Уфа : УГАТУ, 2010 Вестник УГАТУ Т. 14, № 5 (40). С. 198–204

УДК 681.5:004.45

С. С. Валеев, М. Ю. Дьяконов

НЕЙРОСЕТЕВАЯ СИСТЕМА АНАЛИЗА АНОМАЛЬНОГО ПОВЕДЕНИЯ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ В МИКРОЯДЕРНОЙ ОПЕРАЦИОННОЙ СИСТЕМЕ

Предлагается архитектура нейросетевой системы анализа поведения процессов операционной системы реального времени, используемых в специализированных встраиваемых системах. Распознавание аномального поведения процессов производится на базе нейросетевого классификатора. *Информационная безопасность*; *аномальное поведение*; *микроядерная операционная система*; *нейросетевой классификатор*; *встраиваемые системы*; *вредоносный код*

Бортовые компьютеры в системах управления динамическими объектами, коммуникаторы и смартфоны, платежные терминалы и банкоматы, вычислительные интегрированные комплексы технических средств охраны обеспечивают свою функциональность на основе операционных систем реального времени (ОСРВ). В качестве ОСРВ широко используются операционные системы на основе микроядерной архитектуры – микроядерные операционные системы (МОС). Широко известными примерами МОС являются ОСРВ LynxOS, QNX, VxWorks, Symbian, PalmOS, Windows CE [1–6]. МОС обладают рядом преимуществ по сравнению с монолитными ОС, к которым относится, например, ОС UNIX [7]. К этим преимуществам можно отнести высокую надежность, невысокие требования к объему памяти, а модульная структура этих ОС позволяет легко масштабировать и конфигурировать их с учетом особенностей решаемой задачи.

Требования к ОСРВ регламентируются различными стандартами, в том числе и стандартами по защите информации, так как в настоящее время активно используются распределенные системы обработки информации с применением сетевых технологий [8–11]. Основные угрозы безопасности для автоматизированных систем (АС), в том числе и для встраиваемых систем реального времени представлены на рис. 1 [12]. Для подобных систем характерно, что известен состав программного обеспечения (ПО) и круг решаемых задач.

Для противодействия этим угрозам применяются различные методы повышения защищенности встраиваемых систем: использование верифицированных компонентов ПО [8–10]; разграничение доступа процессов к ресурсам [9], сертификация компонентов системного ПО ОС в соответствии с требованиями стандартов [8–10]; во многих случаях применяются средства защиты, характерные для ОС общего назначения (антивирусы, межсетевые экраны и др.) [14]. Решается задача разработки интеллектуальной системы защиты информации на основе анализа вычислительных процессов в ходе решения ОСРВ известного набора задач. Предлагается система обнаружения аномального поведения процессов в МОС на основе классификации процессов на нормальные и аномальные с применением нейросетевого классификатора.

1. СОСТОЯНИЕ ВОПРОСА

Как показал анализ, в известных подходах обеспечения информационной безопасности ОСРВ не учитывается поведение вычислительных процессов в ходе решения известного набора задач. Если известны направления реализации угроз, например, при заражении компьютерным вирусом, переполнении буфера в результате ошибки в коде программы и др., то в некоторых случаях это позволяет достаточно точно определять отклонения в работе ОСРВ.

При использовании известных подходов обнаружения, например, вредоносного кода, средства защиты оперируют информацией, содержащейся в базе данных сигнатур вредоносного кода. В других случаях используется определенный набор запрещающих или разрешающих правил, так называемые эвристики. Эвристики позволяют ограничить воздействие вредоносного кода на вычислительную систему, а также во многих случаях обнаружить такое воздействие. Недостатком известных подходов является то, что в случае несвоевременного обновления базы данных сигнатур возрастает количество уязвимостей в системе и, следовательно, вероятность успешной атаки. При большом числе эвристических правил возрастает количество ошибок второго рода.

Контактная информация: (347) 273-78-76

Рис. 1. Основные угрозы безопасности АС

При использовании предлагаемого подхода используются не фиксированные уникальные последовательности байтов вредоносного кода и наборы правил, а предположение о том, что статистика последовательности системных вызовов процессов и их частотные характеристики остаются в среднем постоянными и любые отклонения от нее могут быть расценены как аномальное поведение. В дополнение к существующим методам, данный подход, например, может использоваться для уведомления о принудительной дополнительной проверке кода процесса на наличие вредоносного кода.

