 Enterprise
Description : Windows 10 Enterprise
Size : 14,781,260,269 bytes

Index : 4
Name : Windows 10 Enterprise N
Description : Windows 10 Enterprise N
Size : 13,958,401,617 bytes

Index : 5
Name : Windows 10 Pro
Description : Windows 10 Pro
Size : 14,782,419,696 bytes

Index : 6
Name : Windows 10 Pro N
Description : Windows 10 Pro N
Size : 13,975,944,782 bytes

Index : 7
```

EXPERIMENT 0.0

DISCOVERY #1

Multiple images in a single WIM file

- Different editions of same Windows version, but same kernel
- Optimize framework by only taking smallest sized image

```
PS C:\Users\Administrator> Dism /Get-ImageInfo /ImageFile:D:\master_experiment\wims\win10-1909-x64\insta
ll.wim

Deployment Image Servicing and Management tool
Version: 10.0.20348.681

Details for image : D:\master_experiment\wims\win10-1909-x64\install.wim

Index : 1
Name : Windows 10 Education
Description : Windows 10 Education
Size : 14,780,927,379 bytes

Index : 2
Name : Windows 10 Education N
Description : Windows 10 Education N
Size : 13,958,508,090 bytes

Index : 3
Name : Windows 10 Enterprise
Description : Windows 10 Enterprise
Size : 14,781,260,269 bytes

Index : 4
Name : Windows 10 Enterprise N
Description : Windows 10 Enterprise N
Size : 13,958,401,617 bytes

Index : 5
Name : Windows 10 Pro
Description : Windows 10 Pro
Size : 14,782,419,696 bytes

Index : 6
Name : Windows 10 Pro N
Description : Windows 10 Pro N
Size : 13,975,944,782 bytes

Index : 7
```

EXPERIMENT 0.0

DISCOVERY #2

Mysterious other kernels in WinSxS

5. Multiple Kernels?

Is it ever possible for multiple kernels to exist at Windows\System32? Also, should I be looking at the other kernels not in this directory, e.g.:

```
In [6]: glob.glob("**/*ntoskrnl.exe", root_dir="C:\\experiment_00\\mount", recursive=True)
Out[6]:
['Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\ntoskrnl.exe',
 'Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\f\\ntoskrnl.exe',
 'Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\r\\ntoskrnl.exe']
```

EXPERIMENT 0.0

DISCOVERY #2

WinSxS = Windows Side-by-Side

Mysterious other kernels in WinSxS

5. Multiple Kernels?

Is it ever possible for multiple kernels to exist at Windows\System32? Also, should I be looking at the other kernels not in this directory, e.g.:

```
In [6]: glob.glob("**/*ntoskrnl.exe", root_dir="C:\\experiment_00\\mount", recursive=True)
Out[6]:
['Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\ntoskrnl.exe',
 'Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\f\\ntoskrnl.exe',
 'Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\r\\ntoskrnl.exe']
```

EXPERIMENT 0.0

DISCOVERY #2

WinSxS = Windows Side-by-Side

Mysterious other kernels in WinSxS

5. Multiple Kernels?

Is it ever possible for multiple kernels to exist at Windows\System32? Also, should I be looking at the other kernels not in this directory, e.g.:

```
In [6]: glob.glob("**/*ntoskrnl.exe", root_dir="C:\\experiment_00\\mount", recursive=True)
Out[6]:
['Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\ntoskrnl.exe',
 'Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\f\\ntoskrnl.exe',
 'Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\r\\ntoskrnl.exe']
```

5. it looks like those might be diffs to produce the same file based on this? seems like only one of them should be necessary. That link gives some info about parsing the diff files, so you can confirm/deny.

EXPERIMENT 0.0

DISCOVERY #2

WinSxS = Windows Side-by-Side

Mysterious other kernels in WinSxS

5. Multiple Kernels?

Is it ever possible for multiple kernels to exist at Windows\System32? Also, should I be looking at the other kernels not in this directory, e.g.:

```
In [6]: glob.glob("**/*ntoskrnl.exe", root_dir="C:\\experiment_00\\mount", recursive=True)
Out[6]:
['Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\ntoskrnl.exe',
 'Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\f\\ntoskrnl.exe',
 'Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\r\\ntoskrnl.exe']
```

5. it looks like those might be diffs to produce the same file based on this? seems like only one of them should be necessary. That link gives some info about parsing the diff files, so you can confirm/deny.

3. WinSxS

He also said the different ntoskrnl.exe files in Windows\WinSxS are previous versions of the kernel for backup purposes. So I started to research this and thought the WinSxS folder deserved a shoutout (forgive me if I'm repeating old/irrelevant news). WinSxS stores multiple versions of system files. I though this somewhat parallels the idea of observing how the system changes given a sequence of CABs.

The article you shared also had an interesting blurb on WinSxS manifests:

> The .manifest files in the patch describe how the patch is to be applied, the files that are part of the patch, the **expected result of the patch in the form of file hashes**, permissions of the resulting files, registry keys to set, and more.

Maybe this "expected result" could also be used in our experiments?

EXPERIMENT 0.0

DISCOVERY #2

WinSxS = Windows Side-by-Side

Mysterious other kernels in WinSxS

5. Multiple Kernels?

Is it ever possible for multiple kernels to exist at Windows\System32? Also, should I be looking at the other kernels not in this directory, e.g.:

```
In [6]: glob.glob("**/*ntoskrnl.exe", root_dir="C:\\experiment_00\\mount", recursive=True)
Out[6]:
['Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\ntoskrnl.exe',
 'Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\f\\ntoskrnl.exe',
 'Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\r\\ntoskrnl.exe']
```

5. it looks like those might be diffs to produce the same file based on this? seems like only one of them should be necessary. That link gives some info about parsing the diff files, so you can confirm/deny.

3. WinSxS

He also said the different ntoskrnl.exe files in Windows\WinSxS are previous versions of the kernel for backup purposes. So I started to research this and thought the WinSxS folder deserved a shoutout (forgive me if I'm repeating old/irrelevant news). WinSxS stores multiple versions of system files. I though this somewhat parallels the idea of observing how the system changes given a sequence of CABs.

The article you shared also had an interesting blurb on WinSxS manifests:

> The .manifest files in the patch describe how the patch is to be applied, the files that are part of the patch, the **expected result of the patch in the form of file hashes**, permissions of the resulting files, registry keys to set, and more.

Maybe this "expected result" could also be used in our experiments?

patch deltas!

EXPERIMENT 0.0

DISCOVERY #2

WinSxS = Windows Side-by-Side

Mysterious other kernels in WinSxS

5. Multiple Kernels?

Is it ever possible for multiple kernels to exist at Windows\System32? Also, should I be looking at the other kernels not in this directory, e.g.:

```
In [6]: glob.glob("**/*ntoskrnl.exe", root_dir="C:\\experiment_00\\mount", recursive=True)
Out[6]:
['Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\ntoskrnl.exe',
 'Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\f\\ntoskrnl.exe',
 'Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\r\\ntoskrnl.exe']
```

5. it looks like those might be diffs to produce the same file based on this? seems like only one of them should be necessary. That link gives some info about parsing the diff files, so you can confirm/deny.

3. WinSxS

He also said the different ntoskrnl.exe files in Windows\WinSxS are previous versions of the kernel for backup purposes. So I started to research this and thought the WinSxS folder deserved a shoutout (forgive me if I'm repeating old/irrelevant news). WinSxS stores multiple versions of system files. I though this somewhat parallels the idea of observing how the system changes given a sequence of CABs.

The article you shared also had an interesting blurb on WinSxS manifests:

> The .manifest files in the patch describe how the patch is to be applied, the files that are part of the patch, the **expected result of the patch in the form of file hashes**, permissions of the resulting files, registry keys to set, and more.

Maybe this "expected result" could also be used in our experiments?

patch deltas!

manifest files!

EXPERIMENT 0.0

DISCOVERY #2

WinSxS = Windows Side-by-Side

Mysterious other kernels in WinSxS

5. Multiple Kernels?

Is it ever possible for multiple kernels to exist at Windows\System32? Also, should I be looking at the other kernels not in this directory, e.g.:

