

Flock driver

burluckij@gmail.com

File system Lock driver – является основным компонентом по защите доступа к объектам файловой системы. Пользователь решает какие объекты файловой системы необходимо скрыть от доступа, для этого он указывает эту информацию в графическом приложении, затем эта информация поступает к драйверу, который занимается обеспечением защиты.

В область защиты входят файлы и папки. Тома *возможно* будут в будущем, сейчас нет необходимости.

Внутреннее устройство

* Первая версия работает исключительно на файловых системах формата NTFS и ReFS, FAT32 не поддерживается из-за отсутствия extended attributes ([https://msdn.microsoft.com/ru-ru/library/windows/desktop/ee681827\(v=vs.85\).aspx](https://msdn.microsoft.com/ru-ru/library/windows/desktop/ee681827(v=vs.85).aspx)); Поддержка FAT32 возможна в более поздних версиях (не вижу смысла в downgrade на старте).

* EAs невозможно удалить, атрибуты можно добавлять, просматривать, но запрещено удалять - <https://github.com/jschicht/EaTools/blob/master/readme.txt>

Основная идея

Защита основывается на скрытии объектов файловой системы, путём удаления информации из списков, возвращаемых операционной системой. В случае прямого доступа к заблокированному объекту, драйвер-фильтр будет возвращать ошибку доступа (access denied error code).

Детали реализации

Драйвер не работает с путями файловой системы, случаи: переименования длинного пути, открытие по уникальному идентификатору, обращения по короткому имени (8.3 стандарт) - ни как, не обрабатываются, это облегчает разработку и эффективность защиты. Решение заключается в использовании дополнительных атрибутов файла (extended attributes - EAs). С контролируемым объектом связывается дополнительная информация, которая позволяет пометить данный объект файловой системы как '*контролируемый*', что позволяет применять логику контроля доступа к запрашиваемому файлу.

При таком подходе возможно изменение и самого имени контролируемого файла, политика контроля доступа будет применима независимо от имени контролируемого объекта. Данный подход позволяет файлы перемещаться в рамках файловой системы – с тома на том, с папки в папку и при этом, оставаться в области прицела драйвера контроля доступа. С файлом ассоциируются зарезервированные и уникальные для системы расширенные атрибуты, после чего он под контролем. Расширенные атрибуты это пары key : value, где key это строка из ascii символов, в нашем случае это выглядят следующим образом:

"FLOCK_META" : structure (FLOCK_META)

```
typedef struct _FLOCK_META
{
    UCHAR signature[16];
    DWORD version;
    UCHAR uniqueId[FLOCK_UNIQUE_ID_LENGTH == 16 bytes];
    DWORD flags;
} FLOCK_META, *PFLOCK_META;
```

.signature – уникальный идентификатор атрибута, сигнатура одинаковая для всех структур данного типа, равняется - {0xB1, 0x0E, 0x21, 0xf4, 0xb2, 0x1E, 0x27, 0x21, 0x12, 0x12, 0x12, 0x12, 0x28, 0x33, 0x92, 0x11};

.version – номер версии формата структуры, предполагается всегда быть равным нулю.

.uniqueId – уникальный идентификатор, ассоциированный с объектом файловой системы, должен быть уникальным в рамках всей системы (желательно и всего мира). Любая политика доступа будет применяться относительно этого поля, коллизий быть не должно.

.flags – поле для хранения дополнительной информации, флагов и характеристик состояния объекта.

Для того чтобы пометить объект защищённым должна выполняться следующая последовательность действий:

1. Пользователь добавляет объект в область контроля доступа, если он ранее не был добавлен. User-mode сервис сохраняет в своём зашифрованном списке контролируемых объектов, новый объект;
 2. Сервис Data Guard отправляет запрос драйверу, чтобы добавить файл в область защиты;
 3. Flock драйвер добавляет в контролируемый файл мета-информацию (**FLOCK_META**), сохраняет информацию о новом файле в списке контролируемых объектов, который полностью управляется драйвером. В ядре имеется своя защищённая копия всех контролируемых объектов, это дает независимость от основного сервиса;
 4. Flock драйвер на данном этапе может осуществлять контроль доступа к добавленному объекту. При каждом обращении к контролируемому объекту, из мета информации будет извлекаться уникальный идентификатор, к которому будет применяться политики доступа. Список с ограничениями доступа к файлу хранится в ядре, в отдельном файле, который защищён драйвером.
- Важно знать, что в метаинформации хранится только идентификатор объекта, а не конечная политика доступа.

