АНАЛИЗ ПОДХОДОВ К ЗАЩИТЕ MDB-ФАЙЛОВ СУБД MICROSOFT ACCESS С ИСПОЛЬЗОВАНИЕМ СРЕДСТВ ШИФРОВАНИЯ

В.В. Пылин

В работе представлен анализ встроенного метода шифрования mdb-файлов MS Access. На основе анализа этого метода предложен усовершенствованный метод защиты с использованием шифрования, позволяющий повысить уровень защиты mdb-файла при сохранении структуры файла и неизменности общего интерфейса работы с mdb-файлами при помощи драйвера MS Jet OLEDB 4.0.

Система управления базами данных (СУБД) MS Access получила широкое распространение во многом из-за простоты использования и настройки. Однако эти преимущества указанной СУБД сказываются на устойчивости работы и безопасности баз данных (БД) [Robinson, 2003]. При этом, что касается устойчивости работы, то БД MS Access в первую очередь хороши для «однопользовательского» режима, то есть режима, при котором база данных представляем собой лишь контейнер для данных со способностью обрабатывать sql-запросы. А проблема безопасности БД до сих пор остаётся актуальной, так как в независимости от режима использования данные, хранящиеся в БД, должны быть защищены.

СУБД MS Access предлагает целый ряд встроенных методов защиты БД [Robinson, 2003]. Однако все из них оказываются так или иначе уязвимыми, когда речь идёт о разграничении доступа к данным. При этом имеем в виду, что БД есть файл операционной системы (ОС) Windows с расширением mdb, который может быть доступен любому (в данной работе не рассматривается вопрос разграничения доступа к файлу на уровне ОС). И в таком случае mdb-файл может быть прочитан с помощью любого средства чтения файлов, вплоть до стандартного текстового редактора. При этом, разумеется, будет неизвестна, например, структура данных, но сами поля данных окажутся доступными. Учитывая это, разработчики MS Access предлагают использовать встроенное в оболочку MS Access шифрование mdb-файлов. При этом само по себе встроенное шифрование защищает только от указанного выше способа доступа к данным, оставляя при этом возможность прочесть данный файл с помощью оболочки MS Access точно так же как и незашифрованный файл. Однако встроенное шифрование может оказаться довольно перспективным способом защиты данных MS Access при его сочетании с

нестандартным способом защиты mdb-файлов MS Access, предлагаемым авторами и описанным ниже. Для этого необходимо сделать анализ встроенного метода шифрования mdb-файлов.

Для анализа встроенного метода шифрования был проведён эксперимент, при котором был создан типовой mdb-файл БД MS Access, далее этот файл был зашифрован с помощью встроенного шифрования MS Access. Опираясь на информацию из бюллетеня MS Access [Microsoft, 2003] о том, что данные mdb-файла шифруются блоками с помощью алгоритма шифрования RC4, при чём каждому блоку ставится в соответствие свой уникальный ключ, длиной 32 бита, эксперимент преследовал следующие цели:

- определение границ и размеров блоков шифрования;
- нахождение соответствий между блоками зашифрованного файла и блоками незашифрованного файла (то есть, какой шифрованный блок соответствует какому незашифрованному блоку);
- вычисление ключей шифрования;
- определение «местоположения» найденных ключей шифрования (то есть смещения в mdb-файле, где хранятся ключи шифрования).

Созданный mdb-файл (база.mdb) имел размер 132 Kb (135168 байт), при этом полученный зашифрованный файл (база_crypted.mdb) также имел размел 132 Kb (135168 байт). Первоначально был проведён «беглый» сравнительный анализ полученных файлов. При сравнении дампа файла база.mdb и дампа база_crypted.mdb для одних и тех же смещений (рис. 1. и рис. 2.) установлено, что файл база_crypted.mdb не содержит в себе какого-либо осмысленного текста и действительно очень похож на массив зашифрованных данных.

