Reading OneDrive Logs



swiftforensics.com/2022/02/reading-onedrive-logs.html



Due to the popularity of OneDrive, it has become an important source of evidence in forensics. Last week, Brian Maloney posted about his research on reconstructing the folder tree from the usercid.dat files, and also provided a script to do so. In this brief post, we explore the format of OneDrive Logs, and provide a tool to parse them. In subsequent posts, I will showcase use case scenarios.

Where to find them?

OneDrive logs have the extension .odl and are found in the user's profile folder at the following locations:

On Windows -

C:\Users\<USER>\AppData\Local\Microsoft\OneDrive\logs\

On macOS -

/Users/<USER>/Library/Logs/OneDrive/

At these locations, there are usually 3 folders - Common, Business1 and Personal, each containing logs. As the name suggests *Business1* is the *OneDrive for Business* version.

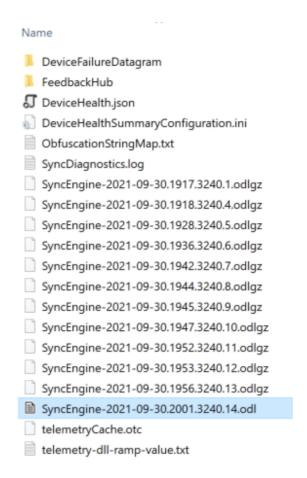


Figure 1 - Contents of Business1 folder

The .odl file is the currently active log, while the .odlgz files are older logs that have been compressed. Depending on which folder (and OS) you are in, you may also see .odlsent and .aodl files, which have a similar format.

What is in there?

These are binary files, and cannot be directly viewed in a text editor. Here is what a .odl file looks like in a hex editor.

	۸	4	2	2		E	-	7	0	0	Α.	В	,	Α.	е	-	01224E 6700A DFDEE
0000h:	45	42	46	47	4F	4E	45	44	8 02	9	A 00	B 00	45	00	00	00	0123456789ABCDEF EBFGONEDE
0010h:	00	00	00	00		00	00	00	01	00	00	00	32	31	2E	31	
0020h:	30	39	2E	30	35	33	30	2E	30	30	30	31	00	00	00	00	09.0530.0001
0030h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0040h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0050h:	00	00	00	00	00	00	00	00	00	00	00	00	31	31	2E	34	11.4
0060h:	2E	30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.0
0070h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0080h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0090h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00A0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00B0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00C0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00D0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00E0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00F0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0100h:	cc	DD	EE	FF	00	00	00	00	84	B5	E4	36	7A	01	00	00	ÌÝîÿ"µä6z
0110h:	66	57	00	00	03	01	00	00	BE	03	3B	7F	SF	8B	09	44	fW%.;<.D
0120h:	9F	ä	65	06	67	A7	96	45	15	5F	8D	9D	01	00	00	00	ŸÍe.g§-E
0130h:	91	00	00	00	00	00	00	00	23	00	00	00	54	65	6C		#Tele
0140h:	6D	65	74	72	79	50	72	6F	78	79	43	6F	6E	66	69	67	metryProxyConfig
0150h:	75	72	61	74	69	6F	6Ē	46	69	6C	65	2E	63	70		2C	urationFile.cpp.
0160h:	00	00	00	0A	00	00	00	_	6E	69	74	69	61	6C	69	7A	Initializ
0170h:	65	52	00	00	00	2F	4C	65	66	74	5A	6F	6F	57	72	79	eR/LeftZooWry
0180h:	2F	4C	6F	67	4F	6E	65	4D	75	67	2F	48	61	6D	55	67	/LogOneMug/HamUg
0190h:	68	56	69	6E	65	2F	4D	75	63	68	44	6F	77	6E	52	69	hVine/MuchDownRi
01A0h:	63	68	2F	51	75	69	6C	6C	52	6F	64	45	67	67	2F	4B	ch/QuillRodEgg/K
01B0h:	6F	69	57	6F	6C	66	54	61	64	2F	4C	61	77	46	6C	79	oiWolfTad/LawFly
01C0h:	4F	77	6C	2E	74	78	74	01	00	CC	DD	EE	FF	00	00	00	Owl.txtlyiv
01D0h:	00	85	B5	E4	36	7A	01	00	00	66	57	00	00	03	01	00	µā6zfW
01E0b+	00	DE	02	20	75		OD.	00	44	ΔE	CD		06	67	47	0.6	v . núfa as

Figure 2 - .odl file in a hex editor

It is a typical binary file format with a 256 byte header and data blocks that follow. Upon first inspection, it seems to be a log of all important function calls made by the program. Having this low level run log can be useful in certain scenarios where you don't have other logging and need to prove upload/download or synchronisation of files/folders or even a discovery of items which no longer exist on disk/cloud.

