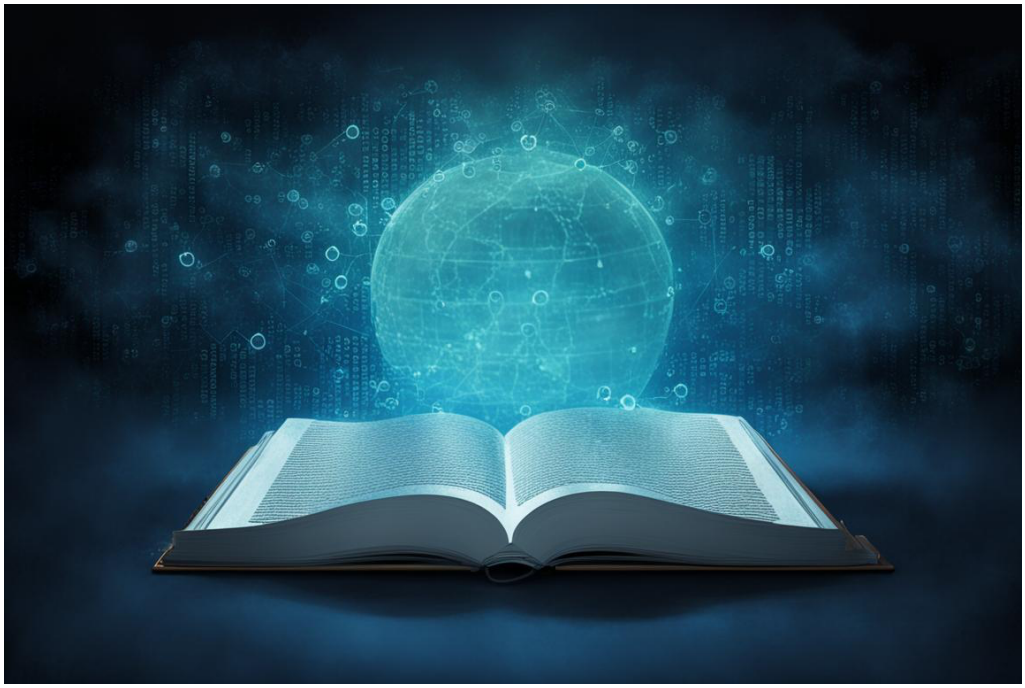
ИТМО

А.А. Менщиков, Д.А. Заколдаев,
А.А. Воробьева

ВВЕДЕНИЕ В КИБЕРБЕЗОПАСНОСТЬ И КИБЕРУСТОЙЧИВОСТЬ

Учебно-методическое пособие



Санкт-Петербург
2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

А.А. Менщиков, Д.А. Заколдаев,
А.А. Воробьева

ВВЕДЕНИЕ В КИБЕРБЕЗОПАСНОСТЬ И КИБЕРУСТОЙЧИВОСТЬ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В
УНИВЕРСИТЕТЕ ИТМО
по направлению подготовки 10.03.01 Информационная безопасность
в качестве учебно-методического пособия для реализации основных профессиональных образовательных программ высшего образования бакалавриата

Санкт-Петербург
2024

Менщиков А.А., Заколдаев Д.А., Воробьева А.А. **Введение в кибербезопасность и киберустойчивость** Учебно-методическое пособие.– СПб: Университет ИТМО, 2024.– 72 с.

Рецензент: Комаров Игорь Иванович, кандидат технических наук, доцент факультета безопасности информационных технологий, Университета ИТМО.

Учебное пособие разработано для методической помощи обучающимся по направлению подготовки 10.03.01 – «Информационная безопасность».
В пособии рассмотрены вопросы обеспечения кибербезопасности и киберустойчивости информационных систем. Изучаются вопросы оценки угроз, принципы обеспечения киберустойчивости компьютерных систем. Формулируются базовые принципы обеспечения защищенности инфраструктуры DevOps и DevSecOps. Приведены практические задания, тесты и вопросы для проверки знаний по оценке и повышению кибербезопасности.
Учебное пособие может быть рекомендовано студентам, осуществляющим подготовку по направлению «Информационная безопасность», а также изучающих дисциплины «DevOps и DevSecOps», «Основы DevOps», «Технологии обнаружения уязвимостей в автоматизированных системах».

ИТМО (Санкт-Петербург) — национальный исследовательский университет, научно-образовательная корпорация. Альма-матер победителей международных соревнований по программированию. Приоритетные направления: IT и искусственный интеллект, фотоника, робототехника, квантовые коммуникации, трансляционная медицина, Life Sciences, Art&Science, Science Communication.
Лидер федеральной программы «Приоритет-2030», в рамках которой реализовывается программа «Университет открытого кода». С 2022 ИТМО работает в рамках новой модели развития — научно-образовательной корпорации. В ее основе академическая свобода, поддержка начинаний студентов и сотрудников, распределенная система управления, приверженность открытому коду, бизнес-подходы к организации работы. Образование в университете основано на выборе индивидуальной траектории для каждого студента.
ИТМО пять лет подряд — в сотне лучших в области Automation & Control (кибернетика) Шанхайского рейтинга. По версии SuperJob занимает первое место в Петербурге и второе в России по уровню зарплат выпускников в сфере IT. Университет в топе международных рейтингов среди российских вузов. Входит в топ-5 российских университетов по качеству приема на бюджетные места. Рекордсмен по поступлению олимпиадников в Петербурге. С 2019 года ИТМО самостоятельно присуждает ученые степени кандидата и доктора наук.

Оглавление

Введение

В рамках данного пособия под обеспечением *кибербезопасности* понимается деятельность, направленная на обеспечение защищенности информационных систем, т.е. на защиту программ, компьютерных систем и сетей от атак [1].

Основная цель кибербезопасности заключается в отражении кибератак, которые направлены на получение доступа к данным, модификацию, уничтожение или нарушение нормального выполнения бизнес-операций вне зависимости от того, исходят ли эти атаки изнутри или извне организации. Современные информационные системы являются настолько сложными, реализованными с использованием огромного стека технологий, что невозможно достичь состояния абсолютной защищенности [2].

В настоящее время прослеживается тенденция перехода от концепции обеспечения кибербезопасности к обеспечению *киберустойчивости* – способности осуществлять свое штатное функционирование в условиях воздействия компьютерных атак в киберпространстве [3].

Цель данного пособия – помочь освоить основы обеспечения кибербезопасности и киберустойчивости информационных систем и овладеть принципами их внедрения на практике.

В рамках данного пособия будут рассмотрены понятия кибербезопасности и киберустойчивости, вопросы оценки уязвимостей, угроз и методики обеспечения кибербезопасности. Пособие познакомит с первыми шагами, необходимыми при проведении комплексного анализа защищенности информационных систем. Будут рассмотрены основы анализа защищенности, базовые понятия, техники и тактики, а также

подходы к построению киберустойчивых инфраструктур с использованием концепций DevOps, DevSecOps и CloudNative.

В рамках данного пособия рассмотрены три тематические блока:

1. Основы кибербезопасности и киберустойчивости;
2. Cloud Native и приложение двенадцати факторов;
3. DevOps и DevSecOps.

Структурно оно состоит из двух основных глав и четырех приложений. В первой представлены теоретические основы, необходимые для выполнения практических работ. Во второй главе даны указания для выполнения практических работ. Практический материал предназначен для подготовки к выполнению практических работ по оценке и повышению защищенности информационных систем. Приложение А содержит тестовое задание для оценки уровня входных знаний и сформированности необходимых компетенций перед изучением курса. В Приложении Б представлены вопросы для самоконтроля и тесты, призванные помочь студентам в достижении результатов обучения по дисциплинам «DevOps и DevSecOps», «Основы DevOps», «Технологии обнаружения уязвимостей в автоматизированных системах». В Приложении В представлены рекомендации для проведения промежуточной аттестации. Также приведены рекомендации для проведения промежуточной аттестации по дисциплине и перечень вопросов. Приложение Г содержит ответы на входное и выходное тестирование. Дополнительно приведен список использованной литературы и список литературы, рекомендуемой для дальнейшего самостоятельного освоения.

