Математические структуры и моделирование
2013. № 2(28). С. 88–102

УДК 004.056.52

РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА ДЛЯ АНАЛИЗА БЕЗОПАСНОСТИ ДОСТУПА К ФАЙЛАМ В ОПЕРАЦИОННЫХ СИСТЕМАХ WINDOWS 7

Д.М. Бречка

В статье рассматривается процесс разработки программного продукта для анализа доступа к объектам файловой системы Windows 7 на основе модели безопасности Take-Grant. Описывается внутренняя организация программного продукта, его функционал и тестирование производительности.

Введение

Нормативные документы по информационной безопасности [1, 2] требуют использования механизмов дискреционного разграничения доступов в современных компьютерных системах, данное требование применимо и к операционным системам. Однако наличие механизма дискреционного разделения доступа в операционной системе не гарантирует защищённость системы от несанкционированного доступа, так как настройка доступа выполняется пользователями системы, в результате чего могут появляться ошибки администрирования. Таким образом, возникает потребность в разработке методов оценки корректности работы механизмов дискреционного разделения доступа в операционных системах.

На сегодняшний день существует ряд формальных моделей дискреционного разделения доступа [3]. Оценивать корректность работы дискреционного механизма в операционной системе разумно на основе какой-либо формальной модели. Однако политика безопасности в конкретной операционной системе может не полностью соответствовать классической модели безопасности. В этом случае возникает проблема адаптации классической модели к политике, применяемой в конкретной системе. В работе [4] описано исследование возможности применения дискреционной модели безопасности Take-Grant для анализа доступа к объектам операционных систем Windows 7 и Ubuntu Linux 10.4.

Целью данной работы является описание процесса разработки и тестирования программного продукта для анализа доступа к объектам файловой системы Microsoft Windows 7 на основе модели Take-Grant.

Омский государственный университет им. Ф.М. Достоевского
E-mail: dbrechkawork@yandex.ru
Работа выполнена при поддержке «ИнфоТеКС Академия»

1. Получение доступа к системным структурам в операционных системах Windows 7

Операционные системы Windows предоставляют широкий набор инструментов для администрирования компьютера. Одним из наиболее удобных инструментов администрирования, доступных для пользовательских программ, является Windows Management Instrumentation (WMI). Windows Management Instrumentation — это технология управления и слежения за работой различных частей компьютерной системы под управлением Windows [5]. В основе WMI лежит объектно-ориентированный подход к представлению компонентов системы. Поля объектов, предоставляемых WMI, являются динамическими, это означает, что значения этих полей нигде не хранятся, а создаются по запросу пользователя. Все классы сгруппированы в пространства имён, которые иерархически упорядочены, то есть допускают наличие классов-потомков и классов-родителей.

WMI предоставляет большое количество классов для доступа к информации о компьютерной системе. Так, например, для доступа к учётным записям пользователей можно воспользоваться классом `Win32_UserAccount`, а для доступа к информации о группах пользователей – `Win32_Group`. Существует ряд классов для доступа к объектам файловой системы, например `Win32_LogicalDisk` предоставляет информацию о логических дисках, а `Win32_Directoty` — о папке. Для доступа к правам пользователя на файл можно использовать класс FileSystemAccessRule [5].

Для обращения к объектам WMI применяется специальный язык запросов — WMI Query Language (WQL), который является разновидностью SQL. Использование WQL позволяет работать с объектами WMI как с таблицами баз данных, что может быть удобным как при обращении к объектам из пользовательских программ, так и из внешних скриптовых оболочек, например Windows PowerShell [5].

2. Разработка проекта программного продукта для анализа доступа к объектам файловой системы

Основной задачей разрабатываемого программного продукта является обнаружение ошибок распределения прав доступа к пользовательским файлам. Наличие таких ошибок может стать причиной появления несанкционированного доступа. Настройкой прав доступа может заниматься администратор системы и владелец файла. Причины появления ошибок настройки прав могут быть следующими:

1) некоторые пользователи могут являться членами групп, обладающих административными правами, и, вследствие этого, получать доступ к файлам как члены административных групп;

2) права пользователя на файл могут быть унаследованы от директории, в которой находится файл;

3) при создании файла пользователь применяет политику по умолчанию, которая может не соответствовать ожиданиям пользователя.