2. ПОСТАНОВКА ЗАДАЧИ

Рассмотрим далее задачу распознавания состояний процессов ОСРВ. В табл. 1 представлены основные характеристики модели вычислительной системы ОСРВ, требуемые для решения задачи распознавания состояний процессов.

Рассмотрим обобщенную модель вычислительной системы, которую представим в виде множества вычислительных процессов *P*. Для реализации определенной функциональности системы каждый вычислительный процесс использует системные вызовы ОСРВ из *S*. Предполагается, что в зависимости от типа вычислительного процесса системные вызовы совершаются с характерной для них частотой, образуя множество *V*. Предполагается, что множество системных вызовов *S*, а также соответствующее множество частот системных вызовов *V* вычислительного процесса остаются постоянными при многократном запуске системного процесса.

Таблица 1

Основные характеристики модели вычислительного процесса ОСРВ

Характеристика модели	Описание
$P=\{p_k \mid k=1\div j\}$	Множество пользовательских процессов, где j – число процессов
$S=\{s_i \mid i=1\div n\}$	Множество системных вызовов, где n – число системных вызовов
$V=\{v_m\}$	Множество частот системных вызовов пользовательских процессов, где $m\in N$
$G: P\times S\times S\to V$	Функция формирования матрицы смежности системных вызовов
$H: P\times S\times S\to V$	Функция формирования матрицы частот последовательных системных вызовов
$I=\left\{v_{ml}\middle\vert \begin{matrix} v_{ml}\in V; m, \\ l=1\div n \end{matrix}\right\}$	Матрица смежности системных вызовов пользовательского процесса p_k, где v_{mm} – частота совершения вызова s_m
$J=\left\{v'_{ml}\middle\vert \begin{matrix} v'_{ml}\in V; m, \\ l=1\div n \end{matrix}\right\}$	Матрица частот системных вызовов пользовательского процесса p_k, где v'_{ml} – частота совершения вызова s_l после s_m

Поставим задачу сбора статистики совершения системных вызовов вычислительного процесса S и соответствующих частот системных вызовов V, а также последующего анализа состояния процессов ОСРВ на базе данной статистики.

Пусть P, S, V, I, J – множества, характеризующие нормальное состояние ОСРВ, полученные при многократном запуске вычислительных процессов в системе до момента времени t, а множества P', S', V', I', J' отражают новое состояние системы в момент времени $t_1 > t$. Таким образом, предполагается, что для анализа состояния процессов ОСРВ может оказаться достаточным сравнение множеств I', J' с множествами I, J.

В итоге, задача анализа состояний процессов ОСРВ по совокупности системных вызовов и их частотных характеристик соответствующих вычислительных процессов сводится к задаче распознавания образов. Задача может быть эффективно решена на базе нейросетевого классификатора.

3. МЕТОДИКА ИССЛЕДОВАНИЯ

Системная организация микроядерной ОС может быть представлена в виде иерархической структуры, в которой можно выделить три функциональных уровня: уровень исполнения, уровень координации и уровень планирования. К исполнительному уровню отнесены устройства ПК и микроядро ОС (рис. 2), взаимодействующие на данном уровне посредством команд прерывания внешних устройств, команд ввода/вывода, команд обмена информацией между компонентами микроядра. Взаимодействие микроядра с вышестоящим уровнем координации обеспечивается посредством системных вызовов; к уровню координации отнесены системные библиотеки, драйверы устройств, менеджеры файловой системы и процессов (см. рис. 2), взаимодействующих с вышестоящим уровнем планирования с использованием интерфейса прикладного программирования; к уровню планирования отнесено прикладное ПО пользователя (см. рис. 2), которое получает команды и данные для обработки от пользователя.

Каждый из иерархических уровней ОСРВ решает свои задачи:

- на уровне планирования (УП) прикладное ПО планируется к выполнению в зависимости от первоначальных установок пользователя;
- на уровне координации (УК) загруженное в память прикладное ПО определяет состояние ОСРВ с использованием прикладного программного интерфейса. Менеджеры управления ресурсами ОСРВ и драйверы устройств данного уровня обслуживают запросы прикладного ПО, получаемые с уровня планирования;
- на исполнительном уровне (ИУ) взаимодействие прикладных программ с уровнем координации обеспечивается благодаря выполнению низкоуровневых системных вызовов. Количество системных вызовов в микроядерных ОС может изменяться от 70 до 250, в зависимости от ее назначения и задач, решаемых разработчиком [15].