Глава 1

Теоретическая часть

1.1 Основы кибербезопасности и киберустойчивости

1.1.1 Кибербезопасность и информационная безопасность

Определение понятия «*кибербезопасность*» сильно зависит от специфики его применения. Часто его трактуют как безопасность киберпространства в целом. Под киберпространством в этом случае понимается некоторое абстрактное (виртуальное) пространство, которое симулируется и опосредуется электронными устройствами (компьютерами, мобильными устройствами и пр.) [4]

В качестве синонима понятия «*кибербезопасность*» часто применяется понятие «*информационная безопасность*», что является некорректным, так как они имеют совершенно различное значение.

Согласно ГОСТ Р 53114-2008, безопасность информации или данных – это состояние защищенности информации, при котором обеспечены ее конфиденциальность, доступность и целостность [5].

- Конфиденциальность – это состояние информации, при котором доступ к ней осуществляют только субъекты, имеющие на него право.
- Доступность – это состояние информации, при котором субъекты, имеющие права доступа, могут реализовать их беспрепятственно.
- Целостность – это состояние информации, при котором отсутствует любое ее изменение либо изменение

осуществляется только преднамеренно субъектами, имеющими на него право.

В ГОСТ Р 50922-2006 содержатся основные термины и определения, касающиеся защиты информации [6]:

- *Угроза безопасности информации* – совокупность условий и факторов, создающих потенциальную или реально существующую опасность нарушения безопасности информации.
- *Уязвимость системы* – свойство информационной системы, обусловливающее возможность реализации угроз безопасности обрабатываемой в ней информации.
- *Атака* – это любое действие нарушителя, которое приводит к реализации угрозы путем использования уязвимостей информационной системы.
- *Риск* – это потенциальная возможность того, что уязвимость будет использоваться для создания угрозы активу или группе активов, приводящей к ущербу для организации.

Соотнесение понятий «информационная безопасность» и «кибербезопасность» дано в международном стандарте ISO 27032 от 2023 года [7]. В нем определено, что кибербезопасность является лишь составной частью информационной безопасности и связана только киберрисками (потерями финансового, репутационного или организационного характера, связанными с какими-либо инцидентами в IT-инфраструктуре) [8]. Также в стандарте указано, что кибербезопасность – это процесс, который обеспечивает конфиденциальность, целостность и доступность информации, но только в рамках киберпространства [9].

В рамках данного пособия используется более узкое определение обеспечения *кибербезопасности*, под которым понимается деятельность, направленная на обеспечение защищенности информационных систем, т.е. на защиту программ, компьютерных систем и сетей от кибератак [1].

Кибератака – это попытка выведения компьютерной системы из строя или хищения информации за счёт уязвимости в устройстве или программном обеспечении (ПО) [10].

Кибератаки почти всегда связаны с получением доступа к приватным или конфиденциальным данным в корыстных целях. Сегодня большая часть данных располагается в облачных

хранилищах, но также они могут храниться на личных устройствах, серверах организаций и устройствах Интернета вещей (IoT).

Организации, внедряющие стратегии кибербезопасности, стремятся свести к минимуму риски и нежелательные последствия кибератак, которые могут привести к репутационному или финансовому ущербу, и повредить бизнес-процессам. Например, компании активируют планы аварийного восстановления, чтобы свести к минимуму ущерб, возникающий при реализации угроз безопасности или при возникновении сбоев при выполнении бизнес-операций [11].

Компании в определенных отраслях или регионах должны соблюдать специфические нормативные требования для защиты конфиденциальных данных от возможных киберрисков. Например, компании, осуществляющие свою деятельность в Европейском Союзе, должны соблюдать Общий регламент по защите данных (GDPR) [12], который требует от организаций принятия надлежащих мер по кибербезопасности для обеспечения конфиденциальности данных. В Российской Федерации осенью 2022 года вступил в силу Федеральный закон № 266-ФЗ, который ужесточил штрафы за утечку персональных данных и сократил перечень ситуаций, когда не нужно уведомлять Роскомнадзор до начала обработки персональных данных [13].

Кибератаки постепенно эволюционируют по мере развития технологий. Злоумышленники используют новые инструменты и изобретают новые стратегии для получения несанкционированного доступа к системам. Организации вынуждены постоянно совершенствовать меры по кибербезопасности, чтобы соответствовать новым и развивающимся технологиям и инструментам, используемым для проведения кибератак.

Практики кибербезопасности могут применяться в самых разных областях – от мобильных устройств обычных пользователей до промышленных систем искусственного интеллекта. Сюда входят сетевая безопасность, защита приложений, облачных систем, восстановление после сбоев и даже обучение пользователей (так называемая программа повышения осведомленности в сфере информационной безопасности, или security awareness.

1.1.2 Угрозы кибербезопасности. Оценка. Тактики и техники злоумышленников

Приведем основные виды угроз, с которыми борется современная кибербезопасность [14]:

- Вредоносное программное обеспечение (ВПО).
- Атаки социальной инженерии – атаки, основанные на человеческом взаимодействии, когда злоумышленники путем обмана вынуждают пользователей нарушить процедуры безопасности, выдать конфиденциальную информацию.
- Фишинговые атаки (фишинг) – атаки, основанные на методах социальной инженерии, когда злоумышленники рассылают пользователям электронные письма или текстовые сообщения, напоминающие сообщения из доверенных источников.
- Целевые атаки – продолжительные и целенаправленные кибератаки, при которых злоумышленник получает доступ к сети и остается незамеченным в течение длительного периода времени.
- Атаки типа «отказ в обслуживании» – злоумышленники перегружают канал связи или генерируют избыточную нагрузку на сервисы, что делает невозможным предоставление услуги легитимным пользователям.
- Атаки на цепочки поставок – эксплуатация доверительных отношений между организацией и ее контрагентами.

Это далеко не полный список угроз. Важно отметить, что, когда мы говорим про большие и сложные системы, все приведенные атаки имеют место.

Одним из основных шагов при обеспечении безопасности является оценка угроз и определение их актуальности для конкретной системы.

Федеральная служба по техническому и экспортному контролю (ФСТЭК) выпустила в 2021 году методику оценки угроз безопасности информации. В данном документе приведены категории нарушителей, выделяют внутренних и внешних нарушителей [15]:

- Внутренние нарушители – нарушители, имеющие права доступа в контролируемую (охраняемую) зону (территорию) и (или) полномочия по автоматизированному доступу к информационным ресурсам и компонентам систем и сетей.
- Внешние нарушители – это нарушители, не имеющие прав доступа в контролируемую (охраняемую) зону (территорию) и (или) полномочий по доступу к информационным ресурсам и компонентам систем и сетей, требующим авторизации

Актуальность возможных угроз безопасности информации определяется наличием сценариев их реализации. Определение сценариев предусматривает установление последовательности возможных тактик и соответствующих им техник нарушителей. Тактики – это технические цели противника, а техники – это то, каким образом данные цели могут достигаться.

Тактики и техники, приведенные в [15], хорошо согласуются с распространенным сегодня понятием киллчейна (killchain), которое определяет структуру атаки [16]. Приведем основные тактики из данного документа:

1. сбор информации об объекте;
2. получение первоначального доступа к компонентам систем и сетей;
3. внедрение и исполнение ВПО;
4. закрепление (сохранение доступа) в системе или сети;
5. управление ВПО и (или) компонентами, к которым ранее был получен доступ;
6. повышение привилегий в системе;
7. сокрытие действий;
8. получение доступа к другим компонентам систем и сетей;
9. сбор и вывод информации, необходимой для дальнейших действий при реализации угроз безопасности информации или реализации новых угроз;
10. несанкционированный доступ и (или) воздействие на информационные ресурсы или компоненты, приводящие к негативным последствиям.

