Научная статья
УДК 004.491
DOI 10.35266/1999-7604-2024-1-4

МОДЕЛЬ ИДЕНТИФИКАЦИИ АГРЕССИВНЫХ ПРОГРАММНЫХ ЭЛЕМЕНТОВ

Ангелина Игоревна Дубровина[1]✉, Теймур Имран Оглы Акперов[2], Татьяна Сергеевна Александрова[3]
[1, 2, 3]Южный университет (ИУБиП), Ростов-на-Дону, Россия
[1]Донской государственный технический университет, Ростов-на-Дону, Россия
[1]ministrelia69@yandex.ru✉, https://orcid.org/0009-0005-8562-9389
[2]iakperov80@gmail.com, https://orcid.org/0009-0003-5028-6975
[3]tanysa08@yandex.ru, https://orcid.org/0000-0001-9852-0423

Аннотация. Рассмотрены вопросы моделирования вычислительных процессов, позволяющих оценить потенциальные возможности используемых программных средств по негативному влиянию на работу различных видов человеко-машинных систем, в том числе обладающих признаками искусственного интеллекта. В ходе численного эксперимента анализировались программы, предоставляющие интеллектуальную поддержку роботам-ассистентам преподавателей и реализующие определенные функции в вычислительном комплексе «умного дома», с учетом их агрессивного поведения. Обсужден ряд вопросов, связанных с результатами этого эксперимента. Показан подход, позволяющий выделить группу операторов машинного языка программирования, имеющих потенциал для формирования программных закладок общего и специального видов, влияющих на информационную экологию и корректную работу компонентов «умного дома». Даны рекомендации по формированию набора признаков агрессивности в зависимости от специфики применения конкретных программных средств.

Ключевые слова: разрушающее программное средство, программная закладка, агрессивный элемент, идентификация, безопасность

Для цитирования: Дубровина А. И., Акперов Т. И., Александрова Т. С. Модель идентификации агрессивных программных элементов // Вестник кибернетики. 2024. Т. 23, № 1. С. 31–36. DOI 10.35266/1999-7604-2024-1-4.

Original article

A MODEL DETECTING AGGRESSIVE SOFTWARE ELEMENTS

Angelina I. Dubrovina[1]✉, Teimur I. Akperov[2], Tatyana S. Aleksandrova[3]
[1, 2, 3]Southern University (IMBL), Rostov-on-Don, Russia
[1]Don State Technical University, Rostov-on-Don, Russia
[1]ministrelia69@yandex.ru✉, https://orcid.org/0009-0005-8562-9389
[2]iakperov80@gmail.com, https://orcid.org/0009-0003-5028-6975
[3]tanysa08@yandex.ru, https://orcid.org/0000-0001-9852-0423

Abstract. The article examines issues of modeling computation processes that help assess the probable capabilities of the software used to negatively affect the operation of various types of human-machine systems, including those with artificial intelligence. Given its aggresive behavior, a numerical experiment analyzed software that provides intelligent support to assistant robots for teaching staff and implements certain functions in the smart home's computation complex. The experiment's findings raise a number of issues, which are discussed. The study demonstrates an approach for highlighting a set of machine programming language statements capable of producing general and special software bugs that affect the information environment

and appropriate operation of smart home components. Guidelines for grouping a set of aggressive properties are given based on the specifics of the software in use.

Keywords: destructive software tool, software bug, aggressive element, identification, security

For citation: Dubrovina A. I., Akperov T. I., Aleksandrova T. S. A model detecting aggressive software elements. *Proceedings in Cybernetics.* 2024;23(1):31–36. DOI 10.35266/1999-7604-2024-1-4.

ВВЕДЕНИЕ

Определение агрессивных программных элементов в исследуемом программном средстве может быть осуществлено с помощью некоторой семантической модели, ориентированной на интерпретацию общих свойств программ, предположительно содержащих разрушающие функции. Основной задачей, решаемой такой моделью разрушающего программного средства (РПС), является выявление состояния вычислительного процесса, нарушающего безопасность и целостность программной среды в результате выполнения программы или отдельных ее участков. Кроме того, следует выделить и ряд дополнительных задач, способствующих решению основной задачи:

1) определение аномалий потока управления: «мертвого» (неисполняемого) [1] кода;

2) определение аномалий потока данных: побочного эффекта функций программного средства – изменения значений глобальных переменных в подпрограммах [1, 2], неинициализированных переменных.

