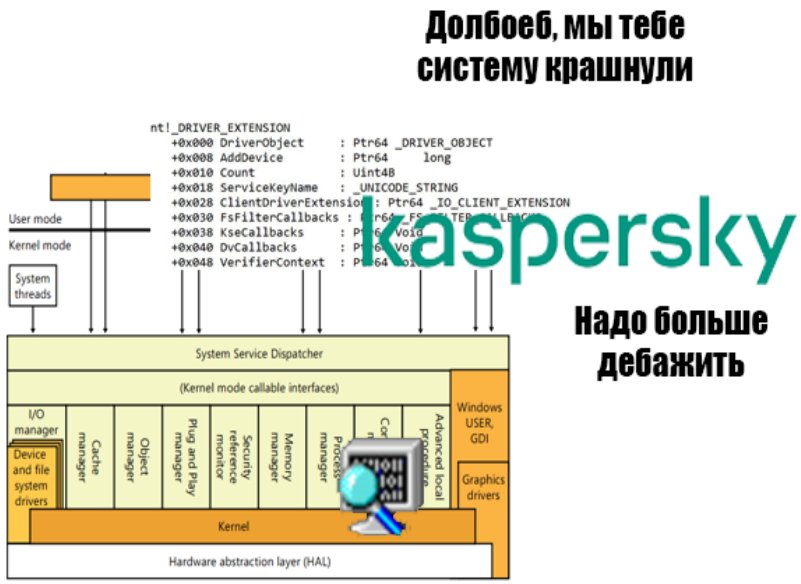


Как мы на Kaspersky в KSE охотились

SKVLLZ.

Где-то примерно неделю назад один мой коллега приметил одну замечательную вещь: Kaspersky интересным образом протянул свои ручки к **Kernel Shim Engine**.

В итоге мы собрались где-то вчера разгребать и смотреть, что он там забыл. Также попытались распилить и отловить шим Касперского, но об этом чуть позже.

Вводные

Начнем с понимания, что это вообще за зверь такой - **Kernel Shim Engine**.
Технически **Kernel Shim Engine** (далее KSE) предоставляет «обертки» для драйверов устройств, а также обеспечивает дополнительную поддержку и обработку ошибок драйверов устройств. У него есть своя специальная база данных, мы затронем её позже.

Также стоит поговорить о кое чем еще.

Инициализация шимов

Если говорить в кратце и не вдаваться в подробности работы, то шимы инициализируются на этапе загрузки системы, где-то сразу после того, как инциализировался **HAL** (Hardware Abstraction Layer) и **WMI** (Windows Management Instrumentation) с **ETW** (Event Tracing for Windows). После прогрузки **HAL**, **WMI** и **ETW** вызывается функция **KseInitialize()**, а после нее **KseRegisterShim()**. Эти функции официально не документированы, но находятся в некоторых заголовочных файлах WDK.

Инициализация «Built-in» шимов драйверов

В конце инициализации HAL происходит инициализация шима **DriverScope**, этот шим предоставляется по стандарту системой:

Имя шима	GUID	Поставщик
DriverScope	{BC04AB45-EA7E-4A11-A7BB-977615F4CAAE}	NT Kernel

Затем, во время инициализации WMI/ETW также инициализируется функция **KseVersionLieInitialize()**. Данная функция инициализирует шимы **KmWin7VersionLie**, **KmWin8VersionLie** и **KmWin81VersionLie** (хоть тесты и производились на десятке, но подобный шим с версией «10» я не обнаружил):

Название шима	GUID	Поставщик
KmWin7VersionLie	{3E28B2D1-E633-408C-8E9B-2AFA6F47FCCB}	NT Kernel
KmWin8VersionLie	{47712F55-BD93-43FC-9248-B9A83710066E}	NT Kernel
KmWin81VersionLie	{21C4FB58-F477-4839-A7EA-AD6918FBC518}	NT Kernel

Потом вызывается функция **KseSkipDriverUnloadInitialize()**, которая инициализирует шим **SkipDriverUnload**.