Действия обработчиков мини-фильтра

Драйвер фильтр реализован в виде мини-фильтра, в котором обрабатываются следующие IRP запросы:

IRP_MJ_CREATE

К примеру, был запрошен следующий ресурс – X:\work\protected\sara\docs\secrets.txt

В то время, как скрыт доступ к подчёркнутой части – x:\work\protected, ожидается что все подкаталоги и файлы должны быть защищены от доступа, в тот момент, когда данный ресурс заблокирован от доступа. Конечный файл secrets.txt не имеет метаинформации, соответственно

для него всегда применяется политика разрешения доступа, чего совершенно невозможно допустить!

Ситуация решается следующим образом – происходит проверка на родительских объектах.

Первая итерация решения конфликта это просмотр метаинформации для – X:\work\protected\sara\docs – который в свою очередь так же не имеет мета информации, вторая итерация – проверка мета информации для – X:\work\protected\sara, тут так же нет метаинформации, поиск продолжается, третья итерация для – X:\work\protected, бинго! Метаинформация присутствует, требуется найти статус для этого объекта контроля в общем хранилище всех контролируемых объектов. Исходя из полученного статуса – либо вернуть ошибку доступа, либо разрешить доступ. Данные Сары будут надёжно защищены!

Более точное техническое описание для *IRP_MJ_CREATE*

* Стоит сразу вспомнить что выполнение pre, post обработчиков синхронизировано, т.е. чтобы выполнить post обработчик на IRLQ меньшем чем DISPATCH_LEVEL и воспользоваться всеми прелестями работы с PASSIVE_LEVEL функциями ядра ОС и подкачиваемой памятью – не нужно возвращать *FLT_PREOP_SYNCHRONIZE* код, достаточно вернуть *FLT_PREOP_SUCCESS_WITH_CALLBACK* и Post-обработчик будет выполнен в контексте вызывающего потока на соответствующем IRLQ.

Pre-operation handler:

- 1) Если на текущий момент нет ни одного объекта, доступ к которому необходимо контролировать, то игнорировать любой контроль доступа – возвращать *FLT_PREOP_SUCCESS_NO_CALLBACK*.
- 2) Если файл открывается без флага *FILE_DELETE_ON_CLOSE*, позволить отработать post-обработчику, выполнить проверку прав на доступ. Если не учесть факта установки соответствующего флага, то при закрытии его дескриптора, даже если мы и запретим выполнение уже на уровне post обработчика, когда дескриптор уже будет создан, то мы пропустим удаление файла, чего нельзя допустить.
- 3) Флаг *FILE_DELETE_ON_CLOSE* установлен, значит проверку на доступ требуется выполнить в pre-обработчике. Такие случаи возникают не часто.
 - a. Запросить атрибуты для X:\work\protected\sara\docs\secrets.txt
 - b. Если файл имеет соответствующие атрибуты, выполнить действие, предусмотренное политикой.
 - c. Атрибуты не найдены, продолжать запрашивать пока не упрёмся в корень диска X: (если не найдём раньше), а вообще запрашивать в следующей последовательности X:\work\protected\sara\docs -> X:\work\protected\sara -> X:\work\protected -> на этом этапе атрибуты будут найдены. Необходимо принять решение на основе политики доступа для данного элемента.

Данная цепочка действий достаточно затратна по эффективности, но всё-таки эффективна.

Если не использовать поиск мета-информации для родительских объектов, то зная точный путь к некоторому файлу, злоумышленник сможет беспрепятственно получить доступ к запрашиваемому файлу.

