САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-Механический факультет
Кафедра системного программирования

Лозов Петр Алексеевич

Метрика в пространстве портретов процессов

Курсовая работа студента 371 группы

Научный руководитель:
старший преподаватель
Баклановский Максим Викторович

Санкт-Петербург
2015

Содержание

Введение

В настоящее время компьютеры используются как в повседневной жизни, так и во многих других сферах. Неудивительно, что вопросы компьютерной безопасности сейчас весьма актуальны, ведь наравне с информационными технологиями развивается и вредоносное программное обеспечение.

Задачу обнаружения файлов, содержащих вредоносный код, решают сигнатурные антивирусные программы.

Однако далеко не всегда вредоносный код содержится файлах. Внедрить его можно также и в процесс. Причем файл, содержащий программу, которую исполняет данный процесс, изменён не был, а значит, сигнатурная антивирусная программа такую активность не обнаружит. Зато эту активность обнаружит CODA[1].

Проект CODA

CODA – это система противодействия вредоносным программам, разрабатываемая с 2009 года на кафедре системного программирования СПбГУ.

В отличие от большинства подобных систем, CODA анализирует процессы, а не файлы. Неформально говоря, CODA запоминает поведение разрешенных процессов и в дальнейшем сравнивает поведение каждого текущего процесса со всеми разрешенными. Чем меньше общего у этих поведений, тем меньше свобод предоставляется данному процессу, вплоть до его уничтожения.

Рассмотрим более подробно, как CODA запоминает поведение разрешенных процессов. Каждый поток каждого процесса с течением времени совершает системные вызовы. Последовательность этих системных вызовов будем называть **следом потока**. Заметим, что каждый системный вызов однозначно описывается своим уникальным номером, поэтому след

можно считать строкой, состоящих из номеров системных вызовов. **Шаблонами процесса** назовём подстроки, которые встречаются в следе какого-либо потока процесса более одного раза. Заметим, что шаблоны могут пересекаться между собой, включаться один в другой, а также являться результатом конкатенации нескольких других шаблонов.

Анализируя следы всех потоков процесса, CODA формирует **портрет процесса –** некоторое подмножество достаточно длинных и достаточно часто встречающихся шаблонов процесса. Именно портрет процесса и выступает в качестве описания поведения процесса.

На данный момент в проекте CODA реализована следующая функциональность[2]:

- сбор системных вызовов и построение следов потоков всех процессов;
- построение базы портретов процессов;
- идентификация процесса по базе портретов в реальном времени.

Другими словами, CODA сравнивает портреты с процессами. О степени сходства портретов между собой CODA ничего сказать не может. Поэтому, в случае добавления новых портретов в существующую базу, могут возникнуть дубликаты: различные портреты одного процесса. В данном случае необходимо выявить эти дубликаты и на их основе построить общий портрет.

Постановка задачи

Целью данной курсовой работы является создание алгоритма сравнения портретов процессов.

Для достижения этой цели мной были сформулированы следующие задачи:

1) исследование существующих методов сравнения строк;
2) реализация алгоритма сравнения портретов;
3) сбор базы портретов реальных процессов;
4) тестирование алгоритма на собранной базе.

Обзор литературы

Чтобы сравнить два набора строк, необходимо научиться решать задачу сравнения строк. А строки можно сравнивать разними способами. Мы рассмотрим несколько методов сравнения, применительно к шаблонам процессов. А именно:

1) точное совпадение;
2) расстояние Хэмминга;
3) расстояние Левенштейна;
4) длина наибольшей общей подстроки;
5) длина наибольшей общей подпоследовательности.

Начнем обзор с точного совпадения двух строк. Данный метод накладывает слишком строгие ограничения на шаблоны процессов. Как было сказано ранее, шаблон может являться конкатенацией нескольких других шаблонов, а значит, он должен быть в достаточной мере схож с шаблонами, из которых он состоит. Однако точное равенство для этих шаблонов не достигается, поэтому сходство между ними обнаружено не будет.