Рис. 1. Дамп файла база.mdb

Рис. 2. Дамп файла база_crypted.mdb

Далее при сравнении первых 4096 байт (4Kb) обоих файлов видно, что они отличаются лишь несколькими байтами в самом начале (рис. 3. и рис. 4., подробнее о разнице в заглавии файлов будет сказано ниже).

Рис. 3. Дамп заголовка файла база.mdb

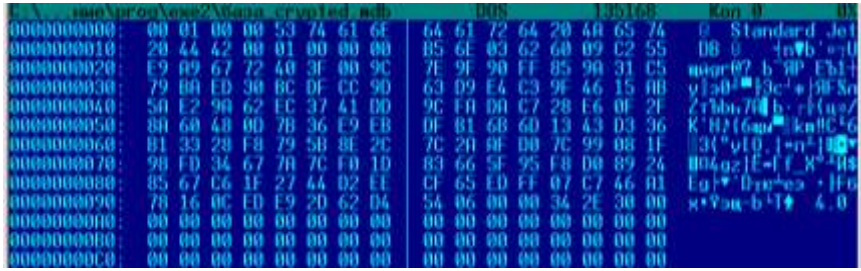


Рис. 4. Дамп заголовка файла база_crypted.mdb

Что же касается дампа сразу после смещения 0x1000[1] (начиная с 4097 байта от начала файла) для обоих файлов, то эти данные резко разнятся. Опираясь на это наблюдение, было сделано предположение, что первый незашифрованный блок данных файла база.mdb начинается именно со смещения 0x1000. Кроме этого было сделано предположение, что этому блоку соответствует блок в зашифрованном файле база_crypted.mdb также начиная со смещения 0x1000.

Для проверки обозначенных предположений был проведён эксперимент, состоящий в поборе ключа для алгоритма шифрования RC4, где шифртекстом является n байт файла база_crypted.mdb начиная со смещения 0x1000, а открытым текстом[2] – n байт файла база.mdb начиная со смещения 0x1000. При этом первоначальный выбор числа n несущественен, так как RC4 является потоковым криптоалгоритмом. Поэтому для увеличения скорости побора n было произвольно выбрано равным 16.

С помощью перебора битовых строк длиной 32 бита данный эксперимент завершился успешно и был найден ключ, значение которого в байтовом представлении есть массив (239,33,230,172). С помощью найденного ключа и перебора числа n установлено, что размер шифртекста и соответствующего ему открытого текста есть 4096 байт. Следующий блок шифртекста длиной 4096 байт, начиная со смещения 0x2000, также соответствует блоку открытого текста, длиной 4096 байт, начиная со смещения 0x2000. При этом найденным ключом является 4-х байтовый массив (236,33,230,172). При сравнении найденных ключей, очевидно, что 3 из 4 байт ключа идентичны, переменным является только первый. Развивая предположение, что данная закономерность распространяется и на остальные ключи, дальнейший перебор ключей

---

[1] *Здесь и далее данная запись есть представление числа в шестнадцатеричном виде.*

[2] *Далее под шифртекстом понимаем блок данных файла база_crypted.mdb, а под открытым текстом понимаем блок данных файла база.mdb.*

состоял лишь в подборе первого байта ключа шифрования (назовём это унифицированным перебором). Таким образом были найдены ключи шифрования до блока начиная со смещения 0xE000 включительно. Однако для блока шифртекста начиная со смещения 0xF000 и соответствующего блока открытого текста с помощью как унифицированного, так полного перебора 32-байтовых ключей искомый не был найден. Отсюда заключение, что порядок следования блоков шифртекста не совпадает с порядком следования блоков открытого текста.

Далее экспериментально необходимо было найти соответствия между блоками шифртекста и блоками открытого текста, а также соответствующие ключи шифрования. С помощью унифицированного перебора ключей и полного перебора блоков шифртекста и блоков открытого текста были найдены соответствия, указанные в таблице 1.

Таблица 1.