```
In [6]: glob.glob("**/*ntoskrnl.exe", root_dir="C:\\experiment_00\\mount", recursive=True)
Out[6]:
['Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\ntoskrnl.exe',
 'Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\f\\ntoskrnl.exe',
 'Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\r\\ntoskrnl.exe']
```

5. it looks like those might be diffs to produce the same file based on this? seems like only one of them should be necessary. That link gives some info about parsing the diff files, so you can confirm/deny.

3. WinSxS

He also said the different ntoskrnl.exe files in Windows\WinSxS are previous versions of the kernel for backup purposes. So I started to research this and thought the WinSxS folder deserved a shoutout (forgive me if I'm repeating old/irrelevant news). WinSxS stores multiple versions of system files. I though this somewhat parallels the idea of observing how the system changes given a sequence of CABs.

The article you shared also had an interesting blurb on WinSxS manifests:

> The .manifest files in the patch describe how the patch is to be applied, the files that are part of the patch, the **expected result of the patch in the form of file hashes**, permissions of the resulting files, registry keys to set, and more.

Maybe this "expected result" could also be used in our experiments?

patch deltas!

Components of an update CAB

manifest files!

EXPERIMENT 0.0

DISCOVERY #2

WinSxS = Windows Side-by-Side

Mysterious other kernels in WinSxS

5. Multiple Kernels?

Is it ever possible for multiple kernels to exist at Windows\System32? Also, should I be looking at the other kernels not in this directory, e.g.:

```
In [6]: glob.glob("**/*ntoskrnl.exe", root_dir="C:\\experiment_00\\mount", recursive=True)
Out[6]:
['Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\ntoskrnl.exe',
 'Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\f\\ntoskrnl.exe',
 'Windows\\WinSxS\\amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.18362.30_none_db566142fbca8bb4\\r\\ntoskrnl.exe']
```

5. it looks like those might be diffs to produce the same file based on this? seems like only one of them should be necessary. That link gives some info about parsing the diff files, so you can confirm/deny.

3. WinSxS

He also said the different ntoskrnl.exe files in Windows\WinSxS are previous versions of the kernel for backup purposes. So I started to research this and thought the WinSxS folder deserved a shoutout (forgive me if I'm repeating old/irrelevant news). WinSxS stores multiple versions of system files. I though this somewhat parallels the idea of observing how the system changes given a sequence of CABs.

The article you shared also had an interesting blurb on WinSxS manifests:

> The .manifest files in the patch describe how the patch is to be applied, the files that are part of the patch, the **expected result of the patch in the form of file hashes**, permissions of the resulting files, registry keys to set, and more.

Maybe this "expected result" could also be used in our experiments?

patch deltas!

Components of an update CAB

manifest files!

Benchmark results

EXPERIMENT 0.0

DISCOVERY #3

Error 2 in DISM leads to MSDelta

- A CAB failed, weird error(s)
  - Trying to debug this while also investigating "patch deltas" discovery

```
PS C:\Users\Administrator> Dism /Image:C:\experiment_00\mount /Add-Package /PackagePath:C:\experiment_00\cab_bag\win10-1
909-x64\windows10.0-kb4495666-x64_34bcea9735afc154bf9115d954a3179537af1c60.cab

Deployment Image Servicing and Management tool
Version: 10.0.20348.681

Image Version: 10.0.18362.30

Processing 1 of 1 - Adding package Package_for_RollupFix~31bf3856ad364e35~amd64~~18362.53.1.5
[==========================100.0%==========================]
An error occurred - Package_for_RollupFix Error: 0x80070002

Error: 2

The system cannot find the file specified.
An error occurred closing a servicing component in the image.
Wait a few minutes and try running the command again.

The DISM log file can be found at C:\Windows\Logs\DISM\dism.log
```

EXPERIMENT 0.0

DISCOVERY #3

Error 2 in DISM leads to MSDelta

- A CAB failed, weird error(s)
  - Trying to debug this while also investigating "patch deltas" discovery

```
PS C:\Users\Administrator> Dism /Image:C:\experiment_00\mount /Add-Package /PackagePath:C:\experiment_00\cab_bag\win10-1
909-x64\windows10.0-kb4495666-x64_34bcea9735afc154bf9115d954a3179537af1c60.cab

Deployment Image Servicing and Management tool
Version: 10.0.20348.681

Image Version: 10.0.18362.30

Processing 1 of 1 - Adding package Package_for_RollupFix~31bf3856ad364e35~amd64~~18362.53.1.5
[==========================100.0%==========================]
An error occurred - Package_for_RollupFix Error: 0x80070002

Error: 2

The system cannot find the file specified.
An error occurred closing a servicing component in the image.
Wait a few minutes and try running the command again.