* Folder Lock – на контролирует доступ к содержимому папки! 54 млн клиентов данный подход вполне устраивает.

Если отказаться от такой “раскрутки”, то предлагаемая защита будет эффективна для штатного проводника Windows (explorer.exe).

! Предлагаю вынести эту углубленную проверку прав на доступ в отдельную “галочку” в настройках. Любые критики неэффективности такого подхода защиты, смогут включить режим углубленной проверки прав доступа.

! При “раскрутке” пути, от дочернего к родителю стоит быть осторожными при чтении EAs из тома (Volume) – это очень затратно по времени! Критически, важно избегать чтения атрибутов из тома, такую информацию нужно кешировать, кеш – наше спасение. Детальное описание кеша будет дано ниже.

4) -

Post-operation handler:

Запрещает доступ к файлу, при наличии флага `FLOCK_FLAG_LOCK_ACCESS`.

При получении списка файлов (`IRP_MJ_DIRECTORY_CONTROL`):

Файлы скрываются в данном обработчике, схема скрытия следующая – скрываемый файл помечается соответствующим атрибутом, а в хранилище с ним ассоциируется флаг `FLOCK_FLAG_HIDE`, который помечает файл как нуждающийся в сокрытии, но это ещё не всё, родительский каталог файла так же помечается соответствующим атрибутом с флагом `FLOCK_FLAG_HAS_FLOCKS`, который нужен чтобы знать – нужно ли производить обработку информации, полученную от низкоуровневых драйверов с целью скрытия файла из списка. Такой подход позволяет избежать излишней нагрузки на файловую систему – наш фильтр будет работать только по нужным каталогам, которые действительно имеют скрытые файлы.

Pre-handler:

1. Проверить в хранилище – есть ли какие-либо файлы, папки, которые требуется скрывать? Если нет ни одного пользовательского объекта файловой системы, который требуется скрыть – прекратить обработку запроса, фильтровать информацию не нужно.
2. Обработывает запросы, для которых которые удовлетворяют условию:

`Data->Iopb->MinorFunction != IRP_MN_QUERY_DIRECTORY`

3. Воспользоваться существующим открытым `FILE_OBJECT`, если есть конечно, считать метаинформацию, если есть - проверить флаг (`FLOCK_FLAG_HAS_FLOCKS`), наличия вложенных для скрытия объектов. Если есть что скрывать, то запланировать выполнение post-обработчика, возвращая `FLT_PREOP_SYNCHRONIZE`.

* Крайне необходимо синхронизировать выполнение post обработчика, потому что он делает системные вызовы, для которых `IRQL < DISPATCH_LEVEL`.

4. —

Post-handler:

1. Обработывает IRP для которых установлен `Irp.MinorFunction = IRP_MN_QUERY_DIRECTORY`.
2. Обработывает полученный список файлов – последовательно открывает каждый из файлов, считывает их расширенные атрибуты (EAs). При наличии флага `FLOCK_FLAG_HIDE` файл будет удален из списка.

* Если файлу одновременно указать `FLOCK_FLAG_HIDE` и `FLOCK_FLAG_LOCK_ACCESS`, то скрыть файл не получится, по причине невозможности считать расширенные атрибуты из-за необходимости открытия файла. Post-operation handler в `IRP_MJ_CREATE` вернёт `STATUS_ACCESS_DENIED`. (Не всегда! Сейчас работает.).

При установке расширенных атрибутов (`IRP_MJ_SET_EA`)

Требуется запрещать удаление метаинформации, записанной Flock'ом. Установку и любые модификации расширенных атрибутов возможно выполнять, если текущим процессом является менеджерский сервис – `DataGuardService.exe`. Менеджерским может быть только один процесс, он регистрируется в момент старта службы вместе со стартом ОС.

