ЗАЩИТА UNIX-ПОДОБНЫХ СИСТЕМНЫХ ОКРУЖЕНИЙ ОТ ЭКСПЛУАТАЦИИ НЕДОСТАТКОВ БЕЗОПАСНОСТИ ПАМЯТИ

Марченко И. В.[1]

DOI: 10.21681/2311-3456-2024-5-89-94

В настоящее время одним из самых распространенных недостатков программного обеспечения, написанного на языках программирования C и C++, является некорректная работа с памятью. Она может приводить к несанкционированному получению информации, исполнению произвольного кода и другим негативным последствиям.

***Целью данной работы** является повышение защищенности программ от атак, использующих недостатки работы с памятью, посредством создания системного окружения, реализующего аппаратно-программную технологию контроля целостности состояния памяти.*

***Методы.** Проведен сравнительный анализ и выбор аппаратных технологий контроля корректности состояния памяти, а также программных технологий, поддерживающих выбранную аппаратную платформу. Предложена методика создания системных окружений для Режима безопасных вычислений аппаратно-программной платформы «Эльбрус», учитывающая особенности данной технологии. Методика учитывает необходимость компиляции исходного кода программ для поддержки Режима безопасных вычислений, а также возможное наличие несовместимых с ним конструкций.*

***Полученные результаты.** На основе предложенной методики разработано базовое системное окружение, функционирующее в Режиме безопасных вычислений. В рамках разработки системного окружения в пакетах с открытым исходным кодом на языке C были выявлены и исправлены конструкции, соответствующие угрозам безопасности памяти.*

***Практическая значимость.** Предложенная методика может применяться для дальнейшей разработки безопасных системных окружений на основе Режима безопасных вычислений платформы «Эльбрус», использующих программные технологии, отличные от представленных в статье. Разработанное системное окружение позволяет предотвратить эксплуатацию недостатков безопасности памяти в программах, входящих в его состав, без потери их функциональности.*

***Ключевые слова:** аппаратно-программная платформа «Эльбрус», Режим безопасных вычислений, ARM MTE, CHERI, язык C, тегирование памяти.*

Введение

В настоящее время одним из актуальных вопросов безопасности информационных систем является обеспечение безопасности памяти. Согласно отчетам, предоставленным MITRE за 2021–2023 гг.[2,3,4], ведущее место в рейтинге 25 самых опасных недостатков программного обеспечения (CWE – Common Weakness Enumeration) занимает запись за границами буфера (CWE-787 – Out-of-bounds Write). Кроме того, в данные рейтинги входят чтение за границами выделенной памяти (CWE-125 – Out-of-bounds Read), использование указателя после освобождения памяти (CWE-416 – Use After Free), разыменование нулевого указателя (CWE-476 – NULL Pointer Dereference). Данные недостатки программного обеспечения относятся к угрозам безопасности памяти и могут приводить к отказу в обслуживании, получению злоумышленником конфиденциальной информации, исполнению произвольного кода, а также эскалации привилегий[5].

В то же время около 70 % обнаруживаемых ежегодно уязвимостей, которые Microsoft признает общеизвестными уязвимостями (CVE – Common Vulnerabilities and Exposures), связаны с проблемами безопасности памяти. Проблемы безопасности памяти несут значительную угрозу безопасности системы.

В первую очередь это касается программ, написанных на языках программирования, небезопасных при работе с памятью, таких как C и C++ [1].

Изначально распространение языка C связано с развитием Unix-подобных операционных систем. Одним из ключевых принципов операционной системы Unix и в дальнейшем Unix-подобных операционных систем стала переносимость на другие платформы. Вместо ассемблера, как стандартного языка разработки программного обеспечения, в операционной системе Unix впервые начал использоваться язык программирования C для машинно-независимого

1 Марченко Ирина Викторовна, аспирант кафедры компьютерных систем и технологий (№12) НИЯУ МИФИ, Москва, Россия. E-mail: Irina.V.Marchenko@mcst.ru

2 CWE VIEW: Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses. URL: https://cwe.mitre.org/data/definitions/1425.html (дата обращения: 14.04.2024).

3 2022 CWE Top 25 Most Dangerous Software Weaknesses. URL: https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html (дата обращения: 14.04.2024).

