Математические структуры и моделирование
2010, вып. 21, с. 141–150

УДК 004.056.57

ЧИСЛО ПОТОКОВ КАК ХАРАКТЕРИСТИКА ЗАРАЖЕННОСТИ ПРОЦЕССА

В.В. Пляшко, Н.Ф. Богаченко

В работе сформулированы и обоснованы критерии, позволяющие, опираясь на число потоков некоторого процесса, дать заключение о том, является ли этот процесс зараженным.

Введение

В рамках построения системы обнаружения вторжений, динамически настраивающейся под вновь обнаруживаемые уязвимости, актуальным становится ответ на вопрос: можно ли каким-то способом ничего не зная о структуре и логике процесса, по его свойствам определить, является ли он опасным.

Данная работа посвящена анализу возможности определения наличия заражения процесса вредоносными программами и срабатывания в процессе логической бомбы, опираясь на число потоков данного процесса. Результаты исследования можно представить в виде марковской цепи числа потоков, что находит отклик в современных подходах к построению систем обнаружения вторжений, использующих марковские модели [5, 7, 11].

Стоит отметить, что на сегодняшний день существует множество реализаций сканеров свойств запущенных процессов, например библиотеки `Psapi.dll` и `tlhelp32.dll`, а также конечные программные продукты, такие как «Стандартный диспетчер задач `Windows`», «`ProcessMonitor`» (`Procmon.exe`), «`ProcessExplorer`» (`procexp.exe`). Но ни один из них не предоставляет удобных инструментов для исследования возможности определения наличия вредоносных потоков в процессе.

1. Нежелательное изменение числа потоков

В этом разделе речь пойдет о том, в каких случаях можно говорить о нежелательном изменении числа потоков исследуемого процесса [10].

Во-первых, вспомним о так называемой логической бомбе, которая может содержаться даже в самых, казалось бы, безобидных приложениях. При этом

Омский государственный университет им. Ф.М. Достоевского.
E-mail: `nfbogachenko@mail.ru`

она может быть как намеренно «вшита» в программу, так и являться результатом работы так называемого EPO-вируса [2].

Логическая бомба — это программа или часть программы, которая запускается при определенных временных или информационных условиях для осуществления вредоносных действий (как правило, несанкционированного доступа к информации, искажения или уничтожения данных). Многие вредоносные программы, такие как вирусы или черви, часто содержат логические бомбы, которые срабатывают в заранее определенное время или при выполнении определенных условий, например в пятницу 13-го. К логическим бомбам, как правило, относят код, который приводит к не сообщенным заранее последствиям для пользователей. Активировать логическую бомбу может и сам пользователь, как умышленно, так и неумышленно, даже не подозревая об этом.

Будем рассматривать запускаемый процесс как совокупность его состояний. Условно разделим все состояния процесса на нормальные (не несущие угроз безопасности) и нежелательные. На рисунке 1 верхней стрелкой обозначено нормальное функционирование процесса, левой – срабатывание логической бомбы или иной способ перехода процесса в нежелательное состояние. Последнее может быть реализовано каким угодно образом, в том числе процесс в этом состоянии может запрашивать дополнительные ресурсы, например открывать новые потоки.

Рис. 1. Общая модель работы процесса

Стоит заметить, что срабатывание логической бомбы может повлечь за собой достаточно серьезные последствия. Ярким свидетельством этого является опыт войны в Персидском заливе: Ирак не смог применить против многонациональных сил закупленные во Франции системы ПВО потому, что их программное обеспечение содержало логические бомбы, которые были активизированы с началом боевых действий [9].

Рассмотрим теперь другой случай. Допустим, что у самого процесса нет вышеописанных нежелательных состояний. Здесь имеет место другой фактор: при написании вредоносных программ злоумышленники часто используют достаточно популярный способ скрыть существование своей программы, замаскировав ее выполнение под поток другого процесса. Схематично это показано на рисунке 2.

