Velociraptor

Digging Deep

This is a sample module from our full “Enterprise Hunting and Incident Response” course.

Register for the full course at https://www.velocidex.com/training/

Forensic Analysis with VQL Pt1

Digging deeper in Windows

Module overview

Velociraptor implements many forensic capabilities in VQL

This module will focus on typical forensic analysis and deep inspection capabilities. We will learn how to put the capabilities together to produce effective artifacts and when to use those.

This module will not use Velociraptor's GUI or even the client/server mode since we are focused on the techniques themselves. Later we can leverage the same VQL across the network at scale, and effectively hunt for artifacts across our infrastructure - keep this in mind through this module.

Searching for files - glob()

One of the most common operations in DFIR is searching for files efficiently.

Velociraptor has the glob() plugin to search for files using a glob expression.

Glob expressions use wildcards to search the filesystem for matches.

- Paths are separated by / or \ into components
- A * is a wildcard match (e.g. *.exe matches all files ending with .exe)
- Alternatives are expressed as comma separated strings in {}
  - e.g. *.{exe,dll,sys}
- A ** denotes recursive search.
  - e.g. C:\Users***.exe

glob

Retrieve files based on a list of glob expressions

Plugin

The `glob()` plugin is one of the most used plugins. It applies a glob expression in order to search for files by file name. The glob expression allows for wildcards, alternatives and character classes. Globs support both forward and backslashes as path separators. They also support quoting to delimit components.

A glob expression consists of a sequence of components separated by path separators. If a separator is included within a component it is possible to quote the component to keep it together. For example, the windows registry contains keys with forward slash in their names. Therefore we may use these to prevent the glob from getting confused:

```
HKEY_LOCAL_MACHINE\Microsoft\Windows\"Some Key With http://www.microsoft.com/"\Some Value
```

Glob expressions are case insensitive and may contain the following wild cards:

- The `*` matches one or more characters.
- The `?` matches a single character.
- Alternatives are denoted by braces and comma delimited: `{a,b}`
- Recursive search is denoted by a `**`. By default this searches 3 directories deep. If you need to increase it you can add a depth number (e.g. `**10`)

By default globs do not expand environment variables. If you need to expand environment variables use the `expand()` function explicitly:

```
glob(globs=expand(string="%SystemRoot%\System32\Winevt\Logs\*"))
```

Exercise: Search for exe

Search user's home directory for binaries.

```
SELECT * FROM glob(globs='C:\\Users\\**\\*.exe')
```

Note the need to escape \ in strings. You can use / instead and specify multiple globs to search all at the same time:

```
SELECT * FROM glob(globs=['C:/Users/**/*.exe',
                          'C:/Users/**/*.dll'])
```


```
C:\Program Files\velociraptor>Velociraptor.exe -v query "SELECT * FROM glob(globs='C:\\Users\\**\\*.exe')"
[INFO] 2020-01-24T20:40:23-08:00 Loaded 145 built in artifacts
[
 {
  "Atime": {
   "sec": 1579866327,
   "usec": 1579866327633739000
  },
  "Ctime": {
   "sec": 1579866327,
   "usec": 1579866327633739000
  },
  "Data": "",
  "FullPath": "\\C:\\Users\\All Users\\Dbg\\sym\\ntoskrnl.exe",
  "GetLink": "",
  "IsDir": true,
  "IsLink": false,
  "ModTime": "2020-01-24T03:45:27.633739-08:00",
  "Mode": 2147484159,
  "Mtime": {
   "sec": 1579866327,
   "usec": 1579866327633739000
  },
  "Name": "ntoskrnl.exe",
  "Size": 0,
  "Sys": {
   "FileAttributes": 16,
   "CreationTime": {
    "LowDateTime": 3317799150,
    "HighDateTime": 30790315
```


Exercise - RunOnce artifact

Write an artifact which hashes every binary mentioned in Run/RunOnce keys.

“Run and RunOnce registry keys cause programs to run each time that a user logs on.”

MSDN

Run and RunOnce Registry Keys

05/31/2018 • 2 minutes to read •

Run and RunOnce registry keys cause programs to run each time that a user logs on. The data value for a key is a command line no longer than 260 characters. Register programs to run by adding entries of the form *description-string=commandline*. You can write multiple entries under a key. If more than one program is registered under any particular key, the order in which those programs run is indeterminate.

The Windows registry includes the following four keys:

- **HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run**
- **HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run**
- **HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce**
- **HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce**

By default, the value of a RunOnce key is deleted before the command line is run. You can prefix a RunOnce value name with an exclamation point (!) to defer deletion of the value until after the command runs. Without the exclamation point prefix, if the RunOnce operation fails the associated program will not be asked to run the next time you start the

Raw registry parsing

In the previous exercise we looked for a key in the HKEY_CURRENT_USER hive.

HKEY_CURRENT_USER	Registry entries subordinate to this key define the preferences of the current user. These preferences include the settings of environment variables, data about program groups, colors, printers, network connections, and application preferences. This key makes it easier to establish the current user's settings; the key maps to the current user's branch in **HKEY_USERS**. In **HKEY_CURRENT_USER**, software vendors store the current user-specific preferences to be used within their applications. Microsoft, for example, creates the **HKEY_CURRENT_USER\Software\Microsoft** key for its applications to use, with each application creating its own subkey under the **Microsoft** key. The mapping between **HKEY_CURRENT_USER** and **HKEY_USERS** is per process and is established the first time the process references **HKEY_CURRENT_USER**. The mapping is based on the security context of the first thread to reference **HKEY_CURRENT_USER**. If this security context does not have a registry hive loaded in **HKEY_USERS**, the mapping is established with **HKEY_USERS\.Default**. After this mapping is established it persists, even if the security context of the thread changes. All registry entries in **HKEY_CURRENT_USER** except those under **HKEY_CURRENT_USER\Software\Classes** are included in the per-user registry portion of a roaming user profile. To exclude other entries from a roaming user profile, store them in **HKEY_CURRENT_USER_LOCAL_SETTINGS**. This handle should not be used in a service or an application that impersonates different users. Instead, call the **RegOpenCurrentUser** function. For more information, see HKEY_CURRENT_USER.