4 2021 CWE Top 25 Most Dangerous Software Weaknesses. URL: https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html (дата обращения: 14.04.2024).

5 CWE-787: Out-of-bounds Write. URL: https://cwe.mitre.org/data/definitions/787.html (дата обращения: 22.07.2024).

кода, позволив расширить круг доступных аппаратных платформ [2].

В настоящее время язык С является основным при написании ядер многих Unix-подобных операционных систем и их базового общего программного обеспечения [3]. Таким образом, ключевые компоненты операционных систем являются подверженными угрозам безопасности памяти.

Существует два подхода к решению проблемы безопасности памяти:

1. Использование безопасных при работе с памятью языков программирования, ближайшим по характеристикам к языку С из которых является Rust;
2. Использование технологий аппаратного контроля корректности состояния памяти, наиболее развитыми из которых являются ARM Memory Tagging Extension (ARM MTE), Capability Hardware Enhanced RISC Instructions (CHERI) и Режим безопасных вычислений аппаратно-программной платформы «Эльбрус».

Использование безопасных языков программирования делает невозможным появление угроз безопасности памяти в программах, однако требует полного переписывания исходных кодов используемых программ с языка С на безопасный язык программирования. Данная задача является крайне трудоемкой. В то же время использование аппаратных технологий контроля корректности состояния памяти позволяет предотвращать эксплуатацию уязвимостей программного обеспечения, связанных с угрозами безопасности памяти, для любой программы на небезопасном языке программирования, запущенной для такой платформы[6].

Для использования технологии аппаратного контроля корректности состояния памяти требуется поддержка со стороны программного обеспечения [4]. Для функционирования безопасного системного окружения необходимо обеспечить безопасность работы всех его компонентов — библиотек и исполняемых файлов.

По этой причине необходимо разработать безопасное системное окружение, реализующее аппаратно-программный комплекс защиты от угроз безопасности памяти, при помощи технологии аппаратного контроля корректности состояния памяти.

1. Аппаратная платформа

Безопасное системное окружение реализует аппаратно-программный комплекс контроля корректного состояния памяти. Для создания безопасного системного окружения требуется определить аппаратную и программную платформы.

На данный момент существует три основных аппаратных технологии контроля безопасности памяти, реализованных в процессорах существующих архитектур: ARM MTE (архитектура ARM [5]), CHERI (архитектура ARM, существуют разработки для архитектур MIPS, RISC-V [6]) и Режим безопасных вычислений «Эльбрус» (архитектура E2K [7]).

Технология ARM MTE осуществляет вероятностный контроль над соответствием указателя выделенной памяти по принципу «ключ-замок»: теги указателя («ключа») и области памяти («замка») должны совпасть при обращении. Размер тега составляет 4 бита, следовательно, количество возможных вариантов тегов равно 16. При однократном запуске вероятность обнаружения ошибки составляет 93% [8].

Технология CHERI не является вероятностной и осуществляет контроль над соответствием указателя и выделенной памяти при помощи дескрипторов (ориг. Capability), представляющих собой указатели с метаданными. Дескрипторы позволяют обратиться по указателю к заданному объекту с учетом верхней и нижней границы выделенной для него области памяти, что позволяет предотвратить выходы за границы объекта [9].

Режим безопасных вычислений не является вероятностной технологией и использует дескрипторы вместо указателей, позволяющие контролировать границы объекта, к которому происходит обращение. Также в Режиме безопасных вычислений предусмотрена технология внешних тегов, не входящих непосредственно в состав дескриптора и позволяющих контролировать тип содержимого каждого 32-битного слова, находящегося в оперативной памяти [10].

Возможности обнаружения типов недостатков программного обеспечения при помощи рассмотренных технологий [11] представлены в табл. 1.

Для дальнейшей работы выбран Режим безопасных вычислений платформы Эльбрус, как реализующий гарантированную невероятностную защиту от эксплуатации большего числа типов недостатков программного обеспечения.

2. Программная платформа

Для выбора программной платформы следует учитывать наличие компонентов, реализующих поддержку Режима безопасных вычислений со стороны операционной системы, компилятора и его компонентов [12].