На рисунке 1.1 представлен пример использования нескольких сценариев реализации угроз безопасности информации в системе.

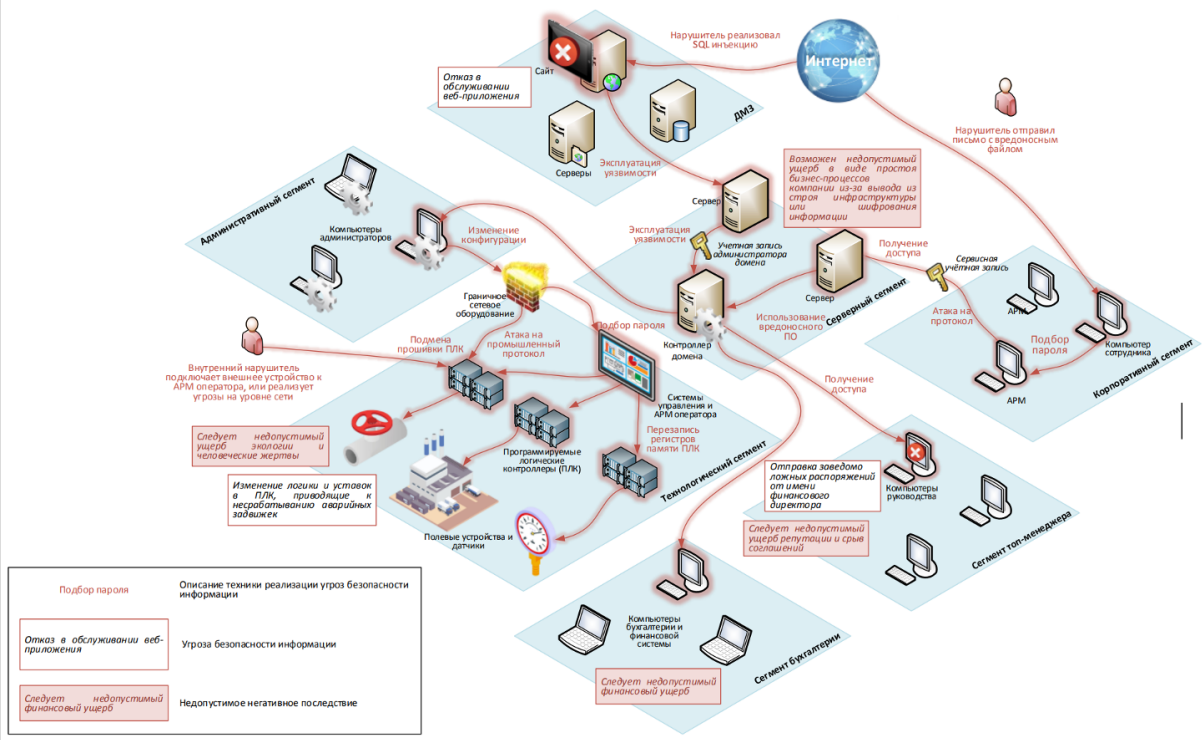


Рисунок 1.1 – Пример сценариев действий нарушителя

В первом сценарии внешний нарушитель инициирует доступ к системе путем отправки письма с вредоносным вложением на почту сотрудника. С компьютера сотрудника он осуществляет подбор пароля на смежный компьютер в сети, через который получает доступ к серверу путем эксплуатации сервисной учетной записи. На сервере он может осуществить поднятие привилегий и завладеть контроллером домена, тем самым получив неограниченный доступ к остальным системам.

В другом сценарии также внешний нарушитель применяет атаку на веб-сайт организации и эксплуатирует уязвимость соседнего сервера для получения доступа ко внутренней сети.

В третьем сценарии внутренний нарушитель подключает специальное устройство к компьютеру в локальной сети и реализует различные сетевые атаки, например, эксплуатирует уязвимость промышленного контроллера для изменения логики его работы.

1.1.3 Уязвимости и угрозы

В предыдущих разделах была приведена общая информация по уязвимостям и угрозам, в этом разделе они будут рассмотрены более подробно.

Существует два основных вида уязвимостей:

- *0-day* (zero-day, уязвимость нулевого дня) – это уязвимость, которая известна, но против которой пока не разработаны защитные механизмы и не выпущены обновления безопасности;
- *1-day* – уязвимость, которая известна, и против которой уже разработаны защитные механизмы или выпущены обновления безопасности.

С понятием уязвимости также тесно связано понятие *эксплойт* – это вредоносный код, который использует ошибки или недостатки ПО для распространения киберугроз.

Важно также понимать, что уязвимости могут быть не только в используемом ПО, но и в так называемых цепочках поставок, то есть в библиотеках и зависимостях, которые используются в других продуктах. И за всем этим нужно постоянно следить.

Common Vulnerability Scoring System (CVSS) или общая система оценки уязвимостей – это открытый стандарт, используемый организациями по всему миру для расчета количественных оценок уязвимости. На текущий момент актуальной является версия 4.0 данного стандарта.

CVSS позволяет определить основные характеристики уязвимости и получить числовую оценку, отражающую ее значимость. Затем числовую оценку можно преобразовать в качественное представление (например: низкий, средний, высокий и критический). Это помогает организациям правильно оценить и расставить приоритеты в своих процессах управления уязвимостями.

Для того чтобы получить числовую оценку, нужно ответить на вопросы из четырех категорий метрик [17]:

- Базовые метрики: включают оценку сложности эксплуатации, потенциального ущерба для конфиденциальности, целостности и доступности информации.

- Временные метрики (метрики угроз): вносят в общую оценку поправку на полноту имеющейся информации об уязвимости, наличии кода эксплойта и доступность исправлений.
- Контекстные метрики: вносят в общую оценку поправки с учетом характеристик информационной среды и влияния на другие компоненты.
- Дополнительные метрики: описывают и измеряют дополнительные внешние атрибуты уязвимости.

Рассмотрим, что такое уязвимость, на примере уязвимости Log4j. В публичной базе уязвимостей CVE ей присвоен номер CVE-2021-44228. Ее суть в том, что Log4j позволяет выполнить любой вредоносный код на сервере при помощи Java Naming and Directory Interface (JNDI). Данной уязвимости присвоен наивысший уровень угрозы – 10 баллов по шкале CVSS 3-й версии. Также в каталоге ExploitDB доступны публичные эксплойты данной уязвимости – код, который достаточно запустить, чтобы использовать данную уязвимость для выполнения произвольных команд на сервере.

Существует огромное множество баз данных уязвимостей. Выделяют государственные, вендорские, коммерческие и различные агрегаторы. Несмотря на то, что все эти базы оперируют примерно одними и теми же сведениями, необходим постоянный их мониторинг для актуализации уязвимых точек и рисков потенциального несанкционированного проникновения.

Существует специальное ПО, которое обновляет данные сведения в автоматизированном режиме и также выполняет проверки, присутствует ли в системе уязвимое ПО. Например, OWASP Dependency-Track – интеллектуальная платформа анализа компонентов, которая позволяет организациям выявлять и снижать риски в цепочке поставок ПО.

Отметим, что для минимизации числа уязвимостей необходимо корректно выстраивать процессы разработки ПО и систем, такие как DevOps, DevSecOps и MLOps.

Список угроз также достаточно формализован. Например, существует банк данных угроз ФСТЭК России, который содержит сведения об основных угрозах безопасности информации и уязвимостях, в первую очередь, характерных для государственных информационных систем и автоматизированных систем

управления производственными и технологическими процессами критически важных объектов [18].

Поскольку по большей части в данном пособии будут рассматриваться системы в контексте сложных веб приложений, потребуется больше информации об угрозах, характерных для веб.