Any artifacts looking in HKEY_USERS using the Windows API are limited to the set of users currently logged in! We need to parse the raw hive to reliably recover all users.

Raw registry parsing

Each user's setting is stored in:

C:\Users\<name>\ntuser.dat

It is a raw registry hive file format. We need to use raw_reg accessor.

The raw reg accessor uses a URL scheme to access the underlying file.

```
:\Program Files\velociraptor>Velociraptor.exe -v query "SELECT * FROM glob(globs=url(scheme='file', path='C:/Users/test/ntuser.dat',
fragment='/**/Run/*').String, accessor='raw_reg')"
INFO] 2020-01-25T06:27:39-08:00 Loaded 147 built in artifacts

{
 "Atime": {
  "sec": 1578620073,
  "usec": 1578620073000000000
 },
 "Ctime": {
  "sec": 1578620073,
  "usec": 1578620073000000000
 },
 "Data": {
  "type": "REG_SZ",
  "data_len": 148,
  "value": "\"C:\\Users\\test\\AppData\\Local\\Microsoft\\OneDrive\\OneDrive.exe\" /background"
 },
 "FullPath": "file:///C:/Users/test/ntuser.dat#Software/Microsoft/Windows/CurrentVersion/Run/OneDrive",
 "GetLink": "",
 "IsDir": false,
 "IsLink": false,
 "ModTime": "2020-01-09T17:34:33-08:00",
 "Mode": 493.
```

Searching data

Searching data

A powerful DFIR technique is searching bulk data for patterns

- ❏ Searching for CC data in process memory
- ❏ Searching for URLs in process memory
- ❏ Searching binaries for malware signatures
- ❏ Searching registry for patterns

Bulk searching helps to identify evidence without needing to parse file formats

YARA - The swiss army knife

YARA is a powerful keyword scanner
Uses rules designed to identify binary patterns in bulk data
YARA is optimized to scan for many rules simultaneously.
Velociraptor supports YARA scanning of bulk data (via accessors) and memory.

yara() and proc_yara()

YARA rules

```
rule X {
    strings:
        $a = “hello” nocase
        $b = “Goodbye” wide
        $c = /[a-z]{5,10}[0-9]/i

    condition:
        $a and ($b or $c)
}
```


Exercise: drive by download

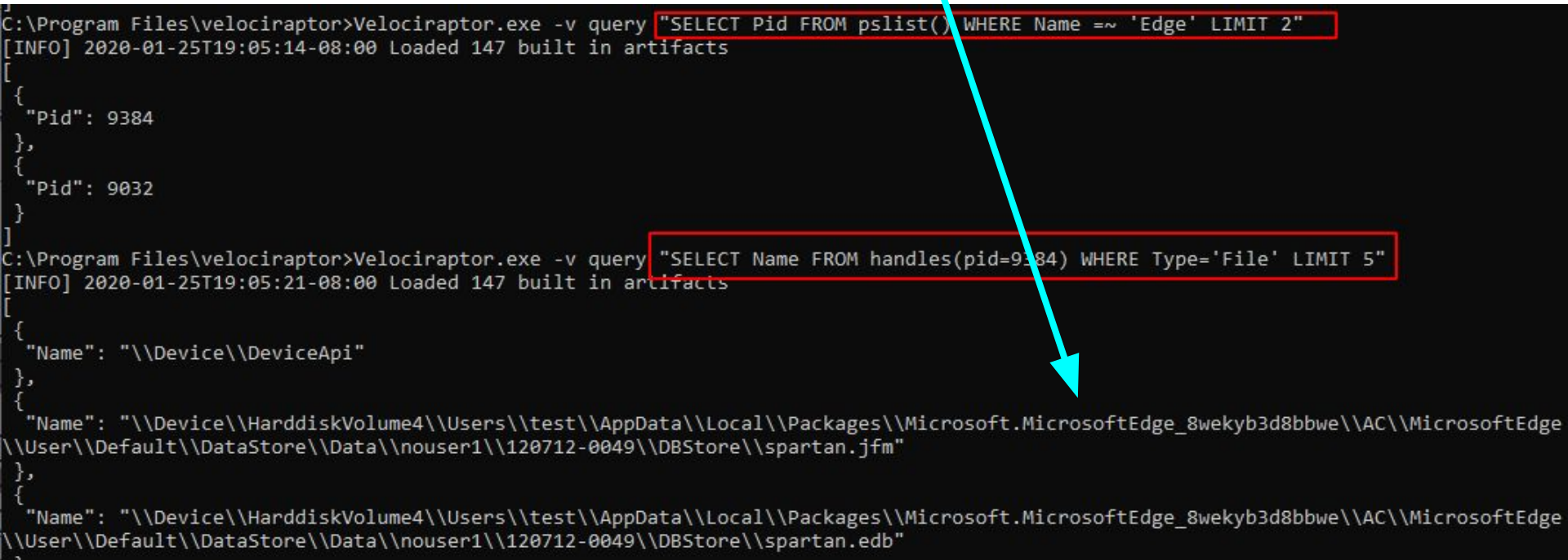
You suspect a user was compromised by a drive by download (i.e. they clicked and downloaded malware delivered by mail, ads etc).

You think the user used the **Edge** browser but you have no idea of the internal structure of the browser cache/history etc.

Write an artifact to extract potential URLs from the Edge browser directory (also where is it?)

Step 1: Figure out where to look

Looks like somewhere in C:\Users\<name>\AppData\Local\Packages\Microsoft.MicrosoftEdge_*/**



Step 2: Recover URLs

We don't exactly understand how Edge stores data but we know roughly what a URL is supposed to look like!