You do notice some funny looking strings (highlighted in red). More on that later...

The Format

With a bit of reverse engineering, the header format is worked out as follows:

```
struct {
  char signature[8]; // EBFGONED
  uint32 unk_version; // value seen = 2
  uint32 unknown2;
  uint64 unknown3; // value seen = 0
  uint32 unknown4; // value seen = 1
  char one_drive_version[0x40];
  char os_version[0x40];
  byte reserved[0x64];
} Odl_header;
```

The structures for the data blocks are as follows:

```
struct {
  uint64
           signature; // CCDDEEFF 0000000
  uint64
           timestamp; // Unix Millisecond time
  uint32
           unk1;
  uint32
           unk2;
  byte
           unk3 guid[16];
  uint32
           unk4;
           unk5; // mostly 1
  uint32
  uint32
           data len;
  uint32
           unk6; // mostly 0
  byte
          data[data len];
} Data block;
struct {
  uint32
          code file name len;
          code file name[code file name len];
  char
  uint32 unknown;
  uint32 code function name len;
  char
          code_function_name[code_function_name_len];
  byte
          parameters[];
} Data;
```

In case of .odlgz files, the Odl_header is the same, followed by a single gzip compressed blob. The blob can be uncompressed to parse the Data_block structures.

Now, we can try to interpret the data. Leaving aside the few unknowns, the data block mainly consists of a timestamp (when event occurred), the function name that was called, the code file that function resides in and the parameters passed to the function. The parameters can be of various types like int, char, float, etc.. and that part hasn't been fully reverse engineered yet, but simply extracting the strings gives us a lot of good information. However the strings are obfuscated!

Un-obfuscating the strings

Since Microsoft uploads these logs to their servers for telemetry and debugging, they obfuscate anything that is part of a file/folder name or url string or username. However file extensions are not obfuscated. The ways this works is that the data identified to obfuscate is replaced by a word which is stored in a dictionary. The dictionary is available as the file *ObfuscationStringMap.txt*, usually in the same folder as the .odl files. To un-obfuscate, one simple needs to find and replace the strings with their original versions.

```
ZigKidDoge
            charley.mifsud@hotmail.com
GadEggZoo
            ftp-20Dec2021
HerBigYup
           Microsoft-Windows-ServerManager
YewZagBat
           SY
GoodDayYup a2eddb08fab0fd15_0
           29103684db90b227_1
Sea0rbLeft
IcyWowGood 1d7e1ff1-a42e-44bf-aada-25e8cd066bef
FarWagNorth 77f30ba3224e16c9_1
BigMawLaw 130805663c7899bb_0
JokeEmuUgh Activities for XA30015400-Paul Lim - 10_1_2021 - 11_11_2021
ZooBagYew def4488c59bda97f_1
BatOafRod 8321e0fd-b276-4c17-9f6c-3f8508b52301
Log0neMug
           ykhatri
IcyWagOwl mi_document
           {9BE9E7F0-F9F1-487B-A2FC-790CD2898388}v3.9.6150
AmGemBlue
           _countHolders
LawFlyPie
TruckUrnAwe config
            93761d7ae5a983e9_0
BugEmuFig
EggUpMuch
            f_000407
FarRedCake login-16
```

Figure 3 - Snipped of ObfuscationStringMap.txt

This is a tab separated file, stored as either UTF-8 or UTF-16LE depending on whether you are running macOS or Windows.

Now, referring back to the original funny looking string in Figure 2 - /LeftZooWry/LogOneMug/HamUghVine/MuchDownRich/QuillRodEgg/KoiWolfTad/LawFlyOwl.txt .. after un-obfuscating becomes ..

/Users/ykhatri/Library/Logs/OneDrive/Business1/telemetry-dll-ramp-value.txt

It is important to note that since extensions are not obfuscated, they can still provide valuable intel even if some or all of the parts in the path cannot be decoded.