Исходя из специфики задач будем использовать конструктивный подход в теории семантики языков программирования. В соответствии с конструктивной точкой зрения язык определяется тем действием, которое он оказывает на машину, т. е. языком называется множество таких программ, выполнение которых на машине имеет совершенно определенные последствия. Построим семантическую модель анализа программы, используя основные идеи Венского метода, разработанного сотрудниками Венской лаборатории фирмы IBM [3].

МАТЕРИАЛЫ И МЕТОДЫ

Введем основные соглашения и обозначения.

В общем случае будем использовать символы для обозначения следующих компонент программы:

x – переменные;

v – выражения;

b – логические выражения;

Q – операторы;

C – списки операторов.

Пусть X – множество всех возможных переменных, а Z – множество всех возможных значений переменных. Каждое из них может быть конечным или счетным. Неопределенное значение Λ также принадлежит Z: $\Lambda \in Z$.

Состояние памяти s определим как конечную функцию из X в Z. Множество всех значений памяти обозначим через S, а его элементы – символами s, s_1 и т. д. Множество S включает три подмножества состояний: $S_л$, $S_н$, $S\grave{}$ – соответственно легитимное, нелегитимное и неустойчивое состояние (одновременное присутствие $S_л$ и $S_н$).

Введем частичную функцию change:

$$\text{Change: } S \times X \times Z \to S,$$

такую, что для всех s, s_1, x и z:

$$s_1 = \text{change}\,(s, x, z) \equiv s_1\,(x) = z \;\&\; s =^{\times} s_1.$$

Функция change описывает эффект изменения значения переменной $x = z$, когда машина находится в состоянии s. Эффект заключается в том, что переменной x теперь приписывается значение z, а значения остальных переменных не изменяются.

Семантическую структуру каждого оператора q представим в виде:

$$q = \{f_i; x_1, \ldots, x_n; y_1, \ldots, y_m\}, \qquad (1)$$

где f_i – одна из следующих функций (тип оператора):

function q:

x: = $v \to f_1\,(x, v)$;

$b * q \to f_2\,(b, \text{q})$;

$(C) \to f_3\,(C)$.

f_1, f_2, f_3 – выражения, определяющие желаемый результат в каждом случае;

$x_1, x_2, \ldots, x_n$ – множество переменных, получающих новые значения при выполнении оператора q;

$y_1, y_2, \ldots, y_m$ – множество переменных, используемых в операторе q и не изменяющих своего значения.

Условное выражение b может быть записано в виде:

($b \rightarrow \ldots$

$d \rightarrow \ldots$).

Если b – истина, то результат берется из первой строчки, а в противном случае – из второй. Пустой оператор обозначим как Ω.

Опишем процесс вычисления программы в терминах модели-вычислителя [4, 5]. В этой модели состояние машины не включает состояние «управление» как в модели-интерпретаторе. Состояние «управление», представляющее собой список операторов, которые предстоит выполнить после q, не влияет на семантику состояния s_i, получаемого в результате выполнения q. Модель определяется заданием функции, отображающей программу в ее вычислении. Под вычислением понимается конечная последовательность состояний памяти. Функция имеет следующий вид:

$$\text{comp: } S \times Q \rightarrow S^* \ (S^* = S_л \cup S_н \cup S`) \qquad (2)$$

comp: (s, Q) =

function Q:

$\Lambda \rightarrow (s`)$

$x := v \rightarrow (s; \text{change } (s, x, v_s))$

$b * q \rightarrow (b_s \rightarrow \text{comp } (s, Q),$

$d \rightarrow (s))$

$(C) \rightarrow \text{comp } (s, C)$

$\Omega \rightarrow (s)$.

Модель-вычислитель является основой построения операционной семантики языков программирования. Операционная семантика задается, как уже было показано, определением абстрактного «состояния машины» и смыслом конструкций языка с точки зрения их влияния на состояние, т. е. функции переходов из состояния в состояние:

Comp: $S \rightarrow S$.