Open Web Application Security Project (OWASP) [19] – это некоммерческая организация, работающая над повышением безопасности ПО. Они разрабатывают открытые проекты, программы защиты, методики, инструкции, проводят образовательные мероприятия с целью обеспечения безопасности Интернета. Одним из их известных их проектов является OWASP Top 10 – перечень десяти самых актуальных угроз для веб приложений, который обновляется периодически (примерно один раз в четыре года). Приведем угрозы, содержащиеся в последней версии данного перечня:

1. ошибки контроля доступа;
2. некорректное использование криптографических библиотек;
3. инъекции, которые реализуются за счет отсутствия проверок введенных пользователем данных и небезопасной архитектуры приложений;
4. ошибки при проектировании;
5. ошибки конфигурации ПО и средств защиты;
6. использование устаревших и скомпрометированных компонентов;
7. ошибки аутентификации и идентификации;
8. недостаточный контроль целостности и доверие данным;
9. ошибки в реализации логирования и мониторинга;
10. подделка запросов на стороне сервера.

Одной из ключевых организаций в области кибербезопасности и киберустойчивости является корпорация MITRE [20], которая занимается созданием различных стандартов, продуктов и подходов для обеспечения безопасности различных систем. MITRE поддерживают базу *MITRE ATT&CK* [21] – общедоступную базу знаний о тактиках и техниках злоумышленников, основанную на реальных наблюдениях. Она является основополагающим

документом для разработки конкретных моделей угроз и методологий защиты.

Заметим, что большинство проблем следуют из того, что архитектура изначально строится без учета принципов киберустойчивости.

1.1.4 Киберустойчивость

Понятие «киберустойчивость» тесно связано с переходом от попытки обеспечить меры, которые помогут защитить продукт от угроз, к тому, чтобы минимизировать вероятность такого события и значительно уменьшить цену восстановления после кибератаки.

Киберустойчивость – это способность предвидеть, противостоять, восстанавливаться и адаптироваться к неблагоприятным условиям, нагрузкам, атакам или компрометациям систем, которые используют или активируются киберресурсами [22]. Можно выделить следующие составные части цели обеспечения киберустойчивости:

1. Предвидение (anticipate) – поддержка состояния осознанной готовности к неблагоприятным обстоятельствам, к которым можно отнести как стихийные бедствия, так и неожиданные нагрузки или целевые атаки. Необходимо выстраивать планы по смягчению последствий, реагированию на события безопасности, а также производить аналитику киберугроз.
2. Противодействие (withstand) – продолжение выполнения основных бизнес-функций, несмотря на невзгоды. Необходимо определить основные бизнес-функции и обеспечивающие их процессы, системы, сервисы, сети и инфраструктуру.
3. Восстановление после инцидентов (recover) – восстановление после неблагоприятного события, включает в себя как восстановление после инцидента, так и определение уровня доверия к восстановленным функциям и данным. Уровень доверия является особенно важным, в случае отсутствия полной уверенности в том, что злоумышленник уже не находится в системе и не может воспользоваться техниками для закрепления в ней.

4. Адаптация (adapt) – модификация бизнес-функций в ответ на прогнозируемые изменения в технических или операционных средах.

Выделяются следующие задачи обеспечения киберустойчивости:

- Предотвращение успешного выполнения атаки или реализации неблагоприятных условий.
- Подготовка и создание реалистичных планов действий, направленных на преодоление прогнозируемых или ожидаемых неблагоприятных обстоятельств.
- Обеспечение живучести, то есть увеличение продолжительности и жизнеспособности основных бизнес-функций в неблагоприятных условиях.
- Ограничение ущерба или противодействие.
- Восстановление системы после окончания атаки.
- Расследование или понимание – это изучение событий и состояний системы до и в момент атаки.
- Преобразование бизнес-функций и вспомогательных систем для повышения эффективности борьбы с трудностями и изменениями в окружающей среде.
- Преобразование архитектуры.

Мы ответили на два вопроса – «зачем нужно строить киберустойчивые системы» и «какие задачи для этого нужно решить». Осталось ответить на вопрос – «каким образом обеспечить киберустойчивость». Это выполняется с использованием при помощи так называемых *техник киберустойчивости* – набора методов и технологий, предназначенных для достижения одной или нескольких целей или задач. Каждая техника характеризуется как возможностями, которые она предоставляет, так и предполагаемыми последствиями использования технологий или процессов, которые она включает [22].

При реализации целей и задач обеспечения киберустойчивости необходимо перейти к техникам, которые необходимо имплементировать в существующую систему. В целом техники можно рассматривать как привычные и понятные технические задачи, которые необходимо решить.

Рассмотрим основные техники, которые используются в различных программах построения киберустойчивых систем. В [22] выделяют:

- Мониторинг – постоянное отслеживание метрик и поведения всех систем, логирование событий.
- Адаптивное реагирование – внедрение гибких инструментов реагирование на риски. Например, возможность быстро добавлять правила межсетевого экрана или назначение ответственных сотрудников за каждый отдельный сервис для ускорения реакции.
- Контекстная осведомлённость – документированность всех процессов. Например, создание карты сети в реальном времени или использование программ для учета и приоритизации рисков.
- Скоординированная защита – работы по максимальной оркестрации всех систем защиты, начиная от разграничения доступа и заканчивая организацией тестирования на проникновение. Самый важный и сложный пункт.
- Обман/запутывание злоумышленника – усложнение атаки путем введения злоумышленника в заблуждение, сокрытия критических активов или создание «приманок». Например, шифрование данных, даже если к ним нет удаленного доступа, встраивание водяных знаков, создание программных ловушек в сети, использование «канареек», которые генерируют оповещение при получении доступа к ним.
- Разнообразие – использование различных архитектур и дизайнов внутри одной системы для исключения повторяемости уязвимости. Например, использование различных операционных систем, языков программирования и фреймворков.
- Динамическое позиционирование – периодическое перемещение ресурсов на другие платформы. Например, переезд микросервисов в другой датацентр, обновление настроек мониторинга. Важным является тот факт, что данные методы могут приводить к увеличению поверхности атаки.
- Очистка состояния – управление ресурсам и данными, что подразумевает их удаление после окончания

использования. Например, очистка лишних кэшей, логов, разрыв долго длящихся соединений, перезагрузка машин.

- Ограничение привилегий – ограничение привилегии на основе атрибутов пользователей, элементов системы, особенностей окружения. Например, принцип минимальных привилегий – организация доступа к ресурсам в системе таким образом, чтобы любая сущность внутри этой системы имела доступ только к тем ресурсам, которые минимально необходимы для успешного выполнения рабочей цели этой сущности [23].
- Реструктуризация – структуризация систем и пользователей ресурсов таким образом, чтобы они соответствовали потребностям выполнения рабочей цели или бизнес-функций, снижали текущие и ожидаемые риски и учитывали эволюцию технической, операционной среды и угроз. Например, избавление от лишних и дублирующихся функций, начиная от ограничения коммуникаций только в выбранных мессенджерах и заканчивая аутсорсингом некритических функций.
- Избыточность – дублирование критичного функционала, бэкапы, репликация баз данных.
- Сегментация – разделение систем и сетей на изолированные компоненты на основе их критичности и надежности. Например, использование микросервисной архитектуры, виртуализация, изоляция сетей и применение программных песочниц.
- Подтверждение целостности – установление факта повреждения критических элементов системы. Также сюда можно отнести использование программ, позволяющих доказать целостность и обеспечить доверие к артефактам и данным, начиная от подписи коммитов в репозитории и заканчивая хранением данных в блокчейне.
- Непредсказуемость – поддержание высокой степени неопределенности для злоумышленника. Изменение поведения системы или ее состояния в моменты времени, которые определяются случайным образом, снижение вероятности экстраполяции злоумышленником прошлых событий. Например, запрос на повторную аутентификацию с произвольным интервалом, выполнение рутинных

действий в разное время суток, ротация ролей и обязанностей, реализация случайного переключения каналов, использование случайной маскировки при динамической маскировке данных.

Все эти техники реализуются через совокупность организационных, технических, программно-аппаратных, морально-этических мер [24].

Описанный выше поход хорош для стратегического понимания. Однако на тактическом уровне необходимы более конкретные методы обеспечения киберустойчивости, которые будут полезны и применимы в реальных условиях.