Таким образом, для выполнения своей основной функции разрабатываемая программа должна располагать сведениями обо всех пользователях компьютерной системы и группах пользователей, а также обо всех файлах на текущем компьютере. Результатом работы программы должен являться отчёт, содержащий сведения о том, какие права имеет определённый пользователь на определённый файл и по каким причинам он имеет эти права. Исходя из сказанного, можно определить минимальный набор режимов работы программы:

1) просмотр имён всех учётных записей пользователей, зарегистрированных на данном компьютере;

2) просмотр всех пользовательских групп на данном компьютере;

3) просмотр файлов, хранящихся на данном компьютере (в указанной директории, на указанном диске или всех файлов на компьютере);

4) проверка прав доступа конкретного пользователя (всех пользователей) на конкретный файл или на группу файлов (файлы в указанной директории).

Согласно принципу декомпозиции [6] следует разделить поставленную задачу на более мелкие части, чтобы упростить процесс ее решения. Разрабатываемый программный продукт можно разделить, как минимум, на три модуля или объекта. Первый модуль будет служить для сбора информации о субъектах и объектах компьютерной системы, на которой запущена программа. Второй модуль будет выполнять все действия по анализу доступа субъектов к объектам. Третий модуль будет предоставлять пользовательский интерфейс.

Для более наглядного представления решаемой задачи и уменьшения количества связей между отдельными частями программы предлагается избрать объектно-ориентированный подход к разработке программ [6]. В этом случае выделенные программные модули могут быть представлены в виде объектов, а отдельные задачи, решаемые этими объектами — в виде методов объектов. В качестве языка программирования для разработки программного продукта был выбран язык C#, который является удобным объектно-ориентированным языком.

3. Разработка класса для анализа доступа

Удобный инструмент для анализа доступа в компьютерной системе предоставляет модель безопасности Take-Grant [3,7–9]. Данная модель представляет компьютерную систему в виде графа доступов, вершинами которого являются субъекты и объекты системы, а дугами — права субъектов на объекты. Для рассматриваемой задачи субъектами графа будут являться пользователи (пользовательские учётные записи), а объектами — пользовательские файлы. Таким

образом, очевидно, что в программе потребуется два типа вершин. Помимо типа в вершине графа полезно хранить дополнительную информацию о вершине такую как: номер вершины в графе и имя субъекта или объекта, представленного данной вершиной. Кроме того, для удобства работы в программе вершины-объекты и вершины-субъекты нумеруются отдельно. В связи со сказанным, для хранения информации о вершине удобно будет создать специальную структуру, содержащую перечисленные поля.

```
struct Vertex {
  public string name; // имя вершины
  public string type; // тип вершины
  public Int64 number; // номер вершины данного типа
  public Int64 ID; // номер вершины в графе
}
```

Существует несколько стандартных способов программного представления графов [10], для разрабатываемой программы был избран способ представления графа в виде матрицы смежности. Столбцами и строками такой матрицы являются вершины графа, а в ячейках находятся веса путей между смежными вершинами. В нашем случае в ячейках матрицы удобно расположить права доступа, которые, в свою очередь, удобно представить в виде битовой строки, где каждый бит идентифицирует наличие или отсутствие определённого права. Так как в разных операционных системах может быть различное количество прав доступа, то и размер битовой строки для разных операционных систем будет разным. Поэтому удобно создать в классе специальное константное поле (BN) для хранения размера битовой строки. Для работы с битовыми строками в языке `C#` удобно использовать класс BitArray [5], тогда массив, задающий матрицу смежности, может быть объявлен следующим образом:

```
BitArray[,] a_matrix.
```

Для хранения списка вершин-субъектов и вершин-объектов можно создать два динамических массива типа Vertex. Также удобно завести отдельные поля в классе для хранения количества субъектов и объектов, во избежание необходимости производить их подсчёт. Таким образом, к полям класса будут относиться следующие переменные:

```
static int BN; // количество битов в строке (количество прав доступа)
int IDs; // переменная для подсчёта общего числа вершин в графе
BitArray[,] a_matrix; //матрица смежности графа
public int S; // количество субъектов в графе
public int O; // количество объектов в графе
public Vertex[] Subjects; // список субъектов в графе
public Vertex[] Objects; // список объектов в графе.
```

Для работы с графом доступов, очевидно, понадобятся операции добавления и удаления субъектов и объектов, а также операции добавления и удаления дуг.

При добавлении субъекта или объекта необходимо увеличить общее число вершин в графе, а также увеличить количество вершин данного типа. Для унификации работы с матрицей смежности в программе используется следующее соглашение: в матрице смежности записи о вершинах-субъектах располагают-

ся до записей о вершинах-объектах. Следовательно, при добавлении субъекта необходимо добавить столбец (строку) в матрицу после всех столбцов (строк), соответствующих субъектам, но до столбцов (строк), соответствующих объектам. При добавлении объекта в матрицу добавляется последний столбец и последняя строка.