The DISM log file can be found at C:\Windows\Logs\DISM\dism.log
```

What if I try to apply the kernel patch delta from this failing CAB? Will it also fail?

EXPERIMENT 0.0

DISCOVERY #3

Error 2 in DISM leads to MSDelta

Patch delta didn't fail, but entire CAB did!

EXPERIMENT 0.0

DISCOVERY #3

Error 2 in DISM leads to MSDelta

Patch delta didn't fail, but entire CAB did!

I applied the kernel-specific patch delta, rather than the whole CAB (which fails), using **MSDelta** (not DISM).

EXPERIMENT 0.0

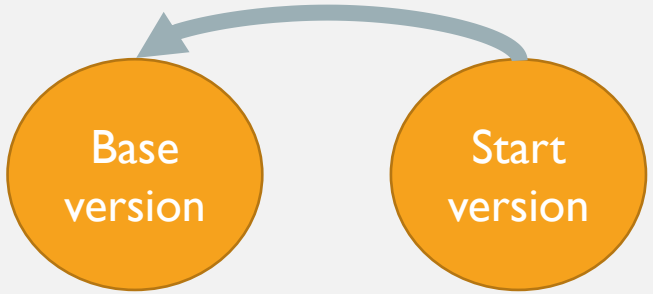
DISCOVERY #3

Error 2 in DISM leads to MSDelta

Patch delta didn't fail, but entire CAB did!

I applied the kernel-specific patch delta, rather than the whole CAB (which fails), using **MSDelta** (not DISM).

- **1. Reverse current kernel to a base state**



EXPERIMENT 0.0

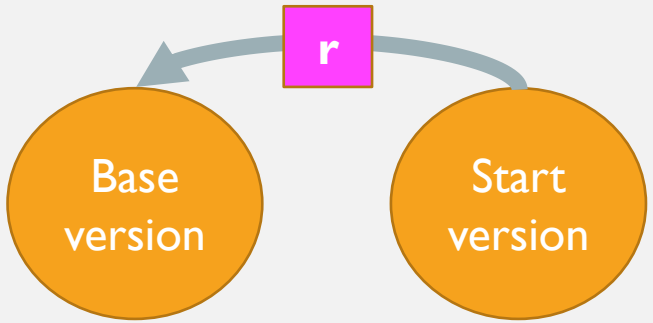
DISCOVERY #3

Error 2 in DISM leads to MSDelta

Patch delta didn't fail, but entire CAB did!

I applied the kernel-specific patch delta, rather than the whole CAB (which fails), using **MSDelta** (not DISM).

- **1. Reverse current kernel to a base state**
  - WinSxS (*inside the WIM mount, **not** cab*) contains a reverse diff for the kernel. We can apply it to roll back our current kernel to a "historical" base version. Then, we start at this checkpoint to apply a new patch (next step).



EXPERIMENT 0.0

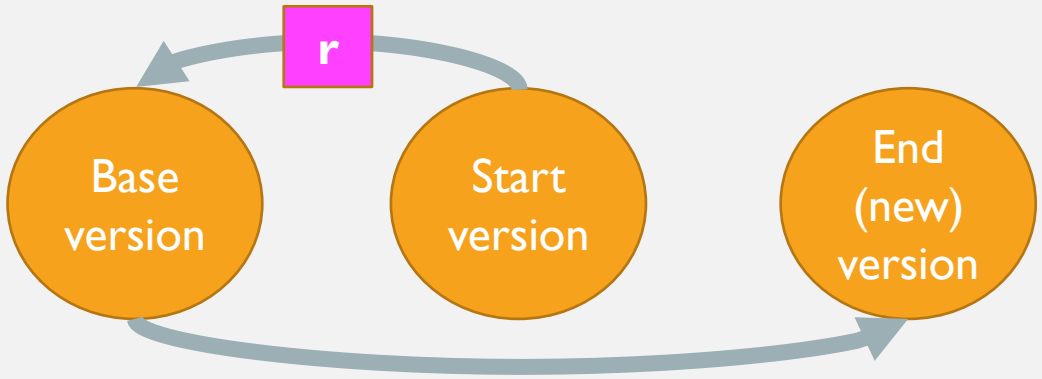
DISCOVERY #3

Error 2 in DISM leads to MSDelta

Patch delta didn't fail, but entire CAB did!

I applied the kernel-specific patch delta, rather than the whole CAB (which fails), using **MSDelta** (not DISM).

- **1. Reverse current kernel to a base state**
  - WinSxS (*inside the WIM mount, **not** cab*) contains a reverse diff for the kernel. We can apply it to roll back our current kernel to a "historical" base version. Then, we start at this checkpoint to apply a new patch (next step).
- **2. Forward to a new state from patch of your choice**



EXPERIMENT 0.0

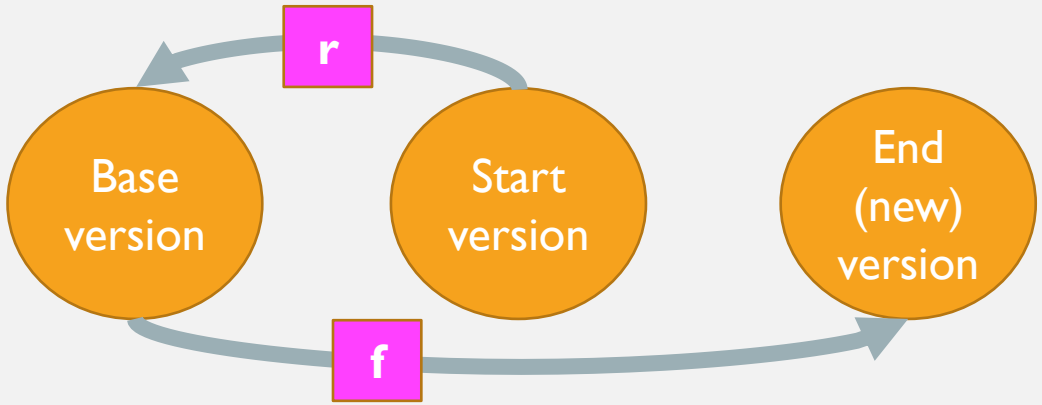
DISCOVERY #3

Error 2 in DISM leads to MSDelta

Patch delta didn't fail, but entire CAB did!

I applied the kernel-specific patch delta, rather than the whole CAB (which fails), using **MSDelta** (not DISM).

- **1. Reverse current kernel to a base state**
  - WinSxS (*inside the WIM mount, **not** cab*) contains a reverse diff for the kernel. We can apply it to roll back our current kernel to a "historical" base version. Then, we start at this checkpoint to apply a new patch (next step).
- **2. Forward to a new state from patch of your choice**
  - Now we turn to the cab (not WIM mount). We isolate the kernel delta, which contains a forward diff. This forward diff must be applied to a known state, the checkpoint from step 1. Then, the manifest hash is produced!

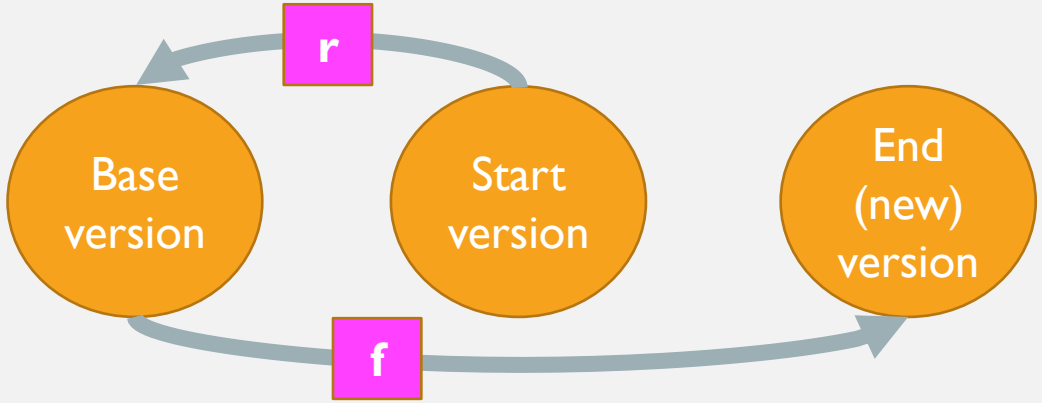


EXPERIMENT 0.0

DISCOVERY #3

Error 2 in DISM leads to MSDelta

- **Key takeaways**

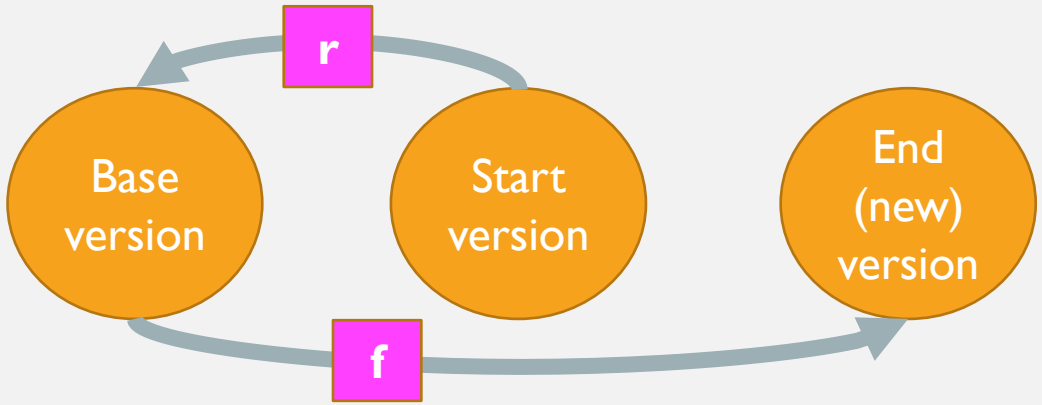


EXPERIMENT 0.0

DISCOVERY #3

Error 2 in DISM leads to MSDelta

- **Key takeaways**
  - Kernel patch component succeeded (MSDelta) despite the entire CAB failing (DISM)

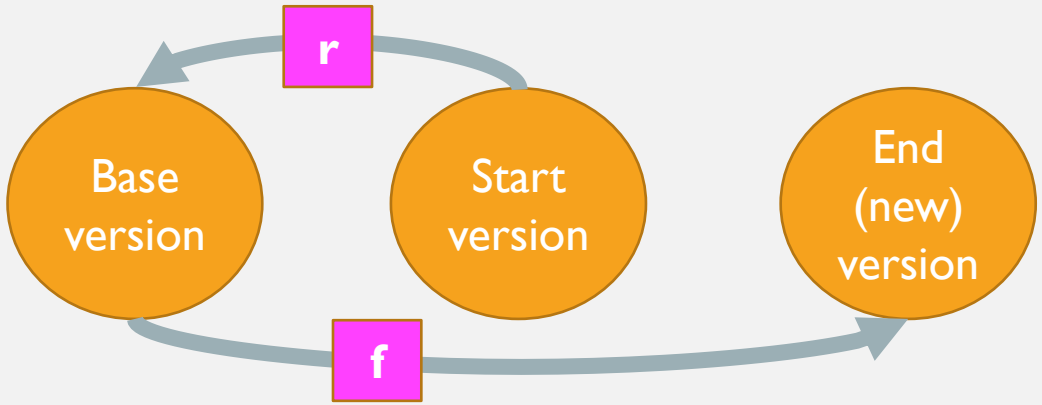


EXPERIMENT 0.0

DISCOVERY #3

Error 2 in DISM leads to MSDelta

- **Key takeaways**
  - Kernel patch component succeeded (MSDelta) despite the entire CAB failing (DISM)
  - Error most likely for another file's patch component, not kernel



IS MSDELTA BETTER THAN DISM?

(Can we use it to more efficiently extract kernel PEs?)

IS MSDELTA BETTER THAN DISM?

- I made an experiment framework that runs trials with both DISM and MSDelta

IS MSDELTA BETTER THAN DISM?

- I made an experiment framework that runs trials with both DISM and MSDelta

vs.

IS MSDELTA BETTER THAN DISM?