Yara is our sledgehammer !

```
rule URL {
  strings: $a = /https?:\\/\\/[a-z0-9\\/+&#:\\?.-]+/i
  condition: any of them
}
```


Step 3: Let's do this!

```
C:\Program Files\velociraptor>Velociraptor.exe query -v "SELECT * FROM foreach(row={ SELECT FullPath from glob(globs='c:\\Users\\*\\AppData\\Local\\Packages\\Microsoft.MicrosoftEdge_*/**')}, query={ SELECT str(str=String.Data) As Hit, String.Offset AS Offset, FileName FROM yara(files=FullPath, rules='rule X { strings: $a = /https?:\\/\\/[a-z0-9\\/+&#:\\?.-]+/i  condition: any of them }')})"
[INFO] 2020-01-25T19:12:18-08:00 Loaded 147 built in artifacts
[
 {
  "Hit": "https://img-s-msn-com.akamaized.net/tenant/amp/entityid/AAdTRyf.img",
  "Offset": 87,
  "FileName": "\\C:\\Users\\test\\AppData\\Local\\Packages\\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\\LocalState\\Favicons\\TopSites\\25248558600\\msapplication.xml"
 },
 {
  "Hit": "https://img-s-msn-com.akamaized.net/tenant/amp/entityid/AAc9vHK.img",
  "Offset": 87,
  "FileName": "\\C:\\Users\\test\\AppData\\Local\\Packages\\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\\LocalState\\Favicons\\TopSites\\6837022160\\msapplication.xml"
 },
 {
  "Hit": "https://img-s-msn-com.akamaized.net/tenant/amp/entityid/AAJX5zh.img",
  "Offset": 87,
  "FileName": "\\C:\\Users\\test\\AppData\\Local\\Packages\\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\\LocalState\\Favicons\\TopSites\\7180116100\\msapplication.xml"
 },
```


C:\Program Files\velociraptor\artifacts\edge.yaml - Notepad++ [Administrator]

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

netstat.yaml | edge.yaml

```
1  name: Windows.Application.EdgeUrls
2  description: |
3      Scan Edge's cache directories to recover anything
4      that looks like a URL.
5
6  parameters:
7  - name: EdgeGlob
8    default: C:\Users\*\AppData\Local\Packages\Microsoft.MicrosoftEdge_*/**
9  - name: URLYaraRule
10   default: |
11     rule URL {
12        strings: $a = /https?:\/\/[a-z0-9\/+&#:\?.-]+/i
13        condition: any of them
14     }
15
16 sources:
17 - queries:
18   -  SELECT * FROM foreach(
19        row={
20          SELECT FullPath from glob(globs=EdgeGlob)
21        }, query={
22          SELECT str(str=String.Data) As Hit,
23                 String.Offset AS Offset, FileName
24          FROM yara(files=FullPath, rules=URLYaraRule)
25        })
```

Administrator: Command Prompt

```
C:\Program Files\velociraptor>Velociraptor.exe --definitions artifacts artifacts collect
Windows.Application.EdgeUrls
[][
 {
  "Hit": "https://img-s-msn-com.akamaized.net/tenant/amp/entityid/AAzObNi.img",
  "Offset": 87,
  "FileName": "\\C:\\Users\\test\\AppData\\Local\\Packages\\Microsoft.MicrosoftEdge_8wek
b3d8bbwe\\LocalState\\Favicons\\TopSites\\13783203330\\msapplication.xml"
 },
 {
  "Hit": "https://img-s-msn-com.akamaized.net/tenant/amp/entityid/BBX4bfQ.img",
  "Offset": 87,
  "FileName": "\\C:\\Users\\test\\AppData\\Local\\Packages\\Microsoft.MicrosoftEdge_8wek
b3d8bbwe\\LocalState\\Favicons\\TopSites\\15624613980\\msapplication.xml"
 },
```

YARA best practice

You can get yara rules from many sources (threat intel, blog posts etc)

YARA is really a first level triage tool:

- ❏ Depending on signature many false positives expected
- ❏ Some signatures are extremely specific so make a great signal

Try to collect additional context around the hits to eliminate false positives.

Yara scanning is relatively expensive! consider more targeted glob expressions and client side throttling since usually YARA scanning is not time critical.

NTFS Analysis

NTFS overview

NTFS is the standard Windows filesystem.

- All files are represented in a Master File Table
- Files can contain multiple attributes:
  - Filename (Long name/Short name)
  - Data attribute – contains file data
  - I30 attribute (contains directory listing)
- Data attributes may be compressed or sparse
- Filename attributes contain their own timestamps

The Master File Table

The NTFS file system contains a file called the *master file table*, or MFT. There is at least one entry in the MFT for every file on an NTFS file system volume, including the MFT itself. All information about a file, including its size, time and date stamps, permissions, and data content, is stored either in MFT entries, or in space outside the MFT that is described by MFT entries.

https://docs.microsoft.com/en-us/windows/win32/fileio/master-file-table

NTFS Concepts

https://www.fireeye.com/blog/threat-research/2012/10/incident-response-ntfs-indx-buffers-part-4-br-internal.html

