5. *C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton.* Stack-Guard: automatic adaptive detection and prevention of buffer-overflow attacks. In Proceedings of the 7th USENIX Security Symposium, January 1998.

6. *C. Cowan, M. Barringer, S. Beattie, and G. Kroah-Hartman.* FormatGuard: automatic protection from printf format string vulnerabilities. In Proceedings of the 10th USENIX Security Symposium, August 2001.

7. *James Newsome, Dawn Song.* Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In Proceedings of the Network and Distributed System Security Symposium, 2005

8. Vulnerabilities in SMBv2 Could Allow Remote Code Execution (975517) http://www.microsoft.com/technet/security/Bulletin/ms09-050.mspx

9. *Michael Sutton, Adam Greene, Pedram Amini.* Fuzzing: Brute Force Vulnerability Discovery. Addison-Wesley Professional; 1 edition (July 9, 2007)

10. *Благодаренко, А.В.* Широковещательное распространение обновлений безопасности для ОС Linux через спутниковый канал // Сборник ЮФУ. Тематический выпуск. Информационная безопасность. Перспектива-2009. – 2009. – С. 210–214.

Благодаренко Артем Васильевич
Технологический институт Федерального государственного образовательного учреждения высшего профессионального образования «Южный федеральный университет» в г. Таганроге.
E-mail: artem.blagodarenko@gmail.com.
347928, г. Таганрог, ул. Чехова, 2, корпус "И".
Тел.: 8 (8634) 312-018.
Кафедра безопасности информационных технологий; аспирант.

Blagodarenko Artem Vasilyevich
Taganrog Institute of Technology – Federal State-Owned Educational Establishment of Higher Vocational Education “Southern Federal University”.
E-mail: artem.blagodarenko@gmail.com.
Block “I”, 2, Chehov str., Taganrog, 347928, Russia.
Phone: 8 (8634) 312-018.
The Department of Security of Information Technologies; post-graduate student.

УДК 004.942 – 056.57

С.Д. Жилкин

РАЗДЕЛЕНИЕ РАБОТЫ ПО НА ФАЗЫ С ЦЕЛЬЮ ПОСТРОЕНИЯ МОДЕЛИ РАБОТЫ ПО

Статья предлагает подходы к решению задачи моделирования поведения программного обеспечения (ПО) с целью дальнейшего выявления отклонений работы. Методы моделирования основаны на системах мониторинга и математическом аппарате нейронных сетей. Основная часть статьи посвящена механизму моделирования работы ПО на протяжении всего рабочего цикла путём выявления определённых фаз работы ПО. Рассмотрена эффективность различных способов.

Модель работы ПО; аномалии работы ПО; фазы работы ПО; нейронные сети.

S.D. Zhilkin

DIVIDING THE SOFTWARE'S PERFORMANCE INTO WORK PHASES FOR BUILDING BEHAVIOR MODELS

This article presents methods for solving the task of modeling software's functioning for further exposure of deflections from functioning model. Modeling methods are based on monitoring systems and neural networks mathematics. The central part of the article focuses on the methods of modeling software's functioning all over the software's life cycle by exposing certain phases of software's functioning. The efficiency of various methods is concerned.

Software's functioning model; functioning deflections; software's functioning phases; neural nets.