В качестве первичного критерия отбора операционной системы выступает поддержка архитектуры E2K. Поддержка операционной системой выбранной архитектуры, предоставляющей технологию контроля корректности состояния памяти, необходима для функционирования безопасного системного окружения.

Еще одним немаловажным критерием при выборе операционной системы является наличие доступа

6 Back To The Building Blocks: A Path Toward Secure And Measurable Software // The White House. Washington. 2024. URL: https://www.whitehouse.gov/wp-content/uploads/2024/02/Final-ONCDTechnical-Report.pdf (дата обращения: 21.07.2024).

Таблица 1.

Возможность обнаружения типов уязвимостей технологиями тегирования памяти

Наименование технологии	Обнаруживаемые недостатки				
	CWE-787	CWE-125	CWE-587	CWE-416	CWE-457
ARM MTE	+	+	+	+	-
CHERI	+	+	+	-	-
Режим безопасных вычислений	+	+	+	+	+

Таблица 2.

Дистрибутивы, поддерживающие архитектуру E2K

Дистрибутив	Поддержка E2K	Доступ к исходному коду	Доступ к системе сборки ОС
ОС Эльбрус	+	+	+
ALT Linux	+	+	-
Astra Linux	+	-	-

к исходному коду пакетов, входящих в состав операционной системы, так как исполнение программы в Режиме безопасных вычислений требует перекомпиляции ее исходного кода.

В качестве возможной операционной системы рассматривались дистрибутивы GNU/Linux, поддерживающие архитектуру E2K: ОС Эльбрус[7], ALT Linux[8] и Astra Linux[9]. Сравнительная таблица данных операционных систем представлена в табл. 2.

Операционной системой для создания безопасного системного окружения выбрана ОС Эльбрус производства АО «МЦСТ».

В качестве разрабатываемого системного окружения выбран LXC-контейнер. Этот вид окружения обеспечивает работу программ в изолированном пространстве процессов, ускоряя отладку контейнера при помощи основной операционной системы и общего ядра Linux.

3. Методика создания безопасного системного окружения

Разработана методика создания безопасного системного окружения. Ее общий алгоритм включает в себя четыре основных этапа:

1. Определение базового списка пакетов;
2. Сборка пакетов;
3. Разработка правил создания и настройки системного окружения;
4. Функциональное тестирование программ.

На этапе определения базового списка пакетов безопасного системного окружения необходимо учитывать наличие пакетов поддержки режима безопасных вычислений, в особенности библиотеки языка C с поддержкой Режима безопасных вычислений. На этапе сборки пакетов из составленного списка возможно обнаружение конструкций, несовместимых с Режимом безопасных вычислений при помощи ошибок и предупреждений компилятора. Разработка правил создания и настройки осуществляется индивидуально для каждого набора выбранных

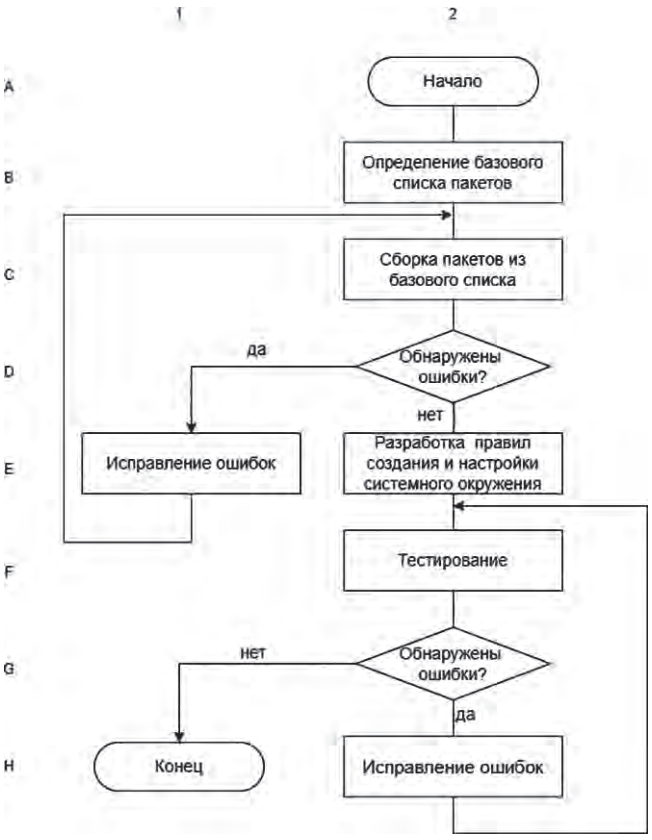

Рис. 1. Блок-схема алгоритма создания безопасного системного окружения

7 Операционные системы «Эльбрус». URL: http://www.mcst.ru/elbrus_os (дата обращения: 27.07.2024).

