ПРОТИВОДЕЙСТВИЕ УЯЗВИМОСТЯМ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. Часть 2. АНАЛИТИЧЕСКАЯ МОДЕЛЬ И КОНЦЕПТУАЛЬНЫЕ РЕШЕНИЯ

Леонов Н. В.[1]

DOI: 10.21681/2311-3456-2024-3-90-95

Цель исследования: *концептуальное противодействие уязвимостям в программном обеспечении.*

Методы исследования: *системный анализ, моделирование, синтез решений.*

Полученные результаты: *во второй части статьи предложена аналитическая модель, формализуемая сущности и взаимосвязи онтологической модели области программного обеспечения с уязвимостями. Оценено влияние сущностей предметной области на реализуемость направлений в ней (программный инжиниринг, внедрение уязвимостей и их нейтрализация), что позволило синтезировать концептуальные пути противодействия уязвимостям.*

Научная новизна *работы определяется полной формализацией объектов и субъектов предметной области, а также их взаимосвязей.*

Ключевые слова: *информационная безопасность, уязвимость, противодействие, аналитическая модель, концептуальные пути решения.*

COUNTERING SOFTWARE VULNERABILITIES. Part 2. ANALYTICAL MODEL AND CONCEPTUAL SOLUTIONS

Leonov N. V.[2]

The goal of the investigation: *conceptual counteraction to software vulnerabilities.*

Research methods: *system analysis, modeling, synthesis of solutions.*

Results: *in the second part of the article, an analytical model is proposed that formalizes the essence and relationship of the ontological model of the software domain with vulnerabilities. The influence of the entities of the subject area on the feasibility of directions in it (software engineering, introduction of vulnerabilities and their neutralization) was assessed, which made it possible to synthesize conceptual ways to counter vulnerabilities.*

The scientific novelty *of the work is determined by the complete formalization of objects and subjects of the subject area, as well as their relationships.*

Keywords: *information security, vulnerability, counteraction, analytical model, conceptual solutions.*

В первой части статьи [1] введена онтологическая модель предметной области, охватившая три ее основных направления – программный инжиниринг, внедрение уязвимостей и их нейтрализация.

1 Леонов Николай Викторович, кандидат технических наук, доцент, начальник лаборатории Государственного научно-исследовательского института прикладных проблем. Москва, Россия. ORCHID: http://orcid.org/0009-0005-1295-5343. E-mail: leonov-nv@yandex.ru

2 Nikolay V. Leonov, Ph.D., Docent, Head of the State Research Institute of Applied Problems Laboratory. Moscow, Russia. ORCHID: http://orcid.org/0009-0005-1295-5343. E-mail: leonov-nv@yandex.ru

Аналитическая модель

Запишем Модель в аналитическом виде для каждого из направлений (с учетом того, что некоторые сущности относятся к нескольким направлениям) предметной области следующим образом.

Направление «Программный инжиниринг»

Получение Программы (*Program*) для решения Задачи (*Task*) Разработчиком (*Developer*) имеет следующий формальный вид:

$$Program = \widehat{Developer}^{CreateProgram}(Task),$$

где «$\widehat{\ }$» – диакритический символ «верхняя скобка» для обозначения продуцированных алгоритмов; $\widehat{Developer}^{CreateProgram}(...)$ – алгоритм создания Программы (перев. на англ. *Create Program*), продуцируемый Разработчиком.

В рамках Модели создание Программы с помощью Средства сборки (*BuildTool*) [2], применяемого Разработчиком, имеет следующий формальный вид:

$$Program^{ExecutableCode} = BuildTool(Program^{SourceCode}, \widehat{DeveloperAlgorithm})^{BuildTool},$$

где $Program^{ExecutableCode}$ и $Program^{SourceCode}$ – Программа в представлении Исполняемого и Исходного кода [3]; $BuildTool(...)$ – алгоритм работы Средства сборки; $\widehat{DeveloperAlgorithm}^{BuildTool}$ – алгоритм по управлению Средством, продуцируемый (*Production*) Разработчиком, т.е.:

$$\widehat{DeveloperAlgorithm}^{BuildTool} := Developer^{Production}(GoalBuilding^{Program}),$$

где «:=» – обозначение назначения алгоритма; $Developer^{Production}(...)$ – продуцирование нового алгоритма Разработчиком; $Goal^{BuildingProgram}$ – цель (перев. на англ. *Goal*) по сборке Программы (перев. на англ. *Build Program*), достигаемая Разработчиком.