Введение

Актуальные задачи сферы компьютерной безопасности уже продолжительное время включают в себя задачу построения модели поведения ПО с целью анализа дальнейших действий ПО относительно построенной модели. Во время построения модели считается, что ПО работает в доверенном режиме, что позволяет получить так называемую эталонную модель поведения. Относительно такой эталонной модели затем можно выявлять отклонения поведения ПО. В зависимости от характеристик, которые описывала эталонная модель, отклонения могут показать: превышение полномочий пользователем, вредоносные действия ПО, подмену ПО другим исполняемым файлом и т.д. Отчасти данная задача решена [1]. В частности, для ПО, имеющего строгий алгоритм работы (системные службы, службы, работающие в фоновом режиме), некоторые ОС предоставляют уже встроенные в дистрибутив средства профилирования, такие как ps-watcher, pwatch и pScan (для Linux-систем). Также существует множество антивирусов, препятствующих выполнению вредоносного кода. Большинство из них основываются на сигнатурном и эвристическом методах анализа исполняемого файла для выявления вредоносного кода. Однако куда меньшее внимание уделено проблеме выявления и локализации недекларированных возможностей ПО. Так как зачастую антивирусы проводят анализ начальных и конечных частей файла в поисках вредоносного кода и, учитывая, что внедрение компьютерной закладки, приводящей, например, к НСД, может быть осуществлено в середине файла, можно говорить о том, что антивирусы не предоставляют полной защиты в данной области. Также учитывая, что код, приводящий к НСД, сам по себе может не являться вредоносным, становится очевидным, что одни антивирусы не обеспечивают должный уровень защиты данных от НСД и НДВ.

Итак, вредоносный код успешно выявляется антивирусами, а для строго алгоритмизированного ПО существуют методы построения точных моделей поведения, поэтому методы, рассмотренные в данной статье, прежде всего, направлены на выявление НДВ в ПО, поведение которого заранее неизвестно. К такому ПО относятся прикладные программные пакеты, имеющие пользовательский интерфейс (например, Adobe Photoshop, 1С, Microsoft Office). Отличительными особенностями такого ПО являются: множество ветвлений алгоритма работы, частая зависимость выполняемых действий от выбора пользователя, что приводит к их не-

предсказуемости, а также то, что время работы ПО заранее неизвестно. Данные особенности сильно затрудняют решение задачи построения эталонной модели работы ПО.

Построение модели поведения ПО

О поведении ПО можно судить по его взаимодействию с операционной системой: обращение к жёсткому диску, сетевым ресурсам, вызовы функций драйверов, работа с реестром и прочее. Так как при моделировании и проведении последующего анализа использован математический аппарат, необходимо описывать поведение ПО некоторым набором числовых значений — вектором координат. Предлагается описывать поведение процесса количественными, логическими и статистическими характеристиками. Количественные характеристики могут включать в себя такие параметры, как число обращений к системным файлам, библиотекам и прочим ресурсам системы. Логические характеристики описывают общее поведение ПО: создавались ли сетевые соединения, порождались ли дополнительные процессы и прочее. Статистические характеристики могут показывать частоту выделения дополнительной памяти, обращений к жёсткому диску и прочие частотные параметры. Чем более подробной будет информация о взаимодействии процесса с операционной системой, тем более развёрнутым будет вектор характеристик, описывающий поведение ПО, и тем более точными будут результаты моделирования и дальнейшего анализа. В дальнейшем вектор, описывающий поведение ПО за некоторое время работы, будем называть профилем работы.

Для того, чтобы получать информацию такого рода о поведении ПО, потребуется дополнительная система аудита. Она может основываться на журнальных файлах операционной системы или, например, на более подробных данных, полученных на уровне драйвера. Так как устройство наиболее популярных пользовательских операционных систем (Windows, Linux, MacOS, Unix, Solaris) имеет схожие принципы архитектуры (архитектура системных уровней, системных вызовов и т.д.), возможно унифицировать сообщения систем аудита для каждой платформы и создать систему моделирования и анализа ПО, запущенного на любой платформе.

Если принять, что вектор координат описывает работу ПО и является эталонной моделью, то необходим механизм, который сможет найти отклонения работы ПО от эталонной модели. То есть нужен механизм, который сравнивает два вектора координат и некоторым образом сигнализирует о несоответствии одного вектора другому. Причём данный механизм должен учитывать следующую особенность: несоответствием является, прежде всего, не количественное отклонение одной координаты в разных векторах, а наличие некоторых характеристик поведения, которые отсутствуют в эталонном векторе. По сути данная задача сводится к задаче распознавания образов.