8 Дистрибутивы ОС Альт для Эльбрус. URL: https://www.altlinux.org/Эльбрус/дистрибутивы (дата обращения: 27.07.2024).

9 Совместимость Astra Linux с процессорами Эльбрус. URL: https://wiki.astralinux.ru/kb/sovmestimost-astra-linux-s-protsessorami-el-brus-187795264.html (дата обращения: 27.07.2024).

программных технологий с учетом их особенностей. Тестирование проводится в первую очередь с целью выявления в утилитах и библиотеках, входящих в состав безопасного системного окружения, конструкций, несовместимых с Режимом безопасных вычислений. Блок-схема алгоритма создания безопасного системного окружения представлена на рис. 1.

Предложенная методика универсальна и не зависит от выбранной технологии создания системного окружения.

4. Реализация

В базовый список пакетов безопасного системного окружения вошли пакеты минимальной базовой системы ОС Эльбрус, а также их сборочные зависимости – 70 пакетов.

Произведена компиляция пакетов из базового списка в режиме безопасных вычислений. Разработаны правила создания и настройки безопасного системного окружения. Проведено функциональное тестирование – программы в Режиме безопасных вычислений при возникновении конструкции с ошибкой завершают свою работу с выдачей диагностики.

Из 70 пакетов, вошедших в состав базового системного окружения, в рамках тестирования интерес представляют пакеты с открытым исходным кодом, написанные на языке С, к таким относятся 53. Распределение пакетов из состава безопасного системного окружения по принадлежности к проприетарному программному обеспечению, пакетам с отрытым исходным кодом на языке С и не на языке С представлено в табл. 3.

Таблица 3.

Разделение пакетов базового списка по категориям

<table>
<tr><th colspan="3"></th><th>Количество пакетов, шт.</th></tr>
<tr><td rowspan="3">Базовый список</td><td colspan="2">Проприетарное ПО</td><td>4</td></tr>
<tr><td rowspan="2">Открытый исходный код</td><td>Исходный код не на языке С</td><td>13</td></tr>
<tr><td>Исходный код на языке С</td><td>53</td></tr>
<tr><td colspan="3">Итого</td><td>70</td></tr>
</table>

По итогам тестирования 28 из 53 исследуемых пакетов содержали конструкции, несовместимые с Режимом безопасных вычислений, что составляет 53%. Некорректные конструкции разделены на 5 категорий, наиболее часто встречающейся из которых стало использование неинициализированных данных. В данные категории включают ранее рассмотренные классы недостатков программного обеспечения, кроме использования указателей после освобождения памяти, примеров такой уязвимости в рассматриваемых пакетах обнаружено не было. Обнаруженные в ходе тестирования некорректные конструкции исправлены. Распределение конструкций представлено в табл. 4.

Таблица 4.

Классы конструкций, несовместимых с режимом безопасных вычислений в пакетах базового списка

Класс конструкции	Количество конструкций, шт.	Процент от общего числа конструкций, %
Использование неинициализированных данных	70	51.9
Запись за границами выделенной памяти	2	1.5
Чтение за границами выделенной памяти	5	3.7
Преобразование переменной целочисленного типа в указатель	38	28.1
Использование фиксированных выравниваний	20	14.8

Реализованное безопасное системное окружение по функциональности полностью соответствует системному окружению в обычном режиме. Таким образом, пользователь безопасного системного окружения получает преимущества использования технологии контроля корректного состояния памяти в виде защиты от эксплуатации ряда недостатков программного обеспечения, в том числе и от еще не выявленных уязвимостей, при этом не будучи ограниченным в своих действиях.