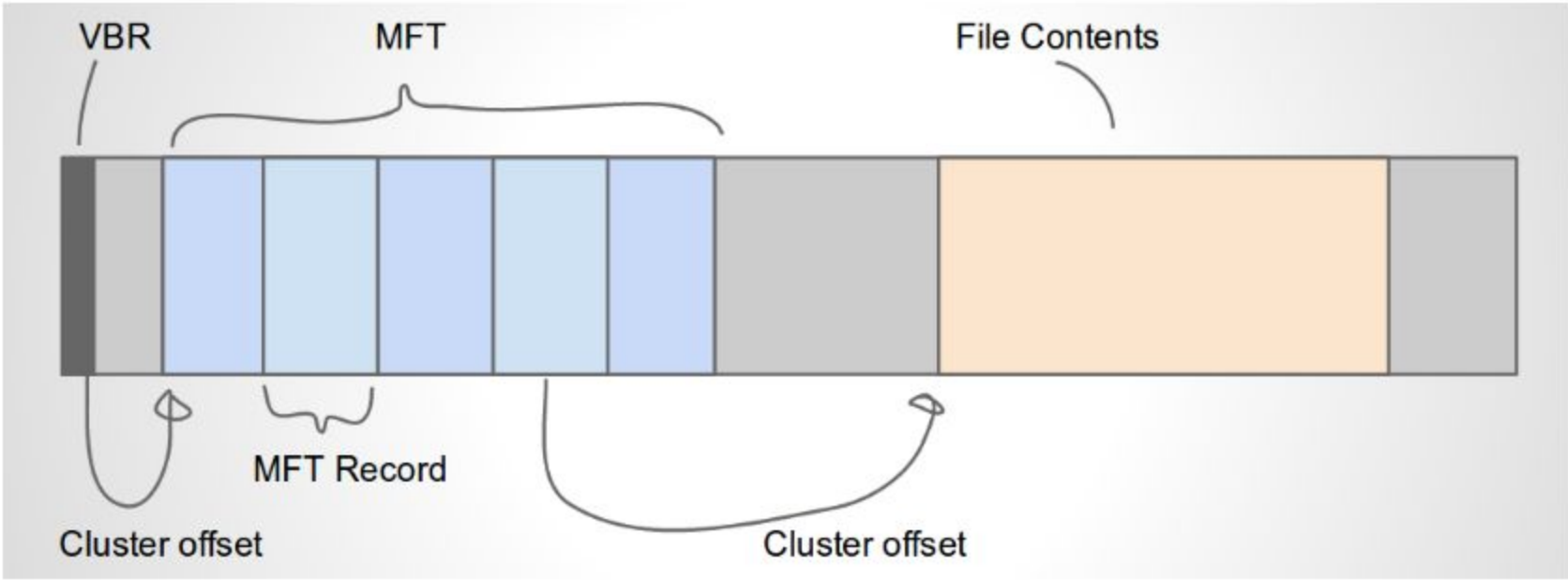


Figure 1: NTFS Volume Layout Showing the $MFT

Velociraptor's NTFS support

Velociraptor has 2 accessors providing access to NTFS

- ntfs - Supports Alternate Data Streams in directory listings.
- lazy_ntfs - much faster but does not detect ADS.

Due to these accessors it is possible to operate on files in the NTFS volume using all the usual plugins.

```
C:\Program Files\velociraptor>Velociraptor.exe -v query "SELECT * FROM glob(globs='C:\\*', accessor='ntfs')"
[INFO] 2020-01-27T02:31:38-08:00 Loaded 147 built in artifacts
[
 {
  "Atime": {
   "sec": 1578682526,
   "usec": 0
  },
  "Ctime": {
   "sec": 1578682526,
   "usec": 0
  },
  "Data": {
   "mft": "4-128-1",
   "name_type": "DOS+Win32"
  },
  "FullPath": "\\\\.\\C:\\$AttrDef",
  "GetLink": "",
  "IsDir": false,
  "IsLink": false,
  "ModTime": "2020-01-10T10:55:26.0379673-08:00",
  "Mode": 493,
  "Mtime": {
   "sec": 1578682526,
   "usec": 0
  },
  "Name": "$AttrDef",
  "Size": 2560,
  "Sys": {
   "mft": "4-128-1",
   "name_type": "DOS+Win32"
  }
 },
```

The NTFS accessor makes NTFS specific information available in the Data field. For regular files it includes the inode string.

The NTFS accessor considers all paths to begin with a device name. For convenience the accessor also accepts a drive letter.

Volume Shadow Copies

NTFS allows for a special copy on write snapshot feature called “Volume Shadow Copy”.

Create a VSS copy on your own machine using WMI:

```
C:\Program Files\velociraptor>wmic shadowcopy call create Volume='c:\'
Executing (Win32_ShadowCopy)->create()
Method execution successful.
Out Parameters:
instance of __PARAMETERS
{
        ReturnValue = 0;
        ShadowID = "{8E45ADC3-EA0F-4316-9C48-02A25B21CA6D}";
};
```

On Windows server OS you can use:

```
vssadmin create shadow
```

NTFS accessor and VSS

When a VSS copy is created, it is accessible via a special device. Velociraptor allows the VSS copies to be enumerated by listing them at the top level of the filesystem.

At the top level, the accessor provides metadata about each device in the “Data” column, including its creation time.

```
C:\Program Files\velociraptor>Velociraptor.exe -v query "SELECT Name, Data FROM glob(globs='/*', accessor='ntfs')"
[INFO] 2020-01-27T02:49:39-08:00 Loaded 147 built in artifacts
[
 {
  "Name": "\\\\.\\C:",
  "Data": {
   "Description": "Local Fixed Disk",
   "DeviceID": "C:",
   "FreeSpace": "39434489856",
   "Size": "63747125248",
   "SystemName": "DESKTOP-3JH0IO0",
   "VolumeName": "",
   "VolumeSerialNumber": "A2856979"
  }
 },
 {
  "Name": "\\\\?\\GLOBALROOT\\Device\\HarddiskVolumeShadowCopy1",
  "Data": {
   "DeviceObject": "\\\\?\\GLOBALROOT\\Device\\HarddiskVolumeShadowCopy1",
   "ID": "{8E45ADC3-EA0F-4316-9C48-02A25B21CA6D}",
   "InstallDate": "20200127023849.152615-480",
   "OriginatingMachine": "DESKTOP-3JH0IO0",
   "VolumeName": "\\\\?\\Volume{15d7d938-95f9-4695-841c-52d0f143e8d3}\\"
  }
 }
]
```


Operating on VSS

Simply use the VSS device name as a prefix to all paths and the ntfs accessor will parse it instead.