Начальное смещение для блока шифртекста в файле база_crypeted.mdb	Начальное смещение для блока открытого текста в файле база.mdb	Ключ шифрования
0x1000	0x1000	239,33,230,172
0x2000	0x2000	236,33,230,172
0x3000	0x3000	237,33,230,172
0x4000	0x4000	234,33,230,172
0x5000	0x5000	235,33,230,172
0x6000	0x6000	232,33,230,172
0x7000	0x7000	233,33,230,172
0x8000	0x8000	230,33,230,172
0x9000	0x9000	231,33,230,172
0xA000	0xA000	228,33,230,172
0xB000	0xB000	229,33,230,172
0xC000	0xC000	226,33,230,172
0xD000	0xD000	227,33,230,172
0xE000	0xE000	224,33,230,172
0xF000	0x11000	225,33,230,172
0x10000	0x12000	254,33,230,172
0x11000	0x14000	255,33,230,172
0x12000	0x15000	252,33,230,172
0x13000	0x16000	253,33,230,172
0x14000	0x1E000	250,33,230,172
0x15000	0x13000	251,33,230,172
0x16000	0x19000	248,33,230,172
0x17000	0x1A000	249,33,230,172

Начальное смещение для блока шифртекста в файле база_crypeted.mdb	Начальное смещение для блока открытого текста в файле база.mdb	Ключ шифрования
0x18000	0x1B000	246,33,230,172
0x19000	0x1C000	247,33,230,172
0x1A000	0x1D000	244,33,230,172
0x1B000	0x17000	245,33,230,172
0x1C000	0x18000	242,33,230,172
0x1D000	0x1F000	243,33,230,172
0x1E000	0xF000	240,33,230,172
0x1F000	0x20000	241,33,230,172
0x20000	0x10000	206,33,230,172

Получив указанные данные, достигнуты 3 из 4 поставленных целей эксперимента. Таким образом, осталось установить место положение ключей шифрования. Возвращаясь к заголовкам шифрованного и незашифрованного файла (рис. 3. и рис. 4) отметим, что они разнятся лишь для байтов, находящихся по смещениям (0x3E, 0x3F, 0x40, 0x41, 0x72, 0x73, 0x74, 0x75). То есть, имеем 2 блока по 4 последовательно идущих байта, первый - (0x3E, 0x3F, 0x40, 0x41) и (0x72, 0x73, 0x74, 0x75). Учитывая, что заголовки разнятся лишь для 8 байтов, а также тот факт, что найденные ключи шифрования отличаются друг от друга одним байтом, утверждаем, что каждый ключ шифрования не хранится где-либо, а формируется из одного ключа, который хранится по указанным смещениям. Таким образом, целью становиться поиск этого ключа, назовём его основным ключом.

Изменяя с помощью любого редактора бинарных файлов какой-либо байт из 2-го блока шифрованного файла и далее открывая этот файл с

Рис 5. Сообщение MS Access об отсутствии разрешения на открытие файла.

помощью MS Access, получаем либо сообщение об отсутствии разрешения на открытие (рис. 5), либо файл корректно открывается. Это свидетельствует о том, что байты из 2-го блока не имеют отношения к основному ключу.

Изменяя же байты из 1-го блока байтов, получаем сообщение о том, что формат базы нераспознаваем (рис. 6). Очевидно, это есть признак

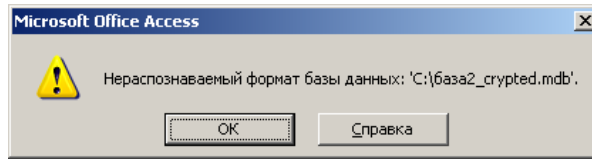


Рис. 6. Сообщение MS Access о нераспознаваемом формате базы.

того, что байты из 1-го блока каким-либо образом формируют основной ключ, и при изменении указанных байтов MS Access не может восстановить нужные ключи и получает ошибочные блоки при расшифровке. Отсюда и сообщение о нераспознаваемом формате данных.

Байты по смещениям 1-го блока представляют собой 4-х байтовую последовательность (21, 171, 90, 226). Сделаем ряд предположений:

- последовательность (21, 171, 90, 226) есть основной ключ, но в зашифрованном виде;
- алгоритмом шифрования есть криптоалгоритм RC4;
- 3 последних байта основного ключа есть (33,230,172) в силу их инвариантности для найденных ключей шифрования.