Уточним содержание понятия состояния памяти. S представляет собой вектор состояния, определяемый совокупностью множеств переменных X. В нашем случае вектор состояния S выглядит следующим образом:

$$S = \{X_д, X_з, X_и, X_п\},$$

где $X_д = \{x_{д1}, x_{д2}, \ldots, x_{дn}\}$ – множество контролируемых глобальных системных переменных по критерию доступа (системные адреса, таблицы данных и т. п.);

$X_з = \{x_{з1}, x_{з2}, \ldots, x_{зm}\}$ – множество контролируемых глобальных системных переменных по критерию запуска (значения системных регистров, размера свободной оперативной и виртуальной памяти, появления дополнительных переменных программы и т. п.);

$X_и = \{x_{и1}, x_{и2}, \ldots, x_{иw}\}$ – множество контролируемых глобальных системных переменных по критерию использования ресурсов среды;

$X_п = \{x_{п1}, x_{п2}, \ldots, x_{пl}\}$ – множество переменных программы.

Операционное определение исследуемой программы дадим в терминах функции Comp от двух аргументов – текста программы и вектора состояния, представляющего текущее состояние вычислений. Кроме того, введем дополнительно следующие функции:

1. Znach, дающая значение выражения, соответствующего текущему состоянию: Znach: $S \times V \rightarrow Z$;

2. End, дающая заключительный вектор состояния конечной последовательности состояний.

Подстановку значения в вектор состояния будем записывать следующим образом: если z – это значение, то вектор состояния s $[x \leftarrow z]$ определяется как:

$s\ [x \leftarrow z]\ \{x\} = z$,

$s\ [x \leftarrow z]\ \{y\} = s\ (y)$ для всех $y \neq x$.

Смысл записи – «присвоить компоненте x вектора s значение z».

Функция Comp определяется для каждого из типов f_i операторов:

$$\text{Comp } (x := v, s) = s\ [x \leftarrow \text{Znach } (v, s)]; \qquad (3)$$

Comp ($s_л, s_н; s$) = Comp ($s_л, s$) || Comp (Comp ($s_н, s$), s);

Comp ($s_1; s_2, s$) = Comp (s_1, s) || Comp (s_2, End (Comp (s_1, s))),

где || используется для обозначения конкатенации последовательностей состояний.

Использование функцией Comp векторов состояний позволяет достаточно просто описать процесс функционирования программного средства.

Таким образом, основное преимущество операционной модели заключается в том, что операционное определение семантики программы может дать приемлемую реализацию [5, 6].

РЕЗУЛЬТАТЫ И ИХ ОБСУЖДЕНИЕ

Цель эксперимента: идентификация конструкций, характерных для разрушающих программных средств с последующим формированием множества первичных признаков РПС.

В основе эксперимента лежит гипотеза о том, что разрушающие программные средства и обычные программы отличаются между собой по частоте использования некоторых конструкций применяемого языка программирования [7, 8], характерного для человеко-машинных систем, например систем «умный дом», систем оценки уровня информационной экологичности информации в учебном процессе [8] и т. д. Исходя из специфики функций, выполняемых РПС, часть операторов и служебных идентификаторов (операндов), отвечающих за их реализацию, редко используется при написании обычных программ. При этом под обычными программами здесь будем понимать программы, не выполняющие совокупность указанных выше разрушающих функций и не имеющих механизмов исследования и маскировки в программной среде. Выявление такого различия позволяет определить первичные признаки РПС в программах, ориентируясь на которые можно повысить эффективность поиска, при этом получая минимальные временные затраты. От определения множества первичных признаков зависит результат идентификации разрушающих функций в программе.

План проведенного эксперимента:

1. Выбор языка программирования, конструкции которого будут контролироваться в процессе анализа программ.

2. Получение статистики применения операторов в программах на заданном множестве разрушающих программных средств и случайной выборки обычных программ.

3. Обработка результатов анализа программ.

4. Формирование множества первичных признаков – операторов программ.