You can use it to analyze older versions of the drive!

```
C:\Program Files\velociraptor>Velociraptor.exe -v query "SELECT Name FROM glob(globs='\\\\?\\GLOBALROOT\\Device\\HarddiskVolumeShadowCopy1' + '/
$*', accessor='ntfs')"
[INFO] 2020-01-27T03:01:18-08:00 Loaded 147 built in artifacts
[
 {
  "Name": "$AttrDef"
 },
 {
  "Name": "$BadClus"
 },
 {
  "Name": "$BadClus:$Bad"
 },
 {
  "Name": "$Bitmap"
 },
```

parse_mft

Scan the $MFT from an NTFS volume.

Plugin

Arg	Description	Type
filename	A list of event log files to parse.	string (required)
accessor	The accessor to use.	string

parse_ntfs

Parse an NTFS image file.

Function

Arg	Description	Type
device	The device file to open. This may be a full path - we will figure out the device automatically.	string (required)
inode	The MFT entry to parse in inode notation (5-144-1).	string
mft	The MFT entry to parse.	int64
mft_offset	The offset to the MFT entry to parse.	int64

parse_ntfs_i30

Scan the $I30 stream from an NTFS MFT entry.

Plugin

Arg	Description	Type
device	The device file to open. This may be a full path - we will figure out the device automatically.	string (required)

Parsing the MFT

You can download the entire $MFT file from the endpoint using the ntfs accessor, then process it offline.

You can also parse the $MFT on the endpoint using Velociraptor.

This is most useful when you need to pass over all the files in the disk - it is more efficient than a recursive glob and might recover deleted files.

```
C:\Program Files\velociraptor>Velociraptor.exe -v query "SELECT * FROM parse_mft(filename='C:/$MFT', accessor='ntfs') WHERE FileName =~ '.exe$'
LIMIT 5"
[INFO] 2020-01-25T19:45:23-08:00 Loaded 147 built in artifacts

{
 "EntryNumber": 197,
 "InUse": true,
 "ParentEntryNumber": 80815,
 "FullPath": "Users/test/AppData/Local/Microsoft/WindowsApps/GameBarElevatedFT_Alias.exe",
 "FileName": "GameBarElevatedFT_Alias.exe",
 "FileSize": 0,
 "ReferenceCount": 2,
 "IsDir": false,
 "Created0x10": "2020-01-11T01:48:51.5704654-08:00",
 "Created0x30": "2020-01-11T01:48:51.5704654-08:00",
 "LastModified0x10": "2020-01-11T01:48:51.5704654-08:00",
 "LastModified0x30": "2020-01-11T01:48:51.5704654-08:00",
 "LastRecordChange0x10": "2020-01-11T01:48:51.5835566-08:00",
 "LastRecordChange0x30": "2020-01-11T01:48:51.5704654-08:00",
 "LastAccess0x10": "2020-01-11T01:48:51.5704654-08:00",
 "LastAccess0x30": "2020-01-11T01:48:51.5704654-08:00"
},
{
 "EntryNumber": 199,
 "InUse": true,
 "ParentEntryNumber": 198,
 "FullPath": "Users/test/AppData/Local/Microsoft/WindowsApps/Microsoft.XboxGamingOverlay_8wekyb3d8bbwe/GameBarElevatedFT_Alias.exe",
 "FileName": "GameBarElevatedFT_Alias.exe",
 "FileSize": 0,
 "ReferenceCount": 2,
 "IsDir": false,
 "Created0x10": "2020-01-11T01:48:51.5844941-08:00",
 "Created0x30": "2020-01-11T01:48:51.5844941-08:00",
 "LastModified0x10": "2020-01-11T01:48:51.5844941-08:00",
 "LastModified0x30": "2020-01-11T01:48:51.5844941-08:00",
 "LastRecordChange0x10": "2020-01-11T01:48:51.5854755-08:00",
 "LastRecordChange0x30": "2020-01-11T01:48:51.5844941-08:00",
 "LastAccess0x10": "2020-01-11T01:48:51.5844941-08:00",
 "LastAccess0x30": "2020-01-11T01:48:51.5844941-08:00"
},
```

Exercise: Find all .exe on the drive

Efficiently find all .exe on disk that were created after Jan 20, 2020

```
C:\Program Files\velociraptor>f:\Velociraptor.exe query -v "SELECT EntryNumber, FullPath, InUse, FileSize, Created0x10 FROM parse_mft
(filename='C:\\$MFT', accessor='ntfs') WHERE FullPath =~ '.exe$' AND Created0x10 > '2020-01-20' "
[INFO] 2020-01-28T05:54:32-08:00 Loaded 147 built in artifacts
[
 {
  "EntryNumber": 197,
  "FullPath": "Users/test/AppData/Local/Microsoft/WindowsApps/GameBarElevatedFT_Alias.exe",
  "InUse": true,
  "FileSize": 0,
  "Created0x10": "2020-01-11T01:48:51.5704654-08:00"
 },
 {
  "EntryNumber": 21312,
  "FullPath": "Users/test/AppData/Local/Microsoft/OneDrive/19.222.1110.0006/FileCoAuth.exe",
  "InUse": true,
  "FileSize": 506216,
  "Created0x10": "2020-01-20T17:04:18.0133036-08:00"
 },
```

MFT Entries

An MFT Entry can have multiple attributes and streams
The previous plugin just shows high level information about each MFT entry - we can dig deeper with the parse_ntfs() plugin which accepts an mft ID.

