Incident Response Intro Exercises

Legend:

Input Command

Output of the previous command

Prerequisites

- Windows Workstation or Server

What you need to know

NTFS File System

NTFS (New Technology File System) is a proprietary file system developed by Microsoft Corporation, introduced with the Windows NT operating system in 1993. It is the default file system for modern versions of Windows, including Windows NT, 2000, XP, Vista, 7, 8, and 10. NTFS offers several advanced features compared to its predecessor, the FAT (File Allocation Table) file system, making it more robust, secure, and suitable for modern computing needs. Here's a detailed explanation of NTFS:

1. File Allocation and Data Structures:

NTFS uses a file allocation system based on clusters, similar to other file systems. However, NTFS clusters can range in size from 512 bytes to 64KB, allowing for more efficient use of disk space, especially on large volumes. NTFS organizes files using several key data structures:

- **Master File Table (MFT):** NTFS maintains a Master File Table, which serves as a central database containing metadata for all files and directories on the disk. Each file and directory entry in the MFT includes information such as file name, timestamps, security descriptors, and pointers to the file's data clusters.
- **File Records:** Each file or directory on an NTFS volume is represented by a file record in the MFT. File records store detailed information about the corresponding file, including data attributes, security information, and file attributes.
- **Attribute Lists:** NTFS supports multiple data streams and file attributes, allowing files to have extended metadata beyond the traditional file contents. Attribute lists within file records store information about file attributes, such as data streams, security descriptors, and file attributes.

2. Security and Permissions:

NTFS offers robust security features, including file and folder permissions, access control lists (ACLs), and encryption capabilities:

- **Access Control Lists (ACLs):** NTFS allows administrators to define fine-grained permissions for files and directories using ACLs. Each file or directory has an associated ACL, which specifies the users or groups that are granted or denied various permissions, such as read, write, execute, or modify.
- **Security Descriptors:** NTFS uses security descriptors to store detailed security information for files and directories, including owner information, ACLs, and auditing settings.
- **Encryption:** NTFS supports file-level encryption through technologies like Encrypting File System (EFS), allowing users to encrypt sensitive data to protect it from unauthorized access.

3. Reliability and Fault Tolerance:

NTFS includes features designed to enhance data reliability and recoverability in case of system failures or disk errors:

- **Journaling:** NTFS uses a transactional journaling system to track changes made to the file system. This journaling mechanism helps ensure file system consistency and allows for faster recovery in the event of system crashes or power failures.
- **Metadata Duplication:** Critical metadata structures, such as the MFT and key system files, are duplicated on the disk to provide redundancy and improve fault tolerance. This redundancy helps protect against data loss caused by disk corruption or hardware failures.
- **Check Disk (Chkdsk):** NTFS includes a built-in utility called Chkdsk, which can scan the file system for errors, repair corrupted files, and recover lost data clusters.

4. Advanced Features:

NTFS offers several advanced features to support modern computing requirements:

- **Sparse Files:** NTFS supports sparse files, allowing applications to allocate disk space efficiently by marking regions of a file as empty or "sparse" without actually storing data on disk until it is needed.
- **Compression:** NTFS provides built-in file compression capabilities, allowing users to compress individual files or entire directories to save disk space.
- **Transaction Support:** NTFS supports file system transactions, enabling atomicity and consistency for operations involving multiple file system changes. This feature is particularly useful for database applications and other scenarios requiring transactional integrity.

Summary:

NTFS is a powerful and feature-rich file system designed for modern computing environments. Its advanced capabilities, robust security features, and fault tolerance mechanisms make it well-suited for a wide range of applications, from personal computing to enterprise storage systems. Despite being primarily associated with Windows operating systems, NTFS can also be accessed and used on other platforms through third-party drivers and utilities.

Journaling

Journaling in NTFS (New Technology File System) is a crucial feature that enhances the reliability and integrity of the file system by maintaining a transactional log, known as the journal. The journal records changes made to the file system before they are committed to disk, providing a mechanism for recovery in case of system crashes or power failures. Here's how journaling works in NTFS:

1. Transaction Logging:

When changes are made to the NTFS file system, such as creating, modifying, or deleting files and directories, these changes are first recorded in the journal before being applied to the actual file system structures on the disk. The journal acts as a temporary record of these changes, ensuring that they are logged in a sequential and atomic manner.

