Dustman APT: Art of Copy-Paste

Archived: 2026-04-05 17:46:09 UTC

Dustman is a piece of data wiping malware with origin believed to be from Iran or if you like - quote from zdnet.com "Iranian state-sponsored hackers".

There is a full technical overview of this malware -> https://www.scribd.com/document/442225568/Saudi-Arabia-CNA-report, I wouldn't waste time fully repeating it, as it gives a brief and enough description of this malware key parameters and capabilities.

Usually I pay zero attention to typical APT hysterics and low quality malware pushed by mass media/various ~~fake~~Av's as "incredible sophisticated" spyware/whatever. With exception if there is anything related to my work, for example copy/pasted from it. Just like in this case.

This is believed shared code with another data wiper called "ZeroCleare" - and IBM did analysis with 28 page PDF where they managed to copy-paste from my github repository without even giving a single credit or link to original. Well, ok, fuck you too IBM IRIS rippers 💩

Why this thing called Dustman? Well authors of this malware were lazy and left full pdb string inside main dropper **C:\Users\Admin\Desktop\Dustman\x64\Release\Dustman.pdb**. This doesn't look like fake and left because Visual Studio (and this one created in it) always sets debug information to Release builds by default (Project settings->Linker->Debugging). It is something from series of small tips just like if you are wondering why some of rootkits pdb paths always at Z: drive - easy to use hotkey while debugging on VMware.

Dustman main executable is a muldrop (SHA-1 e3ae32ebe8465c7df1225a51234f13e8a44969cc).

It contain three more files stored inside executable resource section. They are encrypted with simple xor.

```
for (ULONG i = 0; i < (ResourceSize / sizeof(ULONG_PTR)); i++)
    Buffer[i] ^= 0x7070707070707070;
```

Resource with id 1 (decrypted SHA-1 7c1b25518dee1e30b5a6eaa1ea8e4a3780c24d0c) is a VirtualBox driver. It is ripped by me from WinNT/Turla (another APT, this time "believed" to be from GRU GS AF RF, that one by the way also had some references/inspirations of my/our previous work). Dustman author(s) got it from my github repository called TDL - Turla Driver Loader (https://github.com/hfiref0x/TDL), well not only that driver, half of their work actually blatant copy-paste of this repository.

Resource with id 103 (decrypted SHA-1 a7133c316c534d1331c801bbcd3f4c62141013a1) is Eldos RawDisk modified driver (version 3.0.31.121). It is modified by Dustman authors by removing digital certificate from it. Currently I have no answer why they did this, except Eldos RawDisk certificate is widely blacklisted or detected by intrusion prevention systems/AV as possible sign of threat as it was used before multiple times in different malwares (https://attack.mitre.org/software/S0364/)

Resource with id 106 (decrypted SHA-1 20d61c337653392ea472352931820dc60c37b2bc) is malware agent application that is intended to work with Eldos RawDisk to perform data wipe. It contain pdb string **C:\Users\Admin\Desktop\Dustman\Furutaka\drv\agent.plain.pdb** which is giving you insides on VS solution structure. Furutaka is an internal name that I gave to TDL project executable.

Initial dropper is a modified version of original TDL (Furutaka) version 1.1.5, so it is relatively new, as this is final version in that repository before it was archived at April 2019. Just to show you how much Dustman authors copy-pasted, here is a screenshot of functions which I was able to identify in this malware (while rest of them are various trash from MS runtime).