Choose a file of interest in the previous output and inspect it deeper.

```
:\Program Files\velociraptor>Velociraptor.exe -v query "SELECT parse_ntfs(device='c:/', mft=974) from scope()"
INFO] 2020-01-25T20:06:09-08:00 Loaded 147 built in artifacts

{
 "parse_ntfs(device='c:/', mft=974)": {
  "FullPath": "Program Files/WindowsApps/Microsoft.Xbox.TCUI_1.24.10001.0_x64__8wekyb3d8bbwe/TCUI-App.exe",
  "MFTID": 974,
  "Size": 18432,
  "Allocated": true,
  "IsDir": false,
  "SI_Times": {
   "CreateTime": "2020-01-11T01:49:05.8473789-08:00",
   "FileModifiedTime": "2020-01-11T01:49:08.100144-08:00",
   "MFTModifiedTime": "2020-01-11T01:49:08.1021724-08:00",
   "AccessedTime": "2020-01-11T01:49:08.100144-08:00"
  },
  "Filenames": [
   {
    "Times": {
     "CreateTime": "2020-01-11T01:49:05.8473789-08:00",
     "FileModifiedTime": "2020-01-11T01:49:05.8473789-08:00",
     "MFTModifiedTime": "2020-01-11T01:49:05.8473789-08:00",
     "AccessedTime": "2020-01-11T01:49:05.8473789-08:00"
    },
    "Type": "DOS+Win32",
    "Name": "TCUI-App.exe"
   }
  ],
  "Attributes": [
   {
    "Type": "$STANDARD_INFORMATION",
    "TypeId": 16,
    "Id": 0,
    "Inode": "974-16-0",
    "Size": 72,
    "Name": ""
   },
```

An inode is a triple of mft id, type id and id

e.g. 974-16-0

representing a stream of data

NTFS timestamps

An MFT entry can have up to 16 timestamps!

Timestamps are critical to forensic investigations

- ❑ Determine when files were copied
- ❑ When files were modified
- ❑ And sometimes we can determine when a file was accessed

In NTFS there are timestamps

1. In $STANDARD_INFORMATION stream (usually only 1)
2. In the $FILENAME stream (sometimes 2 or 3)
3. In the $I30 stream of the parent directory (see later)

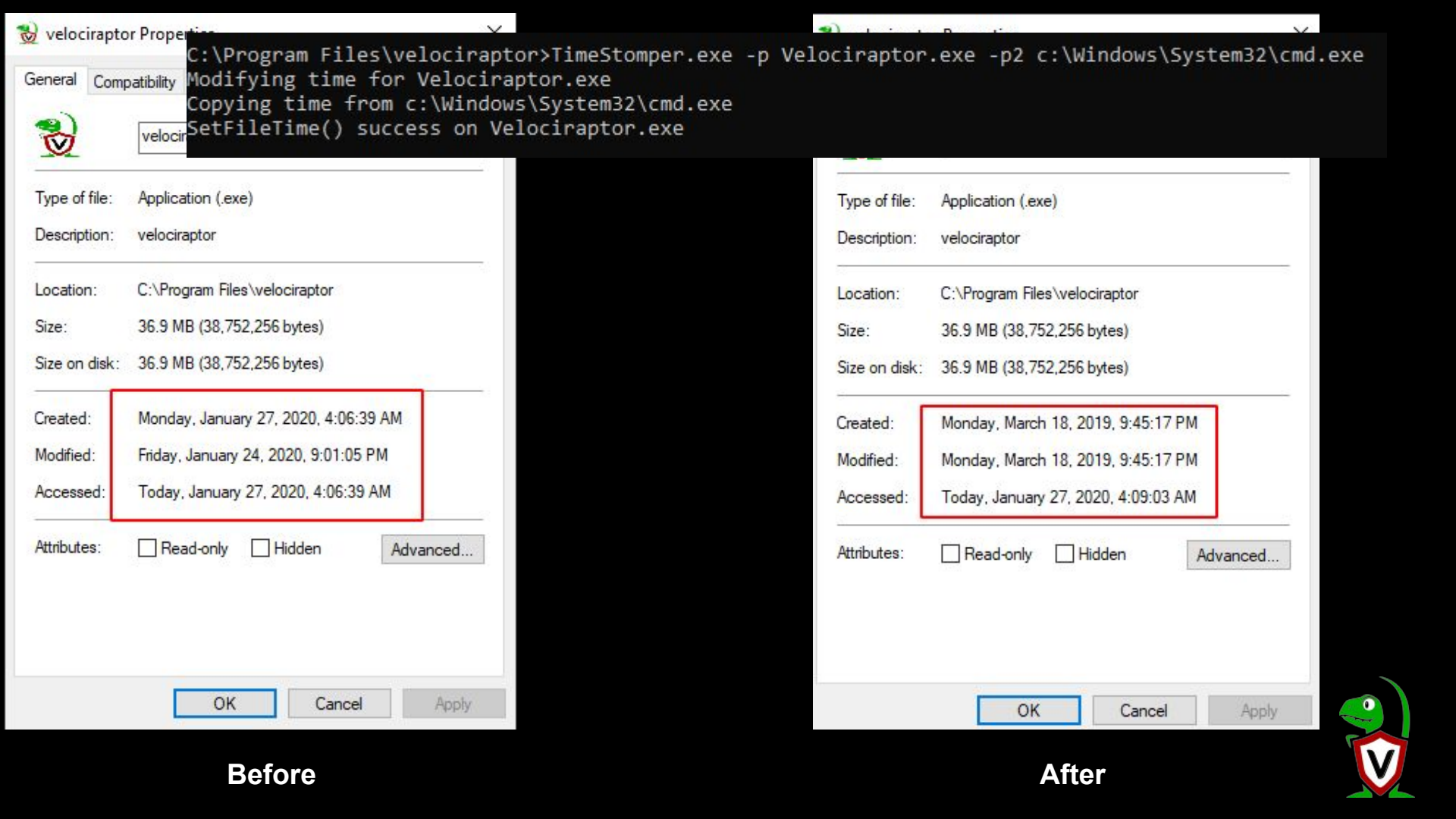
Timestomping

Attackers sometimes change the timestamps of files to make them less obvious. E.g make malware look like it was installed many years ago.