Функции Касперского в ядре

Окей, вроде +- разобрались. Теперь, какие функции были найдены мной, затрагивающие шимы:

```
lkd> x nt!*KseKaspersky*
fffff801`4a4680a8 nt!KseKasperskyInitialize (void)
fffff801`4a608658 nt!KseKasperskyShim = <no type information>
fffff801`4a608e50 nt!KseKasperskyKernelHooks = <no type information>
fffff801`4a609058 nt!KseKasperskyShimHookCollections = <no type information>
fffff801`4a6160a0 nt!KseKasperskyShimGuid = <no type information>
```

Доступ был только к функции к **KseKasperskyInitialize()**:

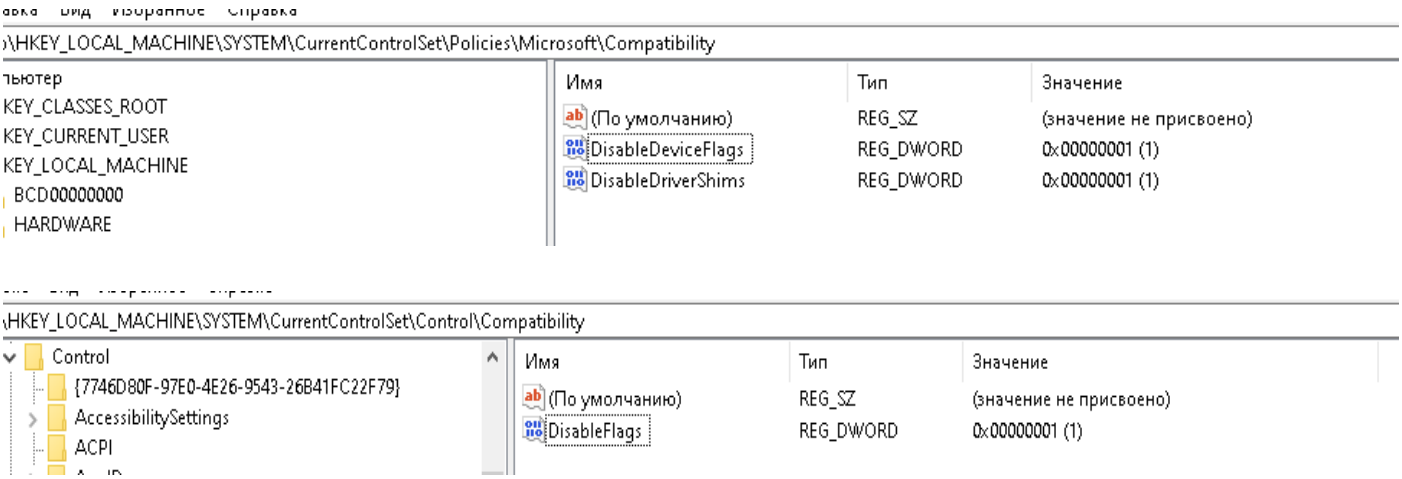
```
nt!KseKasperskyInitialize:
fffff801`4a4680a8 ??              ???
                  ^ Memory access error in 'u nt!KseKasperskyInitialize'
lkd> .fnent nt!KseKasperskyInitialize
Debugger function entry 000002b8`ad5eec68 for:
(fffff801`4a4680a8)   nt!KseKasperskyInitialize   | (fffff801`4a4680d4)   nt!KseSkipDriverUnloadInitialize
Exact matches:
    nt!KseKasperskyInitialize (void)

BeginAddress      = 00000000`00a640a8
EndAddress        = 00000000`00a640cb
UnwindInfoAddress = 00000000`000a8e34

Unwind info at fffff801`49aace34, a bytes
  version 2, flags 0, prolog 4, codes 3
  00: offs 1, unwind op 6, op info 0   UWOP_EPILOG Length: 1. Flags: 0
  01: offs 2, unwind op 6, op info 0   UWOP_EPILOG Offset from end: 2 (FFFFF8014A4680C9)
  02: offs 4, unwind op 2, op info 4   UWOP_ALLOC_SMALL.
```

Сперва мы решили посмотреть, как поведет себя Касперский при манипуляции с реестром, а конкретнее:

- ***HKEY_LOCAL_MACHINE\System\CurrentControlSet\Policies\Microsoft\Compatibility***

  С ключами **DisableDeviceFlags** и **DisableDriverShims** (REG_DWORD), их значения равны единицы, очевидно по названиям ключей, что они делают при значении 0x01.
- ***HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Compatibility***

  С ключем **DisableFlags** (REG_DWORD) со значением 0x01 (отключает функциональность драйвера).