Function name	Segment	Start	Length
FunctionNameByHash	.text	0000000140001000	00000099
FileNameByHash	.text	000000014000109C	000000B1
scmInstallDriver	.text	0000000140001150	00000060
scmOpenDevice	.text	00000001400011B0	00000092
scmRemoveDriver	.text	0000000140001244	00000041
scmStartDriver	.text	0000000140001288	0000005A
scmStopDriver	.text	00000001400012E4	0000007F
RunAgentProcess	.text	0000000140001364	00000108
TDLExploit	.text	000000014000146C	000003B4
TDLGetProcAddress	.text	0000000140001820	00000058
_main	**.text**	**0000000140001878**	**0000023A**
TDLMapDriver_Modified	.text	0000000140001AB4	0000031D
TDLResolveKernelImport	.text	0000000140001DD4	000000A7
TDLStartVulnerableDriver	.text	0000000140001E7C	0000015A
TDLStopVulnerableDriver_Modified	.text	0000000140001FD8	000000C8
TDLVBoxInstalled	.text	00000001400020A0	00000053
_strcat_w	.text	00000001400020F4	0000003C
_strcmpi_w	.text	0000000140002130	0000006D
_strlen_w	.text	00000001400021A0	0000001E
supDetectObjectCallback	.text	00000001400021C0	0000004E
supEnumSystemObjects	.text	0000000140002210	00000201
supGetNtOsBase	.text	0000000140002414	0000003E
supGetSystemInfo	.text	0000000140002454	000000B0
supIsObjectExists	.text	0000000140002504	00000044
supQueryResourceData_Modified	.text	0000000140002548	000000CD
supStopVBoxService	.text	0000000140002618	00000180
supWriteBufferToFile	.text	0000000140002798	000002AA
sub_140002A60	.text	0000000140002A60	00000010
RtlSecureZeroMemory	**.text**	**0000000140002A80**	**00000190**
sub_140002C20	.text	0000000140002C20	00000013
RtlCopyMemory	**.text**	**0000000140002C40**	**00000435**
sub_140003078	.text	0000000140003078	0000020B
sub_140003...A0	.text	0000000140003...A0	00000021

Pic 1. Dustman dropper functions.

It seems Dustman author(s) simple took TDL solution and then modified it by removing console/debug output in code and adapting it for their specific tasks - decrypt, drop resources to the disk, load RawDisk driver and start agent application at final stage. Lets take a look on modifications made by Dustman author(s).

At main (which is a heavily modified TDLMain from original TDL) right at the beginning Dustman attempts to block

multiple copies from installing VirtualBox/mapping Eldos driver by setting mutex with a very specific name "Down With Bin Salman". I do not want to dig into politics and other bullshit but I would like to suggest in case if this is false flag operation (surprise, but we will never know this) use something more creative - like for example "Coded by Soleimani" or "(c) 2019 IRGC", "covfefe" is fine too. If I would doing APT of such kind I would at first refrain from creating such wrong and stupid mutexes or build their unique names based on current environment without using any idiotic constants. Another fun message hidden inside agent executable (dropper resource 106 as mentioned above) "Down With Saudi Kingdom Down With Bin Salman" - very creative (not). Eldos license key is hardcoded in agent executable as "b4b615c28ccd059cf8ed1abf1c71fe03c0354522990af63adf3c911e2287a4b906d47d".

Back to initial dropper, supQueryResourceData (https://github.com/hfiref0x/TDL/blob/master/Source/Furutaka/sup.c#L99) is modified by adding xor decryption loop mentioned above. Below is screenshot of TDLStartVulnerableDriver routine slightly modified by removing console output, code responsible for backup and new file name for dropped file.

```
__int64 __fastcall TDLStartVulnerableDriver(LPWSTR lpBuffer)
{
  LPWSTR v1; // rbx
  _QWORD *v2; // rsi
  SC_HANDLE v4; // rdi
  int v5; // eax
  unsigned int v6; // [rsp+58h] [rbp+10h]
  __int64 v7; // [rsp+60h] [rbp+18h]

  v6 = 0;
  v1 = lpBuffer;
  v7 = -1i64;
  v2 = supQueryResourceData_Modified(1i64, g_hInstance, &v6);
  if ( !v2 )
    return -1i64;
  memset(v1, 0, 0x104ui64);
  if ( GetCurrentDirectoryW(0x104u, v1) )
  {
    v4 = OpenSCManagerW(0i64, 0i64, 0xF003Fu);
    if ( v4 )
    {
      if ( supIsObjectExists(L"\\Device", L"VBoxDrv") )
      {
        supStopVBoxService(v4, L"VBoxUSBMon");
        supStopVBoxService(v4, L"VBoxNetAdp");
        supStopVBoxService(v4, L"VBoxNetLwf");
        Sleep(0x3E8u);
        supStopVBoxService(v4, L"VBoxDrv");
      }
      strcat_w(v1, L"\\assistant.sys");
      v5 = supWriteBufferToFile(v1, v2, v6, 0, 0);
      if ( v5 == v6 )
      {
        if ( !g_VBoxInstalled )
          scmInstallDriver(v4, L"VBoxDrv", v1);
        if ( scmStartDriver(v4, L"VBoxDrv") )
          scmOpenDevice(L"VBoxDrv", &v7);
      }
      CloseServiceHandle(v4);
    }
```

Pic 2. TDLStartVulnerableDriver copy-paste.