Pre-operation-handler:

Вся необходимая информация доступна на данном этапе, нам требуется просмотреть каждый элемент из списка устанавливаемых атрибутов – если имеется атрибут Flock'a («FLOCK_META»), принять следующие действия:

- Отклонить весь запрос (сейчас так и происходит)
 - `Data->IoStatus.Status = STATUS_ACCESS_DENIED;`
 - `return FLT_PREOP_COMPLETE;`
- Если в списке более чем один элемент, удалить который с атрибутом Flock'a (то есть скрыть из списка).
- Изменить название атрибута с FLOCK_META, на некоторый FAKE_META.

Post-operation handler:

* Действий не требуется.

При чтении расширенных атрибутов (`IRP_MJ_QUERY_EA`)

Скрывать метаинформация Flock'a требуется, по причине копирования файла с одного места в другое. Если пользователь запретит доступ к файлу расположенному по адресу `x:\work\file.doc`, затем на какое-то время разрешит к нему доступ, обновит содержимое, а потом решит что нужно скопировать обновлённый файл на новое место, где к нему будет публичный доступ, в рамках одного компьютера. Вся хитрость происходит в этот момент, расширенные атрибуты так же подлежат копированию и если целевая файловая система их поддерживает, они будут скопированы! Доступ к файлу с нового места будет запрещён, хотя пользователь не желал этого.

Если скрывать атрибуты из общего списка, скопированы будут все атрибуты, кроме тех, что принадлежат Flock'у.

Если вызов выполнялся в контексте сервисного процесса Data Guard, никакой фильтрации не требуется, но если вызов был сделан в контексте иного процесса, следует удалить атрибуты Flock'a из общего списка.

При модификации файла (*IRP_MJ_SET_INFORMATION*).

Защищать от удаления.

Cache for EAs searching

В процессе поиска прав на доступ к некоторому ресурсу, происходит поиск расширенных атрибутов с метаданной (Flock-meta), необходимой для принятия решения о доступе. Такой процесс поиска будем называть - раскруткой пути. Ниже представлен лог работы драйвера в процессе раскрутки пути.

0:57:06 FLockFltSearchFirstMetaPath: Delimiter was found -
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData\\Local\\Google\\Chrome\\User Data\\Default\\Cache,
length is 176, delPos 88, rootEndPos 23

0:57:06 FLock!FlockFltOpenAndReadFirstMeta: Success -
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData\\Local\\Google\\Chrome\\User Data\\Default\\Cache
was opened, status code is 0x0 (0)

0:57:06 FLockFltSearchFirstMetaPath: failed - FLock-meta not found in
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData\\Local\\Google\\Chrome\\User Data\\Default\\Cache,
status is 0xc000090b

0:57:06 FLockFltSearchFirstMetaPath: Delimiter was found -
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData\\Local\\Google\\Chrome\\User Data\\Default, length is
164, delPos 82, rootEndPos 23

0:57:06 FLock!FlockFltOpenAndReadFirstMeta: Success -
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData\\Local\\Google\\Chrome\\User Data\\Default was
opened, status code is 0x0 (0)

0:57:06 FLockFltSearchFirstMetaPath: failed - FLock-meta not found in
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData\\Local\\Google\\Chrome\\User Data\\Default, status is
0xc000090b

0:57:06 FLockFltSearchFirstMetaPath: Delimiter was found -
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData\\Local\\Google\\Chrome\\User Data, length is 148,
delPos 74, rootEndPos 23

0:57:06 FLock!FLockFltOpenAndReadFirstMeta: Success -
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData\\Local\\Google\\Chrome\\User Data was opened,
status code is 0x0 (0)

0:57:06 FLockFltSearchFirstMetaPath: failed - FLock-meta not found in
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData\\Local\\Google\\Chrome\\User Data, status is
0xc000090b

0:57:06 FLockFltSearchFirstMetaPath: Delimiter was found -
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData\\Local\\Google\\Chrome, length is 128, delPos 64,
rootEndPos 23

0:57:06 FLock!FLockFltOpenAndReadFirstMeta: Success -
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData\\Local\\Google\\Chrome was opened, status code is
0x0 (0)