В 2021 году корпорация MITRE выпустила документ [25], где определила тактические принципы обеспечения киберустойчивости или лучшие практики. В нем определены действия, которые необходимо предпринять «до взрыва» (т.е. до кибератаки):

- Прерывание атаки – в любой возникающей ситуации первым делом необходимо прервать атаку и усложнить жизнь атакующего.
- Архитектура с точки зрения защиты – основа киберустойчивости, которая включает сегментацию (должны быть заложены принципы изоляции и сегментации сети), скоординированную защиту (комплексная защита, использование нескольких уровней защиты, координация работы сотрудников) и разнообразие используемых технологий и систем.
- Безопасное администрирование – к скоординированной защите добавляются принципы ограничения привилегий и сегментации.
- Контроль доступа – ограничение возможностей атакующего. К принципам предыдущих пунктов добавляется мониторинг.
- Усиление настроек безопасности компонентов – один из ключевых пунктов, развивающий действия по скоординированной защите и ограничению привилегий. Включает минимизацию возможностей скомпрометированного сервиса скомпрометировать другие сервисы компонента и распространиться дальше, ограничение использования служб между компонентами,

определение стратегии, которая выявляет и изолирует уязвимые компоненты.

- Стратегии резервирования данных – разработка стратегии создания резервных копий данных, контроля их целостности и методов восстановления.
- Планирование и выполнение требований и инструкций для обеспечения непрерывности операций в киберпространстве (Cyber Continuity of Operations, COOP).

Также в данном документе определены действия, которые необходимо предпринять «после взрыва» (т.е. для противодействия атакам):

- Обеспечение COOP, включающее адаптивное реагирование, мониторинг и подтверждение целостности восстановленных систем и данных.
- Защищенная коммуникация – это безопасность процессов реагирования и восстановления от компрометации, включает сегментацию (контроль доступа к центру кибербезопасности и его логическое изолирование от остальной сети), подтверждение целостности (механизмы проверки данных для подтверждения целостности данных и подлинности отправителя) и резервирование каналов связи.
- Восстановление ключевых компонентов, включает адаптивное реагирование (динамическое восстановление критически важных активов или возможностей, определение и восстановление функциональных возможностей на основе критичности), координацию мер по восстановлению, обеспечение избыточности и также подтверждение целостности.
- Стратегии восстановления данных – восстановление данных с проверкой на возможность им доверять (адаптивное реагирование, обеспечение избыточности и также подтверждение целостности) .
- Расследование инцидента и проверка того, что системе можно доверять в дальнейшем.
- Улучшение системы с учетом полученного опыта и информации.

В международной практике существуют различные методики оценки киберустойчивости, одной из которых является *Cyber*

Resilience Review (CRR). CRR – это пакет методологических документов, разработанный Министерством внутренней безопасности США. Оценка предназначена для измерения киберустойчивости организации, а также для анализа недостатков с целью улучшения на основе признанной передовой практики. CRR основан на модели управления устойчивостью CERT [26], модели совершенствования процессов, разработанной Институтом разработки программного обеспечения Университета Карнеги-Меллона для управления операционной устойчивостью.

CRR оценивает корпоративные программы и практики в следующих областях [9]:

- управление активами;
- управление средствами контроля;
- управление конфигурацией и изменениями;
- управление уязвимостями;
- управление инцидентами;
- управление непрерывностью обслуживания;
- управление рисками;
- управление внешними зависимостями;
- обучение и осведомленность;
- ситуационная осведомленность.

Помимо самой методики оценки киберустойчивости, CRR содержит и руководства, которые помогут внедрить необходимые практики по каждой из вышеупомянутых областей.

Помимо аналитической оценки, как же проверить, что инфраструктура действительно киберустойчива на практике? Для этого существуют специализированные киберучения.

Киберучения – тренировочные мероприятия, цель которых заключается в оценке эффективности противодействия киберугрозам, включая оценку достаточности мер защиты и оценку и улучшение навыков специалистов, работающих в отделах кибербезопасности организаций. Киберучения заключаются в симуляции разнообразных угроз кибербезопасности виртуальных и физических инфраструктур.

Например, в 2022 году в США прошла серия учений Cyber Storm 8, в рамках которой более 2000 участников отрабатывали

свои планы реагирования на киберинциденты и определяли возможности для координации и обмена информацией. Сценарий учений состоял в появлении эксплойта DVER, который позволял исполнять удаленные команды на сервере для получения первичного доступа в систему. После чего атакующие использовали его для дальнейшего распространения в системе.

В России тоже проводятся подобные учения. Например, на Национальном киберполигоне проводятся как отраслевые учения, так и учения для начинающих специалистов и студентов.

1.2 Подходы к созданию современных приложений

В рамках данного раздела будут рассмотрены подходы и методологии создания приложений, которые на этапе проектирования закладывают свойства киберустойчивости, в отличие от монолитных приложений, которые на этапе создания данные факторы не учитывают.

1.2.1 Подход к созданию приложений в средах облачных вычислений (Cloud Native)

За короткое время облачные технологии и среды облачных вычислений стали движущей тенденцией в индустрии ПО, предоставив новый способ построения больших и сложных систем. Облачная среда меняет способ проектирования, внедрения, развертывания и эксплуатации систем. Сегодня распространены такие архитектуры ПО и подходы к проектированию, как микросервисы, контейнеры, service mesh, Cloud Native.

Cloud Native архитектура и связанные с ней технологии – это подход к созданию, развертыванию и управлению современными приложениями в средах облачных вычислений (частных, гибридных и публичных) [27]. Данный подход позволяет создавать более масштабируемые, отказоустойчивые, высокодоступные, легкоуправляемые, безопасные приложения. Основная задача Cloud Native – обеспечение скорости и гибкости разработки, повышение производительности, минимизация ошибок в условиях усложнения систем.

Приведем отличительные черты приложений Cloud Native архитектуры:

- Каждый сервис автономен и инкапсулирует свой собственный код, данные и зависимости. Он развертывается в программном контейнере и управляется оркестратором контейнеров.
- Вместо использования большой реляционной базы данных (БД) каждая служба владеет собственным хранилищем данных, тип которого зависит от ее потребностей. Некоторые службы могут зависеть от реляционной БД, а другие от БД NoSQL.
- Часть сервисов хранят свое состояние в распределенном кеше.
- Трафик проходит через службу шлюза API, которая отвечает за маршрутизацию трафика к основным сервисам.
- Приложение в полной мере использует функции масштабируемости, доступности и отказоустойчивости.

Полной противоположностью Cloud Native является монолитная архитектура – традиционный подход к разработке ПО, в которой одна база кода используется для выполнения нескольких бизнес-функций. Многие существующие сегодня приложения имеют монолитную архитектуру. Её суть состоит в том, что создается большое ядро приложения, содержащее всю доменную логику. Она может включать в себя различные модули, такие как модули авторизации, идентификации и управления каталогами, CRUD (четыре базовые функции, используемые при работе с БД: создание, чтение, модификация, удаление) и так далее. Все составные части очень тесно связаны друг с другом в рамках одного серверного процесса: модули могут использовать одни и те же подключения к реляционным БД, управлять различными интерфейсам.

Монолитные приложения имеют как свои недостатки, так и достоинства. Например, их очень удобно вертикально масштабировать, они легки в отладке, развертывании, тестировании и сборке. Недостатки в основном зависят от масштаба разработки. Со временем приложение разрастается, и возникают следующие проблемы:

- Приложение стало настолько сложным, что его логику не понимает ни один из членов команды или понимает только один. Иными словами, его «bus factor» (количество

разработчиков в команды, после потери которых проект не может быть дальше продолжен) равен единице.