В нашем случае Касперский никак себя не повел, тем более он спокойно дал изменить эти значения, т.е эти ключи реестра не мониторятся Касперским.

Как поломать KSE

Можно сломать KSE двумя путями: мониторинг драйвера через Verifier и с помощью удаления **drvmain.sdb**.

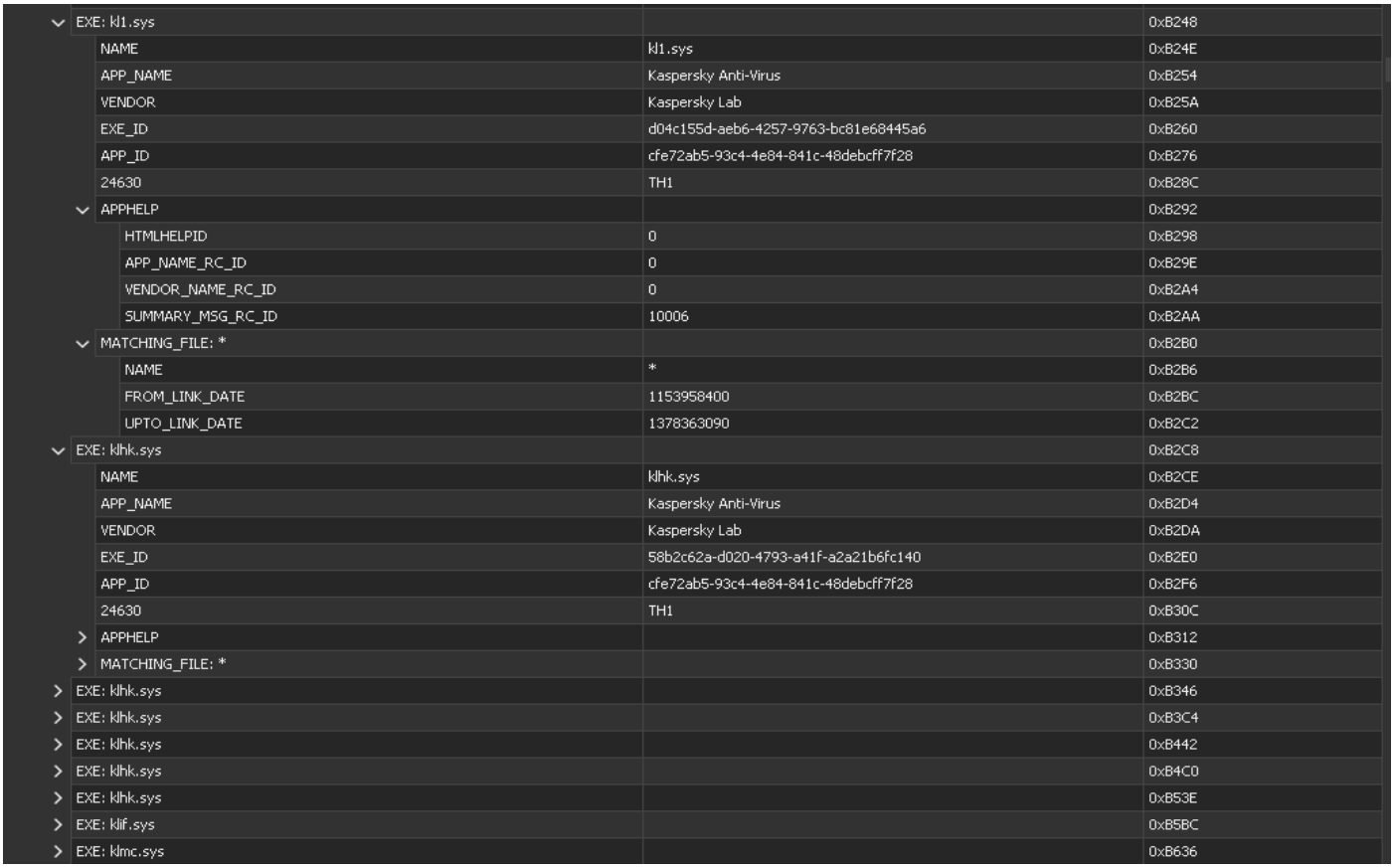
Мы вполне имеем право поставить Verifier на драйвер Касперского, ну или удалить сам drvmain.sdb, но скорее всего это вызовет BSOD.