Алгоритм добавления субъектов в матрицу следующий:

1) создаётся временная матрица размером $(S + O + 1) \times (S + O + 1)$, где S и O — соответственно количество субъектов и объектов в графе доступа;

2) во временную матрицу копируются значения исходной матрицы до тех пор, пока не встретится последняя строка (столбец), соответствующая субъекту;

3) новый субъект пока не имеет связей с другими вершинами графа, поэтому ячейки новой строки и столбца остаются пустыми;

4) после нового субъекта во временную матрицу копируются столбцы и строки, соответствующие объектам;

5) в исходной матрице увеличивается количество строк и столбцов;

6) значения из временной матрицы копируются в исходную.

Добавление объекта происходит аналогично, но новые данные добавляются в конец матрицы.

При удалении субъекта или объекта уменьшается количество вершин данного типа. При изменении матрицы смежности необходимо удалить столбец и строку, соответствующую данной вершине. Алгоритм удаления вершины из матрицы следующий:

1) создаётся временная матрица размером $(S + O - 1) \times (S + O - 1)$, где S и O — соответственно количество субъектов и объектов в графе доступа;

2) во временную матрицу копируются значения исходной матрицы до тех пор, пока не встретится строка (столбец), соответствующая удаляемой вершине;

3) дальнейшее копирование элементов происходит со сдвигом на одну ячейку вперёд, то есть в текущую ячейку временной матрицы копируется значение следующей ячейки исходной матрицы;

4) в исходной матрице уменьшается количество строк и столбцов;

5) значения из временной матрицы копируются в исходную.

Добавление дуги производится простым копированием значения в соответствующую ячейку матрицы смежности, при этом в начале проверяется, существуют ли в графе вершины, между которыми необходимо добавить дугу. Удаление дуги происходит аналогично — путём удаления значений.

Для удобства работы с графом доступа добавлены методы сохранения графа в файл и загрузки графа из файла. Информация о графе хранится в текстовом файле со следующей структурой: в начале файла сохраняется количество субъектов, объектов и общее число вершин в графе. Затем сохраняется количество дуг графа. Далее следует список всех субъектов со значениями всех полей субъектов. Следом за субъектами таким же образом сохраняются объекты. Последним элементом в файле сохраняется матрица смежности графа.

Помимо описанных методов в классе присутствует несколько вспомогательных методов: методы copybits — для копирования битовых строк, метод addSpaces — для добавления пробелов при выводе матрицы смежности, метод showAdjacencyMatrix и метод showMask — для вывода соответственно матрицы смежности и значения ячейки матрицы смежности в строку, которую в дальнейшем можно выводить на экран или в файл.

Структура описанного класса изображена на рисунке 1.

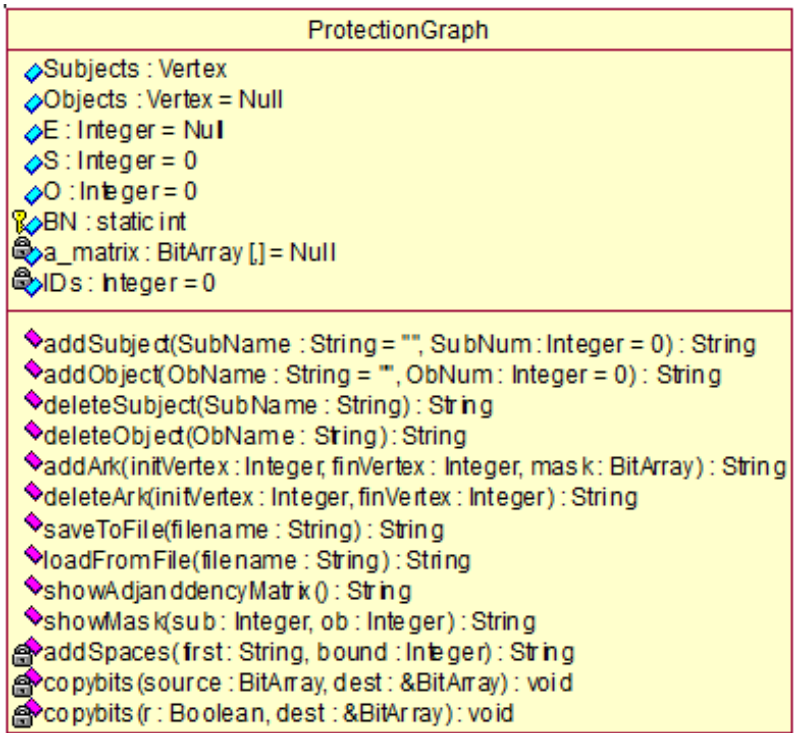

Рис. 1. Структура класса ProtectionGraph в нотации UML

4. Разработка класса для сбора системной информации

При анализе доступа может потребоваться следующая информация о пользователе: имя пользователя, системный идентификатор пользователя (SID), домен пользователя и список групп, в которые входит пользователь. В свою очередь, может потребоваться следующая информация о группе: имя группы и