Так как уже достаточно долгое время для распознавания образов с успехом используются нейронные сети, было принято решение воспользоваться этим математическим аппаратом в данной работе. В предлагаемом методе моделирования и анализа в качестве модели поведения ПО используется некоторая нейронная сеть, на вход которой подаётся профиль поведения (то есть вектор координат). Выходом нейронной сети является число от 0 до 1, показывающее вероятность соответствия профиля модели [2]. Для того чтобы получить такую сеть, требуется провести процесс обучения, который заключается во множестве калибровок коэффициентов и смещений нейронов методом обратного распространения ошибки с целью получения результата, близкого к единице [5].

При обучении следует учитывать, что каждый рабочий цикл ПО несколько отличается от предыдущего [6]. Это, прежде всего, связано с некоторыми оптимизационными механизмами, присущими каждой операционной системе. Так профиль работы, собранный за второй запуск ПО, будет отличаться от профиля для первого запуска, так как операционная система кэширует данные и требует гораздо меньше времени для повторного обращения к тем же самым ресурсам. Поэтому для более точного обучения нейронной сети на поведение ПО необходимо произвести множество запусков целевого ПО, пройдя множество рабочих циклов ПО (рис. 1). Во время рабочего цикла желательно выполнить как можно больше различных действий с помощью целевого ПО или выполнить все легитимные действия, разрешённые оператору данного ПО. Дополнительное обучение осуществляется обучением нейронной сети на обратный результат (близкий к 0) на поведении ПО, схожего по функционалу с целевым ПО.

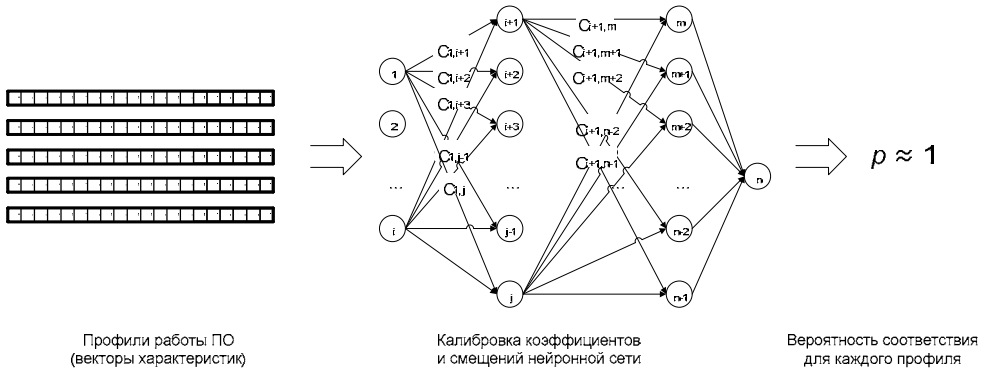

Рис. 1. Обучение нейронной сети на поведение ПО

Таким образом, моделью поведения считается специальным образом обученная нейронная сеть. Такой подход может оказаться действенным только в том случае, если ПО – имеет строгий алгоритм работы. В противном случае, даже вектор для доверенной работы ПО может дать отрицательный результат (далёкий от 1). Следовательно, данная модель не удовлетворяет требованиям и должна быть расширена.

Предлагается расширение модели поведения до как минимум трёх нейронных сетей для разных периодов работы ПО. Первая нейронная сеть должна описывать фазу запуска. Также предлагается выделить некоторое характерное для ПО действие и обучить нейронную сеть на это действие, что даст вторую сеть. В дополнение к двум сетям, обученным на запуск и характерную черту работы ПО, необходима третья сеть, обученная на завершение работы ПО, так как аномальное поведение или НДВ в целях маскировки может проявляться лишь под завершение работы ПО.

На практике сетей второго рода может быть несколько – столько же, сколько различных характерных действий имеет ПО. Под характерным действием понимается действие, которое совершает программа, например, по запросу пользователя. Или это может быть некоторая совершаемая периодически процедура (например, проверка целостности исполняемого файла). Иными словами, характерная черта – это то, что отличает данное ПО ото всех остальных. Дело в том, что запуск и завершение работы некоторых программ очень похожи друг с другом, так как при

запуске используют одинаковые стандартные системные библиотеки, а при завершении работы те же самые системные библиотеки выгружается, что порождает ряд одинаковых событий и, следовательно, одинаковые изменения координат описывающего вектора.