Заключение

Технологии аппаратного контроля корректности состояния памяти предоставляют защиту от эксплуатации угроз безопасности памяти, однако не предназначены для функционирования обособленно, без программной поддержки. В рамках данной работы проведено сравнительное исследование аппаратных технологий и для наиболее оптимальной из них, Режима безопасных вычислений, предложены программные технологии реализации безопасного системного окружения.

Разработана методика создания безопасного системного окружения, учитывающая особенности выбранных технологий. В соответствии с методикой реализовано безопасное системное окружение.

В рамках разработки и отладки безопасного системного окружения в 53 % пакетов с открытым исходным кодом на языке C обнаружены конструкции, несовместимые с Режимом безопасных вычислений, а также найдены 77 потенциально опасных конструкций при работе с памятью. Это подтверждает эффективность использования безопасного системного окружения: оно реализует возможность выявления недостатков программного обеспечения, соответствующих проблемам безопасности памяти. Обнаруженные недостатки исправлены. Безопасное системное окружение предоставляет функциональность, аналогичную функциональности в обычном режиме, в рамках пакетов, входящих в его состав.

Литература

1. *Gavin, T. A proactive approach to more secure code. URL: https://msrc.microsoft.com/blog/2019/07/a-proactive-approach-to-more-secure-code/ (дата обращения: 22.07.2024).*
2. *Цейтин Г. С. UNIX и постановка вопроса о переносимости программного обеспечения // SORUCOM-2011. – 2011. – С. 320–322. URL: https://sorucom.iis.nsk.su/files/page/sorucom-2011_0.pdf (дата обращения: 27.07.2024).*
3. *Tanenbaum A. Modern Operating Systems. Fourth Edition. // Vrije Universiteit, 2014. – 1106 с.*
4. *Jero S. TAG: Tagged Architecture Guide / S. Jero, N. Burow, B. Ward, R. Skowyra // ACM Computing Surveys. 2022. – Vol. 55. – № 6. – Article 124. DOI: https://doi.org/10.1145/3533704.*
5. *Serebryany K. ARM Memory Tagging Extension and How It Improves C/C++ Memory Safety // ;login:. – Summer 2019. – Vol. 44. – № 2. URL: https://www.usenix.org/system/files/login/articles/ login_summer19_03_serebryany.pdf (дата обращения: 26.07.2024).*
6. *Watson R. An Introduction to CHERI. // University of Cambridge, 2019. https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-941.pdf (дата обращения: 28.07.2024).*
7. *Нейман-заде М. И., Королёв С. Д. Руководство по эффективному программированию на платформе «Эльбрус». АО «МЦСТ». 2024. URL: http://mcst.ru/doc/elbrus_prog/elbrus-prog-1.2_2024-02-28.pdf (дата обращения: 26.07.2024).*
8. *Partap A. Memory Tagging: A Memory Efficient Design / A. Partap, D. Boneh // arXiv. Cryptography and Security. – 2022. DOI: https://doi.org/10.48550/arXiv.2209.00307.*
9. *Watson R. Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 9). // University of Cambridge, 2023. URL: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-987.pdf (дата обращения: 28.07.2024).*
10. *Мустафин Т. Р. Безопасная среда исполнения критических приложений во встраиваемых системах на базе вычислительных средств семейства «Эльбрус» / Т.Р. Мустафин, А.И. Алехин, Е.М. Кравцунов, Б.О. Макаев // Радиопромышленность. – 2019. – № 1 – С. 16–22. – EDN YXUUPJ.*
11. *Артемьев И. А. Сравнительный анализ технологий безопасного использования памяти с учетом аппаратно-программных особенностей вычислительных комплексов / И. А. Артемьев, И. В. Марченко, Д. В. Ярапов, Н. А. Шаменков // Цифровые технологии и решения в сфере транспорта и образования. 2023. С. 11–20. – EDN BVFFDD.*
12. *Волконский В. Ю. Безопасная реализация языков программирования на базе аппаратной и системной поддержки // Вопросы радиоэлектроники. – 2008. – Т. 4. – № 2. – С. 98–141. – EDN JTNBAR.*

PROTECTING UNIX-LIKE SYSTEM ENVIRONMENTS FROM EXPLOITATION OF MEMORY SECURITY WEAKNESSES

Marchenko I. V.[10]

Nowadays one of the most common weaknesses of software written in the C and C++ programming languages is incorrect memory handling. It can lead to unauthorized access to information, executing arbitrary code, and other negative consequences.