Направление «Нейтрализация уязвимостей»

Получение Отчета безопасности (*SecurityReport*) [4] путем анализа Программы Средством сканирования (*ScannerTool*) [5], применяемым Экспертом (*Expert*) [6], имеет следующий формальный вид:

$$SecurityReport = ScannerTool(Program, \widehat{ExpertAlgorithm}^{ScannerTool}),$$

где $ScannerTool(...)$ – алгоритм работы Средства сканирования; $\widehat{ExpertAlgorithm}^{ScannerTool}$ – алгоритм по управлению Средством, продуцируемый Экспертом на основании Мета-информации ($MetaInformation^{Expert}$) [7], т.е.:

$$\widehat{ExpertAlgorithm}^{ScannerTool} := Expert^{Production}(Goal^{ScanningProgram}, MetaInformation^{Expert}),$$

где $Expert^{Production}(...)$ – продуцирование нового алгоритма Экспертом; $Goal^{ScanningProgram}$ – цель по сканированию Программы (перев. на англ. *Scan Program*), достигаемая Экспертом.

Получение Патча (*Patch*) [8] по Отчету безопасности путем действий Эксперта имеет следующий формальный вид:

$$Patch = \widehat{Expert}^{CreatePatch}(Task, MetaInformation^{Expert}),$$

где $\widehat{Expert}^{CreatePatch}(...)$ – алгоритм создания Патча (перев. на англ. *Create Patch*), продуцируемый Экспертом на основании Мета-информации.

Внедрение Патча в Программу с помощью Средства внедрения (*EmbeddingTool*), применяемого Экспертом, имеет следующий формальный вид:

$$Program = EmbeddingTool(Patch, \widehat{ExpertAlgorithm}^{EmbeddingTool}),$$

где $EmbeddingTool(...)$ – алгоритм работы Средства внедрения; $\widehat{ExpertAlgorithm}^{EmbeddingTool}$ – алгоритм по управлению Средством, продуцируемый Экспертом на основании Мета-информации, т.е.:

$$\widehat{ExpertAlgorithm}^{EmbeddingTool} := Expert^{Production}(Goal^{PatchingProgram}, MetaInformation^{Expert}),$$

где $Goal^{PatchingProgram}$ – цель по исправлению Программы (перев. на англ. *Patching Program*) с позиции имеющихся в ней ошибок, достигаемая Экспертом.

Запутывание кода Программы с помощью Средства обфускации (*ObfuscationTool*) [9], применяемого Экспертом, имеет следующий формальный вид:

$$Program' = ObfuscationTool(Program, \widehat{ExpertAlgorithm}^{ObfuscationTool}),$$

где «′» – диакритический символ указания новой версии объекта (в данном случае Программы); $Program'$ – Программа после обфускации; $ObfuscationTool(...)$ – алгоритм работы Средства обфускации [10]; $\widehat{ExpertAlgorithm}^{ObfuscationTool}$ – алгоритм по управлению Средством, продуцируемый Экспертом на основании Мета-информации, т.е.:

$$\widehat{ExpertAlgorithm}^{ObfuscationTool} := Expert^{Production}(Goal^{ObfuscatingProgram}, MetaInformation^{Expert}),$$

где $Goal^{ObfuscatingProgram}$ – цель по обфускации Программы (перев. на англ. *Obfuscating Program*), достигаемая Экспертом.

Направление «Внедрение уязвимостей»

Получение Уязвимости (*Vulnerability*) для осуществления Вектора атаки (*AttachVector*) Нарушителем (*Intruder*) [11] имеет следующий формальный вид:

$$Vulnerability = \widehat{Intruder}^{CreateVulnerability}(AttachVector, MetaInformation^{Intruder}),$$

где $\widehat{Intruder}^{CreateVulnerability}(...)$ – алгоритм создания Уязвимости (перев. на англ. *Create Vulnerability*), продуцируемый Нарушителем на основании Мета-информации ($MetaInformation^{Intruder}$).

Запутывание кода Уязвимости с помощью Средства обфускации, применяемого Нарушителем [12], имеет следующий формальный вид:

$$Vulnerability' = ObfuscationTool(Vulnerability, \widehat{IntruderAlgorithm}^{ObfuscationTool}),$$

где $Vulnerability'$ – Уязвимость после обфускации; $\widehat{IntruderAlgorithm}^{ObfuscationTool}$ – алгоритм по управлению Средством, продуцируемый Нарушителем на основании Мета-информации, т.е.:

$$\widehat{IntruderAlgorithm}^{ObfuscationTool} := Intruder^{Production}(Goal^{ObfuscatingVulnerability}, MetaInformationI^{Intruder}),$$

где $Goal^{ObfuscatingVulnerability}$ – цель по обфускации Уязвимости (перев. на англ. *Obfuscating Vulnerability*), достигаемая Нарушителем.