For the next exercise we will stomp over some times. Use the provided TimeStomper tool to stomp over Velociraptor.exe's timestamps.

https://github.com/slyd0g/TimeStomper

https://posts.specterops.io/revisiting-ttps-timestomper-622d4c28a655


C:\Program Files\velociraptor>TimeStomper.exe -p Velociraptor.exe -p2 c:\Windows\System32\cmd.exe
Modifying time for Velociraptor.exe
Copying time from c:\Windows\System32\cmd.exe
SetFileTime() success on Velociraptor.exe
General
Compatibility
Type of file: Application (.exe)
Description: velociraptor
Location: C:\Program Files\velociraptor
Size: 36.9 MB (38,752,256 bytes)
Size on disk: 36.9 MB (38,752,256 bytes)
Created: Monday, January 27, 2020, 4:06:39 AM
Modified: Friday, January 24, 2020, 9:01:05 PM
Accessed: Today, January 27, 2020, 4:06:39 AM
Attributes: Read-only Hidden Advanced...
OK
Cancel
Apply
Type of file: Application (.exe)
Description: velociraptor
Location: C:\Program Files\velociraptor
Size: 36.9 MB (38,752,256 bytes)
Size on disk: 36.9 MB (38,752,256 bytes)
Created: Monday, March 18, 2019, 9:45:17 PM
Modified: Monday, March 18, 2019, 9:45:17 PM
Accessed: Today, January 27, 2020, 4:09:03 AM
Attributes: Read-only Hidden Advanced...
OK
Cancel
Apply
Before
After

```
C:\Program Files\velociraptor>Velociraptor.exe query -v "SELECT parse_ntfs(device=FullPath, inode=Data.mft ) from glob(globs='c:\\pro
gram files\\velociraptor\\velociraptor.exe', accessor='ntfs')"
[INFO] 2020-01-27T04:17:31-08:00 Loaded 147 built in artifacts
[
 {
  "parse_ntfs(device=FullPath, inode=Data.mft)": {
   "FullPath": "Program Files/velociraptor/velociraptor.exe",
   "MFTID": 240524,
   "Size": 38752256,
   "Allocated": true,
   "IsDir": false,
   "SI_Times": {
    "CreateTime": "2019-03-18T21:45:17.086-07:00",
    "FileModifiedTime": "2019-03-18T21:45:17.086-07:00",
    "MFTModifiedTime": "2020-01-27T04:08:18.8459313-08:00",
    "AccessedTime": "2020-01-27T04:17:30.8560161-08:00"
   },
   "Filenames": [
    {
     "Times": {
      "CreateTime": "2020-01-27T04:06:39.4850472-08:00",
      "FileModifiedTime": "2020-01-27T04:06:39.4850472-08:00",
      "MFTModifiedTime": "2020-01-27T04:06:39.4850472-08:00",
      "AccessedTime": "2020-01-27T04:06:39.4850472-08:00"
     },
     "Type": "DOS",
     "Name": "VELOCI~1.EXE"
    },
    {
     "Times": {
      "CreateTime": "2020-01-27T04:06:39.4850472-08:00",
      "FileModifiedTime": "2020-01-27T04:06:39.4850472-08:00",
      "MFTModifiedTime": "2020-01-27T04:06:39.4850472-08:00",
      "AccessedTime": "2020-01-27T04:06:39.4850472-08:00"
     },
     "Type": "Win32",
     "Name": "velociraptor.exe"
    }
   ],
   "Attributes": [
```


Exercise: Detect timestomping

Write an artifact that detects when a file has had its time stomped.

Note: This is not necessarily a smoking gun - many installers will update a file's timestamps during installation.

http://www.forensickb.com/2009/02/detecting-timestamp-changing-utlities.html

```
C:\Program Files\velociraptor>f:\Velociraptor.exe query -v "SELECT FullPath, Created0x10, Created0x30 FROM parse_mft(filename='C:\\$MFT', accessor='ntfs') WHERE Created0x10 < Created0x30 AND FullPath =~ '\\.exe$' AND Created0x30 > '2020-01-20' "
[INFO] 2020-01-27T05:28:40-08:00 Loaded 147 built in artifacts
[
 {
  "FullPath": "Program Files/Common Files/microsoft shared/ink/InputPersonalization.exe",
  "Created0x10": "2019-03-18T21:45:49.9651562-07:00",
  "Created0x30": "2020-01-10T10:55:53.7250247-08:00"
 }
][
 {
  "FullPath": "Program Files/Notepad++/updater/GUP.exe",
  "Created0x10": "2020-01-12T13:42:30-08:00",
  "Created0x30": "2020-01-20T18:34:18.388148-08:00"
 },
 {
  "FullPath": "Program Files/Notepad++/notepad++.exe",
  "Created0x10": "2020-01-12T13:41:40-08:00",
  "Created0x30": "2020-01-20T18:34:18.6069468-08:00"
 }
][
 {
  "FullPath": "Program Files/velociraptor/velociraptor.exe",
  "Created0x10": "2019-03-18T21:45:17.086-07:00",
  "Created0x30": "2020-01-27T04:06:39.4850472-08:00"
 }
]
```

Many binaries are timestomped naturally because they come from CAB or MSI files.
To eliminate noise you can narrow the created time from the $FILE_NAME attribute

The $I30 INDX stream

In NTFS a directory is simply an MFT entry with $I30 streams. The streams contains a B+ tree of the MFT entries in the directory.