- I made an experiment framework that runs trials with both DISM and MSDelta
  - "Both are completely different approaches. One is welding. The other is sewing. Your experiments run them in parallel to see which is superior."
    - Paraphrasing Jason again

vs.

IS MSDELTA BETTER THAN DISM?

Research Objectives

IS MSDELTA BETTER THAN DISM?

Research Objectives

- Figure out logic of a successful CAB (problem in current Windows update processing)

IS MSDELTA BETTER THAN DISM?

Research Objectives

- Figure out logic of a successful CAB (problem in current Windows update processing)
- Determine whether to use MSDelta or DISM

IS MSDELTA BETTER THAN DISM?

Research Objectives

- Figure out logic of a successful CAB (problem in current Windows update processing)
  - **CAB-WIM Success Logic:** Can we predict when a CAB will successfully apply to a WIM?
- Determine whether to use MSDelta or DISM

IS MSDELTA BETTER THAN DISM?

Research Objectives

- Figure out logic of a successful CAB (problem in current Windows update processing)
  - **CAB-WIM Success Logic:** Can we predict when a CAB will successfully apply to a WIM?
  - **CAB Order:** Can we successfully apply CABs out of order? Can we produce a new kernel with a mis-ordered sequence of CABs?
- Determine whether to use MSDelta or DISM

IS MSDELTA BETTER THAN DISM?

Research Objectives

- Figure out logic of a successful CAB (problem in current Windows update processing)
  - **CAB-WIM Success Logic:** Can we predict when a CAB will successfully apply to a WIM?
  - **CAB Order:** Can we successfully apply CABs out of order? Can we produce a new kernel with a mis-ordered sequence of CABs?
- Determine whether to use MSDelta or DISM
  - **MSDelta Reliability:** Is there ever a case where a CAB applies successfully in DISM but a patch component in MSDelta doesn't?

IS MSDELTA BETTER THAN DISM?

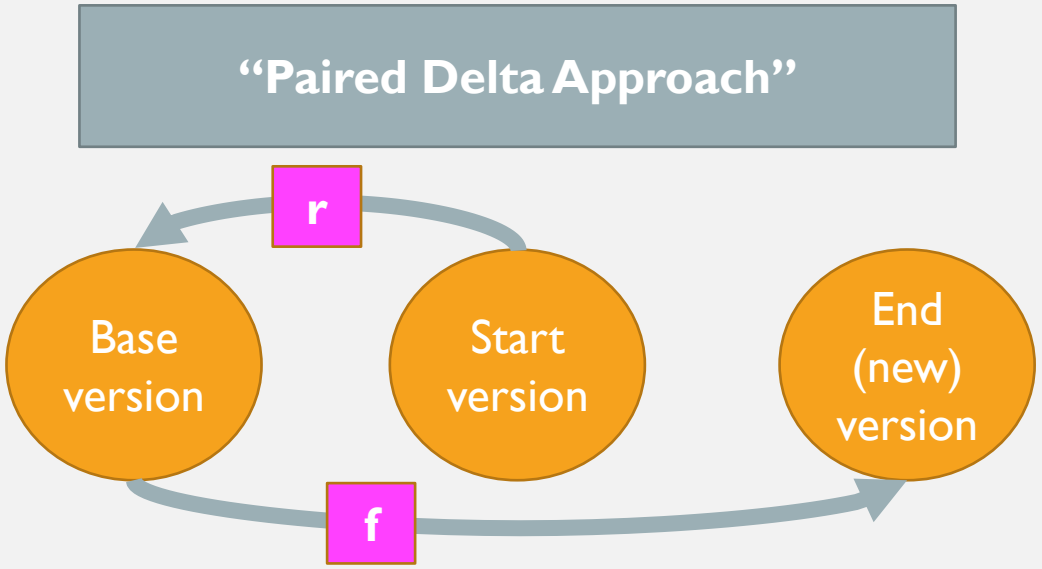
Master Experiment

- **6,071 trials**
  - 13 images (WIM versions)
  - 231 CABs (inner-most)
- **Trial types**
  - 3,003 CAB isolated trials (***DISM***)
  - 3,003 Patch component trials (***MSDelta***)
  - 65 CAB sequential trials (***DISM***)
    - 5 per image (had to do other experiments with smaller range of CABs)

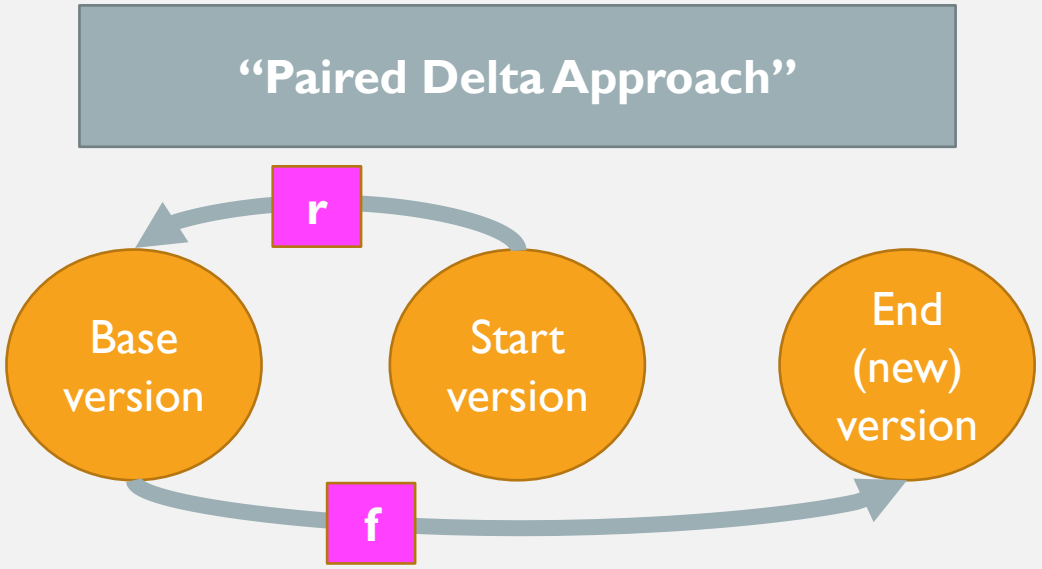
Other Experiments

- **CAB sequential experiments**
  - 860 trials
  - 9 images
  - 144 CABs (not all were applied to every WIM)
- **All 1909 CABs**
  - 2 images
  - 40 CABs "for" 1909
- **Other baby experiments**

PATCH DELTAS



PATCH DELTAS



- **Why this approach? Linear rather than quadratic growth in size of updates**
  - **@Jason, plus more in appendix**

ANATOMY OF A PATCH COMPONENT

- When extracting patch components for a CAB, each component has:

ANATOMY OF A PATCH COMPONENT

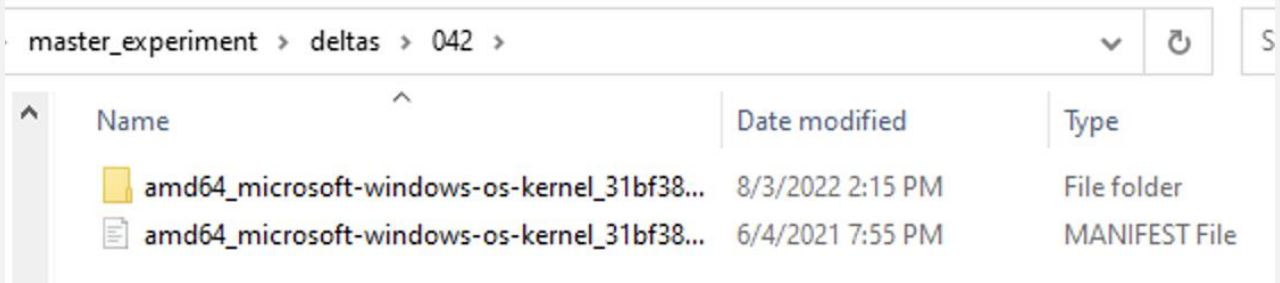
- When extracting patch components for a CAB, each component has:
  - **Patch deltas** (executables applied to a target file, e.g., ntoskrnl.exe)

ANATOMY OF A PATCH COMPONENT

- When extracting patch components for a CAB, each component has:
  - **Patch deltas** (executables applied to a target file, e.g., ntoskrnl.exe)
  - **A corresponding manifest file** (which may be in a different CAB than the exes)

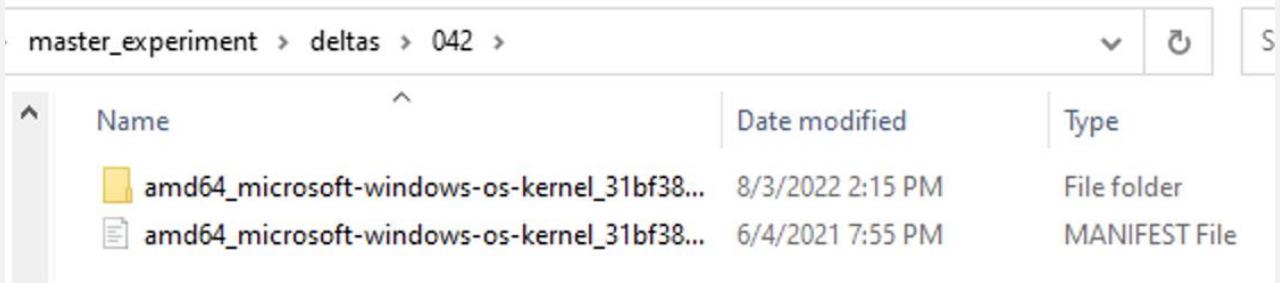
ANATOMY OF A PATCH COMPONENT

- When extracting patch components for a CAB, each component has:
  - **Patch deltas** (executables applied to a target file, e.g., ntoskrnl.exe)
  - **A corresponding manifest file** (which may be in a different CAB than the exes)



ANATOMY OF A PATCH COMPONENT

- When extracting patch components for a CAB, each component has:
  - **Patch deltas** (executables applied to a target file, e.g., ntoskrnl.exe)
  - **A corresponding manifest file** (which may be in a different CAB than the exes)