Внедрение Уязвимости [13, 14] в Программу с помощью Средства внедрения, применяемого Нарушителем, имеет следующий формальный вид:

$$Program = EmbeddingTool(Vulnerability, \widehat{IntruderAlgorithm}^{EmbeddingTool}),$$

где $\widehat{IntruderAlgorithm}^{EmbeddingTool}$ – алгоритм управления Средством внедрения, продуцируемый Нарушителем на основании Метаинформации, т.е.:

$$\widehat{IntruderAlgorithm}^{EmbeddingTool} := Intruder^{Production}(Goal^{InfectingProgram}, MetaInformation^{Intruder}),$$

где $Goal^{InfectingProgram}$ – цель по заражению Программы (перев. на англ. *Infecting Program*), достигаемая Экспертом.

В данном случае применение алгоритма Средства внедрения с формальной точки зрения одинаково как Экспертом для Патча, так и Нарушителем для Уязвимости, поскольку у алгоритма $EmbeddingTool(...,...)$ 1-м параметром идет некоторый код (*Patch* или *Vulnerability*), а вторым – алгоритм управления средством ($ExpertAlgorithm^{EmbeddingTool}$ или $IntruderAlgorithm^{EmbeddingTool}$).

Получение Метаинформации из Программы с помощью Средства реинжиниринга (*ReengineeringTool*) [15, 16], применяемого Экспертом и Нарушителем, имеет следующий формальный вид:

$$\begin{cases} MetaInformation^{Expert} = \text{ReengineeringTool}(\text{Program}, \widehat{ExpertAlgorithm}^{ReengineeringTool}) \\ MetaInformation^{Intruder} = \text{ReengineeringTool}(\text{Program}, \widehat{IntruderAlgorithm}^{ReengineeringTool}) \end{cases},$$

где $ReengineeringTool(...)$ – алгоритм работы Средства реинжиниринга (перев. на англ. *ReengineeringTool*) [17]; $Expert^{ReengineeringTool}$ и $Intruder^{ReengineeringTool}$ – алгоритмы Эксперта и Нарушителя по управлению средством для получения собственной метаинформации $MetaInformation^{Expert}$ и $MetaInformation^{Intruder}$, соответственно, т.е.:

$$\begin{cases} \widehat{ExpertAlgorithm}^{ReengineeringTool} := Expert^{Production}(Goal^{ReverseEngineering}, MetaInformation^{Expert}) \\ \widehat{IntruderAlgorithm}^{ReengineeringTool} = Intruder^{Production}(Goal^{ReverseEngineering}, MetaInformation^{Intruder}) \end{cases},$$

где $Goal^{ReverseEngineering}$ – цель по реверс-инжинирингу Программы (перев. на англ. *Reverse Engineering*), достигаемая Экспертом и Нарушителем в ходе собственной, диаметрально противоположной деятельности.

Корректность и охват записи Модели в аналитическом виде позволяет сделать следующие утверждения.

Во-первых, поскольку результаты каждой аналитической записи используется в какой-либо другой, то, следовательно, все сущности Модели являются необходимыми; иначе, если бы в Модели присутствовала некоторая неиспользуемая сущность, то одна из записей возвращала бы результат, не используемый в других. Например, если бы Эксперт в результате анализа Программы (предположим, Средством сканирования) создавал бы ее электронную подпись – которая, очевидно, бессмысленна для нейтрализации уязвимостей – то присутствовала бы аналитическая запись, результатом которой являлась эта подпись, не используемая ни в каких других записях.

Во-вторых, поскольку для каждой аналитической записи введены или получены используемые в ней параметры, то, следовательно, все сущности Модели являются достаточными; иначе, если бы в Модели отсутствовала некоторая сущность, то и одна из записей была бы невычислима из-за отсутствия необходимого параметра. Например, если для Эксперта отсутствовал бы А-объект – Мета-информация, то создание Экспертом Патча для конкретной Программы (с помощью $\widehat{Expert}^{CreatePatch}$ было бы невозможно из-за отсутствия информации о принципах и деталях ее работы (т.е. $MetaInformation^{Expert}$).

Таким образом, можно утверждать, что все сущности Модели являются необходимыми и достаточными.