Таким образом, встаёт задача разделения всей работы ПО на некоторые фазы: фазу запуска, фазы специфических действий и фазу завершения работы. Ниже будут рассмотрены математические механизмы определения конца одной и начала другой фазы. Однако для того, чтобы эти механизмы были применимы, следует подготовить информацию о работе ПО в другом виде. На данный момент мы обладаем вектором координат, который описывает работы ПО от запуска до завершения. Это не включает в себя информацию об изменении координат во времени, которая необходима для разделения работы ПО на фазы.

Для разрешения этой проблемы предлагается следующий подход при отслеживании действий ПО: создание и хранение в памяти вектора, накопленного за каждую секунду работы ПО. Если ПО проработало в общей сложности m секунд, мы будем располагать m векторами, что позволит отслеживать активность ПО в каждую секунду, что, в свою очередь, поможет разграничить фазы работы ПО.

Приведём пример представления работы ПО набором таких односекундных векторов. Для наглядности будем считать, что вектор содержит всего 5 координат, описывающих работу ПО. Например, 1-я координата показывает число событий чтения системных файлов, 2-я – число событий записи данных на жёсткий диск и т.д. Предположим, что ПО проработало 16 секунд. Таким образом, мы имеем 16 односекундных векторов. Выстраивая эти вектора в хронологическом порядке, получаем детальную картину работы ПО, представленную на рис. 2.

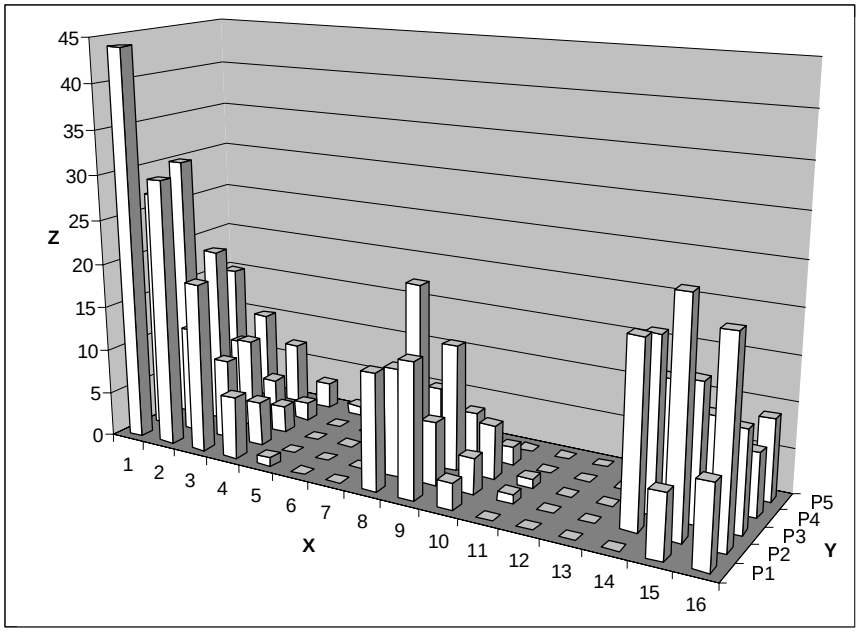

Рис. 2. Представление работы ПО набором односекундных векторов

По оси X отложено время работы ПО в секундах. По оси Y представлены координаты вектора. Ось Z показывает значения координаты. Представление работы ПО в таком графическом виде наглядно разделяет весь жизненный цикл ПО на три

фазы. В примере фаза запуска длится 4-5 секунд, фаза работы – 3-4 секунды, фаза завершения работы – 3 секунды.