Теперь рассмотрим расстояние Хэмминга: это число позиций, в которых соответствующие символы двух слов одинаковой длины различны. Данная функция является метрикой в пространстве строк одинаковой длины. Впервые эта функция была описана Ричардом Хэммингом в 1950 году. Она использовалась для определения меры различия между двоичными векторами[3]. Впоследствии данный метод стал использоваться в вычислительной биологии[5, 569 с.].

К задаче сравнения двух шаблонов данный способ не применим, так как, во-первых, в контексте нашей задачи длины шаблонов могут быть различны, и, во-вторых, данный подход сильно привязан к номерам символов в шаблоне. Например, если у какого-либо шаблона удалить последний символ, а в начало добавить какой-либо символ, то расстояние Хэмминга в

худшем случае будет равняться нулю, хотя оба эти шаблона содержат в себе большую, семантически одинаковую часть.

Следующим методом, который мы изучим, является расстояние Левенштейна (другое название – редакционное расстояние): это минимальное число операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую. Впервые данный метод сравнения сформулировал советский математик Владимир Иосифович Левенштейн в 1965 году при исследовании двоичных последовательностей[4]. В настоящее время этот способ сравнения также используется в вычислительной биологии[5, 270 с.].

Данный метод не подходит для решения нашей задачи, так как вставка, замена или удаление даже одного символа могут поменять семантику, а значит уменьшить схожесть шаблона непредсказуемым образом, а расстояние Левенштейна изменится незначительно.

Перейдём к сравнению шаблонов по длине наибольшей общей подстроки. Выделенная данным способом подстрока не поменяла своей семантики, а значит по ней можно судить о степени схожести шаблонов. Однако сравниваемые шаблоны могут содержать в себе несколько различных достаточно длинных общих подстрок (будем называть их подшаблонами), что с одной стороны повышает сходство шаблонов, а с другой не будет обнаружено данными методом.

Последним, рассматриваемым нами способом сравнения двух строк, является вычисление длины наибольшей общей подпоследовательности. Данный метод, в отличие от предыдущего, не локализируется в каких-либо определенных частях строк, однако может выделять изолированные общие подстроки маленькой длины. Данные части шаблона являются неустойчивыми подшаблонами, так как их семантика, в силу недостаточной длины, сильно зависит от окружающего их контекста в следе.

Последние два метода содержат как достоинства, так и недостатки, поэтому рассмотрим некий компромисс между двумя этими методами:

выявление всех достаточно длинных замкнутых общих подшаблонов, другими словами тех подшаблонов, которые не являются подстроками других выявленных подшаблонов. Данный подход не выделяет коротких нестабильных подшаблонов и не локализируется в какой-либо подстроке. Поэтому данный метод сравнения был выбран в качестве метода сравнения шаблонов.

Для реализации данного алгоритма потребуется построить несколько суффиксных деревьев, поэтому рассмотрим их структуру. Каждое дерево строятся по некоторой строке. Фактически, данная структура является деревом, дуги которого замечены парами чисел, описывающими подстроки исходной строки, а каждый внутренний узел имеет более одного ребёнка. Если для каждого пути от корня до некоторого листа провести конкатенацию всех строк, размещённых на дугах, то получатся все суффиксы исходной строки. Путь от корня до какого-либо узла описывает некоторый префикс суффикса, иначе говоря, подстроку, причем единственным образом. Помимо этого, каждый внутренний узел содержит суффиксную ссылку, переход по которой приводит к узлу, описывающему строку, полученную из строки предыдущего узла, без первого символа.

Описание реализации алгоритма

Данный алгоритм принимает в качестве входных данных P_1, P_2 – два портрета процессов и l – минимальную длину допустимого подшаблона. Результатом работы данного алгоритма является Q – коэффициент сходства портретов.