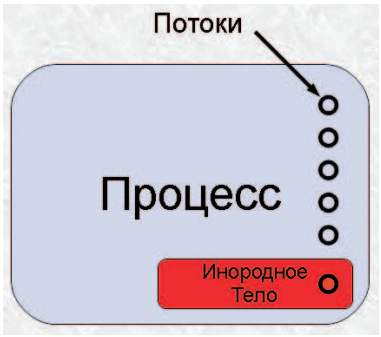

Рис. 2. Схема заражения процесса

Например, это можно сделать следующим образом: зарегистрировать фрагмент памяти, в котором будет лежать код (для этого можно воспользоваться функциями `OpenFileMapping` и `MapViewOfFile` из `kernel32.dll`). В эту область поместить код потока, который будет выполняться в процессе-жертве. Теперь необходимо заставить жертву запустить этот поток. Для этого можно использовать функцию `CreateRemoteThread` [6]. Есть и другие описания подобных механизмов, например в работе [4].

Оба рассмотренных случая демонстрируют, что нежелательное поведение процесса, будь то его собственные функции или внешнее вмешательство, может привести к увеличению числа потоков. Следовательно, есть теоретическая возможность узнать об угрозе безопасности, опираясь на число потоков приложения.

2. Этапы эксперимента и необходимые ограничения

Изучение поведения потоков исследуемого процесса предлагается разбить на два этапа:

– этап наблюдения: изучение поведения в идеальных условиях – *режим обучения*;

– этап аудита: анализ поведения в естественных условиях – *рабочий режим*.

При этом требуется понять, какие ограничения накладываются на планируемый эксперимент. Сразу стоит заметить, что исследование нужно проводить в общем виде, соответственно мы ничего не можем знать о структуре и логике процесса, который нам предстоит изучать. В связи с этим мы априори не сможем отличить нежелательные состояния от нормальных.

Чтобы научиться отличать друг от друга данные состояния, необходимо определить, что такое нормальные состояния процесса.

1. Во время наблюдения за процессом считается, что угрозы безопасности отсутствуют.

Это условие гарантирует, что любое состояние, замеченное во время наблюдения, является нормальным. Но не стоит путать состояние процесса в общем смысле и состояние процесса в смысле количества его потоков, ибо при одном и том же числе потоков процесс может выполнять или не выполнять совер-

шенно разные наборы функций, в том числе и вредоносные. У нас же имеется информация только о числе потоков. Обозначим его ν.

Итак, мы наблюдаем за процессом, собирая информацию обо всех возможных значениях величины ν. Как правило, для сложных приложений возможность получить весь набор значений величины ν очень сомнительна. Поэтому примем следующее допущение:

2. Либо мы имеем достаточно времени и возможностей для получения всех значений величины ν нормального функционирования процесса, либо все значения величины ν, которые не были получены во время наблюдения, являются признаком нежелательного состояния процесса.

3. Параметры исследования

В дальнейшем под *состоянием процесса* будем понимать текущее число потоков этого процесса, то есть значение величины ν. Тогда *разрешенное состояние* – это то, в котором был замечен процесс в режиме обучения. *Запрещенное состояние* – это то, которое не было замечено при обучении.

Опираясь на допущения предыдущего раздела, можно говорить, что *в режиме обучения мы получаем все возможные разрешенные состояния.*

Безусловно, мы сможем определить наличие вредоносных потоков у приложений с постоянным числом потоков, как в принципе и у приложений, имеющих небольшое ограниченное число возможных потоков.

Можно привести простой пример: утилиту `tracert`, которая работает в один поток. Соответственно, если у нее возникает один лишний поток, сразу понятно, что ее поведение не является безопасным.