Способы разделения работы ПО на фазы

Представив работу ПО набором односекундных векторов, мы получаем числовые последовательности, показывающих значение каждой координаты вектора в определённый момент времени. Это является матрицей, пример который приведён в табл. 1, где координаты вектора записаны в столбцах.

Таблица 1

Матрица односекундных векторов

44	30	19	7	1	0	0	13	15	3	0	0	0	0	7	9
27	12	9	5	0	0	0	12	7	4	1	0	0	20	25	22
30	20	10	3	0	0	0	20	14	6	1	0	0	19	15	11
13	8	4	2	0	0	0	7	5	2	0	0	0	13	10	7
15	10	7	3	1	0	0	0	0	0	0	0	0	8	7	9

Воспользовавшись несложными математическими методами, с помощью этих последовательностей можно выделить фазы работы ПО. После того, как фаза выделена, все односекундные вектора, входящие в данную фазу, можно просуммировать и получить обучающий вектор для данной фазы, что, в свою очередь, даст возможность создать нейронную сеть, распознающую поведение в данной фазе.

Для слежения за активностью ПО следует использовать Гёльдеровы нормы односекундных векторов [4]. Гёльдерова норма порядка p для n-мерного вектора вычисляется по формуле [3]:

$$\left\|X\right\|_p = \sqrt[p]{\sum_{i=1}^{n}\left|x_i\right|^p}. \quad (1)$$

В данной статье рассмотрим нормы 1-го, 2-го и бесконечного порядков:

$$\left\|X\right\|_1 = \sum_{i=1}^{n}\left|x_i\right|, \quad (2)$$

$$\left\|X\right\|_2 = \sqrt{\sum_{i=1}^{n}\left|x_i\right|^2}, \quad (3)$$

$$\left\|X\right\|_{\inf} = \max\left|x_i\right|. \quad (4)$$

То есть активность поведения ПО в некоторый момент времени определяется нормой односекундного вектора для данного момента времени. Таким образом, последовательность односекундных векторов превращается в другую числовую последовательность — последовательность норм векторов.

Например, данные, представленные на рис. 2, после вычисления нормы второго порядка (2) будут иметь вид, представленный в табл. 2.

После того, как данные о работе ПО переведены в нормы, легко увидеть время простоя, когда ПО не было занято, и, следовательно, различные фазы работы. На основании табл. 1 можно говорить о трёх фазах, которые имели место с 1-й по 5 секунду, с 8-й по 11-ю и с 14-й по 16-ю. Те же самые выводы были получены для исходных данных, показанных на рис. 2.

Таблица 2

Представление работы ПО с помощью норм

сек.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Норма $\|\|X\|\|_2$	62,9	40,1	24,6	9,8	1,4	0	0	27,6	22,2	8,1	1,4	0	0	31,5	32,4	28,6

Некоторой проблемой при разделении работы ПО на фазы является наличие шума, то есть некоторых мелких действий ПО, которые не зависят ни от пользователя и выполняются периодически и автоматически. Так например, ПО может раз в определённый интервал пытаться читать данные из сетевого стека TCP/IP, но при этом не находить новых данных. В таком случае будет отмечено единичное обращение. Так как оно не повлекло за собой никаких других последствий, не имеет смысла отмечать это как фазу работы. Такой шум в наборе координат разумнее всего игнорировать. Эти рассуждения приводят к тому, что следует ограничивать фазу не в тот момент, когда норма вектора стала равно 0, а когда она стала ниже некоторой константы *C*. Величина константы должна подбираться экспериментально в зависимости от выбранной системы мониторинга. Чем более подробную информацию о работе ПО может предоставлять система мониторинга, тем больше значение константы *C*. Это связано с тем, что высокая подробность о работе ПО ведёт к большому числу узконаправленных характеристик, каждая их которых занимает координату. Чем больше координат, тем больше может быть итоговый шум, поэтому, осуществив выбор используемых координат, следует экспериментальным образом определить величину *C* для каждого порядка нормы: C_1, C_2 и $C_{\inf}$.