Алгоритм состоит из следующих шагов:

1) построение двух общих строк, с помощью конкатенации всех шаблонов через уникальные разделители, для портретов P_1 и P_2;
2) построение суффиксных деревьев для получившихся строк;
3) вычисление суммарных длин L_1 и L_2 всех замкнутых шаблонов для обоих суффиксных деревьев;
4) достраивание суффиксного дерева одного из портретов с помощью общей строки другого портрета до общего суффиксного дерева для обоих портретов;
5) вычисление суммарной длины L_3 всех общих замкнутых подшаблонов обоих портретов;
6) вычисление коэффициента сходства Q между двумя портретами.

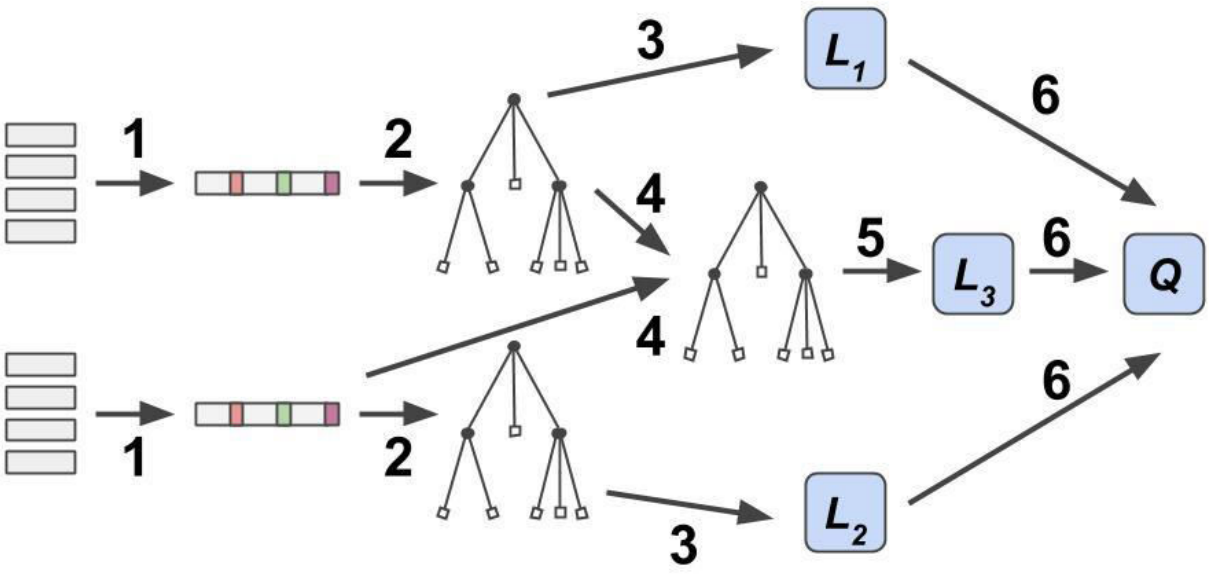

Рисунок 1. Схема алгоритма сравнения портретов

Рассмотрим все стадии алгоритма более подробно. Прежде всего, отметим, что в портрете могут встречаться как дубликаты (в случае, если в нескольких следах был обнаружен этот шаблон), так и незамкнутые шаблоны, то есть шаблоны, являющиеся подстроками других шаблонов. Эти шаблоны необходимо исключить, так как все их подшаблоны уже содержатся в других шаблонах портрета. Данную задачу можно решить с помощью суффиксного дерева. Для этого необходима единая строка, поэтому мы конкатенируем все шаблоны портрета P_1. Уникальные разделители между шаблонами нужны для того, чтобы разделить шаблоны друг от друга. Далее, производится построение суффиксного дерева online алгоритмом Эско Укконена[6]. С помощью обхода всех узлов дерева, выделяются только замкнутые шаблоны, и вычисляется их суммарная длина L_1.

Точно ту же последовательность действий выполняем для второго портрета P_2, вследствие чего получаем суммарную длину всех его замкнутых шаблонов L_2.

Заметим, что построение и анализ суффиксного дерева для каждого портрета производится независимо, поэтому эти задачи можно решать параллельно.