Другой пример – программа `mspaint`. На этапе обучения становится понятно, что данная программа запускает в начале пять потоков. При вызове меню «Открыть файл» количество потоков увеличивается на три, позже постепенно уменьшаясь. Таким образом, мы можем выделить характерные изменения числа потоков: «+3» или «–1». Для наглядности функция изменения числа потоков программы изображена на рисунке 3 в виде направленного графа [1]. Заметим,

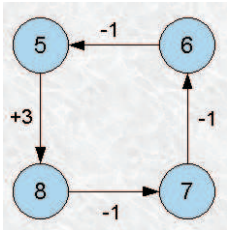

Рис. 3. Возможные изменения числа потоков

что увеличение числа потоков на величину, отличную от трех, в данном случае является неестественным и может свидетельствовать о том, что процесс небезопасен. Следовательно, *возможные изменения числа потоков (изменения величины ν) могут оказаться важной характеристикой.*

Теперь попробуем добавить к исследованиям еще одну функцию приложения `mspaint` – открытие справки. Если в режиме обучения открывать диалог «Открыть файл» и справку в произвольном порядке, то у нас получится более разнообразный набор состояний и переходов. Так, открытие справки влечет за собой прибавку 6-ти потоков. Закрытие же ее приводит к немедленному уменьшению числа потоков на четыре и последующему постепенному уменьшению числа потоков еще на два.

Рассмотрим рисунок 4. Как мы видим, допустимо увеличение числа потоков

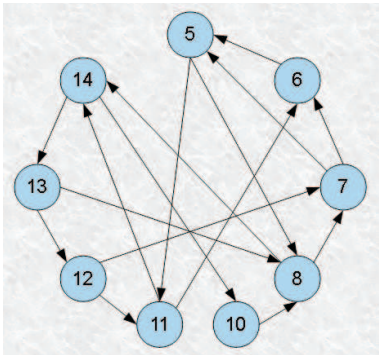

Рис. 4. Возможные изменения числа потоков при более длительном наблюдении

на три. Но из всех ли состояний возможно данное увеличение? Естественно, что заведомо вызовет подозрения, если увеличение на три произойдет с 12-ти, 13-ти или 14-ти потоков, так как итоговое количество не будет даже входить в набор, полученный в режиме обучения.

Нас больше интересуют такие изменения числа потоков на три, при которых мы из разрешенного числа потоков попадаем опять же в разрешенное. Примером может служить несуществующий переход от 7-ми потоков к 10-ти. Это означает следующее: *недостаточно учитывать допустимые изменения числа потоков (изменения величины ν), а следует также фиксировать состояния, из которых возможны данные изменения.*

Необходимость учета таких *исходных* состояний подтверждает и следующий факт. Один и тот же исполняемый файл при разных начальных параметрах может привести к различному поведению соответствующего процесса. Пример возможной модели функционирования процессов, запущенных из того же исполняемого файла, но с разными параметрами, представлен на рисунке 5: изменения числа потоков «+1» и «–1» характерны для обоих вариантов, «+2» и «–2» – только для первого, а «+5» – только для второго.

Теперь представим, что исполняемый файл исследуемого процесса после завершения обучения был инфицирован. И допустим, что пользователю не понадобятся функции открытия файла и справки. В этом случае возможен вариант, что вредоносная программа выделяет для себя необходимые ресурсы, запуская часть себя в отдельном потоке, после чего передает управление своей «жертве». В этом случае получится процесс со стартовым количеством потоков «6», что является разрешенным числом. Но факт того, что файл был заражен, усколь-

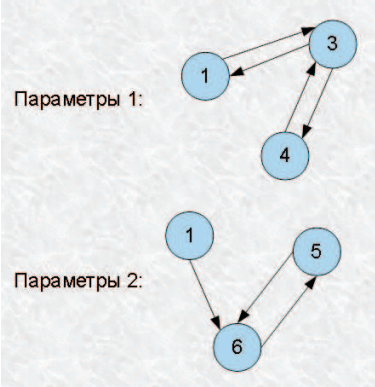

Рис. 5. Поведение процесса при различных начальных параметрах

зает. Чтобы этого не произошло, можно также проверять *стартовое число потоков процесса*.