0:57:06 FLockFltSearchFirstMetaPath: failed - FLock-meta not found in
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData\\Local\\Google\\Chrome, status is 0xc000090b

0:57:06 FLockFltSearchFirstMetaPath: Delimiter was found -
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData\\Local\\Google, length is 114, delPos 57, rootEndPos
23

0:57:06 FLock!FLockFltOpenAndReadFirstMeta: Success -
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData\\Local\\Google was opened, status code is 0x0 (0)

0:57:06 FLockFltSearchFirstMetaPath: failed - FLock-meta not found in
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData\\Local\\Google, status is 0xc000090b

0:57:06 FLockFltSearchFirstMetaPath: Delimiter was found -
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData\\Local, length is 100, delPos 50, rootEndPos 23

0:57:06 FLock!FLockFltOpenAndReadFirstMeta: Success -
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData\\Local was opened, status code is 0x0 (0)

0:57:06 FLockFltSearchFirstMetaPath: failed - FLock-meta not found in
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData\\Local, status is 0xc000090b

0:57:06 FLockFltSearchFirstMetaPath: Delimiter was found -
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData, length is 88, delPos 44, rootEndPos 23

0:57:06 FLock!FLockFltOpenAndReadFirstMeta: Success -
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData was opened, status code is 0x0 (0)

0:57:06 FLockFltSearchFirstMetaPath: failed - FLock-meta not found in
\\Device\\HarddiskVolume1\\Users\\admin0\\AppData, status is 0xc000090b

0:57:06 FLockFltSearchFirstMetaPath: Delimiter was found - \Device\HarddiskVolume1\Users\admin0, length is 72, delPos 36, rootEndPos 23

0:57:06 FLock!FLockFltOpenAndReadFirstMeta: Success - \Device\HarddiskVolume1\Users\admin0 was opened, status code is 0x0 (0)

0:57:06 FLockFltSearchFirstMetaPath: failed - FLock-meta not found in \Device\HarddiskVolume1\Users\admin0, status is 0xc000090b

0:57:06 FLockFltSearchFirstMetaPath: Delimiter was found - \Device\HarddiskVolume1\Users, length is 58, delPos 29, rootEndPos 23

0:57:06 FLock!FLockFltOpenAndReadFirstMeta: Success - \Device\HarddiskVolume1\Users was opened, status code is 0x0 (0)

0:57:06 FLockFltSearchFirstMetaPath: Success - FLock-meta was found in \Device\HarddiskVolume1\Users

Метаинформация была найдена в \Device\HarddiskVolume1\Users, следует прекратить поиск, перейти к принятию решения о доступе.

Если бы метаинформация не была найдена, то мы пошли на следующий этап – проверка прав на доступ к корневому каталогу, а он том - \Device\HarddiskVolume1, как говорилось ранее, поиск метаинформации среди расширенных атрибутов для тома – космически затратная, дорогая операция, несколько последовательных запросов могут полностью приостановить работу системы! Эту информацию следует всегда искать в кеше.

0:57:06 FLockFltSearchFirstMetaPath: Delimiter was found - \Device\HarddiskVolume1, length is 46, delPos 23, rootEndPos 23

0:57:06 FLockFltSearchFirstMetaPath: Ignore reading EAs from volume - FLock-meta not found in \Device\HarddiskVolume1

Проблема производительности

Система часто пытается открывать одни и те же файлы (возможно это просто особенность мини-фильтров), привожу скриншот с подтверждением.