Ещё одним способом разграничения фаз является анализ скорости изменения односекундных векторов. Для этого следует пересчитать элементы изначальной матрицы, представленной в табл. 1. Пересчёт элементов происходит по следующей формуле:

$$a^*_{i,j} = a_{i,j} - a_{i-1,j-1}. \quad (5)$$

В итоге получится матрица, показывающая ускорение каждой координаты в каждый промежуток времени. В случае с матрицей из табл. 1 получится матрица, представленная в табл. 3.

Построив последовательность норм для каждого вектора в такой матрице, можно также судить о начале или конце фазы работы (табл. 4).

Таблица 3

Матрица ускорения координат

44	-14	-11	-12	-6	-1	0	13	2	-12	-3	0	0	0	7	2
27	-15	-3	-4	-5	0	0	12	-5	-3	-3	-1	0	20	5	-3
30	-10	-10	-7	-3	0	0	20	-6	-8	-5	-1	0	19	-4	-4
13	-5	-4	-2	-2	0	0	7	-2	-3	-2	0	0	13	-3	-3
15	-5	-3	-4	-2	-1	0	0	0	0	0	0	0	8	-1	2

Таблица 4

Представление ускорения координат с помощью норм

сек.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Норма $\|X\|_2$	62,9	23,9	16,0	15,1	8,8	1,4	0	27,6	8,3	15,0	6,9	1,4	0	31,5	10	6,5

Преимуществом данного подхода является то, что он может выявить фазы работы поверх существенного шума, который может быть обусловлен специфической чертой ПО (например, постоянное обращение к жёсткому диску). Без такого подхода для каждого ПО приходилось бы считать уровень постоянного шума и затем учитывать его при делении работы на фазы.

Также следует учитывать, что некоторые координаты могут быть с точки зрения обеспечения безопасности, важнее остальных. Например, события записи данных в системный реестр имеют большую важность, чем событие открытия текстового файла на чтение. Следовательно, может возникнуть необходимость ввода веса для каждой координаты, что приведёт формулу (1) вычисления нормы порядка p к следующему виду:

$$\|X\|_p = \sqrt[p]{\sum_{i=1}^{n} k_i |x_i|^p}, \quad (6)$$

где k_i – некоторая константа, подбираемая для каждой координаты в зависимости от её важности.

Выбор способа разделения работы ПО на фазы

Как было сказано выше, выбор способа разделения работы ПО на фазы сильно зависит от подробности и количества характеристик, которыми описывается поведение ПО, а также от самого ПО. В большинстве случаев метод представления работы ПО матрицей ускорений координат не требуется, так как постоянный шум (то есть, постоянная активность) характерен для системных ПО и служб, для которых задача моделирования работы решена. Прикладные пользовательские пакеты, на моделирование работы которых нацелены методы этой статьи, большую часть своего времени проводят в стадии ожидания. Поэтому для разделения работы ПО на фазы следует использовать обычную матрицу односекундных векторов (табл. 1). Остаётся только выбрать порядок нормы, которой будет оцениваться активность ПО в каждый момент времени. Это полностью зависит от под-

робности и общего числа характеристик, которыми описывается поведение ПО. Например, если для описания поведения есть всего 5 характеристик, следует выбирать норму бесконечного порядка (формула (4)). Такой выбор объясняется тем, что маленькое количество координат означает, что каждая из них имеет большое значение при описании поведения. Следовательно, для отслеживания фазы хватит максимального значения координат вектора в данный момент времени. Уровень шума, который можно игнорировать, следует приравнять минимальному значению координаты, так как при маленьком числе координат даже минимальное изменение координаты может говорить о начале новой фазы.

Чем больше число характеристик, описывающих поведение ПО, тем более узконаправленными они становятся, тем меньшую роль играют минимальные изменения в какой-нибудь отдельно взятой координате. Поэтому тем ниже должен быть порядок нормы.

В ходе написания данной статьи была выбрана модель поведения ПО с 53 координатами. Они показаны в табл. 5 (для экономии места представлены группами группы).