2. Atomicity:

One of the key benefits of journaling is atomicity, which ensures that file system operations are either fully completed or not performed at all. In NTFS, changes recorded in the journal are committed to disk in atomic transactions. If a system crash or power failure occurs during a file system operation, NTFS can use the information stored in the journal to either complete or roll back the transaction, maintaining the consistency of the file system.

3. Recovery Mechanism:

In the event of a system crash or unexpected shutdown, NTFS uses the journal to perform recovery operations during the next system boot. When the system restarts, the NTFS driver checks the journal to identify any incomplete transactions or corrupted file system structures. It then uses the information in the journal to replay or undo the logged transactions, restoring the file system to a consistent state.

4. Improving Reliability:

Journaling significantly improves the reliability and fault tolerance of NTFS by reducing the risk of data loss or file system corruption due to system crashes or hardware failures. By maintaining a transactional log of changes, NTFS can ensure that file system operations are resilient to interruptions and can be safely rolled back if necessary.

5. Performance Considerations:

While journaling provides important benefits for data integrity and recovery, it may also have some performance implications. The overhead associated with logging changes in the journal can impact file system performance, particularly on systems with high disk I/O activity. However, the benefits of improved reliability and recovery often outweigh the performance overhead for most users and organizations.

6. Configurability:

NTFS provides options for configuring the size and behavior of the journal to accommodate different usage scenarios and performance requirements. System administrators can adjust journaling settings using tools such as Group Policy or registry settings to optimize performance or prioritize data integrity based on their specific needs.

Conclusion:

Journaling is a critical feature of NTFS that enhances the reliability, integrity, and recoverability of the file system. By maintaining a transactional log of changes, NTFS can ensure that file system operations are atomic and resilient to system crashes or power failures. Journaling plays a vital role in minimizing the risk of data loss and file system corruption, making NTFS a robust and dependable file system for modern computing environments.

MetaData Dedupe

Metadata duplication in NTFS involves replicating critical file system metadata structures, such as the Master File Table (MFT) and key system files, across multiple locations on the disk. This duplication strategy is implemented to enhance fault tolerance and improve data reliability in the face of disk corruption or hardware failures. Here's an expanded explanation:

1. Master File Table (MFT) Duplication:

The Master File Table (MFT) is a central database in NTFS that stores metadata for all files and directories on the disk. To ensure the integrity of the file system, NTFS duplicates critical portions of the MFT at multiple locations on the disk. This duplication includes crucial entries and attributes essential for file system operation.

2. Redundancy for Fault Tolerance:

By duplicating critical metadata structures like the MFT, NTFS creates redundancy within the file system. Redundancy means that if one copy of the MFT or other critical metadata becomes corrupted due to disk errors or hardware failures, there are additional copies available to serve as backups. This redundancy significantly improves fault tolerance and increases the likelihood of recovering from data corruption without data loss.

3. Protection Against Disk Corruption:

Disk corruption can occur due to various factors, including physical damage to the storage media, software bugs, or abrupt power loss. When disk corruption affects critical metadata structures like the MFT, it can lead to file system inconsistencies and data loss. By duplicating these structures, NTFS provides an additional layer of protection against such corruption. Even if one copy of the metadata becomes corrupted, NTFS can use redundant copies to repair the file system and restore data consistency.

4. Improved Data Reliability:

Metadata duplication contributes to overall data reliability in NTFS by reducing the risk of data loss and file system corruption. With redundant copies of critical metadata structures spread across the disk, NTFS can detect and recover from errors more effectively, ensuring that file system operations remain robust and data integrity is maintained.

5. Recovery Mechanisms:

In the event of disk corruption or hardware failures, NTFS employs various recovery mechanisms to restore the file system to a consistent state. These mechanisms may include using redundant copies of metadata structures, performing consistency checks, and repairing corrupted file system elements. Metadata duplication plays a crucial role in facilitating these recovery processes and minimizing data loss.

6. Performance Considerations:

While metadata duplication enhances fault tolerance and data reliability, it may have some performance implications. Duplicating critical metadata structures requires additional disk space and imposes overhead on file system operations. However, the benefits of improved fault tolerance and data integrity typically outweigh the minor performance impact, making metadata duplication a worthwhile trade-off for most users and organizations.

Summary:

Metadata duplication is a fundamental aspect of NTFS that contributes to its reliability and fault tolerance. By replicating critical metadata structures like the MFT across the disk, NTFS enhances data reliability, protects against disk corruption, and improves the file system's ability to recover from errors. Metadata duplication is a key feature of NTFS that ensures the integrity and availability of data in modern computing environments.