Since INDX streams are a B+ tree when a record is deleted, the tree will be reordered. Sometimes this leaves old entries in the slack space.

```
C:\Program Files\velociraptor>f:\Velociraptor.exe query -v "SELECT parse_ntfs(device=FullPath, inode=Data.mft) FROM glob(globs='C:\\Program Files\\Velociraptor', accessor='ntfs')"
[INFO] 2020-01-27T05:39:20-08:00 Loaded 147 built in artifacts
[
 {
  "parse_ntfs(device=FullPath, inode=Data.mft)": {
   "FullPath": "Program Files/velociraptor",
   "MFTID": 1075,
   "Size": 0,
   "Allocated": true,
   "IsDir": true,
```

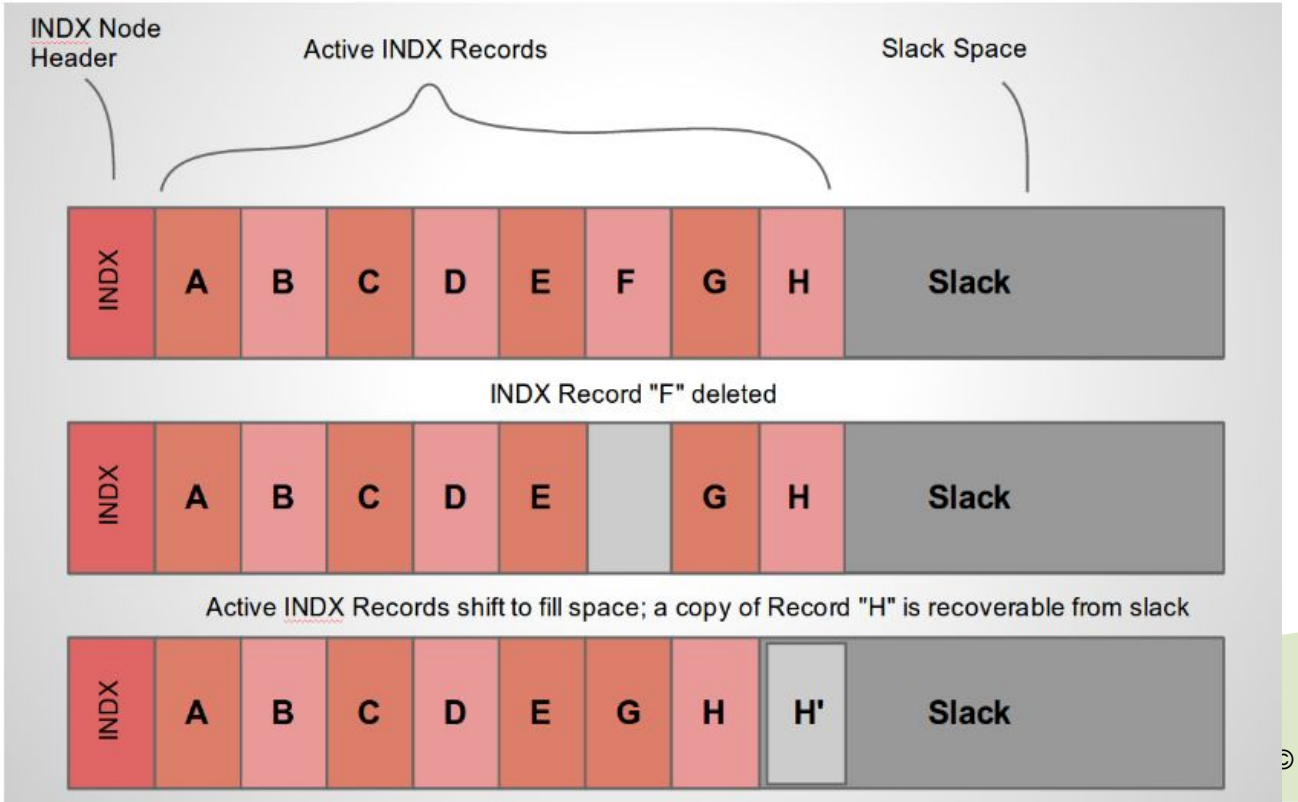
```
"Attributes": [
 {
  "Type": "$STANDARD_INFORMATION",
  "TypeId": 16,
  "Id": 0,
```

```
 {
  "Type": "$INDEX_ROOT",
  "TypeId": 144,
  "Id": 7,
  "Inode": "1075-144-7",
  "Size": 56,
  "Name": "$I30"
 },
 {
  "Type": "$INDEX_ALLOCATION",
  "TypeId": 160,
  "Id": 5,
  "Inode": "1075-160-5",
  "Size": 4096,
  "Name": "$I30"
 },
```

INDX stream is allocated in 4096 bytes. Contains information about the directory contents.

Carving INDX headers

https://www.fireeye.com/blog/threat-research/2012/10/incident-response-ntfs-indx-buffers-part-4-br-internal.html



Exercise: Experiment with $I30 carving

Add and remove files from a directory and observe which files can be carved from the $I30 stream.

See previous slide to verify the process.

```
C:\Program Files\velociraptor>del Z1.txt

C:\Program Files\velociraptor>f:\Velociraptor.exe query -v "SELECT Name, IsSlack, SlackOffset FROM parse_ntfs_i30(device='C:\\', inod
e='1075') WHERE Name =~ 'txt'  "
[INFO] 2020-01-27T17:06:14-08:00 Loaded 147 built in artifacts
[
 {
  "Name": "Z1.txt",
  "IsSlack": true,
  "SlackOffset": 976
 },
 {
  "Name": "Z1.txt",
  "IsSlack": true,
  "SlackOffset": 1072
 }
]
```

Exercise: Write an artifact

Sometimes we need to prove that a file used to exist in a directory - just the presence of the name and timestamps is significant!

Example:

- FIN8 deletes prefetch files https://attack.mitre.org/techniques/T1107/

Write an artifact that recovers the filenames of deleted files in directories.

Exercise: Write an artifact

```
SELECT * FROM foreach(
   row={
     SELECT FullPath, Data.mft AS MFT
     FROM glob(globs=DirectoryGlobs, accessor="ntfs")
     WHERE IsDir
   },
   query={
     SELECT FullPath, Name, NameType, Size, AllocatedSize,
            IsSlack, SlackOffset, Mtime, Atime, Ctime, Btime, MFTId
     FROM parse_ntfs_i30(device=FullPath, inode=MFT)
})
```


End of sample module

Register for the full course on our training page

https://www.velocidex.com/training/