Данные предположения подтвердились, так методом полного перебора был найден ключ, при котором последовательность (21, 171, 90, 226) при расшифровании представляла собой последовательность (218,33,230,172) – основной ключ шифрования. Найденный ключ шифрования есть последовательность (1,25,149,242), назовём его глобальным ключом. Заметим, что разница между значениями первого байта основного ключа (218,33,230,172) и первого байта ключа шифрования для первого 4096-байтного блока шифрования (239,33,230,172), есть число 21. Очевидно, что полученные ключи шифрования (таблица 1.) формируются на базе основного ключа путем изменения значения его первого байта.

Отметим, что идентичный эксперимент был проведён для другой пары файлов (незашифрованный/зашифрованный), и при этом найденный глобальный ключ оказался таким же, как и для первой пары файлов. Причём разница между значениями первого байта основного ключа и первого байта ключа шифрования для первого 4096-байтного блока шифрования, так же есть число 21. Соответствия блоков шифртекста и блоков открытого текста оказались идентичны данным, указанным в таблице 1.

Подытожив, отметим основные результаты эксперимента:

1. Встроенное шифрование MS Access шифрует файл базы данных в формате mdb блоками, размером по 4Kb каждый с помощью алгоритма шифрования RC4;
2. Месторасположение блоков шифрования в зашифрованном и исходном файлах могут находиться по различным смещениям (см. таблицу 1), однако алгоритм формирования размещения блоков в шифрованном файле инвариантен;
3. Ключи шифрования блоков отличны друг от друга лишь на один байт и формируются на основании одного случайного ключа с помощью статического алгоритма изменения первого байта ключа;
4. Указанный случайный ключ шифрования записывается в заголовок шифрованного файла по смещениям (0x3E, 0x3F, 0x40, 0x41) в зашифрованном виде с помощью алгоритма шифрования RC4;
5. Ключ шифрования основного ключа есть постоянная величина для конкретной инсталляции MS Access, то есть наиболее вероятно храниться в одной из системных библиотек MS Access.

Результаты эксперимента свидетельствуют о том, что для дешифровки mdb-файла, предварительно зашифрованного встроенным алгоритмом шифрования MS Access, достаточно получить глобальный ключ, в нашем случае это последовательность байт (1,25,149,242), а далее процесс дешифрования тривиален. При чём данный ключ может быть получен путём перебора в соответствии с вышеописанной схемой, а перебор 4-байтового ключа не займёт много времени (например, на персональном компьютере это составляет в среднем 7-10 часов). Таким образом, основная слабость заключается в том, что ключ шифрования не задаётся пользователем, а определяется системой, причём этот ключ идентичен для одной и той же инсталляции MS Access. С этой точки зрения способ шифрования mdb-файлов с помощью средств MS Access, безусловно, ненадёжен.

Авторами предложен способ усиления встроенного шифрования MS Access с помощью использования шифрования заголовка mdb-файла. Под заголовком в данном случае понимаем первые 65536 байт файла базы данных. Шифрование осуществляется на основе заданного пользователем симметричного алгоритма и ключа шифрования. При этом сохраняется возможность восстановить исходный заголовок при известном ключе шифрования.

Первоначально при помощи специально созданного windows-приложения заголовок mdb-файла шифруется (согласно рис. 7) на основе

заданного симметричного алгоритма шифрования и секретного ключа. Ключ шифрования передаётся легитимному пользователю.

Специально созданное windows-приложение для чтения данных mdb-файла с зашифрованным заголовком (далее программа) работает по следующей схеме. Windows-приложения для работы с файлами базы данных MS Access в формате mdb используют драйвер Microsoft Jet OLEDB 4.0. При заданных параметрах соединения с mdb-файлом производиться подключение, происходит чтение первых 65536 байт файла данных. В данный момент срабатывает защита. Программа перехватывает вызов API функции ReadFile, сама считывает заголовок БД, и возвращает данные, расшифрованные на основе заданного алгоритма шифрования и ключа шифрования. В случае успешной расшифровки программа продолжает корректно работать с данными mdb-файла, иначе происходит сбой, сигнализирующий о неверной расшифровке.