На уровне исполнения МОС за неделимую единицу вычислительного процесса примем процесс или поток. Информация о процессах и потоках представляется в МОС структурой данных в виде таблицы с набором полей, идентифицируемой МОС и хранящей информацию о текущем состоянии процесса или потока [16].

Рис. 2. Иерархические уровни микроядерной ОСРВ

Рис. 3. Архитектура системы анализа состояния МОС на основе поведения процессов

Эти таблицы используются при планировании процессов на исполнение и при управлении процессорным временем, а также, например, переключении контекста [12].

В нашем случае предполагается, что последовательность системных вызовов в ходе выполнения вычислительных процессов, а также частота их использования остаются постоянными при повторном запуске той или иной задачи.

Предлагаемый метод анализа состояния МОС с применением нейронной сети был апробирован в МОС Minix 3 [13]. На рис.3 представлена архитектура разработанной системы анализа состояния МОС на основе нейросетевого классификатора процессов.

Функционирование системы анализа поведения процессов МОС описывается следующей последовательностью шагов:

Шаг 1. Некоторый процесс из множества *P* совершает системный вызов из *S*.

Шаг 2. Обработчик передачи сообщений (ОПС) блокирует прохождение сообщения к обработчику системных вызовов (ОСВ) и передает идентификатор процесса (PID) модулю сбора статистики (МСС).

Шаг 3. МСС осуществляет проверку наличия в базе статистики данных о процессе.

Шаг 3.1. В случае наличия данных МСС передает информацию о процессе в модуль анализа состояния процесса (МАСП) и переходит к шагу 4.

Шаг 3.2. В случае отсутствия данных МСС заводит новую запись о процессе и функционирует в режиме сбора статистики для процесса *M* и переходит к шагу 7.

Шаг 4. МАСП формирует входной вектор состояния процесса из данных, содержащихся в системных таблицах ОС, и передает на вход нейросетевого модуля классификации.

Шаг 5. Нейросетевой модуль передает результаты классификации в МАСП.

Шаг 6. МАСП формирует код ошибки и передает результат в МСС.

Шаг 7. МСС добавляет запись о состоянии процесса кодом ошибки и передает код ошибки ОПС.

Шаг 7.1. В случае возникновения ошибки ОПС пересылает код ошибки процессу *P*, блокируя выполнение системного вызова из *S* и переходит к шагу 10.

Шаг 7.2. В случае, если код ошибки равен нулю (т.е. отсутствие ошибки), ОПС пересылает аргументы системного вызова ОСВ и переходит к следующему шагу.

Шаг 8. ОСВ исполняет системный вызов из *S* и возвращает результаты обработки ОПС.

Шаг 9. ОПС возвращает результаты системного вызова из *S* процессу из *P*.

Шаг 10. ОПС ожидает совершения следующего системного вызова.

На основе предложенного алгоритма анализа реализован исследовательский прототип для ОС Minix 3. В качестве задач были выбраны наиболее часто используемые системные утилиты (см. табл. 2).

В табл. 3 представлены некоторые стандартные библиотечные функции, реализующие системные вызовы ОС Minix 3.

Для каждой из выбранных утилит при обращении к ним был выполнен анализ частоты используемых системных вызовов. Результаты численного эксперимента представлены в табл. 4, а на рис. 4 представлены гистограммы частот использования системных вызовов для утилит getty, init, rm.

Как следует из результатов численного эксперимента для ОС Minix 3, при запуске системных утилит используется определенный набор системных вызовов. В ходе численного эксперимента также учитывалась взаимосвязь пар системных вызовов, то есть частота появления следующего системного вызова.