Результаты выполнения программы характеризует гистограмма (рисунок), отображающая среднее арифметическое частот реализаций тех операторов, которые наиболее часто встречались в РПС по сравнению с обычными программами.

Столбцы диаграммы темного цвета характеризуют результаты эксперимента на множестве разрушающих программных средств. Столбцы светлого тона – соответственно ре-

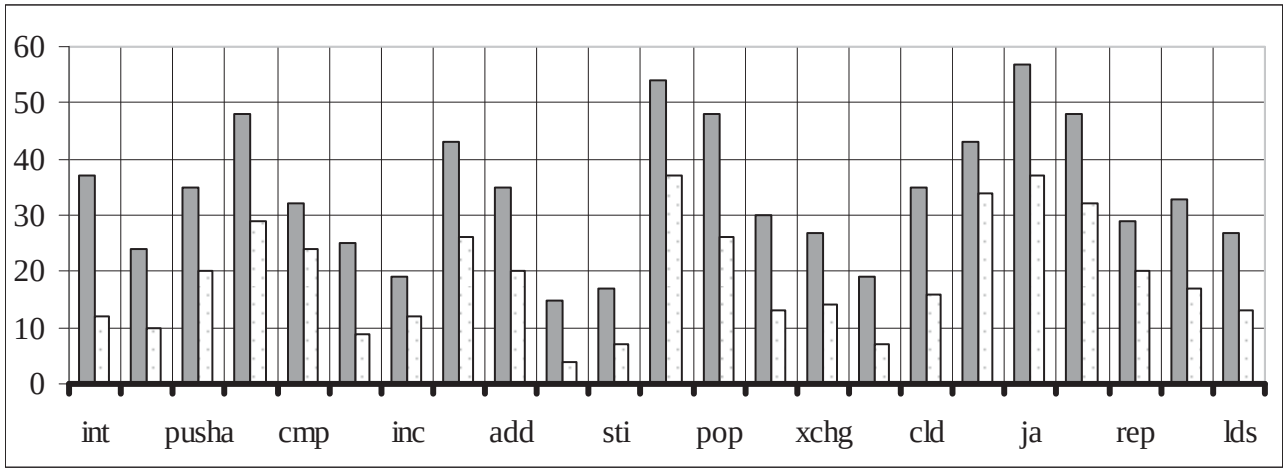

Рисунок. Результаты эксперимента
Примечание: составлено авторами на основании данных, полученных в исследовании.

зультаты эксперимента на обычных программных средствах.

Таким образом, эксперимент подтвердил предположение из [9] о том, что существует различие в реализации на операторном уровне между разрушающими программными средствами и обычными программами. Следует полагать, что данные операторы являются основой построения разрушающих программных конструкций и могут быть использованы в качестве «опорных» точек в методике идентификации программных закладок.

ЗАКЛЮЧЕНИЕ

1. Разработанная методика идентификации программных закладок позволяет определять в исходных текстах программ неизвестные разрушающие функции (агрессивные программные элементы) на основе описанной логической системы распознавания.

2. Предложенные методы семантического анализа программы, ее моделирования в рамках квантового представления являются универсальными с точки зрения стилистики программ, их реализации на любых языках программирования, особенностей ВС.

3. Эксперимент с контрольной группой разрушающих программных средств позволил сформировать множество первичных признаков РПС и показал отличительные особенности реализации разрушающих функций в РПС.