Для перехвата вызова API функции ReadFile используется локальный перехват с использованием раздела импорта. Локальный перехват реализован посредством подмены адреса перехватываемой функции в таблице импорта. Изменяется адрес импорта функции ReadFile во всех модулях процесса (процесс может вызывать её не только из exe-модуля, но и из dll-модулей). В нашем случае функция ReadFile вызывается из модуля msjet40.dll – библиотека драйвера MS Jet OLEDB 4.0. Адрес импорта необходимо изменить только в этом модуле.

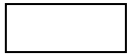
После замены адреса библиотека msjet40.dll при чтении будет вызывать новую функцию, назовем её SpecialReadFile. При этом необходимо сначала вызвать оригинальную функцию ReadFile, предварительно запомнив её адрес в модуле kernel32.dll. После этого вызова ReadFile возвратит буфер с зашифрованными данными, размером 64Кб. Далее эти данные расшифровываются и записываются по адресу этого же буфера. После этого управление возвращается вызывающему модулю. Таким образом, происходит расшифрование налету.

В качестве симметричного алгоритма шифрования предложено использовать алгоритм Blowfish, который обладает высокой стойкостью и позволяет задавать ключи длиной до 448 бит.

При перехвате вызова ReadFile факт чтения именно заголовка mdb-файла определяем на основе анализа размера буфера для чтения. Если размер буфера составляет 65536 байт, то происходит чтение данных именно заголовка. В этот момент и происходит расшифровка. Кроме этого библиотека msjet периодически (период порядка 10 секунд) считывает 512 байт с 3584 байта от начала файла. Таким образом, предварительно отдельно шифруется блок длиной 3584 байта от начала файла до смещения 3583, и блок, начиная со смещения 4096 до 65536. А блок длиной 512 байт оставляем незашифрованным (Рис. 7).

Рис. 7. Структура зашифрованного заголовка mdb-файла.

Где  –данные, подвергающиеся шифрованию/расшифрованию,

 – данные, не подлежащие шифрованию/расшифрованию.

Представленный подход связан с трудностью определения текущего положения курсора в БД. То есть, неизвестно, какие именно 512 байт необходимо прочитать: те, что являются частью заголовка или это байты, лежащие после заголовка БД.

Предлагаемый способ защиты mdb-файлов СУБД Microsoft Access позволяет повысить уровень защиты mdb-файла при сохранении структуры файла и неизменности общего интерфейса работы с mdb-файлами при помощи драйвера MS Jet OLEDB 4.0. При совмещении предложенного способа с встроенным шифрованием MS Access у злоумышленника, однако, остаётся возможность расшифровать данные лежащие за пределами зашифрованного заголовка по схеме описанного выше эксперимента. Но полученные таким образом блоки данных не позволят полностью восстановить структуру базы и получить данные в полном объёме. Что же касается дешифрования самого заголовка, то при использовании, например, 32-битного ключа это не представляется возможным, так как стойкость используемого алгоритма симметричного шифрования имеет доказанную и обоснованную стойкость, позволяющую полагаться на его надёжность. Кроме этого при синтезе методов полученный файл будет полностью зашифрованным, и по его виду невозможно определить является ли файл mdb-файлом базы данных MS Access. Таким образом, сохранение в секрете не только ключа шифрования заголовка, но и всей процедуры защиты mdb-файла также повышает уровень безопасности содержащихся в файле данных.

Список литературы

[Robinson, 2003] Robinson G. Real World Microsoft Access Database Protection and Security. – Berkeley, CA.: Apress, 2003.

[**Microsoft, 2003**] ACC: How Microsoft Access Uses Encryption. Microsoft Knowledge Base Article No. 140406, May 2003.