Таблица 2

Список использованных утилит ОС Minix 3

Имя системной утилиты	Краткое описание назначения утилиты
su	Повышение пользовательских привилегий до уровня root (администратора)
init	Начальная загрузка всех служб и процессов-демонов
rm	Удаление файлов из файловой системы
getty	Запуск терминала пользователя
login	Аутентификация пользователя
more	Просмотр содержимого файлов
cp	Копирование файлов в файловой системе
ls	Просмотр содержимого каталога файловой системы
mem	Информация о свободном пространстве ОЗУ

Таблица 3

Основные функции, реализующие системные вызовы [13]

Системный вызов	Описание	№
exit(status)	Завершает выполнение процесса и возвращает статус	1
pid = fork( )	Создает дочерний процесс, идентичный родительскому	2
n = read(fd, buffer, nbytes)	Читает данные из файла в буфер	3
n = write(fd, buffer, nbytes)	Пишет данные из буфера в файл	4
fd=open(file, how,...)	Открывает файл для чтения, записи или того и другого	5
s = close(fd)	Закрывает открытый файл	6
…	…	…
s = sigaction (sig, &act, &oldact)	Устанавливает реакцию на сигнал	71
s = sigsuspend(sigmask)	Устанавливает маску сигналов для процесса и приостанавливает его	72
s = sigpending(set)	Определяет набор блокированных сигналов	73
s = sigprocmask(how, &set, &old)	Определяет или устанавливает маску сигналов для процесса	74
s = sigreturn(&context)	Возвращается из обработчика сигнала	75

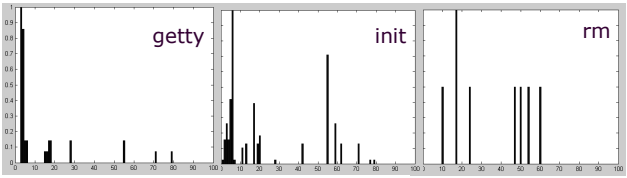

Рис. 4. Гистограммы частот использования системных вызовов

Полученные матрицы системных вызовов *I* представлены для наглядности в графическом виде (см. рис. 5). Как видно из рисунка, для каждого из рассмотренных заданий характерно уникальное множество используемых системных вызовов. Моделирование показало, что данное множество остается в среднем постоянным при многократном запуске соответствующих заданий, а влияние на значения элементов матрицы оказывает входной поток данных.

Таблица 4

Частоты использования системных вызовов утилитами

№	Наименование утилиты							
	su	init	rm	getty	lo-gin	more	cp	ls
	Частота совершения системного вызова							
1	–	1	–	–	–	–	–	–
2	–	6	–	–	1	–	–	–
3	2	10	–	14	38	897	2	3
4	–	6	–	12	10	1178	1	2
5	2	16	–	2	10	2	2	1
6	2	38	–	2	15	2	2	1
7	–	1	–	–	–	–	–	–
10	–	–	1	–	–	–	–	–
11	–	4	–	–	1	–	–	–
12	–	–	–	–	1	–	–	–
13	–	5	–	–	1	–	–	–
15	–	–	–	1	1	–	–	–
16	–	–	–	1	1	–	–	–
17	1	15	2	2	4	9	2	3
18	–	–	–	2	4	–	3	1
19	–	5	–	–	2	1	-	-
20	–	7	–	–	–	–	-	-
23	1	–	–	–	1	–	-	-
24	1	–	1	–	–	–	1	1
27	–	–	–	–	2	–	–	–
28	–	1	–	2	5	1	1	1
33	–	–	–	–	1	–	–	1
42	–	5	–	–	1	–	–	–
46	1	–	–	–	1	–	–	–
47	1	–	1	–	–	–	1	1
50	–	–	1	–	–	–	–	1
54	–	–	1	–	–	300	1	3
55	4	27	–	2	11	1	–	2
59	–	10	–	–	–	–	–	–
60	–	–	1	–	–	–	1	–
62	–	5	–	–	–	–	–	–
68	–	–	–	–	–	–	–	–
71	–	5	–	1	25	5	–	–
74	–	–	–	–	–	1	–	–
77	–	1	–	–	–	–	–	–
79	–	1	–	1	–	–	–	–

Решение задачи анализа состояния ОСРВ основано на обучении нейросетевого классификатора. В качестве нейросетевого классификатора использовалась многослойная нейронная сеть [17].