системный идентификатор группы (SID). Для хранения информации о группах и пользователях в программном продукте используются следующие структуры:

```
struct Group {
  public string Name;
  public string SID;
}
struct User
  public string Name;
  public string SID;
  public string Domain;
  public List<Group> UGroups;
}
```

Поле UGroups структуры User хранит список групп, в которые входит пользователь. Это поле имеет тип List, предоставляемый платформой Microsoft .NET Framework.

Для хранения информации о файле в программном продукте используется следующая структура:

```
public struct SObject {
  public Int64 ID;
  public string name;
}
```

Как видно, в программном продукте хранится идентификатор файла и его имя. Для хранения списков пользователей, групп и файлов в классе предусмотрены следующие поля:

```
List<Group> Groups; // список групп
List<User> Users; // список пользователей
List<SObject> SObjects; // список файлов.
```

Кроме того, класс имеет статическое BN поле, в котором хранится размер битовой строки для представления прав доступа. Для Windows 7 значение этого поля равно 23.

Для получения списка пользователей системы с перечислением всех групп, в которые входят пользователи, применяется метод getSubjects. Данный метод не имеет параметров. В начале метод получает список всех групп и список всех пользователей в системе с помощью WMI-объектов `Win32_Group` и `Win32_UserAccount` соответственно, информация о пользователях и группах сохраняется в полях Groups и Users. Затем метод перебирает всех пользователей и все группы, чтобы выяснить членство пользователей в группах. Для проверки членства пользователя в группе используется метод checkMembership, принимающий на вход имя домена, имя группы и имя пользователя. Метод checkMembership возвращает значение «истина» либо «ложь», в зависимости от того, является указанный пользователь членом указанной группы или нет.

Для получения списка файлов в указанной пользователем директории используется метод getObjects, принимающий на вход имя директории. Этот метод для рекурсивного обхода каталогов запускает метод TraverseTree. TraverseTree обходит подкаталоги и сохраняет информацию о найденных фай-

лах в поле SObjects. Для доступа к информации о файлах TraverseTree использует объект System.IO.FileInfo из .NET Framework.

Для создания матрицы смежности графа используется метод createMatrix. Существует два варианта этого метода. Первый вариант метода просто создаёт матрицу смежности, второй вариант создаёт матрицу и возвращает эту матрицу по ссылке в процедуру, вызвавшую метод. Метод createMatrix создаёт новую матрицу размером $(S + O) \times (S + O)$, где S и O — соответственно количество субъектов и объектов в системе, и в каждую ячейку матрицы копирует либо права субъекта на объект, либо пустую битовую строку, если прав нет. Для копирования битовых строк используется вспомогательный метод cpbits.

Для выяснения прав субъекта на объект метод createMatrix вызывает метод getMask. Метод getMask принимает в качестве параметра объект типа SObject и субъект типа User. Метод перебирает все ACE из DACL данного файла и выясняет, есть ли в DACL записи, соответствующие группам данного пользователя или индивидуальные записи для данного пользователя. Если такие записи находятся, то вызывается метод createMask, который формирует списки запрещённых и разрешённых действий для данного пользователя над данным объектом. Списки действий возвращаются методом createMask в виде битовых строк. Учитывая то, что явно запрещённые действия имеют более высокий приоритет, чем разрешённые, getMask формирует итоговую битовую строку, которая должна быть записана в матрицу смежности. Итоговая строка возвращается по ссылке методу createMatrix. Для доступа к DACL файла метод getMask использует объекты типа File и FileSecurity, предоставляемые .NET Framework.

Метод createMask принимает в качестве параметра объект типа FileSystemAccessRule, и, в зависимости от типа права (разрешающее или запрещающее), формирует выходные битовые строки, где каждое право, доступное пользователю расположено в определённом разряде битовой строки. Например, если какое-либо право явно разрешено пользователю, то в разрешающей ботовой строке бит, соответствующий этому праву, устанавливается. Аналогично формируется строка запрещающих прав. Особенным образом обрабатывается право FullControll. Если такое право встречается в строке, то все биты строки устанавливаются.