Data Streams

In NTFS (New Technology File System), data streams are a feature that allows files to contain multiple streams of data within a single file. This capability enables the storage of additional information or resources associated with a file without altering its primary data. Data streams provide a flexible mechanism for organizing and managing data, supporting various applications and use cases. Here's an explanation of data streams in NTFS:

1. Primary Data Stream:

Every file in NTFS has a primary data stream, which contains the main content or payload of the file. This primary data stream represents the conventional file data that users typically interact with, such as text, images, executables, or other types of binary data. The primary data stream is accessed and manipulated through standard file I/O operations.

2. Alternate Data Streams (ADS):

In addition to the primary data stream, NTFS supports alternate data streams (ADS), which are additional streams of data associated with a file. Alternate data streams are named and can contain different types of information, such as metadata, extended attributes, or embedded resources. Unlike the primary data stream, which is accessed directly by file system APIs, alternate data streams are accessed using special syntax that specifies the stream name within the file path.

3. Syntax for Accessing Alternate Data Streams:

To access an alternate data stream associated with a file, users can use a special syntax in the file path, indicating the name of the stream after a colon (`:`) delimiter. For example:

```
C:\path\to\file.txt:streamName
```

This syntax allows applications and utilities to read from or write to specific alternate data streams associated with a file. However, many standard file manipulation tools and APIs may not be aware of alternate data streams and may inadvertently ignore or overwrite them.

4. Use Cases for Alternate Data Streams:

Alternate data streams provide a versatile mechanism for storing additional information or resources within files, supporting various use cases:

- **Metadata:** Alternate data streams can be used to store metadata associated with a file, such as author information, file versioning details, or custom attributes.
- **Resource Forks:** In some scenarios, alternate data streams are used to store resource forks, which contain supplementary data related to file resources, such as icons, thumbnails, or embedded media.
- **Execution Context:** Alternate data streams can be leveraged to store execution context or configuration settings for applications, allowing them to customize behavior or store temporary state information.

5. Security Implications:

While alternate data streams provide flexibility and versatility, they also have security implications. Malicious actors may exploit alternate data streams to hide or disguise malicious code or data within seemingly innocuous files. Therefore, it's essential for security-conscious users and administrators to monitor and scrutinize alternate data streams to ensure the integrity and safety of the file system.

6. History:

The concept of alternate data streams (ADS) in NTFS has its roots in the development of the NTFS file system itself, which was introduced with the Windows NT operating system in the early 1990s. Alternate data streams were not initially part of the design of NTFS but emerged as a result of evolving requirements and the need for enhanced functionality. Here's a brief overview of the history behind alternate data streams in NTFS:

Development of NTFS:

NTFS was developed by Microsoft as a modern file system to replace the aging FAT (File Allocation Table) file system used in previous versions of Windows. NTFS was designed to offer improved reliability, scalability, and performance, as well as support for advanced features not available in FAT.

Support for Multiple Data Streams:

During the development of NTFS, Microsoft recognized the need for a file system that could support multiple data streams within a single file. This capability was not present in FAT and was seen as a valuable addition for supporting various types of data and metadata associated with files.

Incorporation of Alternate Data Streams:

Alternate data streams were introduced as a feature of NTFS to address the requirement for supporting multiple streams of data within files. With alternate data streams, NTFS provided a mechanism for storing additional information, metadata, or resources alongside the primary data content of a file.

4. Evolution and Adoption:

Alternate data streams in NTFS were initially introduced with limited fanfare and were not widely publicized by Microsoft. However, over time, the functionality gained attention from developers, administrators, and security professionals who recognized its potential applications and implications.

5. Use Cases and Applications:

As awareness of alternate data streams grew, various use cases and applications emerged. Developers began leveraging alternate data streams to store metadata, resource forks, custom attributes, and embedded resources within files. Security researchers also explored the security implications of alternate data streams, highlighting their potential for hiding or disguising malicious code.

6. Security Concerns:

While alternate data streams offer versatility and flexibility, they also raise security concerns due to their potential for misuse. Malicious actors may exploit alternate data streams to hide malware, bypass security controls, or evade detection by antivirus software. As a result, alternate data streams have been the subject of scrutiny and debate within the cybersecurity community.