4. Поиск критерия

Пусть на этапе обучения собрана некоторая информация о динамике изменения числа потоков исследуемого процесса, назовем ее статистикой.

Не будем забывать, что поведение процессов может существенно отличаться, и некоторые из них являются более предсказуемыми в плане определения наличия вредоносных потоков, а некоторые могут быть очень «неудобными» в этом смысле. В связи с этим было бы целесообразно найти такой критерий, который, исходя из собранной информации, мог бы давать нам представление о том, насколько каждый конкретный процесс является *предсказуемым* в вопросе определения наличия вредоносных потоков.

Естественно, что одного математического ожидания числа потоков будет явно недостаточно. Проанализируем, насколько подходит дисперсия на роль критерия [3, 8].

Рассмотрим процесс с постоянным числом потоков. Очевидно, что математическое ожидание числа потоков равно самому числу потоков, а дисперсия числа потоков такого процесса будет равна нулю:

$$M_1 = \text{const}, \ D_1 = 0.$$

Рассмотрим процесс с двумя равновероятными состояниями: $\nu_1 = 1, p_1 = 0.5$, $\nu_2 = 2, p_2 = 0.5$. Математическое ожидание и дисперсия числа потоков:

$$M_2(\nu) = 1.5, \ D_2(\nu) = 0.25.$$

Теперь предположим, что есть три равновероятных состояния: $\nu_1 = 1$, $p_1 = 1/3$, $\nu_2 = 2$, $p_2 = 1/3$, $\nu_3 = 3$, $p_3 = 1/3$. Математическое ожидание и дисперсия числа потоков:

$$M_3(\nu) = 2, \ D_3(\nu) = 0.667.$$

Нетрудно заметить, что в приведенных примерах с ростом возможного числа потоков при равной вероятности их появления дисперсия возрастает: $D_1 < D_2 < D_3$. Можно предположить, что чем больше дисперсия, тем более разнообразно ведет себя процесс и тем сложнее определить, присутствует ли в нем вредоносный поток. Попробуем найти пример, противоречащий данному предположению: $\nu_1 = 2$, $p_1 = 0.5$, $\nu_2 = 6$, $p_2 = 0.5$. Математическое ожидание и дисперсия числа потоков:

$$M_4(\nu) = 4,\ D_4(\nu) = 4.$$

С одной стороны, выполняется следующая цепочка неравенств: $D_1 < D_2 < D_4$. С другой – как в первом, так и в четвертом примерах прибавка одного вредоносного потока приводит к запрещенному состоянию, что автоматически указывает на нежелательное состояние процесса. Во втором примере та же самая прибавка одного вредоносного потока может привести к переходу из разрешенного состояния опять же в разрешенное.

Таким образом, дисперсия не подходит для оценки возможности определения наличия вредоносных потоков. Необходимо найти подход, учитывающий не только разброс возможного числа потоков, но и сам спектр принимаемых значений.

Поместим все разрешенные состояния процесса на числовую прямую, например так, как показано на рисунке 6. При фиксированных минимальном и

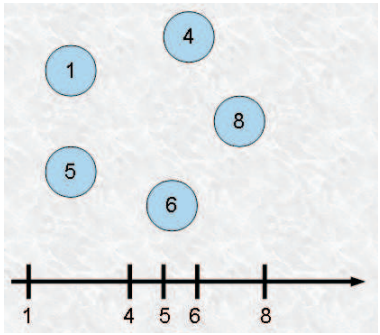

Рис. 6. Размещение значений величины ν на числовой прямой

максимальном значениях количество гипотетически возможных целочисленных значений величины ν известно. В данном случае – 8. Так же возможно подсчитать число разрешенных значений – 5. Очевидно, что *чем больше в данном промежутке разрешенных состояний и чем меньше запрещенных, тем меньше шансов обнаружить нежелательное увеличение числа потоков.*

Рассмотрим два примера, показанных на рисунке 7: в обоих случаях длина промежутка и количество разрешенных значений величины ν совпадают. Но из-за того, что во втором примере разрешенные состояния располагаются «кучно», будет сложнее определить наличие, скажем, одного вредоносного потока. В то время как первый пример позволяет сделать это почти во всех разрешенных состояниях.