- Сложность внедрения гибких методологий разработки.
- Сложность внесения изменений, так как каждое изменение может иметь непредсказуемые последствия.
- Реализация нового функционала или внесение исправлений становятся сложной, трудоемкой и дорогостоящей задачей, требующей пересборки всего приложения.
- Нестабильность одного компонента может привести к сбою всей системы.
- Ограничения в использовании новых технологий и библиотек.
- Со временем кодовая база ухудшается из-за внесения бесконечных «быстрых исправлений», наступает архитектурная эрозия.

Рассмотрим отличия сервисной архитектуры от монолитной на простом, но интересном примере, через одну из основных концепций DevOps – «домашние животные против крупного рогатого скота» (или «монолитная модель против сервисной»). Сравнение приведено в таблице 1.1.

Таблица 1.1 – Сравнение монолитной и сервисной моделей на примере «домашние животные против крупного рогатого скота»

Монолитная модель	В модели обслуживания домашних животных каждому серверу домашних животных даются имена, такие как Зевс, Арес, Аид, Посейдон и Афина. Они «уникальны, выращены с любовью, о них заботятся, и когда они заболевают, вы их лечите». Вы масштабируете их, делая их крупнее, и когда они недоступны, все это замечают. Примеры домашних серверов – мейнфреймы, одиночные серверы, балансировщики нагрузки, брандмауэры, системы баз данных и т.д.
Сервисная модель	В модели обслуживания крупного рогатого скота серверам присваиваются идентификационные номера, такие как web01, web02, web03, web04 и web05. Каждый сервер «почти идентичен друг другу» и «когда один болеет, вы заменяете его другим». Вы масштабируете их, создавая все новые и новые, и когда один из них недоступен, никто не замечает. Примеры серверов

	крупного рогатого скота – массивы веб-серверов, NoSQL кластеры, кластеры очередей, поисковые кластеры, хранилища данных с несколькими мастер-нодами, такие как Cassandra, сервера больших данных и т.д.

Обозначим эпохи развития технологий разработки ПО.

В железный век вычислительной техники только внедрение аппаратной виртуализации привело к появлению систем управления серверами. Использовались инструменты конфигурации изменений, такие как Puppet (2005 г.), CFEngine 3 (2008 г.) и Chef (2009 г.). Они позволили операторам настраивать парки систем с помощью автоматизации.

Первый облачный век – начальная эпоха, виртуализация была расширена, чтобы предложить IaaS (инфраструктура как услуга), которая виртуализировала всю инфраструктуру (сети, хранилище, память, процессор). Популярными платформами, предлагающими IaaS, являлись Amazon Web Services (2006 г.), Microsoft Azure (2010 г.), Google Cloud Platform (2011 г.). Такие сервисы привели к появлению инструментов оркестрации на основе push модели, таких как Salt Stack (2011 г.), Ansible (2012 г.) и Terraform (2014 г.). Эти инструменты позволили координировать состояние между облачным провайдером и приложением и, по сути, позволяли запрограммировать инфраструктуру в виде шаблона, называемого «инфраструктура как код» [1].

Во второй облачный век появилась возможность автоматизации для виртуализации инфраструктуры и изоляции приложений без необходимости виртуализации оборудования, что дублирует операционную систему для каждого приложения. Некоторые из этих технологий включают OpenVZ (2005 г.), Linux Containers или LXC (2008 г.) и Docker (2015 г.). В это время был разработан новый набор технологий для распределения ресурсов для контейнеров и планирования этих контейнеров в кластере серверов: Apache Mesos (2009 г.), Kubernetes (2014 г.), Nomad (2015 г.), Swarm (2015 г.). Внедрение контейнеров стало взрывным, поскольку Docker сам применяться повсеместно.

Век неизменяемой инфраструктуры (immutable production) начался, когда появился подход к управлению службами и

[1] Подход «инфраструктура как код» (IaC) – это процесс управления ИТ-инфраструктурой, при котором для управления ресурсами облачной инфраструктуры применяются рекомендации из сферы DevOps-разработки.

развертыванием программного обеспечения на ИТ-ресурсах, при котором компоненты заменяются, а не изменяются. Приложение или служба эффективно повторно развертываются каждый раз, когда происходит какое-либо изменение, а одноразовые контейнеры настраиваются при развертывании.

В современных условиях активно применяются платформы облачных вычислений, подразумевающие программирование с использованием подхода IaC с помощью контейнеров и оркестрации. Для нишевых случаев, когда требуется управлять особыми серверами, все еще используются платформы Ansible, Terraform и Chef. Kubernetes теперь является наиболее распространённым при управлении контейнерами с реализациями на всех популярных облачных платформах: Google Kubernetes Engine, Azure Kubernetes Service и AWS Elastic Kubernetes Service. В системах обработки больших данных и потоковой передачи в настоящий момент наиболее распространены инструменты Spark, Kafka, Flink, Storm, Hadoop, Cassandra.

Облачные системы включают *микросервисы* – популярный архитектурный паттерн для создания современных приложений.

Микросервисы спроектированы как распределенный набор небольших независимых сервисов, которые взаимодействуют через общую структуру. Вместе они составляют приложение. При этом каждый микросервис обладает следующими характеристиками [27]:

- реализует определенную и конкретную бизнес-задачу в более широком контексте предметной области;
- разрабатывается автономно и может быть развернут независимо;
- работает автономно, инкапсулируя собственную технологию хранения данных, зависимости и программную платформу;
- функционирует в рамках своего собственного процесса и взаимодействует с другими с помощью стандартных протоколов, таких как HTTP/HTTPS, gRPC, WebSockets или AMQP.

Плюсы микросервисов очевидны: независимость разработки, легкость масштабирования, прозрачные зависимости, гибкость тестирования и так далее.

Однако существует и ряд недостатков:

- Проблемы со взаимодействием. Тут можно выделить следующие вопросы, требующие внимания. Как интерфейсные клиентские приложения будут взаимодействовать с внутренними базовыми микросервисами? Возможно ли прямое общение или необходимо абстрагировать внутренние микросервисы с помощью шлюза, который обеспечивает контроль и безопасность? Разрешены ли прямые HTTP-вызовы, которые могут увеличить связанность и негативно влиять на производительность и гибкость?
- Устойчивость. Архитектура микросервисов переводит систему из внутрипроцессного сетевого взаимодействия во внепроцессное. Что происходит в распределенной архитектуре, когда служба *B* не отвечает на сетевой вызов службы *A*? Что происходит, когда служба *C* становится временно недоступной, а другие службы, вызывающие ее, блокируются?
- Распределение данных. Каждый микросервис инкапсулирует свои собственные данные, предоставляя доступ к операциям через общедоступный интерфейс. Как запрашивать данные? Как реализовать транзакцию в нескольких службах?
- Распределение секретов. Как микросервисы будут безопасно хранить секреты и конфиденциальные данные конфигурации? Как микросервисы будут управлять ими?

1.2.2 Методология создания приложений «программное обеспечение как услуга (The Twelve-Factor App)

Программное обеспечение как услуга (SaaS) – это модель распространения программного обеспечения, при которой облачный провайдер размещает приложения и делает их доступными конечным пользователям через Интернет [28]. В этой модели независимый поставщик ПО может заключить контракт со сторонним облачным провайдером на размещение приложения. Или облачный провайдер может также быть поставщиком программного обеспечения, в случае крупных компаний.

SaaS является одной из трех основных категорий облачных вычислений, наряду с инфраструктурой как сервисом (IaaS) и платформой как сервисом (PaaS).

The Twelve-Factor App или 12factor app – это методология для создания SaaS-приложений [29].

В данной методологии для описания процесса установки и настройки используется декларативные форматы для автоматизации настройки, которые значительно сокращают затраты времени и ресурсов для новых разработчиков, присоединяющихся к проекту. Приложения, созданные по этой методологии, имеют чистый контракт с базовой операционной системой, который обеспечивает максимальную переносимость между различными средами выполнения. Такие приложения подходят для развертывания на современных облачных платформах, что позволяет избежать необходимости использования собственных серверов и управления системами. Расхождения между средой разработки и средой выполнения минимизируются, что позволяет использовать непрерывное развертывание для достижения максимальной гибкости. Кроме того, технологии могут масштабироваться без существенных изменений в инструментах, архитектуре и методологии разработки.