Для создания пустой битовой строки (если между субъектом и объектом не может быть прав) предусмотрен перегруженный вариант метода getMask. Для вывода списка пользователей, групп и файлов в классе предусмотрены соответствующие методы: listUsers, listGroups, listFiles.

Описание класса в нотации UML приведено на рисунке 2.

5. Разработка класса для предоставления пользовательского интерфейса

Пользовательский интерфейс предоставляется классом Program. Данный класс имеет два метода. Метод parseMask преобразует битовую строку, представляющую набор прав субъекта на объект, в строку, содержащую названия

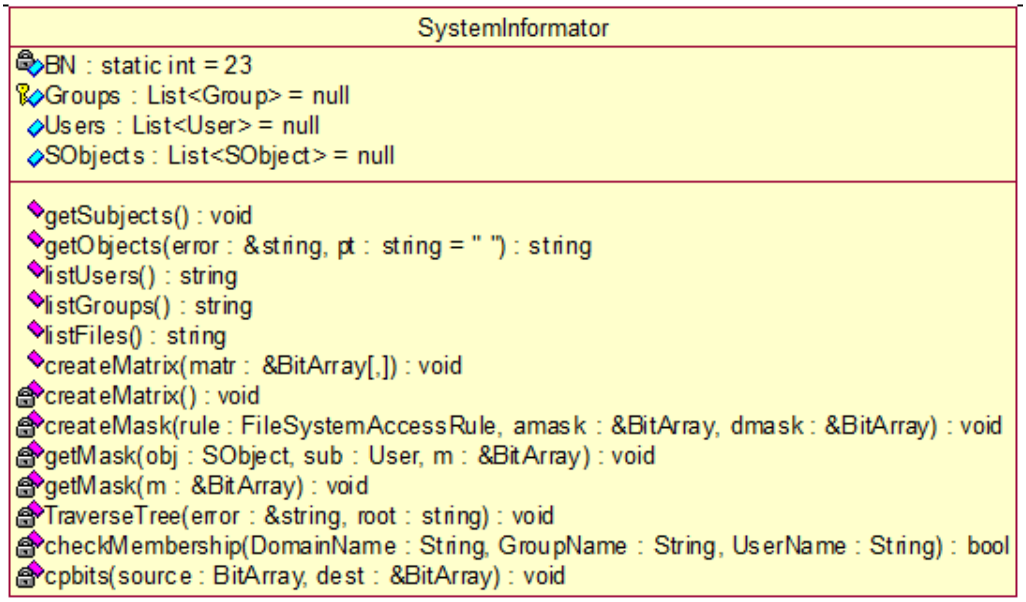

Рис. 2. Структура класса SystemInformator в нотации UML

прав для вывода их на экран. Метод Main предоставляет консольный пользовательский интерфейс, обеспечивающий возможность доступа ко всем перечисленным выше режимам работы программы. При запуске программы метод Main предлагает пользователю ввести директорию, для которой будет производиться анализ доступа. Пользователь может не вводить директорию, тогда программа будет анализировать доступ ко всем локальным файлам компьютера. Для анализа доступа строится граф доступов, вершинами-субъектами в котором являются пользователи системы, а вершинами-объектами файлы из директории, указанной пользователем (либо все локальные файлы).

Метод Main предоставляет пользователю меню, в котором имеется возможность выбрать следующие действия:

1) просмотр списка пользователей;

2) просмотр списка групп;

3) просмотр списка файлов;

4) проверка доступа;

5) вызов справки.

Первые три действия выполняются посредством вызова специальных методов объекта класса SystemInformator, описание которого приведено выше. Проверка прав доступа производится при помощи вызова метода showMask объекта класса ProtectionGraph для заданных строки и столбца. Перед выводом на экран маска преобразуется из битовой строки в список доступных прав

при помощи метода parseMask объекта Program. Вывод прав доступа может быть осуществлён в следующих режимах:

1) вывод прав конкретного пользователя на конкретный файл;

2) вывод прав конкретного пользователя на все файлы;

3) вывод прав всех пользователей на конкретный файл;

4) вывод прав всех пользователей ко всем файлам.

Кроме того, у пользователя есть возможность перенаправить вывод запрошенной информации в файл.

Программа выводит информацию о доступе к файлу в следующем виде: в первой строке выводится имя пользователя, во второй — имя файла, далее следует список действий, доступных для данного пользователя.

Структура класса Program представлена на рисунке 3.