Теперь необходимо выделить общие замкнутые подшаблоны обоих портретов. Для этого выберем из построенных ранее суффиксных деревьев то, которое образовано от большей по длине строки. Благодаря online подходу алгоритма Укконена мы достраиваем это дерево с помощью второй строки до общего дерева для обоих портретов. В данном дереве, двигаясь от листьев к корню, перебираем все узлы, определяем к какому портрету принадлежат строки, образованные путями от корня, до этих узлов (данные строки является подстроками одного или нескольких шаблонов) и фиксируем эту информацию в узлах. Одновременно с этим выбираем те узлы, которые принадлежат обоим портретам. После этого для каждого найденного узла и всех узлов, достижимым по суффиксным ссылкам, исключаем всех их предков, а также исключаем все узлы, описывающие подшаблоны длинной

меньше l. В итоге остались узлы, описывающие все общие подшаблоны обоих портретов, не являющиеся подстроками других выбранных подшаблонов, иначе говоря, все замкнутые общие подшаблоны. Наконец, вычисляем L_3 – суммарную длину всех общих замкнутых шаблонов обоих портретов.

Осталось вычислить итоговый коэффициент сходства портретов, по формуле $Q = \frac{L_3}{\min(L_1, L_2)}$. Данная формула подобрана эмпирически. Фактически, Q отражает, отношение размера общей части двух портретов к размеру меньшего из портретов.

Перейдём к оценкам по времени и памяти для данного алгоритма. Количество узлов во всех построенных суффиксных деревьях линейно, относительно суммы длин всех шаблонов, дополнительная информация константного размера сохранялась только в узлах, поэтому итоговая оценка используемой памяти $O(\sum|t|)$, где суммирование производится по всем шаблонам портретов P_1 и P_2.

Оценка времени работы алгоритма определяется самой вычислительно затратной частью алгоритма, а именно построением суффиксного дерева. Для произвольного алфавита A строки S время работы алгоритма Укконена оценивается, как $O(|S|log|A|)$. В нашем случае, алфавит системных вызовов константен, однако количество добавленных уникальных разделителей определяется, как $|P_1| + |P_2|$. Поэтому итоговая временная оценка составляет $O((\sum|t|)log(|P_1| + |P_2|))$.

Тестирование алгоритма

Для первичного тестирования алгоритма использовалось два набора, каждый из которых содержит по десять портретов приложений, типичных для Windows 7: Explorer, Opera, Minesweeper, Task Manager, Paint W7, Note Pad ++, WinRAR, Calculator W7, VS 2012 и Word Pad. Построение каждого портрета проводилось в течение 40 минут, причем моё поведение в приложениях никак не регламентировалось. Далее, каждый портрет из первого набора сравнивался со всеми портретами из второго набора. Округленные результаты сравнения отображены в *таблице 1*.

	Explorer	Opera	Minesweeper	Task Manager	Paint W7	Note Pad ++	WinRAR	Calculator W7	VS 2012	Word Pad
Explorer	63	0	3	2	2	3	5	3	3	3
Opera	0	52	8	0	1	0	1	1	1	0
Minesweeper	2	5	88	2	2	2	3	3	1	2
Task Manager	2	0	3	34	2	2	7	5	1	1
Paint W7	4	0	2	2	47	5	4	3	2	15
Note Pad ++	3	0	3	2	2	59	6	4	2	1
WinRAR	5	1	5	6	3	5	40	6	3	4
Calculator W7	2	1	3	4	2	3	3	67	1	1
VS 2012	2	1	1	1	2	2	2	1	50	2
Word Pad	4	0	2	1	12	2	4	2	2	12

Таблица 1. Результаты первичного тестирования

Если в качестве минимального значения, при котором можно считать, что портреты построены от одного процесса, взять, например, 25, то ошибочный результат получился только при сравнении двух портретов приложения Word Pad. Это обусловлено тем, что моё поведение при построении портретов сильно различалось. Поэтому было решено проверить, насколько поведение пользователя в приложении отражается на построенном портрете этого приложения. Для этого были построены два набора по пять портретов приложения Word Pad. Данные портреты строились в течение пяти

минут, причем на протяжении этого промежутка времени для первых четырёх портретов каждого набора выполнялось только одно действие, своё для каждой пары портретов. Для последней пары портретов выполнялись все эти действия поочерёдно. Результаты сравнения этих двух наборов представлены в *таблице 2*.