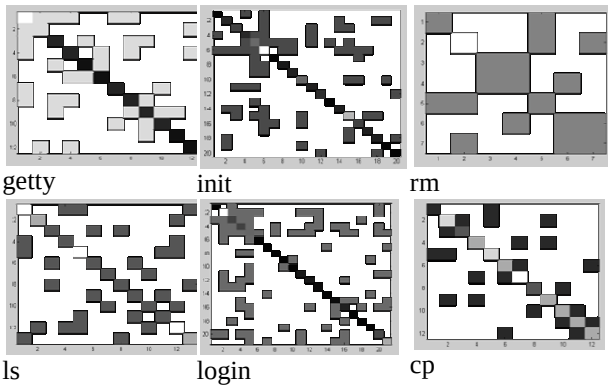

Рис. 5. Графическое представление матриц системных вызовов

ВЫВОДЫ

В результате выполненных исследований показано, что поведение процессов в МОС характеризуется определенным уникальным набором системных вызовов с определенными частотными характеристиками их использования. Применение нейросетевого классификатора оказывается достаточно эффективным для задачи распознавания последовательности системных вызовов процессов и их частотных характеристик, ввиду простоты реализации и скорости работы. Архитектура нейросетевого классификатора является подходящей для его использования в МОС, используемых во встраиваемых системах для распознавания аномального поведения процессов в режиме реального времени.

СПИСОК ЛИТЕРАТУРЫ

1. The real-time operating system for complex embedded systems [Электронный ресурс] (http://www.lynuxworks.com/rtos/rtos.php).

2. QNX Neutrino RTOS Secure Kernel [Электронный ресурс] (http://www.qnx.com/products/neutrino_rtos/ secure_kernel.html).

3. Introducing New Solutions for Critical Business Needs from the World's Most Widely Adopted RTOS [Электронный ресурс] (http://www.windriver.com/solutions/network-equipment/).

4. Symbian [Электронный ресурс] (http://www.symbian.com/).

5. Palm OS [Электронный ресурс] (http://ru.wikipedia.org/wiki/PalmOS).

6. Windows CE [Электронный ресурс] (http://ru.wikipedia.org/wiki/Windows_CE).

7. **Вахалия, Ю.** UNIX изнутри. Классика CS. СПб.: Питер, 2003. 844 с.

8. DO-178B, Software Considerations in Airborne Systems and Equipment Certification.

9. ARINC 653 (Avionics Application Standard Software Interface).

10. Common Criteria for Information Technology Security Evaluation (Общие критерии оценки безо-

пасности информационных технологий) — международный стандарт (ISO/IEC 15408, ИСО/МЭК 15408-2002).

11. ГОСТ Р 51904-2002 «Программное обеспечение встроенных систем. Общие требования к разработке и документированию».

12. **Гайдамакин Н. А.** Теоретические основы компьютерной безопасности: учеб. пособие. Екатеринбург: изд-во Урал. ун-та, 2008. 212 с.

13. **Таненбаум Э.** Операционные системы. Разработка и реализация (+CD). Классика CS. СПб.: Питер, 2007. 704 с.

14. **Warrender C., Forrest S., Pearlmutter B.** Detecting Intrusion Using System Calls: Alternative Data Models // IEEE Symposium on Security and Privacy. 1999. P. 133–145.

15. **Forrest S., Hofmeyr S. A., Longstaff T. A.** A sense of self for UNIX processes // Proceedings of the 1996 IEEE Symposium on Security and Privacy. P. 120-128, Los Alamos, CA, 1996. IEEE Computer Society Press.

16. Anomaly detection using call stack information / H. Feng [et al] // Proceedings of IEEE Symposium on Security and Privacy, Berkley, California, IEEE Computer Society, 2003.

17. **Хайкин С.** Нейронные сети: полный курс. М.: ООО «И.Д. Вильямс», 2006. 1104 с.

ОБ АВТОРАХ

Валеев Сагит Сабитович, проф., зав. информатики. Дипл. инж.-электромех. (УАИ, 1970). Д-р техн. наук. по упр. в техн. системах (УГАТУ, 2005). Иссл. в обл. интеллект. упр. сложными объектами.

Дьяконов Максим Юрьевич, асс., асп. каф. вычисл. техн. и защ. инф. Дипл. спец. по защите инф. (Уфа, 2007). Исс. в обл. интеллектуальн. систем защиты инф-ции в операц. системах.