Таблица 5

Координаты, составляющие вектор

№ координат	Описание координат
1 – 20	Работа с файловой системой (системные файлы, пользовательские, сетевые, файлы приложений, съёмных носителей)
20 – 24	Работа с библиотеками (запись, открытие, линковка)
25 – 26	Работа с конфигурационными файлами
27 – 30	Работа с документами офисных пакетов и мультимедиа файлами
30 – 36	Работа с системными ключами и значениями реестра (разделение таких веток, как hklm\software, hkcu\software и т.д.)
37 – 46	Установление или принятие TCP и UDP соединений, передача данных
47 – 53	Запуск процессов от имени моделируемого ПО, отношения с другими процессами, характеристики процесса

Для такого набора координат самым удачным оказался выбор нормы второго порядка. Константа для уровня шума C при этом взята равной 3. В таком случае роль шума снижена по сравнению с нормой первого порядка, если шум не превышает значения 1, так как единица, возведённая в квадрат, остаётся единицей. Иными словами, при измерении активности нормой второго порядка данные вектора $X = \{1; 0; 0; \ldots 0; 0; 1; 1; 1\}$ будут сочтены за шум ($\|X\| = \sqrt{4} < 3$), что верно в случае 53 координат. Однако в случае нормы первого порядка это было бы расценено как начало новой фазы ($\|X\| = 4 > 3$). Для того, чтобы не пропустить наиболее критичные события, как шум, некоторым координатам (описывающим события записи, запуска и принятия данных по сети) присвоен вес 2.

Таким образом, этот механизм позволяет выделить среди данных о работе ПО определённые фазы, суммирующий вектор которых затем используется для обучения соответствующих нейронных сетей. Чтобы избежать повторного обучения нейронной сети на одно и ту же фазу, следует пропускать вектор для новой обнаруженной фазы через нейронные сети для существующих фаз. В случае высокой

вероятности соответствия считается, что нейронная сеть для этой фазы уже существует, однако её можно дополнительно обучить полученным вектором.

Анализ работы ПО и выявление аномалий

Процесс анализа и выявления аномального поведения ПО использует методы, применяемые при построении модели поведения. Так при запуске ПО начинают накапливаться односекундные векторы работы, чтобы в любой момент была возможность определить конец некоторой фазы и получить общий вектор за определённый период работы ПО (например, вектор для фазы запуска). Такие общие векторы пропускаются через соответствующую нейронную сеть модели процесса. Результаты, близкие к 1, на всех нейронных сетях модели говорят о соответствии работы ПО его модели. Если выход на какой-нибудь сети оказался значительно меньше 1, значит, в той фазе, которую данная сеть описывает, обнаружено аномальное поведение. Аномальное поведение, в свою очередь, указывает на НДВ или превышение полномочий пользователем данного ПО.

Так как есть вероятность наличия в модели ПО нескольких сетей второго рода, следует предусмотреть возможность создания условий прохождения этих сетей. ПО может иметь несколько характерных действий во время работы, однако, не все эти характерные действия могут проявляться одновременно за один жизненный цикл. В таком случае с помощью логических операторов И/ИЛИ можно описать возможный порядок проявления характерных действий ПО, если имеются такие экспертные данные. Если такие данные недоступны, по умолчанию следует выбирать логический оператор ИЛИ для всех характерных действий. Тогда заключение о соответствии работы ПО его эталонной модели будет установлено в том случае, если результат, близкий к 1, будет достигнут хотя бы на одной сети второго рода.