7. Ongoing Support and Maintenance:

Despite the security considerations, Microsoft has continued to support alternate data streams as a feature of NTFS in subsequent versions of the Windows operating system. However, efforts have been made to educate users and administrators about the risks associated with alternate data streams and best practices for mitigating potential security threats.

Conclusion:

Alternate data streams in NTFS represent a feature that evolved from the development of the NTFS file system to address the need for supporting multiple data streams within files. While providing valuable functionality for various use cases, alternate data streams also raise security concerns due to their potential for misuse. Understanding the history and context behind alternate data streams is essential for effectively leveraging their capabilities while mitigating associated risks.

References

https://blog.netwrix.com/2022/12/16/alternate_data_stream/

https://attack.mitre.org/techniques/T1564/004/

https://www.varonis.com/blog/the-malware-hiding-in-your-windows-system32-folder-part-iii-certutil-and-alternate-data-streams

https://www.malwarebytes.com/blog/news/2015/07/introduction-to-alternate-data-streams

https://denwp.com/unveiling-the-stealth/

https://insights.sei.cmu.edu/blog/using-alternate-data-streams-in-the-collection-and-exfiltration-of-data/

The echo command

In the Windows Command-Line Interface (CLI), the `echo` command is used to display messages or enable/disable echoing of commands within batch scripts or directly in the command prompt. It is similar to the `echo` command found in Unix/Linux shells.

Here's how the `echo` command works in Windows CLI:

1. Basic Usage:

The basic syntax of the `echo` command is:

echo [message]

Where `[message]` is the text you want to display.

For example:

echo Hello, World!

```
Hello, World!
```

Special Characters:

The `echo` command can also display special characters using escape sequences:

- `\n`: Newline
- `\t`: Tab
- `\\`: Backslash
- `\"`: Double quote

For example:

echo Line 1\nLine 2

Line 1

Line 2

Echoing Environment Variables:

You can also use the `echo` command to display the values of environment variables. Simply prefix the variable name with `%` symbols.

For example:

```
echo The value of PATH is: %PATH%
```

Redirecting Output:

You can redirect the output of the `echo` command to a file using the `>` operator.

For example:

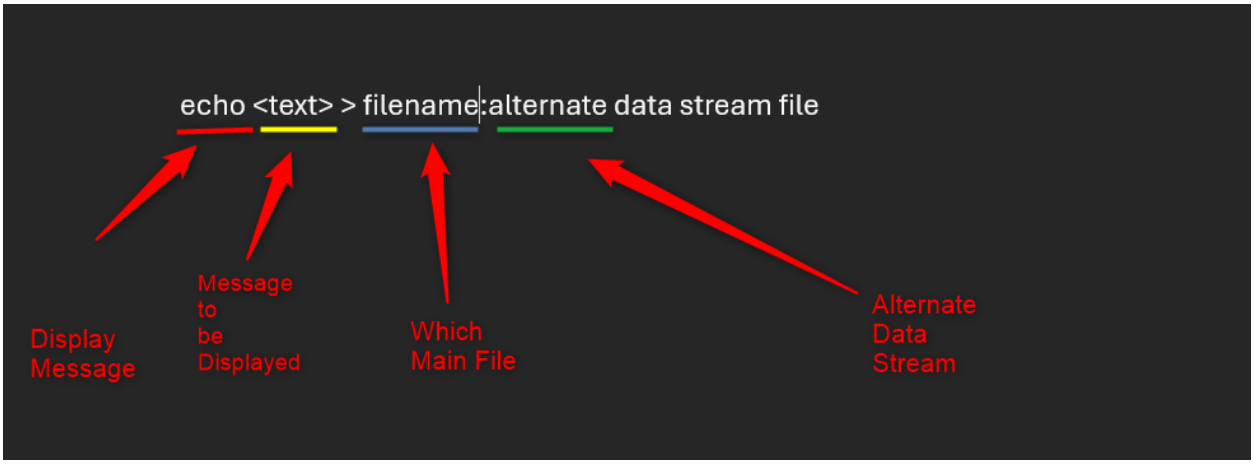
```
echo Hello, World! > output.txt
```

This command will write `Hello, World!` to a file named `output.txt`.