Методология 12factor app может быть применена для приложений, написанных на любом языке программирования и использующих любые комбинации сторонних служб (БД, очереди сообщений, кэш-памяти, и т.д.).

Приложение, создаваемое по данной методологии, должно соответствовать нескольким базовым факторам (принципам):

1. Использование одной кодовой базы, отслеживаемой в системе контроля версий на множество развёртываний.
2. Необходимость явного объявления и изолирования зависимостей.
3. Конфигурация должна храниться в среде выполнения.
4. Сторонние службы должны считаться подключаемыми ресурсами. Код приложения не делает различий между локальными и сторонними сервисами (доступ по URL-адресу).
5. Каждый релиз должен иметь уникальный идентификатор.
6. Приложение должно запускаться как один или несколько процессов, не сохраняющих внутреннее состояние (или stateless).

7. Сервисы должны экспортироваться через привязку портов. Приложение должно быть полностью самодостаточным и не полагаться на инъекции веб-сервера во время выполнения для того, чтобы создать веб-сервис.
8. Приложение должно быть масштабируемым с помощью процессов. Любая программа после запуска представляет собой один или несколько работающих процессов.
9. Должна обеспечиваться возможность быстрого запуска и корректного завершения работы, т.е. процессы приложения могут быть запущены и остановлены в любой момент.
10. Паритет окружений разработки и исполнения. Приложение должно быть спроектировано для непрерывного развёртывания за счет минимизации различий между разработкой и работой приложения.
11. Должно выполняться логирование, которое обеспечивает наглядное представление поведения работающего приложения.
12. Разработчикам периодически необходимо выполнять разовые задачи администрирования и обслуживания приложения.

Методология 12factor app предоставляет четко определенные рекомендации по разработке микросервисов и является широко используемым шаблоном для запуска, масштабирования и развертывания приложений. Однако, если более плотно поработать с реальными крупными приложениями, работающими в Kubernetes, можно обнаружить, что некоторые важные факторы, о которых разработчикам следовало бы думать, отсутствуют в исходной методологии.

Разберем семь дополнительных факторов (принципов). Основные примеры будут приведены с ориентацией на Kubernetes, так как на сегодняшний день – это де-факто стандарт оркестрации сложных технических систем в веб [30].

Фактор №13 Наблюдаемость.

Приложения должны обеспечивать видимость текущего состояния и показателей. Распределенными системами обычно сложно управлять, поскольку несколько микросервисов работают одновременно в рамках одного приложения. По сути, многие части должны работать синхронно, чтобы система функционировала.

Если один микросервис дает сбой, система должна обнаружить его и исправить автоматически. Kubernetes предоставляет отличные возможности для защиты, такие как проверка готовности (rediness probe) и живости (liveness probe).

Разберем проверку готовности. Kubernetes использует проверки готовности, чтобы убедиться, что приложение готово к приему трафика. Если проверка дает сбой, Kubernetes прекращает отправку трафика в модуль до тех пор, пока она не вернет статус успеха.

Рассмотрим приложение, состоящее из трех микросервисов: интерфейс, бизнес-логика и базы данных. Для этого приложения интерфейс должен иметь конечную точку (endpoint) проверки готовности, чтобы проверить, готовы ли бизнес-логика и базы данных, прежде чем принимать трафик.

В качестве проверки может использоваться HTTP запрос, команда или даже TCP протокол, также может быть выполнено управление конфигурациями проверки. Например, можно указать, как часто они должны запускаться, каковы пороги успеха и неудачи и как долго ожидать ответов. Существует один очень важный параметр, который необходимо настроить при использовании тестов готовности – это параметр *initialDelaySeconds*, чтобы проверка не запускалась до тех пор, пока приложение не будет готово.

Второй параметр – проверка живучести. Kubernetes использует тесты, чтобы проверить, «живо» ли приложение или «мертво». Если приложение не работает, Kubernetes удаляет его и запускает новое для замены. Это подтверждает необходимость того, чтобы микросервисы не сохраняли состояние и были одноразовыми (фактор №10).

Иногда приложения имеют специфичные метрики, которые необходимо отслеживать. Для этого настраиваются пороговые значения и оповещения для этих метрик (например, количество транзакций в секунду).

Для этого удобно использовать стек мониторинга, состоящего из Prometheus и Grafana.

Фактор №14 Прогнозируемость

Приложения должны предоставлять рекомендации по ожидаемым ограничениям ресурсов.

Допустим, руководство выбирает команду для экспериментов с проектом на Kubernetes. Команда усердно работает над настройкой среды, и в итоге получается приложение, работающее с образцовым временем отклика и производительностью. Затем другая команда создает свое приложение и размещает его в той же среде. Когда второе приложение запускается, исходное приложение начинает испытывать снижение производительности. Когда первая команда начинает устранять неполадки, первое, на что нужно обратить внимание, – это вычислительные ресурсы, назначенные контейнерам (CPU и память), которых вероятно не хватает контейнерам.

В Kubernetes есть возможность устанавливать запросы и ограничения для контейнеров. Если контейнер запрашивает ресурс, Kubernetes планирует его только на узле, который может предоставить ему этот ресурс. Ограничения же гарантируют, что контейнер никогда не превысит определенное значение.

Фактор №15 Обновляемость

Приложения должны обновлять форматы данных предыдущих поколений. Часто требуются исправления безопасности или логики, и важно обновлять приложения без прерывания работы службы. Kubernetes предоставляет последовательные обновления (*RollingUpdate*) для приложений, которые можно обновлять без отключения службы, которые позволяют выполнять обновление одного модуля за раз, не отключая весь сервис.

Фактор №16 Принцип наименьших привилегий

Необходимо думать о каждом разрешении, которое выдается в контейнере, как о потенциальной атаке. Например, если контейнер работает с правами *root*, то любой, у кого есть доступ к контейнеру, может внедрить в него вредоносный процесс. Kubernetes предоставляет политики безопасности пода (Pod Security Policies), которые следует использовать для ограничения доступа к файловой системе, ресурсам и сервисам. Под – это абстрактный объект, представляющий собой группу из одного или нескольких контейнеров приложения.

Фактор №17 Проверяемость

Возможность аудита имеет решающее значение для любых действий, выполняемых в кластерах Kubernetes или в приложении. Например, если приложение обрабатывает платежные транзакции, то необходимо включить аудит, чтобы отслеживать

каждую транзакцию. Существует стандартный отраслевой формат Cloud Auditing Data Federation (CADF), не зависящий от облака. Его часто используют, передавая события, содержащие: идентификатор пользователя, выполнившего операцию; специальный целевой адрес и выполняемое действие, описывающее операцию; тип ресурса.

Фактор №18 Защищаемость

Необходимо защищать свое приложение и ресурсы. Приложениям нужна сквозная безопасность при работе. Для этого необходимо обеспечивать следующие механизмы:

- Проверка подлинности. Как правило, это выделенная служба, к которой подключаются микросервисы пользовательского интерфейса для проверки личности пользователей.
- Авторизация. К ней подключаются микросервисы пользовательского интерфейса для принудительного управления доступом на основе ролей к различным возможностям, предоставляемым в пользовательском интерфейсе.
- Управление сертификатами, т.е. создание, хранение и обновление цифровых сертификатов.
- Защита данных при передаче и хранении.
- Автоматическое сканирование на уязвимости в конвейере сборки.
- Статическое и динамическое сканирование исходного кода.

Фактор №19 Измеримость

Использование приложения должно быть измеримым. Вычислительные ресурсы, выделенные для запуска контейнеров, должны поддаваться измерению, а организации, использующие кластер, должны нести ответственность и иметь возможность оценивать затраты на поддержание инфраструктуры.