Список источников

1. Алекперов И. Д., Храмов В. В., Горбачева А. А. и др. Информационная безопасность и защита информации в цифровой экономике: элементы теории и тестовые задания. Ростов н/Д. : Южный университет (ИУБиП), 2020. 114 с.
2. Арапова Е. А., Бочаров А. А., Востроknутов И. Е. и др. Возможности искусственного интеллекта в совершенствовании информационного образовательного пространства регионов России : моногр. М. : ООО «Издательский Центр РИОР», 2022. 140 с.
3. Храмов В. В. Особенности использования принципа информационного следа при поиске программных закладок // Вопросы защиты информации. 2001. № 3. С. 39–40.
4. Храмов В. В., Трубников А. Н. Модель специальной программной закладки // Вопросы защиты информации. 1998. № 1–2. С. 36–37.
5. Akperov I. G., Akperov G. I., Alekseichik T. V. et al. Soft models of management in terms of digital transformation. Vol. 2. Monograph. Rostov-on-Don: PEI HE SU (IUBIP); 2020. 256 p.
6. Линденбаум Т. М., Попов О. Р., Храмов В. В. Введение в информационную экологию: технологические предпосылки // Актуальные проблемы и перспективы развития транспорта, промышленности и экономики России : сб. науч. тр. Междунар. науч.-практич. конф., 09–11 ноября 2020 г., г. Ростов-на-Дону. Ростов н/Д. : Ростовский государственный университет путей сообщения, 2020. С. 136–139.
7. Khramov V. V., Trubnikov A. N. Analysis of the aggressiveness of a software product. *Automatic Control and Computer Sciences.* 1999;33(2):28–34.
8. Khramov V. V. Development of a human-machine interface based on hybrid intelligence. *Modern Informa-*

References

1. Alekperov I. D., Khramov V. V., Gorbacheva A. A. et al. Informatsionnaia bezopasnost i zashchita informatsii v tsifrovoi ekonomike: elementy teorii i testovye zadaniia. Rostov-on-Don: Southern University (IMBL); 2020. 114 p. (In Russian).
2. Arapova E. A., Bocharov A. A., Vostroknutov I. E. et al. Vozmozhnosti iskusstvennogo intellekta v sovershenstvovanii informatsionnogo obrazovatelnogo prostranstva regionov Rossii. Monograph. Moscow: RIOR Publishing; 2022. 140 p. (In Russian).
3. Khramov V. V. Osobennosti ispolzovaniia printsipa informatsionnogo sleda pri poiske programmnykh zakladok. *Information Security Questions.* 2001;(3):39–40. (In Russian).
4. Khramov V. V., Trubnikov A. N. Model spetsialnoi programmnoi zakladki. *Information Security Questions.* 1998;(1–2):36–37. (In Russian).
5. Akperov I. G., Akperov G. I., Alekseichik T. V. et al. Soft models of management in terms of digital transformation. Vol. 2. Monograph. Rostov-on-Don: PEI HE SU (IUBIP); 2020. 256 p.
6. Lindenbaum T. M., Popov O. R., Khramov V. V. Introduction to information ecology: Technological background. In: *Proceedings of the International Research-to-Practice Conference "Aktualnye problemy i perspektivy razvitiia transporta, promyshlennosti i ekonomiki Rossii",* November 9–11, 2020, Rostov-on-Don. Rostov-on-Don: Rostov State Transport University; 2020. p. 136–139. (In Russian).
7. Khramov V. V., Trubnikov A. N. Analysis of the aggressiveness of a software product. *Automatic Control and Computer Sciences.* 1999;33(2):28–34.
8. Khramov V. V. Development of a human-machine interface based on hybrid intelligence. *Modern Informa-*

tion Technologies and IT-Education. 2020;16(4):893–900.

9. Храмов В. В., Горбачева А. А., Фомичев Д. П. Моделирование недекларированной активности программного средства в условиях нечеткости исходных данных // Информационная безопасность: вчера, сегодня, завтра : сб. ст. по материалам III Междунар. науч.-практич. конф., 23 апреля 2020 г., г. Москва. М. : Российский государственный гуманитарный университет, 2020. С. 124–130.

tion Technologies and IT-Education. 2020;16(4):893–900.

9. Khramov V. V., Gorbacheva A. A., Fomichev D. P. Modeling of unclaimed activity of the software under the conditions of fuzzy background. In: *Proceedings of the 3rd International Research-to-Practice Conference "Infomatsionnaia bezopasnost: vchera, segodnia, zavtra",* April 23, 2020, Moscow. Moscow: Russian State University for the Humanities; 2020. p. 124–130. (In Russian).

Информация об авторах

А. И. Дубровина – аспирант.
Т. И. Акперов – аспирант.
Т. С. Александрова – аспирант.

Information about the authors

A. I. Dubrovina – Postgraduate.
T. I. Akperov – Postgraduate.
T. S. Aleksandrova – Postgraduate.