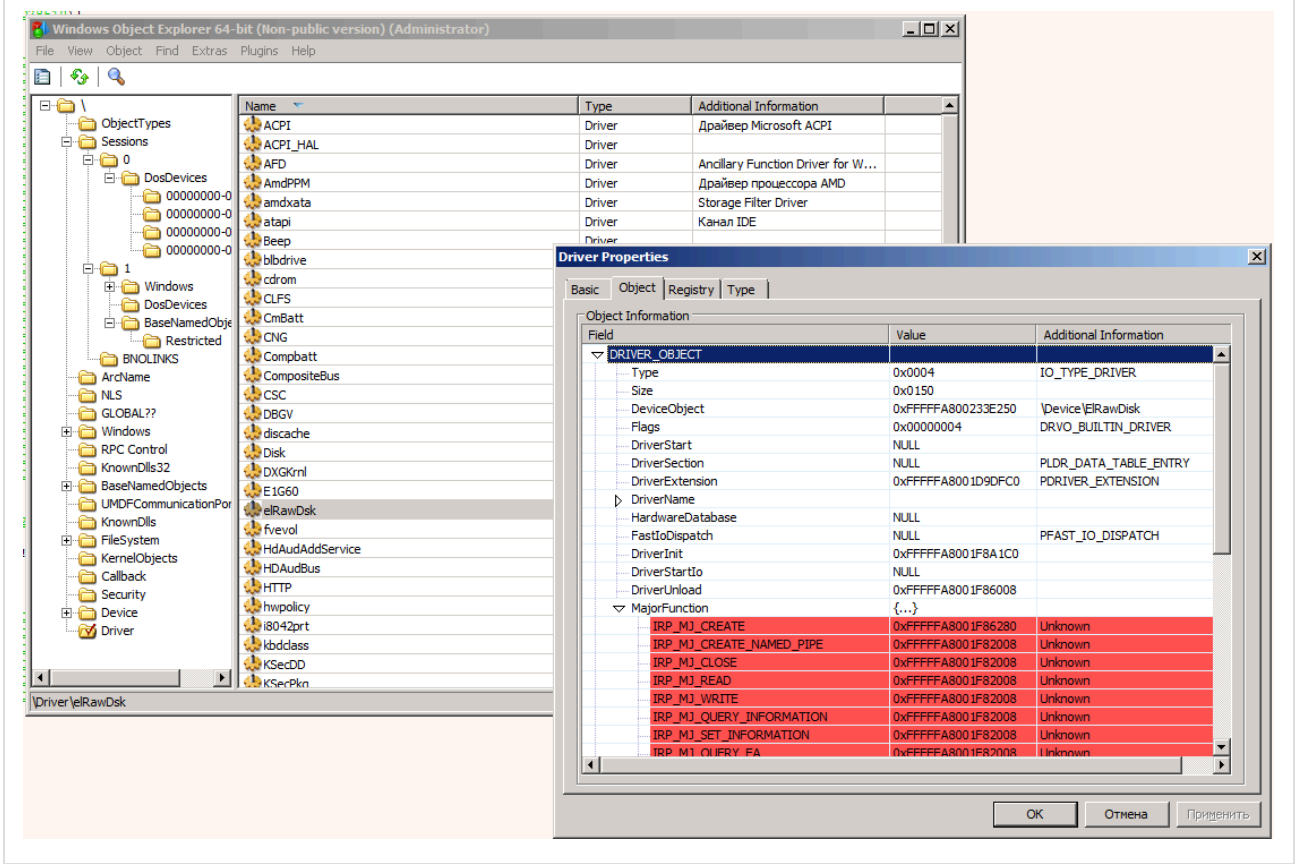
Original routine https://github.com/hfiref0x/TDL/blob/master/Source/Furutaka/main.c#L498