Направление «Нейтрализация уязвимостей»

Исходя из того, что три направления предметной области (Программный инжиниринг, Нейтрализация и Внедрение уязвимостей) построены на общем терминологическом базисе, при этом в единой нотации (используя жестко заданные и ограниченные субъекты, объекты и их связи), можно сделать следующее предположение:

«Влияние, оказываемое на отдельные сущности, будет оказывать влияние и на реализуемость направлений».

Следовательно, снижение эффективности действий Нарушителя может быть достигнуто через влияние на сущности, с которыми он «работает»; сложность такого противодействия уязвимостям заключается и в потенциальном обратном влиянии на другие, легальные направления. Например, достаточно смелое «удаление» из онтологической модели основополагающей сущности Программы, как совокупности Исходного и Исполняемого кода со средством его сборки (т.е. некий «фантастический» сценарий полного отказа от информационных технологий), хотя и не позволит Нарушителю закладывать в нее Уязвимости и проводить Вектор атаки (при автоматическом избавлении Эксперта от необходимости анализа Программы и создания для нее Патча), тем не менее и Разработчик не сможет решать Задачу программным способом. Примером менее радикального способа является запрет использования или строгий контроль любых средств модификации Программ (таких, как Средство внедрения), что хотя и не позволит исправлять случайные ошибки Разработчиков с помощью Патчей, но и существенно затруднит Нарушителям встраивание вредоносного кода. Связь каждого объекта Модели с реализуемостью каждого из направлений приведены в табл. 1, где введены следующие обозначения степени влияния: «+» – необходимость сущности для направления (зеленый фон); «–» – отсутствие необходимости (синий фон); «+/–» – повышение эффективности его реализуемости (белый фон). Подобные связи для субъектов будут опущены, поскольку их наличие влияет исключительно на каждое из направлений. Влияние же взаимодействий между сущностями сложно анализируемо и контролируемо (например, при участии трех сущностей).

Дадим ряд пояснений касательно выставленных степеней влияния каждой сущности на направления и качественных способов противодействия путем управления ее (см. табл. 1) – т.е. повышения эффективности нейтрализации Уязвимостей и/или снижения эффективности внедрения Уязвимостей с сохранением направления Программного инжиниринга.

Во-первых, наличие «+» только для одного из направлений тождественно отсутствию влияния сущности на другие направления. Таким образом, исключение Вектора атаки и Уязвимости из предметной области (т.е. их полная нейтрализация) позволяет противодействовать Внедрению уязвимостей, для чего, впрочем, уже существует достаточное количество программных, технических, организационных и иных решений. Соответственно, подобные («однонаправленные») сущности (Задача, Отчет безопасности, Патч, Средство сканирования) для своих направлений не должны быть исключены или ограничены.

Во-вторых, наличие «+» для двух и более направлений тождественно необходимости наличия сущностей во всех из них; это верно для Программы. Таким образом, нейтрализация Исполняемого кода привела бы и к противодействию направления «Программная инженерия», что, естественно, недопустимо.

Таблица 1

Влияние сущностей предметной области на реализуемость в ней направлений

Сущность	Направление			Способ противодействия
	Программный инжиниринг	Нейтрализация уязвимостей	Внедрение уязвимостей	
Задача	+	–	–	Отсутствует
Исходный код	+	+/–	+/–	Ограничение
Исполняемый код	+	+	+	Ограничение
Вектор атаки	–	–	+	Исключение
Уязвимость	–	–	+	Исключение
Отчет безопасности	–	+	–	Отсутствует
Патч	–	+	–	Отсутствует
Метаинформация	–	+	+	Ограничение
Средство сборки	+	+/–	+/–	Ограничение
Средство сканирования	–	+	–	Отсутствует
Средство внедрения	–	+	+	Ограничение
Средство обфускации	–	+/–	+/–	Исключение Ограничение
Средство реинжиниринга	–	+/–	+	Исключение Ограничение

Нейтрализация же Метаинформации и Средства внедрения привела бы к двойному эффекту. Так, как Нарушитель не смог бы анализировать код, разрабатывать под него Уязвимости и встраивать их в Программу, так и Эксперт остался бы без Отчетов безопасности и возможности применения Патчей. Однако, в этом случае возможным способом противодействия может стать ограничение доступа к сущности со стороны Нарушителя, оставляя при этом полный доступ для Разработчика и Эксперта. Так, если ограничение доступа Нарушителя к Программе представляется сложно осуществимым, то препятствование распространению Метаинформации о различных популярных или используемых в критических областях Программах является более чем разумным и возможным; Средства внедрения и вовсе могут быть отнесены к разрешенным для использования только в специальных организациях, занимающихся вопросами информационной безопасности. Исключение Исполняемого кода из предметной области, как очевидно, привела бы к полному прекращению функционирования любого ПО, а меры ограничения доступа к нему со стороны Нарушителя являются практически неосуществимыми на практике.