Заключение

Предоставленный механизм обнаружения аномалий с помощью нейронных сетей имеет ряд преимуществ и недостатков. Основным преимуществом является то, что он предлагает модель поведения, которая разделяет всю работу ПО на фазы и тем самым описывает поведение ПО на протяжении всего рабочего цикла. Также преимуществом является то, что средства создания модели процесса не зависят от операционной системы или платформы, на которой запускается ПО. Тем самым, построив профиль поведения некоторого ПО один раз, можно использовать для любых СВТ, на которых работает смоделированное ПО. Также с применением некоторой экспертной оценки для любого ПО может быть подобран свой набор координат, учитывающий особенности данного ПО, и эти особенности будут заложены в эталонную модель поведения, что значительно повысит её эффективность. Кроме того, предоставленный механизм построения эталонных профилей поведения учитывает работу ПО на протяжении всего жизненного цикла. Немаловажно, что размеры получаемых сетей относительно небольшие (до 150 нейронов), поэтому обработка вектора реального поведения ПО при современных вычислительных мощностях будет производиться почти моментально. То есть обнаружение аномалии происходит почти сразу же после того, как она произошла. Учитывая, что нейронных сетей в одном профиле и самих профилей может быть много, при необходимости возможно дополнительно ускорить вычислительные возможности сетей [8].

Основным недостатком на данный момент является процесс обучения, а именно стадия сбора обучающих векторов. При размерности вектора около 40 ко-

ординат число обучающих векторов для нейронной сети любого рода согласно теории [7] должно быть порядка 10000, однако такое количество учебных прогонов ПО не всегда возможно (и нужно) осуществить [9]. Здесь практика идёт в разрез с теорией, и число обучающих векторов может быть на порядок меньше. Прежде всего это связано со спецификой задачи, так как основная цель обучающего набора заключается в том, чтобы описать как можно больше различных вариантов запуска (например, с различными вариантами аргументов запуска), а не один и тот же вариант запуска по много раз. Тем не менее, автоматизированное построение профиля поведения ПО потребует некоторого времени (до нескольких часов) и участия пользователя, а также, желательно, экспертных знаний, что может затруднить процесс построения моделей для всего имеющегося в наличии ПО.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Ясницкий Л.Н.* Введение в искусственный интеллект. – М: Издательский центр “Академия”, – М., 2005. – 176 с.

2. *Kevin Swingler* Applying Neural Networks. A practical Guide, Pap/Dsk edition, Morgan Kaufmann, 1996. – P. 53 – 55.

3. *Дынкин Е.Б., Юшкевич А.А.* Управляемые марковские процессы и их приложения. – М., 1975. – 36 с.

4. *Бриллинджер Д.* Временные ряды. Обработка данных и теория. – М: Изд-во Мир, 1980. – С. 417 – 420.

5. *Harris Drucker, Yann Le Cun* Improving Generalization Performance Using Backpropagation, IEEE Transactions on Neural Networks, Vol.3, N5, 1992. – P. 991 – 997.

6. *Malki H.A., Moghaddamjoo A.* Using the Karhunen-Loe`ve Transformation in the Back-Propagation Training Algorithm, IEEE Transactions on Neural Networks, Vol.2, N1, 1991. – P. 162 – 165.

7. *Paul J. Werbos* Backpropagation Through Time: What It Does and How to Do It, Artificial Neural Networks: Concepts and Theory, IEEE Computer Society Press, 1992. – P. 309-319.

8. *Крисилов В.А., Олешко Д.Н., Лобода А.В.* Методы ускорения нейронных сетей // Вестник СевГТУ. Информатика, электроника, связь. – Одесса, 2001. – Вып. 32 – С. 19.

9. *Alain Petrowski, Gerard Dreyfus, Claude Girault* Performance Analysis of a Pipelined Backpropagation Parallel Algorithm, IEEE Transactions on Neural Networks, Vol.4, N6, 1993. – P. 970 – 981.

Жилкин Сергей Дмитриевич
Московский инженерно-физический институт.
E-mail: z-h-i-l-k-i-n@yandex.ru.
115409, г. Москва, Каширское ш., 31.
Тел.: 8 (499) 127-24-50.
Кафедра 43 «Стратегические информационные исследования»; аспирант.

Zhilkin Sergey Dmitrievich
Moscow Engineering Physics Institute.
E-mail: z-h-i-l-k-i-n@yandex.ru.
31, Kashirskoe schosse, Moscow, 115409, Russia.
Phone: 8 (499) 127-24-50.
Chair 43 «Strategic informational researches»; post-graduate student.