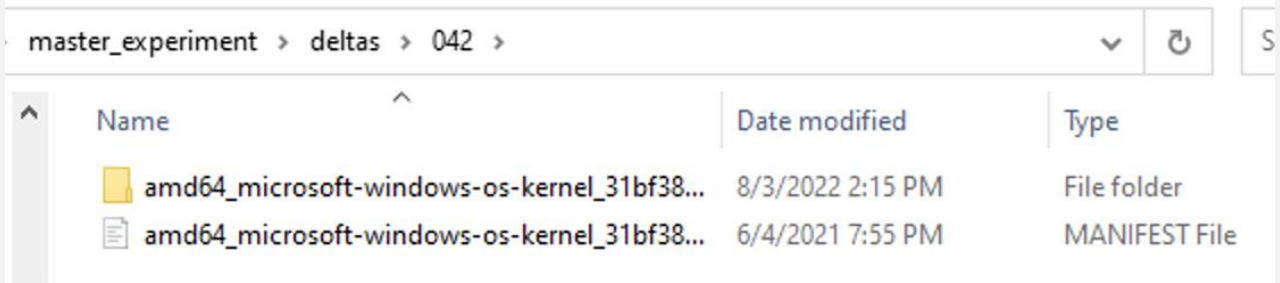


manifest files!

- Contains expected result of the patch in the form of hashes (and kernel versions)

ANATOMY OF A PATCH COMPONENT

- When extracting patch components for a CAB, each component has:
  - **Patch deltas** (executables applied to a target file, e.g., ntoskrnl.exe)
  - **A corresponding manifest file** (which may be in a different CAB than the exes)



manifest files!

- Contains expected result of the patch in the form of hashes (and kernel versions)

WinSxS Manifests

The .manifest files in the patch describe how the patch is to be applied, the files that are part of the patch, the expected result of the patch in the form of file hashes, permissions of the resulting files, registry keys to set, and more. They define the effects that happen to the system other than replacing the file that is being updated.

ANATOMY OF A PATCH COMPONENT

patch deltas!

credit: wumb0 article

Scenario #1: Paired Deltas

- **Forward diffs (f) –** brings the base binary (.1) up to a particular patch level

ANATOMY OF A PATCH COMPONENT

patch deltas!

credit: wumb0 article

Scenario #1: Paired Deltas

- **Forward diffs (f) –** brings the base binary (.1) up to a particular patch level
- **Reverse diffs (r) –** reverts the applied patch back to the base binary (.1)

ANATOMY OF A PATCH COMPONENT

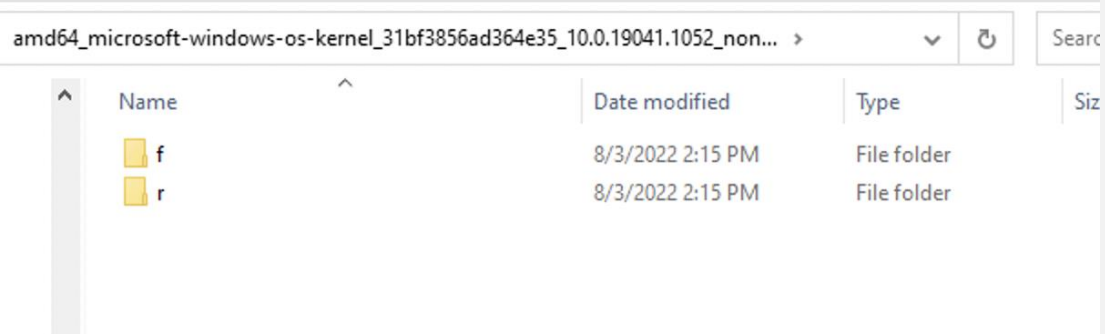
patch deltas!

credit: wumb0 article

Scenario #1: Paired Deltas

- **Forward diffs (f) –** brings the base binary (.1) up to a particular patch level
- **Reverse diffs (r) –** reverts the applied patch back to the base binary (.1)

"You will always see r and f folders together inside of a patch because you need to be able to revert the patch later on to apply a newer update." –wumb0



ANATOMY OF A PATCH COMPONENT

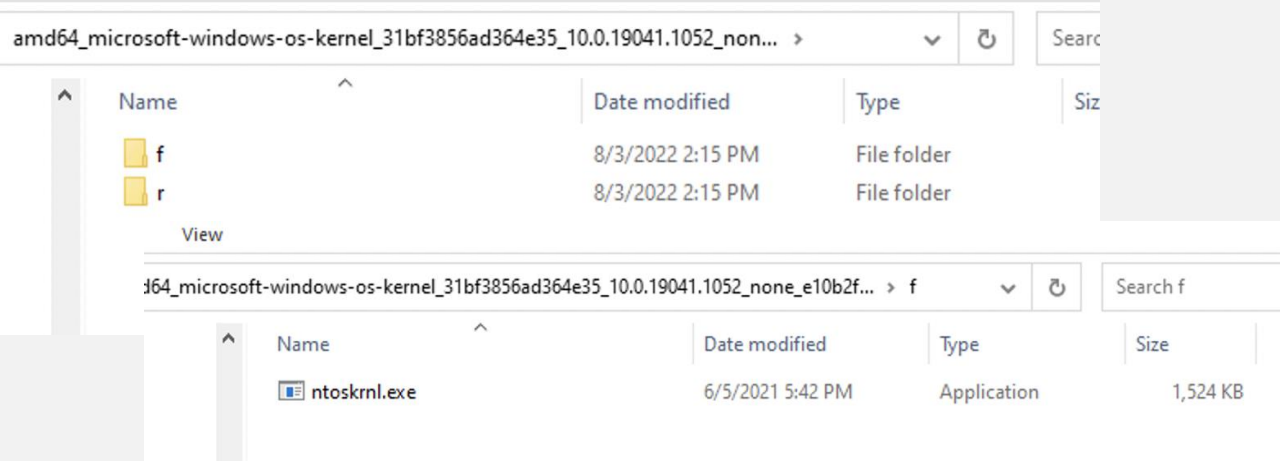
patch deltas!

credit: wumb0 article

Scenario #1: Paired Deltas

- **Forward diffs (f) –** brings the base binary (.1) up to a particular patch level
- **Reverse diffs (r) –** reverts the applied patch back to the base binary (.1)

"You will always see r and f folders together inside of a patch because you need to be able to revert the patch later on to apply a newer update." –wumb0



ANATOMY OF A PATCH COMPONENT

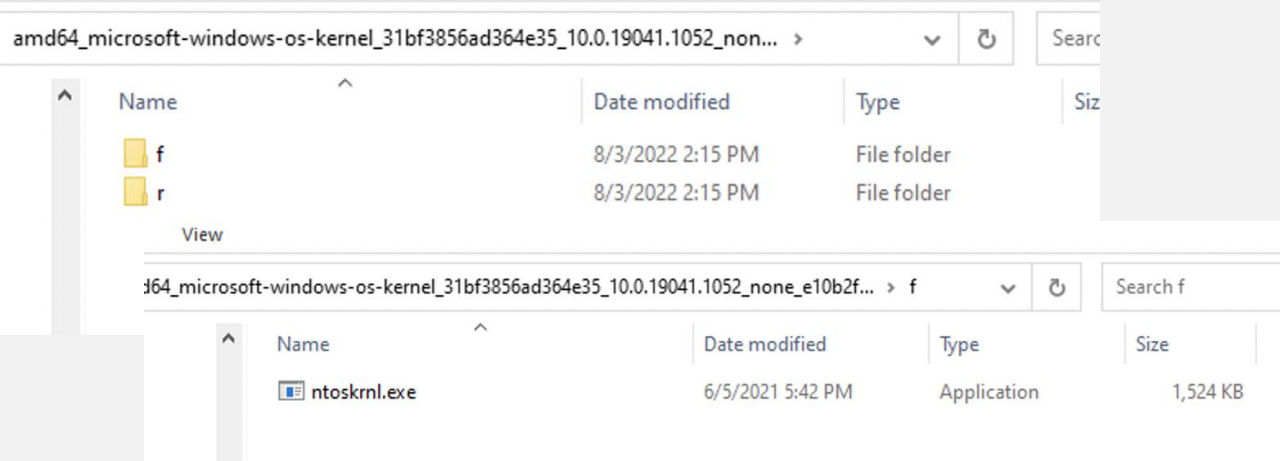
patch deltas!

credit: wumb0 article

Scenario #1: Paired Deltas

- **Forward diffs (f) –** brings the base binary (.1) up to a particular patch level
- **Reverse diffs (r) –** reverts the applied patch back to the base binary (.1)

"You will always see r and f folders together inside of a patch because you need to be able to revert the patch later on to apply a newer update." –wumb0



Not an executable! An MSDelta patch file (PA30)

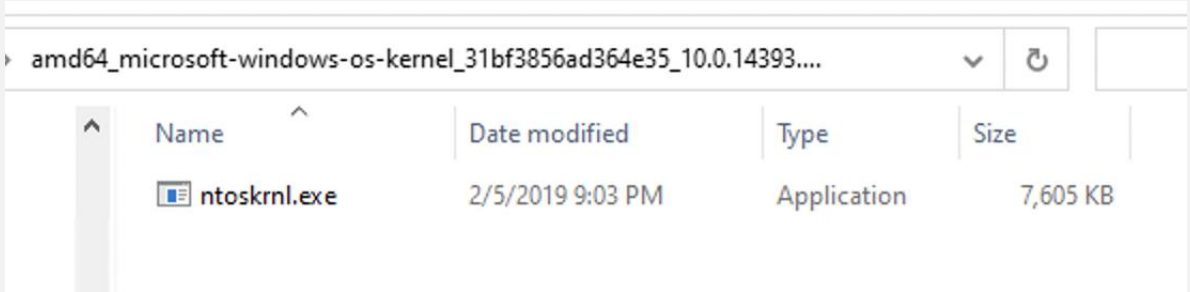
ANATOMY OF A PATCH COMPONENT

patch deltas!

credit: wumb0 article

Scenario #2: Null Diff

- **Null diffs (n) –** a completely new file, just compressed; apply to an empty buffer to get the full file
  - **Root or n subdirectory**



ANATOMY OF A PATCH COMPONENT

patch deltas!

credit: wumb0 article

Scenario #2: Null Diff

- **Null diffs (n) –** a completely new file, just compressed; apply to an empty buffer to get the full file
  - **Root or n subdirectory**

amd64_microsoft-windows-os-kernel_31bf3856ad364e35_10.0.14393....

Name	Date modified	Type	Size
ntoskrnl.exe	2/5/2019 9:03 PM	Application	7,605 KB

This *is* an executable

ANATOMY OF A KERNEL BUILD NUMBER

A kernel build number:

10.0.17763.55

ANATOMY OF A KERNEL BUILD NUMBER

A kernel build number:

10.0.17763.55

Major.Minor.Build.Revision

ANATOMY OF A KERNEL BUILD NUMBER

A kernel build number:

10.0.17763.55

Major.Minor.Build.Revision

- According to Microsoft:
  - ***Base version:*** *A major software release with significant changes, such as Windows 10, version 1809 (Windows 10 Build 17763.1)*
  - ***Revision:*** *Minor releases in between the major version releases, such as KB4464330 (Windows 10 Build 17763.55)*

Research Objective:
Figure out logic of successful CAB

CAB-WIM SUCCESS LOGIC

Q: Can we predict when a CAB will successfully apply to a WIM?

Research Objective:
Figure out logic of successful CAB

CAB-WIM SUCCESS LOGIC

Q: Can we predict when a CAB will successfully apply to a WIM?

- **A:** ***Yes.***

Research Objective:
Figure out logic of successful CAB

CAB-WIM SUCCESS LOGIC

Q: Can we predict when a CAB will successfully apply to a WIM?

- **A:** ***Yes.***
  - Success is based on *both* **build** AND **revision** number

Research Objective:
Figure out logic of successful CAB

CAB-WIM SUCCESS LOGIC

Q: Can we predict when a CAB will successfully apply to a WIM?

- **A:** ***Yes.***
  - Success is based on *both* **build** AND **revision** number

10.0.17763.55

Major.Minor.Build.Revision

Research Objective:
Figure out logic of successful CAB

CAB-WIM SUCCESS LOGIC

Q: Can we predict when a CAB will successfully apply to a WIM?

- **A: *Yes.***
  - Success is based on *both* **build** AND **revision** number
  - Success = no errors + new kernel
    - More nuanced in the full results (e.g., CABSs without kernel deltas)

10.0.17763.55

Major.Minor.Build.Revision

Research Objective:
Figure out logic of successful CAB

CAB-WIM SUCCESS LOGIC

- **Successful Patches (CAB and Patch Component)**

Major.Minor.Build.Revision

Research Objective:
Figure out logic of successful CAB

CAB-WIM SUCCESS LOGIC

- **Successful Patches (CAB and Patch Component)**
  - All successful patches had:

Major.Minor.Build.Revision

Research Objective:
Figure out logic of successful CAB

CAB-WIM SUCCESS LOGIC

- **Successful Patches (CAB and Patch Component)**
  - All successful patches had:
    - **build numbers equal to** the build number of a given WIM, and

10.0.17763.60

10.0.17763.55

Major.Minor.Build.Revision

Research Objective:
Figure out logic of successful CAB

CAB-WIM SUCCESS LOGIC

- **Successful Patches (CAB and Patch Component)**
  - All successful patches had:
    - **build numbers equal to** the build number of a given WIM, and
    - **a revision number greater than** that of the WIM

10.0.17763.60

10.0.17763.55

Major.Minor.Build.Revision

Research Objective:
Figure out logic of successful CAB

CAB-WIM SUCCESS LOGIC

- **Successful Patches (CAB and Patch Component)**
  - All successful patches had:
    - **build numbers equal to** the build number of a given WIM, and
    - **a revision number greater than** that of the WIM

10.0.17763.60

10.0.17763.55

Major.Minor.Build.Revision

Golden Rule for a Successful Patch

A patch component for 10.0.x.y will only successfully apply to 10.0.a.b if x==a and y>b.

Note: the edge/trivial case of x==a AND y==b (patch is the same as the image) was not tested in this project.

CAB ORDER

Q: Can we successfully apply CABs out of order?

Q: Can we produce a new kernel with a mis-ordered sequence of CABs?

CAB ORDER

Q: Can we successfully apply CABs out of order?

Q: Can we produce a new kernel with a mis-ordered sequence of CABs?

- **A: *Sometimes.***

Research Objective:
Figure out logic of successful CAB

CAB ORDER

Q: Can we successfully apply CABs out of order?

Q: Can we produce a new kernel with a mis-ordered sequence of CABs?

- **A: *Sometimes.***

Not super important. MSDelta is now hot shot.

Research Objective:
Figure out logic of successful CAB

CAB ORDER

Q: Can we successfully apply CABs out of order?

Q: Can we produce a new kernel with a mis-ordered sequence of CABs?

- **A: *Sometimes.***