#	Time	Debug Print
362	0.23034669	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Users
363	0.23037291	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\
364	0.23037660	Flock!FlockFltReadFirstMeta: success - Flock meta found.
365	0.23037790	Flock!FlockFltReadFirstMeta: success - Flock meta was read and validated.
366	0.23039760	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Users
367	0.23056430	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Users
368	0.23058470	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Users
369	0.23267449	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\ntmarta.dll
370	0.23279130	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\ntmarta.dll
371	0.23301341	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\Wldap32.dll
372	0.23311649	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\Wldap32.dll
373	0.23470791	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\clbcatq.dll
374	0.23482341	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\clbcatq.dll
375	0.23529020	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\cryptsp.dll
376	0.23540939	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\cryptsp.dll
377	0.23566140	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\rsaenh.dll
378	0.23571390	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\rsaenh.dll
379	0.23579609	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\rsaenh.dll
380	0.23584580	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\rsaenh.dll
381	0.23594511	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\rsaenh.dll
382	0.23599760	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\rsaenh.dll
383	0.23606680	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\rsaenh.dll
384	0.23611450	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\rsaenh.dll
385	0.23618960	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\rsaenh.dll
386	0.23624200	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\rsaenh.dll
387	0.23895440	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\rsaenh.dll
388	0.23906830	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\rsaenh.dll
389	0.23927490	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\Globalization\Sorting\SortDefault.nls
390	0.24198849	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\AudioSes.dll
391	0.24210580	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\AudioSes.dll
392	0.24287780	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\Registration\R0000000000006.clb
393	0.24312051	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\AudioEng.dll
394	0.24323060	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\AudioEng.dll
395	0.24387240	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\avrt.dll
396	0.24401470	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\avrt.dll
397	0.24477540	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\setupapi.dll
398	0.24488540	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\setupapi.dll
399	0.24508379	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\cfgmgr32.dll
400	0.24518530	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\cfgmgr32.dll
401	0.24535890	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\devobj.dll
402	0.24545769	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\devobj.dll
403	0.24572480	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\ru-RU\setupapi.dll.mui
404	0.24636240	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\AUDIOKSE.dll
405	0.24649531	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\AUDIOKSE.dll
406	0.24895340	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\
407	0.24896550	Flock!FlockFltReadFirstMeta: success - Flock meta found.
408	0.24896690	Flock!FlockFltReadFirstMeta: success - Flock meta was read and validated.
409	0.24899291	Flock!FlockPostCreate: open - \Device\HarddiskVolume1\
410	0.24900930	Flock!FlockFltReadFirstMeta: success - Flock meta found.

В ядре Windows есть возможность воспользоваться Generic и AVL деревьями, но нет возможности использовать готовые хеш-таблицы, нужно реализовать самостоятельно. Таблица должна умещать в себе как минимум 1000 записей, сама таблица очень быстро превратится в обычный массив с линейным поиском, таблица для элементов 1000, должны иметь минимальный размер 10000 ячеек, а при условии, что мы храним в таблице md5 хеш строки с полным путём к файлу и логическое значение, которое говорит о необходимости чтения атрибутов с диска.

16 byte Hash : 1 byte value – 17 byte на одну запись, $17 * 1000 = 17\ 000$, 170 000 на таблицу из 10000 тыс. ячеек → около 167 килобайт памяти в ядре нужно будет держать под одну таблицу.

Можно немного похитрить, ниже строка, которая пришла в post-operation handler для IRP_MJ_CREATE

Flock!FlockPostCreate: open - \Device\HarddiskVolume1\Windows\System32\imm32.dll

Можно запоминать имя файла - imm32.dll и отправлять запрос на чтение атрибутов только если конечное имя совпадает с одним из тех, которое имеется в хранилище. Так же при таком подходе

требуется приводить строки к одному регистру, а так же каждый раз считывать атрибуты с диска, в случае, если конечное имя задано в 8.3 формате.

Storage

Flock – объект файловой системы, доступ к которому необходимо ограничить, скрыть.

Технически, Flock'ом может являться любой файл, каталог, который в списке своих расширенных атрибутов будет иметь специальную сигнатуру, по которой драйвер будет распознавать его как объект, к которому необходимо применить дополнительную проверку безопасности. Политика доступа хранится отдельно в хранилище.