Assistant.sys here is VirtualBox driver which is loaded as shown on picture above. Have no idea why Dustman authors left VirtualBox USB/Network drivers unload code intact. In original TDL this is required to load driver on machine with VirtualBox installed and this is requirement because VBoxHardenedLoader is depends on this. However this is not required in APT and can be removed, but it seems Dustman author(s) had mediocre understanding of what they are doing. It is a little doubtful that target machines has VirtualBox running which can produce incompatibilities with TDL.

Our next stop is TDLMapDriver routine. In original TDL proof-of-concept it setups shellcode that next will be executed in kernel by VBoxDrv, maps input file, processes it imports and merges it with shellcode. Next VBoxDrv memory mapping executed and finally exploit called. In shellcode original TDL allocates memory for driver mapping using **ExAllocatePoolWithTag** routine with tag 'SldT' (Tdl Shellcode), processes image relocs, creates system thread (**PsCreateSystemThread**) with parameter set to driver entry. TDL mapped drivers must be specially designed as DriverEntry parameters in such way of loading will be invalid. Finally thread handle closed with **ZwClose**. Function pointers passed to shellcode through registers by small bootstrap code which is constructed in user mode. Dustman author(s) modified this loading scheme in the following way:

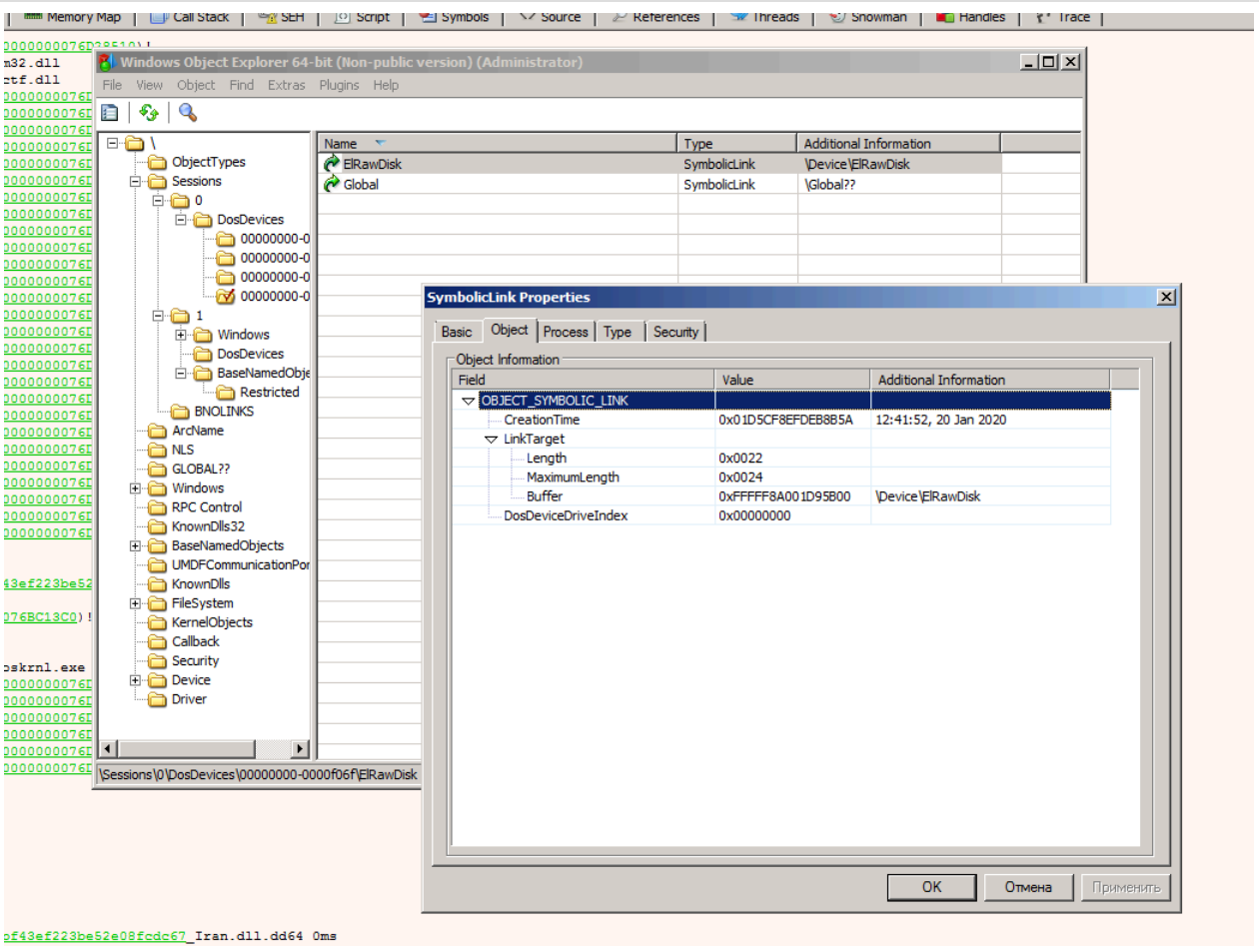
1) Encryption for module/function names, funny note that the following string used to decrypt strings in runtime *"I'm 22 and looking for fulltime job!"*. Because this is copy-paste from open source and original TDL is very well detected by various ~~fake~~AVs (https://www.virustotal.com/gui/file/37805cc7ae226647753aca1a32d7106d804556a98e1a21ac324e5b880b9a04da/detection) this maybe an attempt to remove *some* of these detections.
2) They remember **ExAllocatePoolWithTag**, **PsCreateSystemThread** and **IoCreateDriver** however they never use **PsCreateSystemThread** despite checking it resolving success and instead in their shellcode simple call **IoCreateDriver** with pointer to driver entry point as *InitializationFunction* param.