	Просмотр	Вставка текста	Изображение	Курсив	Всё вместе
Просмотр	49	7	9	6	16
Вставка текста	5	51	2	2	19
Изображение	5	2	94	2	21
Курсив	5	3	2	99	99
Всё вместе	8	22	15	71	97

Таблица 2. Результаты сравнения портретов приложения Word Pad

Как видно из данной таблицы, поведение пользователя в приложении Word Pad оказывает очень сильное влияние на структуру портретов. При различных поведениях сходства процессов не наблюдается.

Однако для более сложных приложений поведение пользователя накладывает гораздо меньший отпечаток на портрет, что демонстрируется в аналогичном тестировании, но для приложения Opera (*таблица 3*).

	Google	Wikipedia	Google map	YouTube	Всё вместе
Google	75	53	13	23	51
Wikipedia	51	71	14	20	36
Google map	17	14	61	14	21
YouTube	25	29	11	54	61
Всё вместе	46	41	22	58	69

Таблица 3. Результаты сравнения портретов приложения Opera

Для большинства пар портретов их сходство было обнаружено, однако и для этого приложения некоторые поведения сильно отличаются от остальных.

Последнее тестирование было нацелено на поиск ответа на обратный вопрос: влияет ли одинаковое поведение в различных приложениях на сходство их портретов? Для этого было выбрано шесть текстовых редакторов: Word Pad, Note Pad, Note Pad ++, Far Manager, Sublime Text, VS 2012. Как и в предыдущих тестах, было построено два набора, содержащих портреты всех вышеперечисленных приложений. В каждом из этих приложений выполнялось фиксированная последовательность действий над одним и тем же текстовым файлом.

	Word Pad	Note Pad	Note Pad ++	Far Manager	Sublime Text	VS 2012
Word Pad	93	9	12	1	1	4
Note Pad	13	92	17	1	1	10
Note Pad ++	14	15	73	2	1	10
Far Manager	1	1	2	99	0	3
Sublime Text	1	1	2	0	91	1
VS 2012	13	18	20	3	1	81

Таблица 4. Результаты сравнения портретов текстовых редакторов

В результате сравнения этих наборов, представленных в *таблице 4*, было выявлена независимость схожести портретов различных текстовых редакторов от поведения пользователя.

Подводя итоги данного тестирования, выделим главные обнаруженные особенности:

- результат сравнения портретов простого приложения сильно зависит от поведения пользователя в процессе сбора следа;
- результат сравнения портретов сложного приложения в некоторых случаях зависит от поведения пользователя в процессе сбора следа;
- результат сравнения портретов различных приложений практически не зависит от поведения пользователя в процессе сбора следа.

Заключение

Таким образом, в рамках данной курсовой работы мной были получены следующие результаты:

1) исследованы существующие методы сравнения строк;
2) реализован алгоритм сравнения портретов;
3) собрана база портретов реальных процессов;
4) алгоритм протестирован на собранной базе.

Список литературы

1. Баклановский М.В., Ханов А.Р. CODA – новая система компьютерной безопасности: обзор архитектуры системы // Материалы секции 22, XXXVIII Академические чтения по космонавтике. 2014. С. 649–650.

2. Баклановский М. В., Ханов А. Р. Поведенческая идентификация программ // Моделирование и анализ информационных систем. Ярославль, Том 21, Номер 6, 2014. С. 120–130.

3. Hamming R. W. Error detecting and error correcting codes // Bell System Technical Journal. New York, 1950. P. 147–160.

4. Левенштейн В. И. Двоичные коды с исправлением выпадений, вставок и замещений символов // Доклады Академий Наук СССР. М. Том 163, 1965. С. 707–710.

5. Гасфилд Д. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология. СПб., 2003.

6. Ukkonen E. On-line construction of suffix-trees // Algorithmica. New York, 1995. P. 249–260.