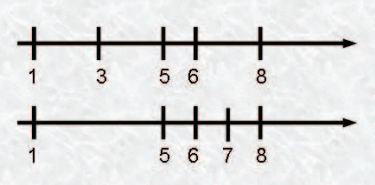

Рис. 7. Разрешенные состояния процессов

Таким образом, *имеет смысл учитывать не только спектр принимаемых параметром значений, а также особенности их взаимного размещения.*

Помимо вышесказанного стоит также учитывать специфику возможного нежелательного поведения процесса. Хотя вредоносные средства и пишутся таким образом, чтобы быть менее заметными в системе и использовать как можно меньше ресурсов, но они могут быть реализованы не только в одном, но и в нескольких потоках.

Обозначим количество вредоносных потоков через X. Теперь мы можем сформулировать так называемый *критерий X-непредсказуемости процесса.* Рассмотрим пример, изображенный на рисунке 8. Для начала определим, на-

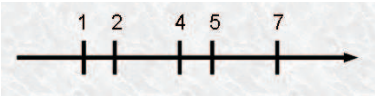

Рис. 8. Разрешенные состояния процесса

сколько удобен процесс для определения наличия одного вредоносного потока. Для этого разделим все его разрешенные состояния на 1-предсказуемые и 1-непредсказуемые. *1-предсказуемыми* назовем те состояния, увеличение числа потоков на один в которых приведет к запрещенному состоянию. Соответственно *1-непредсказуемыми* состояниями назовем те, увеличение числа потоков на один в которых приведет к разрешенному состоянию. На рисунке 8 состояния «1» и «4» являются 1-непредсказуемыми.

Изучение поведения процесса на этапе обучения позволяет вычислить вероятности[1] нахождения процесса в каждом состоянии. Теперь мы можем говорить о вероятности нахождения процесса в 1-непредсказуемом состоянии в каждый момент времени: для этого нужно сложить вероятности нахождения процесса в каждом из 1-непредсказуемых состояний. Найденная вероятность и будет значением критерия 1-непредсказуемости.

По аналогии можно вычислить значения критерия для двух и более вредоносных потоков: 2-непредсказуемыми являются состояния «2» и «5»; 3-непредсказуемыми – состояния «1», «2» и «4»; 4-непредсказуемое состояние «1» и т.д. Для расчета критерия X-непредсказуемости в каждом случае надо сложить вероятности X-непредсказуемых состояний.

[1]Здесь и далее речь пойдет о вероятностях, экспериментально полученных в ходе обучения.

5. Разработка программы

Для проведения численного эксперимента на языке `C++` [13] в среде разработки `Microsoft Visual Studio` [14] была реализована программа-сканер, отслеживающая число потоков запущенных процессов.

Предусмотрено три режима работы сканера – простой, режим обучения и рабочий режим. Программа-сканер сохраняет собранную на этапе обучения статистику в файл. Для сохранения данных был выбран формат XML [12]. Имеется функция обнуления накопленной статистики и возможность отключать слежение для каждого конкретного процесса.

Для удобства на этапе обучения для каждого процесса сохраняется не величина изменения числа потоков, а состояние, в которое процесс попадает при таком изменении (см. таблицу 1). Фиксируется количество попаданий процесса

Таблица 1. Описание разрешенного поведения процесса, представленного на рисунке 5

Разрешенное состояние	1	3	4	5	6
Разрешенные состояния перехода	3, 6	1, 4	3	6	5

в каждое состояние и количество переходов из одного допустимого состояния в другое. Далее происходит расчет математического ожидания и дисперсии числа потоков, и вычисляется значение критерия 1-непредсказуемости процесса.