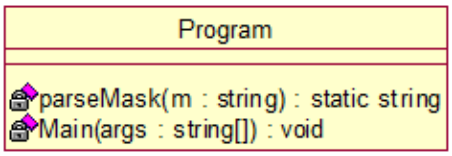

Рис. 3. Структура класса Program в нотации UML

6. Оценка производительности программного продукта

Наиболее трудоёмкими действиями, выполняемыми программным продуктом, являются: сбор сведений о субъектах системы (пользователях и группах), сбор сведений об объектах системы, выявление доступов субъектов к объектам и построение матрицы смежности. Время, затрачиваемое на анализ возможности доступа, будет зависеть от количества субъектов и объектов в системе, то есть от размера матрицы смежности.

Сбор сведений о субъектах системы включает следующие этапы:

1) при помощи системных вызовов формируются списки групп и пользователей, зарегистрированных в системе, эти списки сохраняются во внутренних структурах программы;

2) выясняется членство пользователей в группах, для этого перебираются все пользователи, группы и записи о пользователях в каждой группе, формируется список групп каждого пользователя.

Как видно, системные вызовы применяются только на первом этапе. Пусть для получения информации об одной группе требуется A единиц времени, а для получения информации об одной записи пользователя требуется B единиц

времени. Пусть в системе зарегистрировано N пользователей и M групп, тогда время завершения первого этапа сбора системной информации можно оценить как

$$T_{s_1} = AM + BN.$$

Для выяснения членства пользователя в группе необходимо проверить все записи в текущей группе и сравнить их идентификаторы с идентификатором текущего пользователя. Пусть на операцию сравнения затрачивается C единиц времени, тогда время на выяснение членства пользователя в группе вычисляется как

$$T_{s_2} = \sum_{i=1}^{M} u_i NC,$$

где u_i — количество пользователей в группе i. Тогда общее время этапа сбора информации о субъектах системы оценивается как

$$T_s = T_{s_1} + T_{s_2} = AM + BN + \sum_{i=1}^{M} u_i NC.$$

Сбор сведений об объектах системы происходит путём рекурсивного обхода каталогов, алгоритм обхода таков:

1) корень каталога заносится в стек;

2) с помощью системных вызовов определяется список подкаталогов и файлов в текущем каталоге;

3) данные каждого файла копируются в системную структуру программы;

4) каждый подкаталог заносится в стек;

5) условием выход из рекурсии является отсутствие записей в стеке.

Пусть на операции занесения и возврата из стека совокупно тратится W единиц времени, время получения списка подкаталогов равно X, а время получения списка файлов равно Y. Пусть также Z — время получения информации об одном файле. Тогда время получения списка объектов можно оценить по формуле:

$$T_o = l(X + Y + W \sum_{i=1}^{l} k_i + Z \sum_{i=1}^{l} f_i),$$

где l — глубина вложенности, k_i и f_i соответственно количество каталогов и файлов на i-м уровне дерева каталогов.

Выявление прав доступа субъекта на объект выполняется по следующему алгоритму:

1) просматривается список доступа объекта и выявляются записи, соответствующие группам пользователя;

2) если найдена запись, соответствующая группе пользователя, то права, регламентируемые этой записью, сохраняются во внутренних структурах программы;

3) просматривается список доступа объекта и выявляются записи, соответствующие пользователю;

4) если найдена запись, соответствующая пользователю, то права, регламентируемые этой записью, сохраняются во внутренних структурах программы;

5) после выполнения шагов 1–4 программа формирует списки разрешённых и запрещённых действий, после чего из полученных списков получается единый список прав в виде битовой строки.

Положим, l — длина списка i-го файла, e — количество записей о группе j в списке i, h — количество записей о текущем пользователе в списке i, p — количество групп, в которые входит пользователь. Тогда время выявления прав доступа текущего пользователя к текущему файлу:

$$T_d = Kl(pe + h) + E,$$

где K — время сохранения прав на объект во внутренней структуре программы, E — время формирования общего списка прав.

Формирование матрицы смежности происходит путём копирования маски доступа в ячейку матрицы. Пусть в системе S субъектов, O объектов и время копирования битовой строки занимает τ единиц времени, тогда время создания матрицы смежности:

$$T_m = \tau T_d (S + O)^2.$$

Общее время сбора информации о системе и формирования внутренних структур программы можно вычислить как:

$$T_{SI} = T_s + T_o + T_m.$$

После формирования внутренних программных структур время получения информации о правах доступа пользователя i на объект j будет определяться временем копирования битовой строки из матрицы смежности, разбора строки и вывода информации на экран. Обозначим это время через T_{ij}. Тогда время получения информации о доступе всех пользователей к объекту j будет определяться как:

$$T_{sj} = ST_{ij},$$

где S — количество субъектов в системе. Время получения информации о доступе пользователя i ко всем объектам системы:

$$T_{io} = OT_{ij},$$

где O — количество объектов в системе.