В общем процесс доступа выглядит так:

1. Can I open the file? -> 2. read flock_id from Bank_accounts.doc -> 3.
driver_storage[flock_id_XXX] = policy_rule;

Хранение информации с политиками доступа должно осуществляться на нескольких уровнях. Первый и основной – пользовательский режим, со стороны сервиса, второй уровень – драйвер режима ядра, в нем должна быть информация только об идентификаторах Flock'ов и флаги доступа – lock, hide. Пути к файлам хранятся только на уровне приложения в юзермоде, на диске они лежат в зашифрованном виде и расшифровываются для отображения пользователю в окне графического клиента.

Сервис пользовательского режима Data Guard (далее просто сервис) управляет двумя списками Flock'ов.

1. Драйверный – должен быть доступен драйверу, ещё до того момента, когда будет загружен сервисный процесс. Так, к примеру, в безопасном режиме загрузки наш сервис безопасности может быть вообще не загружен, а драйверу необходимо осуществлять контроль доступа при любых вариантах загрузки системы. Элементы списка драйвера имеют следующие поля:
 - a. Номер версии структуры;
 - b. Уникальный 16-байтовый идентификатор Flock'a;
 - c. 4-байтовый список флагов, отражающий набор применяемых политик безопасности, к примеру – заблокировать доступ, скрыть из общего списка файлов.
- Набор информации крайне куцый, что замечательно с точки зрения безопасности, никого полного пути к файлу найти невозможно. Есть только идентификатора объекта и политика доступа.
2. Сервисный – хранится в зашифрованном виде. Любая модификация списка требует пользовательской аутентификации в главном приложении. Каждый элемент имеет следующую информацию:
 - a. Оригинальный путь к объекту файловой системы.
 - b. Идентификатор.
 - c. Политика доступа.
 - d. Дата создания.

е. Дата модификации последней.

* Информация зашифрована, чтобы злоумышленник не смог прочитать путь к охраняемому ресурсу, в случае успешного взлома системы безопасности. Полный путь к файлу не всегда актуальный, часть его пути может быть подвержена переименованию, но нас это не страшит, мы закладываемся на идентификатор, зашитый в расширенные атрибуты! Fuck yeah!

Пример добавления FLock'a в область контроля доступа:

1. Сгенерировать уникальный 16-байтовый идентификатор.
2. Записать сгенерированный идентификатор в Extended Attributes. Это необходимо, чтобы пометить файл, говоря на нашем жаргоне – сделать его FLock'ом.
3. Отправить в Flock-драйверу запрос, чтобы он добавил новый FLock к списку уже имеющихся FLock'ов.
4. Отправить драйверу flush-запрос, для сброса всех последних изменений на диск, чтобы не потерять актуальную информацию.

* Более полное описание будет позднее.

Список команд для работы с драйвером:

1. *Инициализация сервисного процесса. Позволяет выполнять привилегированные операции.
2. *Удаление ранее зарегистрированного сервисного процесса. Возможно только в контексте процесса, ранее зарегистрированного сервисным.
3. *Получение идентификатора сервисного процесса.
4. *Получение информации о доступности файлового хранилища в ядре. Открыт ли файл с FLock'ами или нет.
5. Запрос состояния хранилища списка flock'ов в ядре. В некоторых случаях могут познать сложности с загрузкой списка – повреждение структуры данных, хакерская атака и т.п.
6. Принудительная загрузка списка flock'ов.
7. Сброс (flush) списка flock'ов из памяти ядра на диск.
8. *Получение списка flock'ов из памяти ядра.
9. *Получение одного flock'a из общего списка ядра.
10. *Удаление flock'a из списка ядра.
11. *Добавление flock'a в список ядра.
12. *Изменение политики доступа для flock'a в общем списке ядра.
13. *Запрос на чтение FLock атрибутов с файла на диске.
14. *Запрос на модификацию FLock атрибутов файла на диске.
15. *Инвалидация FLock атрибутов файла на диске. Операция равносильна удалению атрибутов, но из-за отсутствия возможности удаления атрибутов, они просто нарушаются (портится контрольная сумма идентификатора).