Exercise 1

Alternate Data Streams

echo <text> > filename:alternate data stream file



Example:

echo This is it! > file1.txt:hiddenfile.txt

Task #1

Create a simple text file using this echo command: echo Normal File > file_normal.txt

Task #2

Using the above method generate a hidden message as an alternate data stream.

echo Evil Malware > badfile.txt:hiddenfile.txt

Task #3

While in the directory that you created the file with the ADS, run the following command:

dir /r *.txt

The command `dir /r *.txt` is used in the Windows Command Prompt to list files in a directory, including alternate data streams, for files with a `.txt` extension. Let's break down the command:

- `dir`: This is the command used to list the contents of a directory. It stands for "directory."
- `/r`: This is an option or switch for the `dir` command. It instructs the `dir` command to display alternate data streams for each file listed. Alternate data streams are a feature of the NTFS file system that allows files to have multiple streams of data. By default, `dir` does not display alternate data streams.
- `*.txt`: This part of the command is a file pattern or wildcard filter. It tells the `dir` command to only list files with a `.txt` extension. The `*` character is a wildcard that matches any sequence of characters, and `.txt` specifies files with the `.txt` extension.

So, when you run `dir /r *.txt`, the Command Prompt will list all files with a `.txt` extension in the current directory, along with any alternate data streams associated with each file.

Task #4

What was the output of that `dir` command, what did that output mean?

Exercise 2

Alternate Data Streams – Things aren't always as they seem

Prerequisite

fciv must be in the system path. (protip, save it to your nmap folder, it is already in your path. C:\Program Files (x86)\Nmap)

Task #1

Create a simple text file using this echo command: echo Normal File > file2.txt

Task #2

Obtain a hash of file2.txt

```
C:\Users\yenri\Downloads>echo Normal File > file2.txt

C:\Users\yenri\Downloads>fciv file2.txt
//
// File Checksum Integrity Verifier version 2.05.
//
27d306fd5ac51bee8414d5d3ecbcc481 file2.txt
```

Should be: 27d306fd5ac51bee8414d5d3ecbcc481

Task #3

Add an alternate data stream to the file.

echo Evil Malware > file2.txt:evil.txt

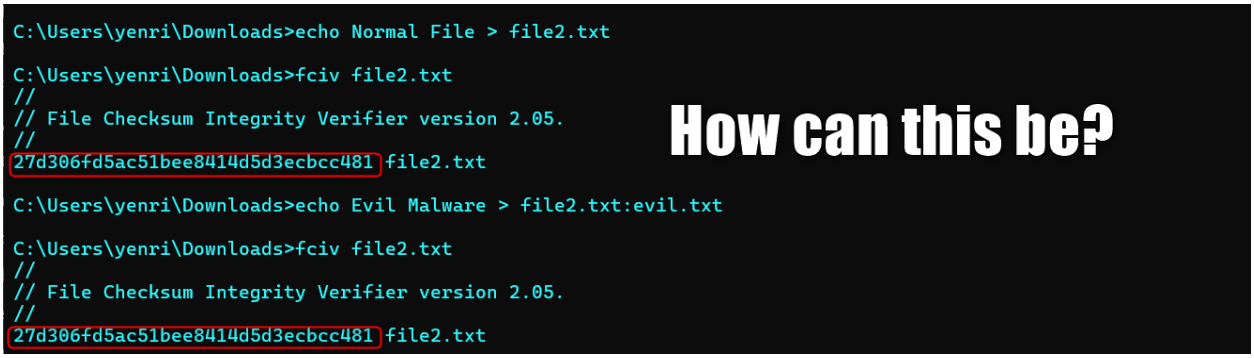
```
C:\Users\yenri\Downloads>echo Evil Malware > file2.txt:evil.txt
```

Task #4

Where are your data integrity gods now?

RE- Obtain a hash of file2.txt

It is STILL: 27d306fd5ac51bee8414d5d3ecbcc481



When adding data to an alternate data stream (ADS) in NTFS (New Technology File System), it **does not change the hash of the file** because the hash is typically calculated only on the primary data stream of the file.

1. Primary Data Stream:

In NTFS, every file has a primary data stream, which contains the main content or payload of the file. When calculating the hash of a file, most hashing algorithms, such as SHA-256 or MD5, operate on the contents of the primary data stream.

2. Alternate Data Streams (ADS):

Alternate data streams, on the other hand, are additional streams of data associated with a file. When data is added to an alternate data stream, it does not modify the primary data stream of the file. Instead, it adds supplementary data alongside the existing content.