В представленной работе подробно освещены алгоритмы режима обучения. Стоит заметить, что расчет критерия X-непредсказуемости пока что не учитывает условные вероятности перехода из одного состояния в другое – это вопрос дальнейших исследований.

Накопленная в ходе обучения статистика в рабочем режиме используется для диагностики процессов. Описание алгоритмов этого режима выходит за рамки данной статьи.

В заключении построим граф состояний некоторого процесса (см. рис. 9): здесь вес ребра – полученное на этапе обучения число соответствующих пере-

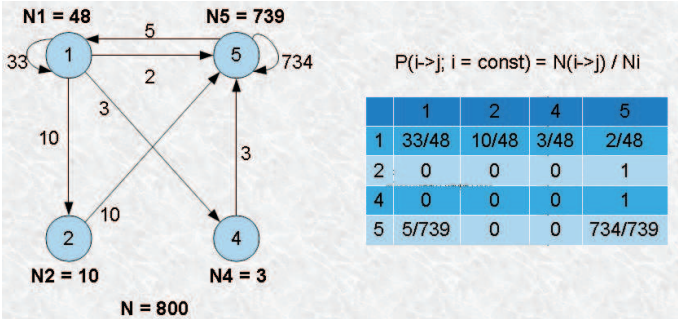

Рис. 9. Граф состояний и матрица условных вероятностей перехода

ходов. Используя количественные показатели, вычисляются условные вероятности перехода из одной вершины в другую. Таким образом, можно говорить о

марковской цепи с дискретным временем, матрица которой также представлена на рисунке 9.

Не стоит забывать, что программа написана для работы в пользовательском режиме. Если же иметь доступ к таблице прерываний, можно точно определять время запуска потоков, и тогда речь уже пойдет о марковской цепи с непрерывным временем. Это может дать дополнительные возможности исследования поведения процессов.

Литература

1. Берж К. Теория графов и ее приложения. М.: ИЛ, 1962. С. 11-14.
2. Бойцев О. Защити свой компьютер на 100 % от вирусов и хакеров. СПб.: Питер, 2004. С. 98.
3. Гмурман В.Е. Теория вероятностей и математическая статистика. М.: «Высшая школа», 2003. С. 85-89.
4. Зайцев О. Технологии вредоносных программ и угрозы информационной безопасности // КомпьютерПресс. 2007. N. 3.
   URL: http://www.compress.ru/article.aspx?id=17361 (дата обращения: 10.01.2010).
5. Ивашко Е.Е. Построение системы защиты электронных библиотек от несанкционированного копирования документов.
   URL: http://rcdl2007.pereslavl.ru/papers/paper_41_v1.pdf (дата обращения: 27.03.2010).
6. Касатекно И. Пишем свой стелс // Хакер. 2003. N. 35(10). С. 52.
7. Кашаев Т.Р. Система активного аудита на основе скрытых марковских моделей // Информационное противодействие угрозам терроризма. Научно-практический журнал.
   URL: http://www.contrterror.tsure.ru/site/magazine4/Pdf/Journal4full.pdf (дата обращения: 27.03.2010).
8. Кибзун А.И. Теория вероятностей и математическая статистика. М.: Физматлит, 2002. С. 58.
9. Поздняков А.И. Информационная безопасность личности, общества, государства // Военная мысль. 1993. С. 16.
10. Рихтер Д. Windows для профессионалов. Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. СПб.: Питер, 2004. С. 130-154.
11. Соколов А.М. Обнаружение вторжений в компьютерную систему с помощью марковских цепей переменного порядка.
    URL: http://www.iai.dn.ua/public/JournalAI_2002_4/Razdel1/11_Sokolov.pdf (дата обращения: 27.03.2010).
12. Спенсер П. XML. Проектирование и реализация. М.: Лори, 2001.
13. Хортон А. Visual C++ 2005 Базовый курс. М.: Диалектика, 2007.
14. Kruglinski D.J., Wingo S., Shepherd G. Programming Microsoft Visual 6.0. Microsoft Press, 1998.