Время получения информации о доступе всех пользователей ко всем объектам:

$$T_{so} = T_{sj} + T_{io} = T_{ij}(S + O).$$

7. Тестирование производительности программного продукта

Как видно из предыдущего раздела, скорость работы программного продукта зависит от множества параметров. Для сокращения объёма данной статьи приведём зависимости скорости работы продукта только от ключевых параметров. Для текущего тестирования количество групп и пользователей не менялось, не менялось также членство пользователей в группах. Основной целью тестирования являлось выявление зависимости скорости работы программного продукта от количества объектов системы.

Параметры вычислительной системы, на которой производилось тестирование программного продукта, приведены в таблице 1.

Таблица 1. Характеристики вычислительной системы для тестирования программного продукта

Открытый текст Модель процессора	AMD Athlon 64 x2 Dual-Core 6000
Частота процессора	3,01 МГц
Объем физической памяти	2048 Мбайт
Скорость доступа к жёсткому диску	67 Мбайт/с
Операционная система	Microsoft Windows 7 Профессиональная 64 бит
Версия Microsoft .NET	4.5
Количество пользователей	5
Количество пользовательских групп	20

Среднее время сбора информации о субъектах системы после 10 запусков программы составило 18,4 с.

Зависимость времени сбора сведений о файлах от количества файлов представлена на рисунке 4. При составлении диаграммы изменялось количество файлов в одноуровневой директории, вложенные директории отсутствовали.

Зависимость времени построения матрицы смежности от количества файлов приведена на рисунке 5.

Из приведённых диаграмм видно, что, если пренебречь погрешностью, то время сбора сведений о файлах растёт практически линейно в зависимости от количества файлов. Также видно, что время построения матрицы смежности для графа доступа имеет экспоненциальную зависимость от количества файлов. Таким образом, экспериментальные данные подтверждают справедливость теоретической оценки производительности, описанной в предыдущем разделе.

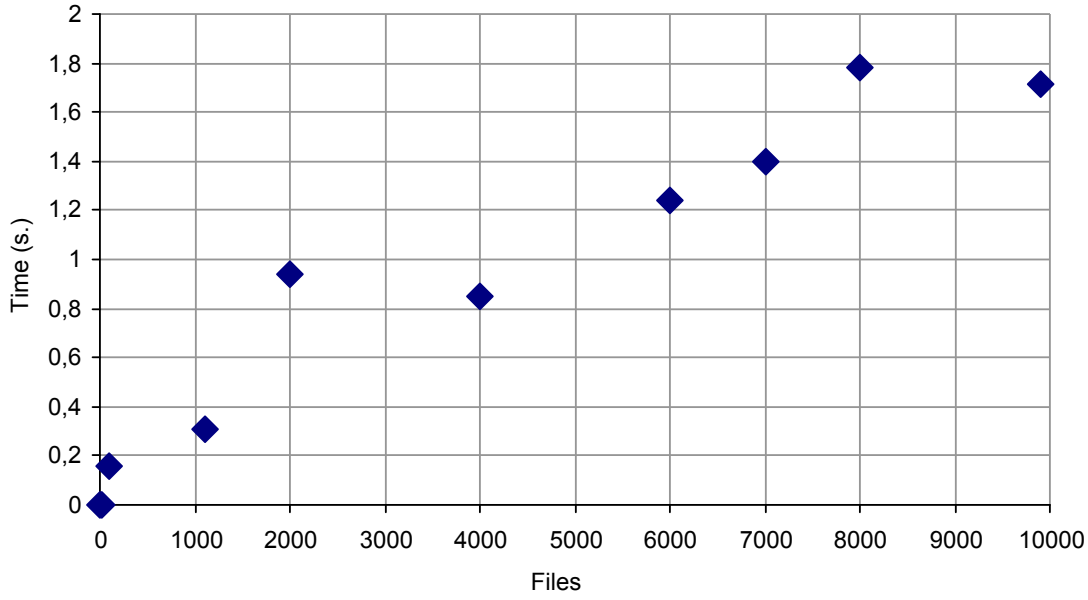

Рис. 4. Диаграмма зависимости времени сбора сведений о файлах от количества файлов

Заключение

В данной работе описывается процесс разработки и тестирования программного продукта для анализа доступа к объектам файловой системы Windows 7. Работа является продолжением исследований в области применения дискреционной модели безопасности Take-Grant для анализа защищённости компьютерных систем от несанкционированных доступов. Основные результаты текущей работы следующие:

1) разработан программный продукт для анализа доступа к объектам файловой системы Windows 7;
2) проведена оценка производительности разработанного программного продукта;
3) проведено тестирование производительности разработанного программного продукта на локальном компьютере под управлением Windows 7.