  - Depends on whether there are CABs in the sequence with the potential to successfully apply (build number matches WIM)

Not super important. MSDelta is now hot shot.

CAB ORDER

Q: Can we successfully apply CABs out of order?

Q: Can we produce a new kernel with a mis-ordered sequence of CABs?

- **A: *Sometimes.***
  - Depends on whether there are CABs in the sequence with the potential to successfully apply (build number matches WIM)
  - If so, the final kernel takes the CAB with the most recent kernel version, even if that CAB wasn't the last CAB applied.

Not super important. MSDelta is now hot shot.

CAB ORDER

Q: Can we successfully apply CABs out of order?

Q: Can we produce a new kernel with a mis-ordered sequence of CABs?

- **A: *Sometimes.***
  - Depends on whether there are CABs in the sequence with the potential to successfully apply (build number matches WIM)
  - If so, the final kernel takes the CAB with the most recent kernel version, even if that CAB wasn't the last CAB applied.
  - The ability of a CAB sequence to produce a new kernel depends on whether a failed CAB in the sequence (if there is one) corrupts the mount image, thereby ruining subsequent CABs from successfully applying.

Not super important. MSDelta is now hot shot.

MSDELTA RELIABILITY

Q: Is there ever a case where a CAB applies successfully in DISM but a patch component in MSDelta doesn't?

MSDELTA RELIABILITY

Q: Is there ever a case where a CAB applies successfully in DISM but a patch component in MSDelta doesn't?

- **A: *No.***

MSDELTA RELIABILITY

Q: Is there ever a case where a CAB applies successfully in DISM but a patch component in MSDelta doesn't?

- **A: *No.***
  - For all observed trials, when MSDelta fails, the corresponding DISM trial also fails. There are no observed cases of an MSDelta trial failing and the corresponding DISM trial succeeding.

MSDELTA RELIABILITY

Q: Is there ever a case where a CAB applies successfully in DISM but a patch component in MSDelta doesn't?

- **A: *No.***
  - For all observed trials, when MSDelta fails, the corresponding DISM trial also fails. There are no observed cases of an MSDelta trial failing and the corresponding DISM trial succeeding.
  - In addition, DISM trials are less reliable than MSDelta trials, since it is still possible for a DISM trial to fail despite a matching build number.

MSDELTA RELIABILITY

Q: Is there ever a case where a CAB applies successfully in DISM but a patch component in MSDelta doesn't?

- **A: *No.***
  - For all observed trials, when MSDelta fails, the corresponding DISM trial also fails. There are no observed cases of an MSDelta trial failing and the corresponding DISM trial succeeding.
  - In addition, DISM trials are less reliable than MSDelta trials, since it is still possible for a DISM trial to fail despite a matching build number.
    - We do not see this in MSDelta trials **due to the reverse diffing process of applying patch components.**

MSDELTA RELIABILITY

Q: Is there ever a case where a CAB applies successfully in DISM but a patch component in MSDelta doesn't?

- **A: *No.***
  - For all observed trials, when MSDelta fails, the corresponding DISM trial also fails. There are no observed cases of an MSDelta trial failing and the corresponding DISM trial succeeding.
  - In addition, DISM trials are less reliable than MSDelta trials, since it is still possible for a DISM trial to fail despite a matching build number.
    - We do not see this in MSDelta trials **due to the reverse diffing process of applying patch components.**
  - *Therefore, we can predict whether MSDelta will successfully apply a patch component just based on build numbers*

CONCLUSIONS

CONCLUSIONS

MSDelta beats DISM

CONCLUSIONS

MSDelta beats DISM

Golden Rule for successful patch

MSDELTA BEATS DISM