В-третьих, наличие «+/–» для нескольких направлений указывает на желательность, но не на необходимость данной сущности; так, применение Средства обфускации носит как положительный эффект – снижает эффективность внедрения Уязвимости в Программу, так и отрицательный – затрудняет поиск этой Уязвимости. Таким образом, исключение Средства обфускации из предметной области неоднозначно (а скорее – несущественно) повлияет на безопасность Программы. Впрочем, ограничение доступа Нарушителям к средствам затруднения анализа кода Программы при наличии такового для Эксперта даст преимущество последнему.

И, в-четвертых, наличие «+» для одного направления и «+/–» для другого означает повышение эффективности второго при необходимости для первого. Соответственно, исключение данной сущности позволит оказать качественное противодействие первому направлению, при этом лишь снизив эффективности реализуемости второго. Так, исключение из предметной области Средства реинжиниринга существенно затруднит Нарушителю внедрение Уязвимости в Программу (поскольку, получение им Метаинформации будет невозможно, а «слепое» изменение Исполняемого кода скорее всего приведет к нарушению функционирования); при этом без такого Средства Эксперт также сможет получать Отчеты безопасности и внедрять Патчи, хотя и с меньшей эффективностью (поскольку, Разработчики также заинтересованы в устранении Уязвимости и будут более активно взаимодействовать с Экспертами). Аналогично предыдущему пояснению, ограничение доступа к Средствам реинжиниринга существенно затруднит анализ Нарушителем Программы при неизменности подобного анализа для Эксперта, что является наиболее предпочтительным вариантом. С другой стороны, исключение Исходного кода и Средства сборки хотя и снизит эффективность внедрения и нейтрализации Уязвимости, однако решение Задач с применением Программ окажется практически невозможным.

Исходя из сделанных пояснений, можно предложить следующие, достаточно строго полученные, предпосылки, которые должны лечь в основу соответствующих концептуальных путей противодействия Уязвимостям в Программах.

1) Вектор атаки и Уязвимость должны быть исключены из предметной области;
2) Доступ к Исходному коду, Метаинформации, а также Средствам сборки и внедрения должны быть ограничены для Нарушителя;
3) Средства обфускации и реинжиниринга могут быть исключены из предметной области, однако более целесообразным вариантом должно быть ограничение доступа к ним Нарушителя;
4) Остальные сущности (Исполняемый код, Задача, Отчет безопасности, Патч и Средство сканирования) должны присутствовать в предметной области.

Естественно, для отражения реального состояния дел в предметной области требуется более «тонкая» (по возможности – количественная) оценка влияния сущностей на направления, а также способов воздействия на них. Так, Средство реинжиниринга неявно присутствует в любой работе Эксперта (например, для связи областей Исходного или Исполняемого кода Программы с Уязвимостями в Отчете безопасности), что также требуется учитывать.

Заключение

В работе рассмотрена основополагающая проблема противодействия уязвимостям ПО с позиции трех действующих «игроков» сферы информационной безопасности – Разработчика, Нарушителя и Эксперта. Для этого введена онтологическая модель предметной области – разработка Программы, внедрение в нее уязвимостей и их нейтрализации, – построенная по единому (шаблонированному) принципу связей ее сущностей и являющаяся основным научным результатом.

Частный научный результат состоит в таблице влияния сущностей предметной области на реализуемость ее основных направлений, а также в предложенных качественных способах противодействия.

Новизна основного научного результата заключается в использовании строгих правил (или шаблонов) связей между ее сущностями предметной

области, а также в представлении модели в полностью аналитическом виде.

Теоретическая значимость состоит в установлении строгой взаимосвязи между Разработчиком программы, внедряющим в нее уязвимости Нарушителем, и противодействующим уязвимостям Экспертом посредством общих (т.е. разделяемых) сущностей предметной области.

Практическая значимость состоит в возможности выработки концептуальных способов влияния на сущности предметной области (путем их исключения или ограничения доступа) в интересах решения глобальной задачи обеспечения безопасности ПО.