Since **IoCreateDriver** expects *DriverName* as pointer to UNICODE_STRING modified shellcode also contain "**\Driver\elRawDsk**" string stored as local array of bytes. **IoCreateDriver** will create driver object with specified name and pass it to the InitializationRoutine as parameter, exactly what Eldos RawDisk need at it driver entry. Thus original TDL limitation bypassed and mapping code can work with usual drivers. As result of successful exploitation Eldos RawDisk will be mapped to the kernel and it DriverEntry executed.



Pic 3. Eldos driver object as seen by WinObjEx64.

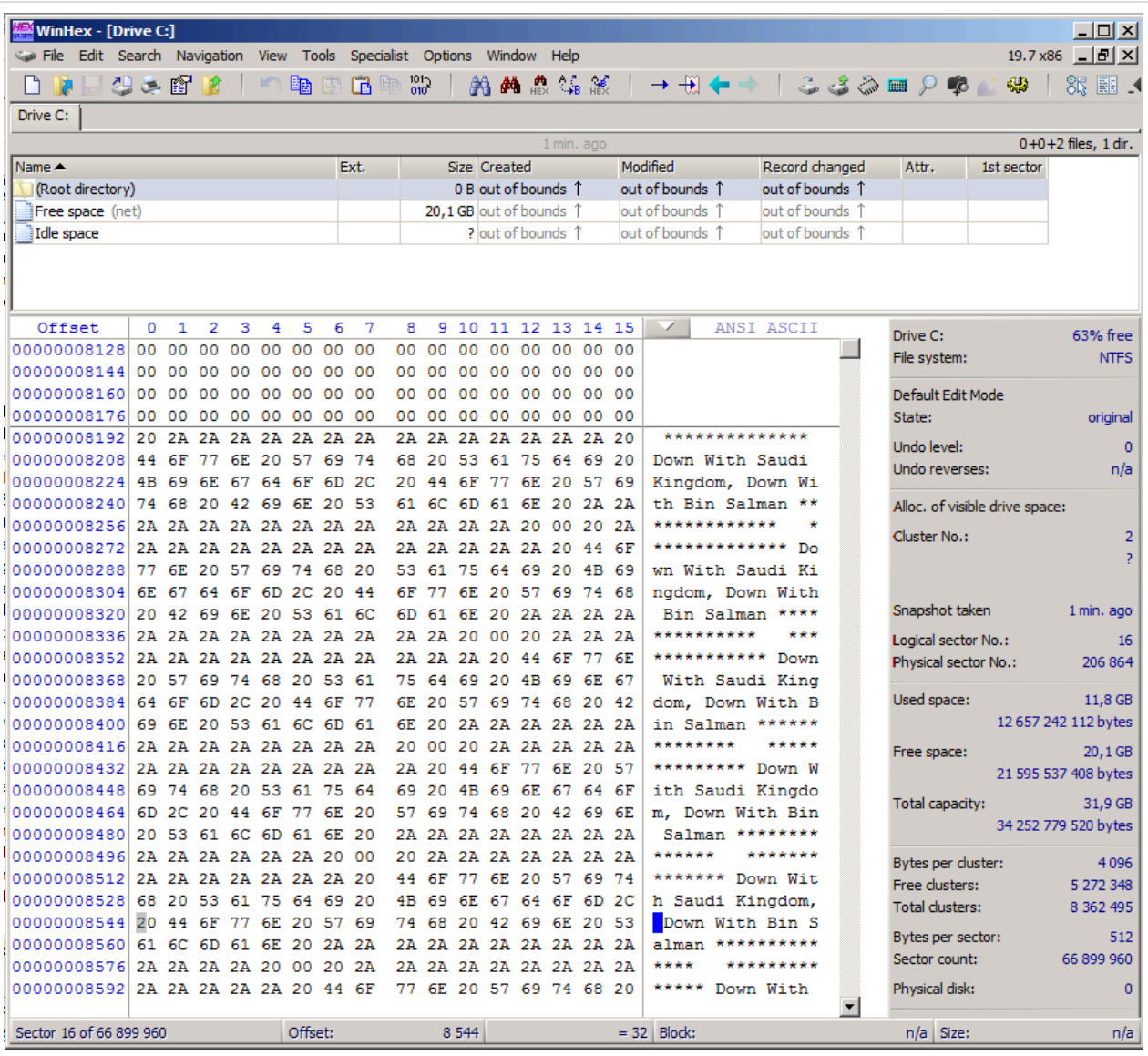
Because driver was mapped without involving Windows loader it doesn't have corresponding entry in **PsLoadedModulesList** therefore WinObjEx64 shows it driver object major functions as belonging to unknown memory area which is always automatically suspicious and usually mean kernel mode malware activity. While Eldos RawDisk DriverEntry execution it creates a symbolic link to provide access for the applications. It also can be seen with WinObjEx64.



Pic 4. ElRawDisk symbolic link.

Here is a mystery or at least question. Why do they use TDL at all? If you look at Eldos RawDisk previous versions, for example https://www.virustotal.com/gui/file/c5c821f5808544a1807dc36527ef6f0248d6768ef9ac5ebabae302d17dd960e4/details you will notice it is digitally signed. As I said at the beginning of this post there can be IPS/AV blocking Eldos driver by it certificate. However why use Eldos RawDisk if you can write your own driver which will be much simpler/smaller (because it will miss useless license check) and use it with TDL? It seems author(s) of Dustman prefer simplest ways and incapable of writing anything beyond simple copy-pasting with small additions. State sponsored hackers, rofl? It of course depends on effectiveness of such methods but I think someone need a bigger budget. However if you take this entire Dustman as false flag operation it looks pretty much ok, because Dustman thing can be built in 4-5 hours and cost almost nothing, while doing severe impact as informational warfare.

A little about agent application, a little because as fact there is nothing interesting inside. It is built as typical C++ MS runtime based application full of ineffective code unrelated to main purpose - wipe data on disk. To do this agent calls Eldos RawDisk with mentioned above license. As data to fill it uses "Down With Saudi Kingdom Down With Bin Salman" string. If agent launched without elevation it will crash with error due to its code quality, state sponsored hackers do you remember?



Pic 5. Wipe in progress.

Source: https://swapcontext.blogspot.com/2020/01/dustman-apt-art-of-copy-paste.html