MSDELTA BEATS DISM

- **More precise**
  - Just the kernel patch, not entire CAB

MSDELTA BEATS DISM

- **More precise**
  - Just the kernel patch, not entire CAB
- **Much faster**
  - No mounting/unmounting for every trial – only need base version
  - 30 mins in MSDelta vs. 2.5 hours in DISM

MSDELTA BEATS DISM

- **More precise**
  - Just the kernel patch, not entire CAB
- **Much faster**
  - No mounting/unmounting for every trial – only need base version
  - 30 mins in MSDelta vs. 2.5 hours in DISM
- **More reliable**
  - Succeeds in some cases where DISM fails (never vice versa)
  - Theory: isolates kernel patch from corruption in other components' patches

MSDELTA BEATS DISM

- **More precise**
  - Just the kernel patch, not entire CAB
- **Much faster**
  - No mounting/unmounting for every trial – only need base version
  - 30 mins in MSDelta vs. 2.5 hours in DISM
- **More reliable**
  - Succeeds in some cases where DISM fails (never vice versa)
  - Theory: isolates kernel patch from corruption in other components' patches
- **Likely to be able to run on other platforms**
  - DLL – doesn't require Windows system services (distributed with Wine)
  - Whereas DISM depends on other Windows system services

MSDELTA BEATS DISM

- **More precise**
  - Just the kernel patch, not entire CAB
- **Much faster**
  - No mounting/unmounting for every trial – only need base version
  - 30 mins in MSDelta vs. 2.5 hours in DISM
- **More reliable**
  - Succeeds in some cases where DISM fails (never vice versa)
  - Theory: isolates kernel patch from corruption in other components' patches
- **Likely to be able to run on other platforms**
  - DLL – doesn't require Windows system services (distributed with Wine)
  - Whereas DISM depends on other Windows system services
- **Bonus**: sheds light on how DISM's "Add-Package" command works
  - Used to apply a full CAB

MSDELTA BEATS DISM

- **More precise**
  - Just the kernel patch, not entire CAB
- **Much faster**
  - No mounting/unmounting for every trial – only need base version
  - 30 mins in MSDelta vs. 2.5 hours in DISM
- **More reliable**
  - Succeeds in some cases where DISM fails (never vice versa)
  - Theory: isolates kernel patch from corruption in other components' patches
- **Likely to be able to run on other platforms**
  - DLL – doesn't require Windows system services (distributed with Wine)
  - Whereas DISM depends on other Windows system services
- **Bonus**: sheds light on how DISM's "Add-Package" command works
  - Used to apply a full CAB

"There's work to do."

- Right now, just a proof of concept

GOLDEN RULE

GOLDEN RULE

A patch component for 10.0.x.y will only successfully apply to 10.0.a.b if x==a and y>b.

GOLDEN RULE

A patch component for 10.0.x.y will only successfully apply to 10.0.a.b if x==a and y>b.

TLDR

- Build numbers must match
- Revision number of patch must be greater than that of WIM/base version

GOLDEN RULE

- We figured out something about the logic of a CAB!
- Help predict whether kernel patch component will successfully apply.

A patch component for 10.0.x.y will only successfully apply to 10.0.a.b if x==a and y>b.

TLDR

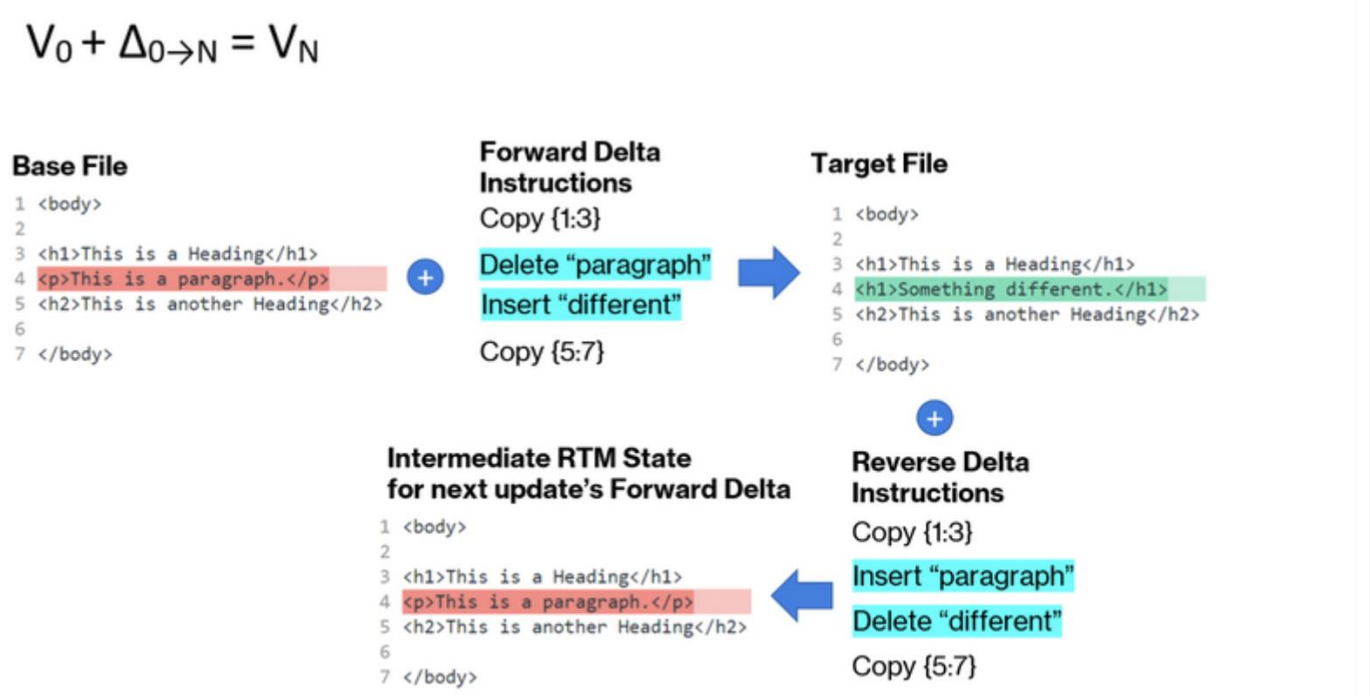
- Build numbers must match
- Revision number of patch must be greater than that of WIM/base version

THANKS FOR AN AWESOME INTERNSHIP

QUESTIONS

APPENDIX

PAIRED DELTAS DEEP DIVE



The delta pairs used in Windows Updates. The endpoints that have the base version of the file (V0) hydrate the target revision (VN) by applying a delta.

WORKS REFERENCED

- Extracting and Diffing Windows Patches in 2020 - wumb0in'
- Windows Updates using forward and reverse differentials - Windows Deployment | Microsoft Docs
- How Microsoft reduced Windows 11 update size by 40% - Microsoft Tech Community