В качестве продолжения работы планируется детализация представленной Модели, добавление новых противодействующих друг другу участников, более тонкий учет связей между сущностями Модели, а также разработка конкретных путей противодействия Уязвимостям в Программах.

Литература

1. *Леонов Н. В. Противодействие уязвимостям программного обеспечения. Часть 1. Онтологическая модель // Вопросы кибербезопасности. № 2(60). 2024. DOI: 10.21681/2311-3456-2024-2-87-92*
2. *Миронов С. В., Батраева И. А., Дунаев П. Д. Библиотека для разработки компиляторов // Труды Института системного программирования РАН. 2022. Т. 34. № 5. С. 77–88. DOI: 10.15514/ISPRAS-2022-34(5)-5.*
3. *Афонин М. В. Компиляция. Сборка и связывание проектов // Инновационный потенциал развития общества: взгляд молодых ученых: сборник научных статей 3-й Всероссийской научной конференции перспективных разработок (Курск, 01 декабря 2022 года). Том 3. 2022. С. 115–118.*
4. *Якимук А. Ю., Устинов С. А., Лазарев Т. П., Коваленко А. С. Методы формализации описания сценариев кибератак // Электронные средства и системы управления. Материалы докладов Международной научно-практической конференции. 2022. № 1–2. С. 73–76.*
5. *Суздалов Д. В., Некрасов А. Н. Разработка сканера уязвимостей // Наука молодых: сборник материалов Межрегиональной молодежной научной конференции, посвященной памяти Ф. А. Бабушкина, (Сыктывкар, 25–26 мая 2023 года). 2023. С. 139–143.*
6. *Вареница В. В., Марков А. С., Савченко В. В., Цирлов В. Л. Практические аспекты выявления уязвимостей при проведении сертификационных испытаний программных средств защиты информации // Вопросы кибербезопасности. 2021. № 5 (45). С. 36–44. DOI: 10.21681/2311-3456-2021-5-36-44.*
7. *Израилов К. Е. Методология реверс-инжиниринга машинного кода. Часть 2. Статическое исследование. Труды учебных заведений связи // 2023. Т. 9. № 6. С. 68–82. DOI: 10.31854/1813-324X-2023-9-6-68-82.*
8. *Коржев А. А. Обеспечение безопасности программного обеспечения // Стратегическое развитие инновационного потенциала отраслей, комплексов и организаций: сборник статей XI Международной научно-практической конференции (Пенза, 10–11 октября 2023 года). 2023. – С. 237–241.*
9. *Градский Д. Ю. Методы обфускации кода // Оригинальные исследования. 2020. Т. 10. № 5. С. 177–180.*
10. *Иванов М. А., Коннова И. Г., Саликов Е. А., Степанова М. А. Обфускация логических схем генераторов псевдослучайных чисел на регистрах сдвига с линейными и нелинейными обратными связями // Безопасность информационных технологий. 2021. Т. 28. № 1. С. 74–83. DOI: 10.26583/bit.2021.1.06.*
11. *Лукацкий А. В. Обзор мировых трендов по промышленной кибербезопасности // Релейщик. 2020. № 1 (36). С. 60–62.*
12. *Ерохин В. В., Притчина Л. С. Анализ и совершенствование методов обнаружения шелл-кодов (shellcode) в компьютерных системах // Прикладная информатика. 2021. Т. 16. № 2 (92). С. 103–122.*
13. *Руднев Н. О., Герасимова В. Ф., Шагапов И. А. Метод закрепления доступа в системе посредством инъекции кода в операционной системе Windows // Естественные и технические науки. 2022. № 12 (175). С. 398–403.*
14. *Нефедов В. В. Методы внедрения кода в исполняемые файлы PE-формата // Молодежная научная школа кафедры «Защищенные системы связи». 2021. Т. 1. № 2 (4). С. 61–68.*
15. *Маркин Д. О., Макеев С. М. Система защиты терминальных программ от анализа на основе виртуализации исполняемого кода // Вопросы кибербезопасности. 2020. № 1 (35). С. 29–41. DOI: 10.21681/2311-3456-2020-01-29-41.*
16. *Буйневич М. В., Ганов Г. А., Израилов К. Е. Интеллектуальный метод визуализации взаимодействий программ в интересах аудита информационной безопасности операционной системы // Информатизация и связь. 2020. № 4. С. 67–74.*
17. *Фомин А. И. Оценка сложности исследования дизассемблированного кода исполняемых программ // Естественные и технические науки. 2021. № 7 (158). С. 210–211.*