Тестирование программного продукта показало его пригодность для использования в локальных системах с относительно небольшим количеством файлов.

Продолжение исследований в данной области планируется в следующих направлениях:

1) разработка программного продукта для анализа доступа к прочим объектам Windows 7;
2) анализ применимости модели Take-Grant для исследования безопасности компьютерных сетей;
3) анализ применимости модели Take-Grant для исследования безопасности систем управления базами данных.

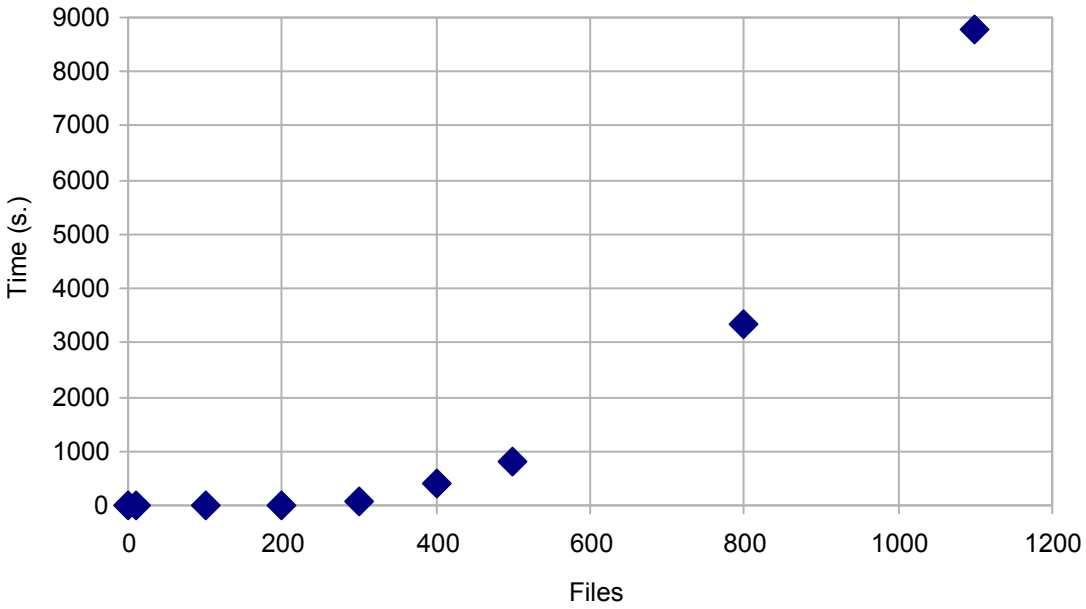

Рис. 5. Диаграмма зависимости времени построения матрицы смежности от количества файлов

4) анализ применимости модели Take-Grant для исследования безопасности мобильных операционных систем.

Литература

1. Common Criteria for Information Technology Security Evaluation (CCEB). Version 1.0. 96.01.31.
2. Trusted Computer System Evaluation Criteria (TCSEC), US DoD 5200.28-STD, 1983.
3. Девянин П.Н. Модели безопасности компьютерных систем: Учебное пособие для студентов высших учебных заведений. М. : Издательский центр «Академия», 2005.
4. Проблемы обработки и защиты информации. Книга 3. Модели разграничения доступа / Белим С.В., Бардычев В.Ю., Бречка Д.М. и др.; под редакцией С.В. Белима Омск : «ООО Полиграфический центр КАН», 2013.
5. Microsoft Developer Network. URL: http://msdn.microsoft.com (дата обращения 16.01.2013).
6. Лафоре Р. Объектно-ориентированное программирование в C++ 4-е изд. СПб. : Питер, 2004.
7. Jones A., Lipton R., Snyder L. A Linear Time Algorithm for Deciding Subject Security // Journal of the ACM (Addison-Wesley). 1977. N. 3. P. 455–464.
8. Snyder L. Theft and conspiracy in the Take-Grant protection model // Journal of Computer and System Sciences. Dec 1981. 23(3):333—347.
9. Frank J., Bishop M., Extending the Take-Grant protection system. Technical report. Department of Computer science, University of California in Davis, 1996. 14 p.
10. Вирт Н. Алгоритмы и структуры данных. Пер. с англ. М. : Мир, 1989.