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 информацию поступает к драйверу, который занимается обеспечение защиты.

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

Внутреннее устройство

* Первая версия работает исключительно на файловых системах формата NTFS, FAT32 не поддерживается из-за отсутствия Extended attributes (https://msdn.microsoft.com/ru-ru/library/windows/desktop/ee681827(v=vs.85).aspx); Поддержка fat32 будет введена в более поздних версиях.

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

Идея защиты 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).

Драйвер не работает с путями файловой системы, случаи: переименования длинного пути, обращения по короткому имени - ни как, не обрабатываются, что облегчает разработку и эффективность защиты. Решение заключается в использовании дополнительных атрибутов файла (Extended Attributes). 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.

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

1. Пользователь добавляет объект в область контроля доступа, если он ранее не был добавлен.
2. Сервис Data Guard отправляет запрос драйверу, чтобы добавить файл в область защиты.
3. FLock драйвер добавляет в файл мета-информацию, сохраняет информацию о новом файле в общем списке контролируемых объектов.
4. FLock драйвер на данном этапе может осуществлять контроль доступа к добавленному объекту.

Действия обработчиков мини-фильтра:

1. При открытии файла (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 IRQL.

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. Обрабатывает 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. Обрабатывает IRP для которых установлен Irp.MinorFunction = *IRP_MN_QUERY_DIRECTORY*.
2. Обрабатывает полученный список файлов – последовательно открывает каждый из файлов, считывает их расширенные атрибуты (EAs). При наличии флага FLOCK_FLAG_HIDE файл будет удалятся 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'а («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'а (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'а 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'а 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 метаинформацией (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.

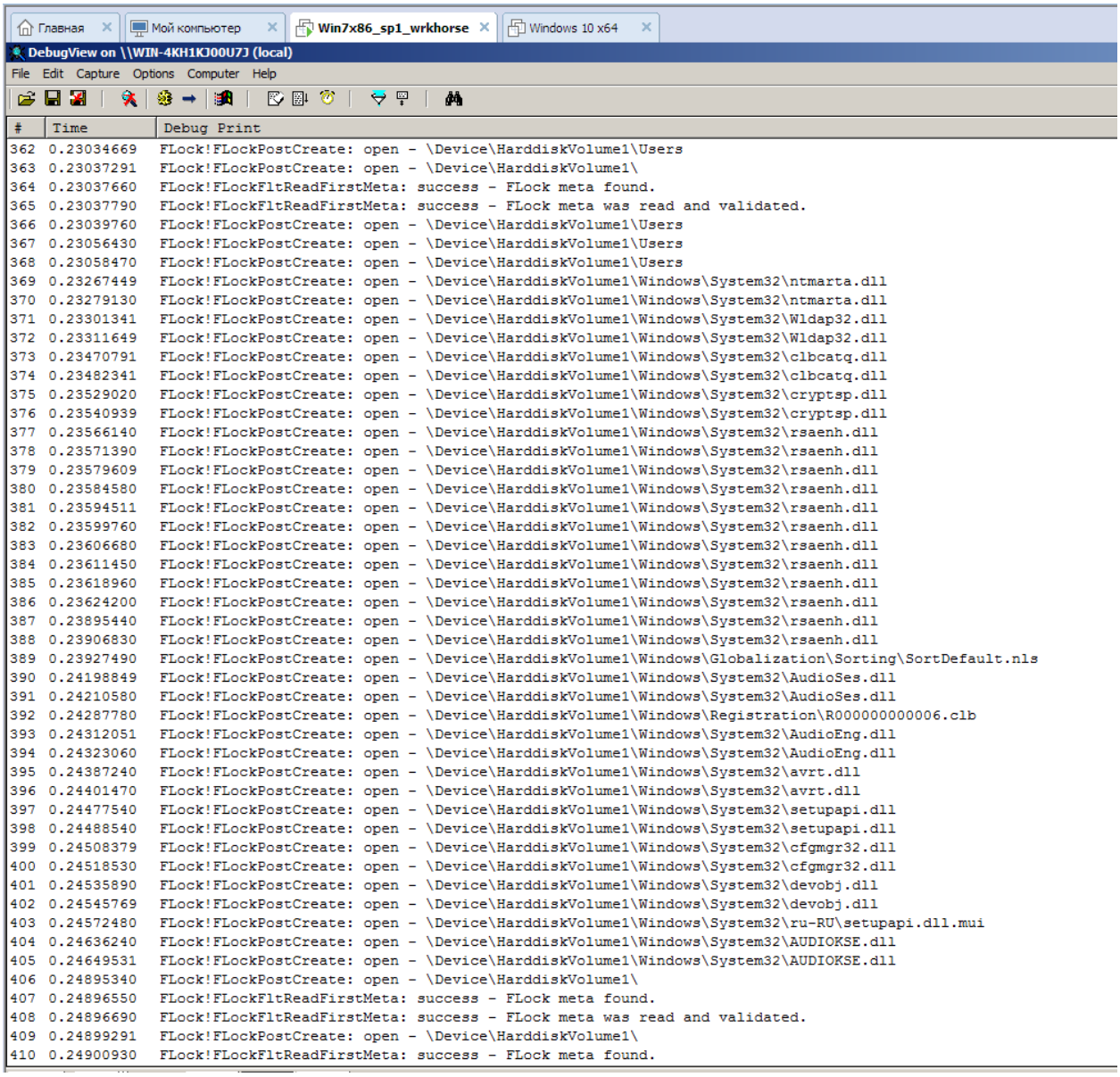
Если бы метаинформация не была найдена, то мы пошли на следующий этап – проверка прав на доступ к корневому каталогу, а он том - \Device\HarddiskVolume1, как говорилось ранее, поиск метаинформации среди расширенных атрибутов для тома – космически затратная, дорогая операция, несколько последовательных 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.



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 – объёкт 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_accaunts.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 расшифровуются 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'а;
    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.

e. 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’а в область контроля доступа:

1. Сгенерировать уникальный 16-байтовый идентификатор.
2. Записать сгенерированный идентификатор в Extended Attributes. Это необходимо, чтобы пометь 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 изменения на диск, чтобы не потерять актуальную информацию.

* Более полное описание будет позднее.