Now this process seems easy, however not all obfuscated strings are file/folder names or parts of paths/urls. Some are multi line strings. Another problem is that the words (or keys to the dictionary) are reused! So you might see the same key several times in the *ObfuscationStringMap*. The thing to remember is that new entries get added at the top in this file, not at the bottom, so when reading a file, the first occurrence of a key should be the latest one. Also sometimes, the key is not found, as it's cleaned out after a period of time. Also there is no way to tell if an entry in the dictionary is stale or valid for a specific log file being parsed. All of this just means that the decoded strings *need to be taken with a grain of salt*.

Based on the above, a python script to parse the ODL logs is available <u>here</u>. A snippet of the output produced by the script is shown below.

Timestamp	Code_File	Function	Params_Decoded
2022-02-13 03:46:01	WorkerThreadPool.cpp	AddThread	Diagnostics
2022-02-13 03:46:01	WorkerThreadPool.cpp	WorkerMainLoop	Diagnostics
2022-02-13 03:46:01	SyncHelpers.cpp	RecordStallErrorsSummary	[2e88b632-4fdd-486a-91a2-01be1bf02de1', 'cav;', 'a5d8a4ef80c94e17boe44c8d9a0b97d8+15', 'Stalla9', 'Stalla_TeamSite_SyncStall: No
2022-02-13 03:46:01	SyncProgressAudit.cpp	SendDiagnosticInfo	[2x88b632-4fdd-486a-91a2-01be1bf02de1H', 'None', 'Sosacb05-195a-4do5-a5cb-ecc8b7d45d99', 'b008e1a4-9c2c-97b9-ec9c-52859.
2022-02-13 03:46:01	Diagnostics.cpp	ReportDiagnosticInfoWorker	['sync_progress_id\$', '2e88b632-4fdd-486a-91s2-01be1b/02de1']
2022-02-13 03:46:01	ScenarioQosWrapper.cpp	RecordQosTelemetry	[StorageSyncVerificationScenario', 'Storage', 'StorageSyncVerificationScenario', 'Success\$', 'Tb4b4403-7ba1-4358-97a1-3d2bc91c16ac
2022-02-13 03:46:01	ScenarioTracking.cpp	-ScenarioBase	['DiagnosticsScenario\$', '8c79d19f-009d-444a-8224-c3a083649791']
2022-02-13 03:46:03	file8ystem.cpp	copyFile	[%ConfigPath%/3f838a67-f3fb-49a2-95ab-athtdc08f3820.in/, %ConfigPath%/logUploaderSettings.in/]
2022-02-13 03:46:12	Deviceld.mm	GetUserName	ykhatri
2022-02-13 03:46:12	Deviceld.mm	AppendHostNamelfNeeded	Update ring ramp
2022-02-13 03:46:12	Deviceld.mm	ReportHostNameChangesIfNeeded	Update ring ramp
2022-02-13 03:46:12	OneAuthUserAuthentication.	AuthenticateToService	DRX_E_AUTH_NO_VALID_CREDENTIALS
2022-02-13 03:46:12	BaseUserAuthenticationInter	GetTenantidForSpoResource	https://onedrive.live.com
2022-02-13 03:46:12	AuthPlatformHttpRequest.cp	Create	[OPTIONS', 'https://onedrive.live.com<']
2022-02-13 03:46:12	AuthPlatformHttpRequest.cp	OnRequestComplete	
2022-02-13 03:46:12	AuthPlatformHttpRequest.cp	Wait	
2022-02-13 03:46:12	BaseUserAuthenticationInten	GetTenantidForSpoResource	https://onedrive.live.com
2022-02-13 03:46:12	OneAuthWrapper.cpp	operator()	[OneAuth:Infe:9zkae:0e5f8dbc-4c72-46bb-a32a-8a0c61a55aca] Start task AcquireCredentialSilently [76]
2022-02-13 03:46:12	OneAuthWrapper.cpp	operator()	TID = 8552 ADAL 4.0.9 Mac 12.1.0 [2022-02-13 03:46:12 - 065F6DBC-4C72-468B-A32A-BADC61A55ACA] ##### BEGIN acquireToken S

Figure 4 - output snippet

In a subsequent post, we'll go through the items of interest in these logs like Account linking/unlinking, uploads, downloads, file info, etc..