3. Calculation of File Hash:

When calculating the hash of a file, hashing algorithms process the bytes of the primary data stream, generating a hash value based on the content of that stream. Since the alternate data streams are separate from the primary data stream, changes made to them do not affect the hash value calculated from the primary data stream.

4. Immutable Primary Data Stream:

The primary data stream of a file is typically considered immutable when calculating its hash. This means that any modifications made to alternate data streams do not alter the hash of the primary data stream or the file as a whole. As a result, the hash of the file remains unchanged even if data is added, modified, or removed from alternate data streams.

5. Impact on Integrity Verification:

While alternate data streams provide a means of storing additional data within a file, they do not impact the integrity verification process based on file hashes. Integrity checks performed using file hashes will continue to validate the content of the primary data stream, regardless of the presence or contents of alternate data streams.

6. Considerations for Security and Forensics:

Security-conscious users and forensic investigators should be aware of the existence of alternate data streams and their potential implications for data integrity and security. Although changes to alternate data streams do not affect file hashes, they may still contain valuable information or pose security risks if used to hide malicious content or sensitive data.

Summary:

In NTFS, adding data to an alternate data stream does not change the hash of the file because hashing algorithms typically operate on the contents of the primary data stream. This distinction between primary and alternate data streams ensures that modifications made to alternate streams do not impact the integrity verification process based on file hashes, maintaining the reliability and consistency of hash-based integrity checks.

Exercise 3

Alternate Data Streams - Detection

In PowerShell, you can use various cmdlets and techniques to detect alternate data streams (ADS) associated with files in the NTFS file system. One common approach involves using the `Get-Item` cmdlet along with stream information.

Using `Get-Item` Cmdlet:

The `Get-Item` cmdlet retrieves information about the specified file or directory, including its alternate data streams.

Here's how you can use it to detect alternate data streams:

```
Get-Item -Path "C:\path\to\file.txt" -Stream *
```

- Replace `"C:\path\to\file.txt"` with the path to the file you want to inspect.
- The `-Stream *` parameter retrieves information about all streams associated with the file, including the primary data stream and any alternate data streams.

Or for our example. Is to use *.txt to select all the files in the present working directory.

```
Get-Item *.txt -Stream *
```

When you run the above command, PowerShell will display information about the primary data stream and any alternate data streams associated with the specified file.

```
PSPath        : Microsoft.PowerShell.Core\FileSystem::C:\Users\yenri\Downloads\file1.txt::$DATA
PSParentPath  : Microsoft.PowerShell.Core\FileSystem::C:\Users\yenri\Downloads
PSChildName   : file1.txt::$DATA
PSDrive       : C
PSProvider    : Microsoft.PowerShell.Core\FileSystem
PSIsContainer : False
FileName      : C:\Users\yenri\Downloads\file1.txt
Stream        : :$DATA
Length        : 0

PSPath        : Microsoft.PowerShell.Core\FileSystem::C:\Users\yenri\Downloads\file1.txt:hidden.txt
PSParentPath  : Microsoft.PowerShell.Core\FileSystem::C:\Users\yenri\Downloads
PSChildName   : file1.txt:hidden.txt
PSDrive       : C
PSProvider    : Microsoft.PowerShell.Core\FileSystem
PSIsContainer : False
FileName      : C:\Users\yenri\Downloads\file1.txt
Stream        : hidden.txt
Length        : 17
```

In this example:

- `File1.txt` is the primary data stream of the file.
- `File1.txt:stream1.txt` and `file1.txt:stream2.txt` are alternate data streams associated with `file1.txt`.

Additional Information:

- You can use the `-Force` parameter with `Get-Item` to include hidden or system files in the search.
- The output includes information such as the file mode, last write time, length, and name of each stream.

4. Advanced Techniques:

You can also use PowerShell scripting to iterate through directories and detect alternate data streams for multiple files. For example, you can use a `foreach` loop combined with the `Get-ChildItem` cmdlet to process multiple files.

```
Get-ChildItem -Path "C:\path\to" -Recurse | ForEach-Object {
    Get-Item -Path $_.FullName -Stream *
}
```

This script will recursively search the specified directory and its subdirectories, displaying information about alternate data streams for each file encountered.

PowerShell provides powerful capabilities for detecting and inspecting alternate data streams associated with files in the NTFS file system. By using cmdlets such as `Get-Item` and `Get-ChildItem`, you can retrieve information about primary and alternate data streams, enabling you to better understand and manage file metadata and associated resources.