drvmain.sdb

Мы все же решили для начала порыться в drvmain.sdb. Стоит сказать, что до мониторинга и удаления мы не дошли, а почему – вы поймете позже.

drvmain.sdb – стандартная база данных «легальных» драйверов и не только. Помимо драйверов, там также находится информация о шимах и других вещах. Этот файл находится в «***%systemroot%/apppatch/***». Файлы с расширением sdb можно открыть, к примеру, через SDBExplorer.

Видим в drvmain вот такую картину:



EXE: kl1.sys		0xB248
NAME	kl1.sys	0xB24E
APP_NAME	Kaspersky Anti-Virus	0xB254
VENDOR	Kaspersky Lab	0xB25A
EXE_ID	d04c155d-aeb6-4257-9763-bc81e68445a6	0xB260
APP_ID	cfe72ab5-93c4-4e84-841c-48debcff7f28	0xB276
24630	TH1	0xB28C
APPHELP		0xB292
HTMLHELPID	0	0xB298
APP_NAME_RC_ID	0	0xB29E
VENDOR_NAME_RC_ID	0	0xB2A4
SUMMARY_MSG_RC_ID	10006	0xB2AA
MATCHING_FILE: *		0xB2B0
NAME	*	0xB2B6
FROM_LINK_DATE	1153958400	0xB2BC
UPTO_LINK_DATE	1378363090	0xB2C2
EXE: klhk.sys		0xB2C8
NAME	klhk.sys	0xB2CE
APP_NAME	Kaspersky Anti-Virus	0xB2D4
VENDOR	Kaspersky Lab	0xB2DA
EXE_ID	58b2c62a-d020-4793-a41f-a2a21b6fc140	0xB2E0
APP_ID	cfe72ab5-93c4-4e84-841c-48debcff7f28	0xB2F6
24630	TH1	0xB30C
APPHELP		0xB312
MATCHING_FILE: *		0xB330
EXE: klhk.sys		0xB346
EXE: klhk.sys		0xB3C4
EXE: klhk.sys		0xB442
EXE: klhk.sys		0xB4C0
EXE: klhk.sys		0xB53E
EXE: klif.sys		0xB5BC
EXE: klmc.sys		0xB636

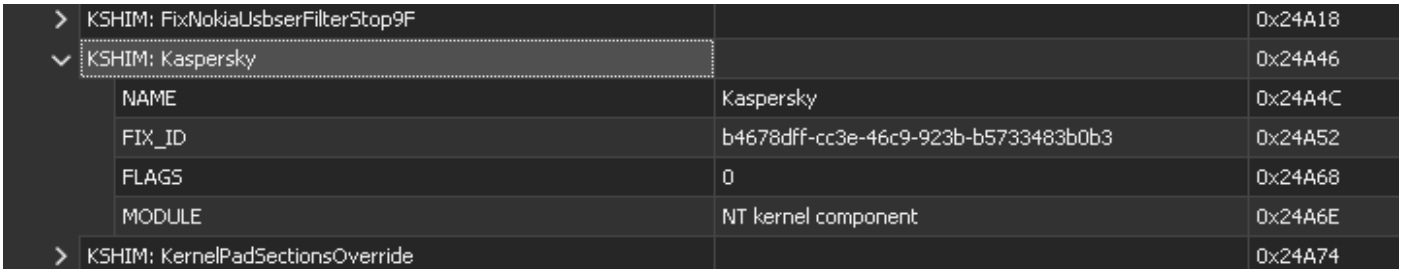
Да, список драйверов Касперского, ничего удивительного. Можно было бы ставить Verifier на klhk.sys и идти отлавливать шим Касперского, но не все так просто.