***The purpose of this work** is increasing the protection of programs from attacks using memory safety weaknesses by implementing a protected system environment using hardware memory integrity monitoring technology.*

***Methods.** A comparative analysis and selection of hardware and software memory integrity monitoring technology, as well as software technologies supporting the selected hardware platform, are performed. A methodology for creating system environments for Secure computing mode is proposed, taking into account the features of this technology. The methodology takes into account the need to compile the source code of the program to support Secure computing mode compilation option, as well as the possible existence of incompatible constructions.*

***Results.** Based on the proposed methodology, a basic Secure computing mode protected system environment has been developed. During the development of the system environment in C language open source packages, constructions corresponding to memory security threats were identified and corrected.*

***Practical significance.** The proposed methodology can be used for further development of protected system environments based on the Secure computing mode while using software technologies other than those presented in the article.*

10 Irina V. Marchenko, Postgraduate Student, Department of Computer Systems and Technologies (No12), National Research Nuclear University MEPhI, Moscow, Russia. E-mail: Irina.V.Marchenko@mcst.ru

The developed system environment allows preventing the exploitation of memory safety weaknesses in the software included in it, without reducing functionality for the user.

Keywords: *Elbrus hardware and software platform, Secure computing mode, ARM MTE, CHERI, C language, memory tagging.*

References

1. *Gavin, T. A proactive approach to more secure code. URL: https://msrc.microsoft.com/blog/2019/07/a-proactive-approach-to-more-secure-code/ (accessed: 22.07.2024).*
2. *Tseytin G. S. UNIX and the Statement of Software Portability Problem // SORUCOM-2011. – 2011. – P. 320–322. (in Russian). URL: https://sorucom.iis.nsk.su/files/page/sorucom-2011_0.pdf (accessed: 27.07.2024).*
3. *Tanenbaum A. Modern Operating Systems. Fourth Edition. // Vrije Universiteit, 2014. – 1106 p.*
4. *Jero S. TAG: Tagged Architecture Guide / S. Jero, N. Burow, B. Ward, R. Skowyra // ACM Computing Surveys. 2022. – Vol. 55. – № 6. – Article 124. DOI: https://doi.org/10.1145/3533704.*
5. *Serebryany K. ARM Memory Tagging Extension and How It Improves C/C++ Memory Safety // ;login:. – Summer 2019. – Vol. 44. – № 2. URL: https://www.usenix.org/system/files/login/articles/ login_summer19_03_serebryany.pdf (accessed: 26.07.2024).*
6. *Watson R. An Introduction to CHERI. // University of Cambridge, 2019. https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-941.pdf (accessed: 28.07.2024).*
7. *Neiman-Zade M. I., Korolev S. D. Guide to Effective Programming on the Elbrus Platform. MCST. 2024. (in Russian). URL: http://www.mcst.ru/doc/elbrus_prog/elbrus-prog-1.2_2024-02-28.pdf (accessed: 26.07.2024).*
8. *Partap A. Memory Tagging: A Memory Efficient Design / A. Partap, D. Boneh // arXiv. Cryptography and Security. – 2022. DOI: https://doi.org/10.48550/arXiv.2209.00307.*
9. *Watson R. Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 9). // University of Cambridge, 2023. URL: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-987.pdf (accessed: 28.07.2024).*
10. *Mustafin T. R. Secure execution environment for critical applications in embedded systems based on Elbrus family computing facilities / T. R. Mustafin, A. I. Alekhin, E. M. Kravtsunov, B. O. Makaev // Radio Inustry. – 2019. – № 1 – P. 16–22. (in Russian). – EDN YXUUPJ.*
11. *Artemiev I. A. Comparative analysis of technologies for the safe use of memory, taking into account the hardware and software features of computing complexes / I. A. Artemiev, I. V. Marchenko, D. V. Yarapov, N. A. Shamenkov // Digital technologies and solutions in the field of transport and education. 2023. P. 11–20. (in Russian). – EDN BVFFDD.*
12. *Volkonskii V. Y. Secure implementation of programming languages based on hardware and system support // Issues of radio electronics. – 2008. – T. 4. – № 2. – P. 98–141. (in Russian). – EDN JTNBAR.*