KSHIM и почему все печально закончилось

В drvmain, как и упоминалось выше, есть не только перечисление драйверов, а также некоторые шимы. Представляются они в виде структуры **KSHIM**, которая по сути является структурой **KSE_SHIM**, только слегка измененной:

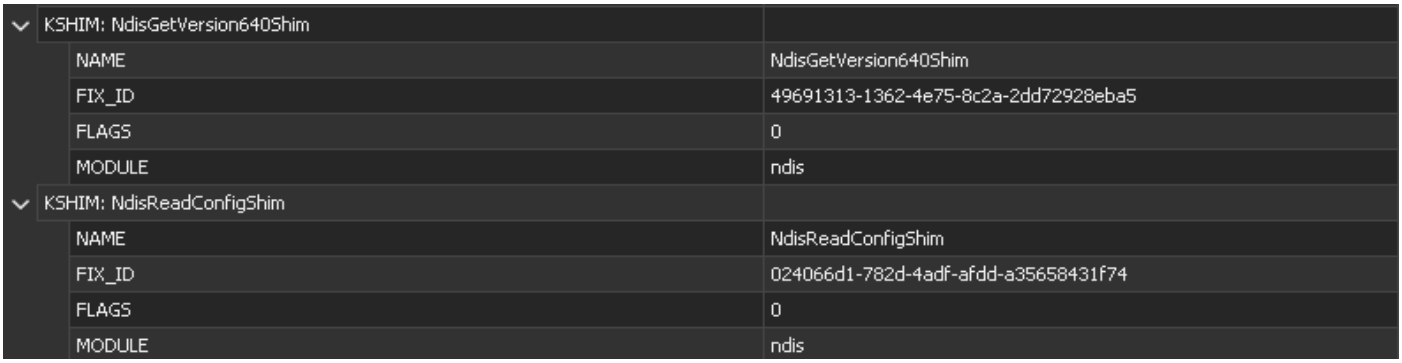
```
typedef struct _KSE_SHIM {
    _In_ ULONG Size;
    _In_ PGUID ShimGuid;
    _In_ PWCHAR ShimName;
    _Out_ PVOID KseCallbackRoutines;
    _Inopt_ PVOID ShimmedDriverTargetedNotification;
    _Inopt_ PVOID ShimmedDriverUntargetedNotification;
    _In_ PVOID HookCollectionsArray;
} KSE_SHIM, *PKSE_SHIM;
```

Вот как выглядит KSHIM Касперского:



KSHIM: FixNokiaUsbserFilterStop9F		0x24A18
KSHIM: Kaspersky		0x24A46
NAME	Kaspersky	0x24A4C
FIX_ID	b4678dff-cc3e-46c9-923b-b5733483b0b3	0x24A52
FLAGS	0	0x24A68
MODULE	NT kernel component	0x24A6E
KSHIM: KernelPadSectionsOverride		0x24A74

Заметьте, что поле **MODULE** несет значение «NT kernel component». А теперь можем посмотреть на ndis шимы:



KSHIM: NdisGetVersion640Shim	
NAME	NdisGetVersion640Shim
FIX_ID	49691313-1362-4e75-8c2a-2dd72928eba5
FLAGS	0
MODULE	ndis
KSHIM: NdisReadConfigShim	
NAME	NdisReadConfigShim
FIX_ID	024066d1-782d-4adf-afdd-a35658431f74
FLAGS	0
MODULE	ndis

Да, тут поле **MODULE** со значением «ndis». Думаю, можно догадаться, в чем проблема.

Ага, это значит, что шим Касперского существует в пространстве самого ядра системы. А это значит, что кроме как накидывая Verifier на ядро мы отследить шим Касперского не сможем. Вот, только если накидывать его на ядро, сломается не только Касперский, но и сама система. Прикольно, правда?

Возможный путь отлова шима Касперского

В мою голову лезет только анализ дампа системы. Т.е мы должны завершить систему аварийно, при этом сгенерировав дамп системы, где был запущен Касперский. Команда «.crash» (если не ошибаюсь) из windbg вполне решит эту проблему. Я, правда, не уверен, что мы сможем вытащить что-то дельное, но попытка не пытка, как говорится.

Послесловие и догадки

Теоретически, **_KSE_SHIM** - пользовательский ввод. Есть смысл посидеть над драйвером ndis.sys (сама структура оттуда). Однако, стучать туда через непривелегированного пользователя мы не сможем, но вполне вероятно, что до туда сможет достать **TrustedInstaller**.