1.3 Методологии разработки DevOps и DevSecOps

1.3.1 Методология разработки DevOps

DevOps – это методология автоматизации технологических процессов сборки, настройки и развёртывания программного

обеспечения. Она включает набор процессов и инструментов, которые позволяют компании создавать и улучшать продукты быстрее, чем при использовании традиционных подходов к разработке ПО [31].

Термин DevOps – это комбинация слов «разработка» (development) и «эксплуатация» (operations), которая отражает интеграцию этих процессов в единый и непрерывный. Разработчики и тестировщики отвечают за development, а администраторы – за operations.

Данная методология способствует достижению следующих преимуществ:

- Скорость. Для эффективного внедрения новых функций и достижения задач бизнеса необходимо адаптироваться к быстро меняющимся рынкам. Микросервисы и непрерывная доставка позволяют оперативно управлять сервисами и быстро обновлять их.
- Быстрая доставка. Быстрое внедрение новых функций и исправлений позволяет оперативно реагировать на потребности рынка и создавать конкурентные преимущества. Непрерывная интеграция и непрерывная доставка автоматизируют процесс выпуска ПО, начиная с его сборки и заканчивая развертыванием.
- Надежность. Применение методов непрерывной интеграции и непрерывной доставки позволяет проверить функциональность и безопасность каждого изменения кода. Также следует отметить, что мониторинг и ведение журналов позволяют отслеживать производительность в режиме реального времени.
- Масштабирование. Автоматизация помогает эффективно управлять сложными или изменяющимися системами, при этом снижая риски отказов. Система может автоматически расширяться при высокой рабочей нагрузке и сокращаться обратно, когда потребности снижаются до нормальных. Использование подхода IaC облегчает управление средами разработки, тестирования и production, что позволяет повторно воспроизводить эти среды.
- Оптимизированная совместная работа. Группы разработки и эксплуатации тесно взаимодействуют, разделяют задачи и объединяют свои рабочие процессы. Это позволяет избежать

неэффективных действий и экономить время, например, сокращается время передачи задач от разработчиков инженерам по эксплуатации и исключается необходимость адаптации кода к конкретной среде его запуска.

- Безопасность. Методология DevOps может быть внедрена без высоких рисков кибербезопасности, благодаря автоматизированной политике соблюдения требований, точной настройке и управлению конфигурациями. Применение IaC и «политики как кода» позволяет определить требования и следить за их соблюдением на любом масштабе.

При всех достоинствах этого подхода можно выделить следующие недостатки:

- Ошибки проектирования приводят к трудно прогнозируемым сложностям на поздних циклах производства, когда изменения в инфраструктуре будут требовать значительных финансовых затрат.
- Высокая сложность организации эффективного взаимодействия между командами разнопрофильных специалистов.

Одна из практик DevOps – *непрерывная интеграция и доставка* (Continuous integration и Continuous delivery или CI/CD) – это методы, которые автоматизируют процесс выпуска ПО, от сборки и до развертывания. CI/CD за счет автоматизации способствует минимизации ошибок, повышению темпов сборки и качества разрабатываемого продукта в целом.

Одно из решений, которые необходимо принять, прежде чем применять IaC, состоит в том, какой же подход выбрать для автоматизации изменений среды: императивный или декларативный? Разница состоит в следующем: при императивном подходе программе указывают, как выполнить задачу, а при декларативном – просто конечную цель. Чаще всего применяется последний.

При императивном подходе в соответствии со скриптом изменения будут последовательно применены к контейнеру, виртуальной машине и виртуальному частному облаку. Такой подход позволяет подробно описать желаемые изменения, но если конфигурацию понадобится снова изменить после ее доставки на множество машин, придется менять и скрипт.

Для автоматического внесения изменений при декларативном подходе указывается конечная цель. Например, вместо использования интерфейса командной строки и перечисления всех шагов для создания конфигурации виртуальной машины просто запрашивается создание виртуальной машины в определенном домене. Задача будет автоматически выполнена. Декларативный подход позволяет просто указать то, что должно быть выполнено средствами автоматизации.

GitOps – это методология, которая позволяет перенести лучшие практики из области разработки ПО в инфраструктурные проекты. При ее использовании Git-репозитории становятся единственным источником достоверной информации о желаемом состоянии системы: код приложений и конфигурация инфраструктуры. При этом процесс ее развертывания автоматизируется за счет использования уже привычных инструментов CI/CD, что позволяет увеличить скорость создания, надежность и воспроизводимость решений.

Предположим, у нас есть Git-репозиторий с приложением и манифестами для его развертывания в конкретные окружения. И есть CI конвейер[2], который запускается на любое изменение по триггеру и развертывает приложение в Kubernetes.

Проблема состоит в том, что если что-то изменяется в production инфраструктуре (это может произойти по различным причинам), то описанное в конфигурационном файле состояние будет отличаться от фактического.

Подход GitOps основан на идее того, что существует не прекращаемый цикл «reconciliation loop» – который продолжает следить за источником достоверной информации (в данном случае за Git-репозиторием) и непрерывно синхронизировать его состояние в реальный мир (в данном случае в Kubernetes-кластер).

GitOps подразумевает, что существует некий GitOps-оператор или контроллер, который делает то же самое, что и контроллеры в Kubernetes, но смотрит не в Kubernetes API, а в конкретный Git-репозиторий. То есть, берет состояние Git-репозитория и перекладывает его в Kubernetes. А также следит за тем, чтобы оно всегда ему соответствовало.

[2] CI конвейер (CI pipeline) – процесс автоматизации(последовательность шагов), который используют для сборки, запуски тестов и развертывания программного продукта на протяжение всего жизненного цикла разработки [32].

Эталонный вариант имплементации GitOps предлагает использование двух Git-репозиториев:

1. Репозиторий, в котором хранится то, что не зависит от конкретного окружения: исходные файлы приложения, файлы конфигураций инфраструктуры и файлы конфигураций контейнеров.
2. Репозиторий манифестов описывает конкретное приложение и как оно должно быть запущено в кластере. Для тестовой среды существует файлы конфигураций инфраструктуры с одними значениями, а для production среды с другими.

Второй репозиторий используется GitOps-оператором для получения достоверной информации. Всё, что в нём описано, должно быть синхронизировано с Kubernetes. Может быть создан конвейер, который будет обновлять состояние второго репозитория из первого.

Может использоваться несколько Git-репозиториев, а GitOps-оператор (например, Flux) может синхронизировать их состояние с несколькими Kubernetes-кластерами. В итоге один и тот же GitOps-оператор может использоваться для нескольких команд разработчиков и заниматься доставкой сразу нескольких не зависящих друг от друга приложений.

Важным является выбор подходящего метода организации репозитория. Могут быть использованы:

- монорепозиторий, описывающий все приложения, который отдается на управление GitOps-оператору.
- отдельные Git-репозитории на каждое приложение, которые будут обновляться независимо.
- гибридный вариант, когда для каждой команды разработки выделяется монорепозиторий, содержащий все необходимые манифесты в рамках одного проекта. Такой подход зарекомендовал себя как наиболее простой и не погружающий в «Ад зависимостей»[3].

Когда манифесты отправляются на развертывание, необходима их валидация. Прежде чем объединить две или

[3] Ад зависимостей (dependency hell) – это антипаттерн управления конфигурацией, разрастание графа взаимных зависимостей программных продуктов и библиотек, приводящее к сложности установки новых и удаления старых продуктов [33].

несколько веток разработки, сливая изменения в одну общую целевую ветку (например, *main*), нужно выполнить *helm template*, либо *kubectl apply* с ключём *--dry-run*, чтобы удостовериться, что с манифестами все в порядке и синтаксис не нарушен.

Секреты в репозитории необходимо хранить безопасно, так чтобы их нельзя было просто извлечь (в том числе пользователям с соответствующими правами на доступ). Это, например, необходимо, чтобы предотвратить риск утечки кодового репозитория. Можно было бы хранить их отдельно, например, в хранилище Vault, но это нарушало бы принципы GitOps.

Для обеспечения безопасности применяется асимметричное шифрование, так что расшифровать секрет сможет только контроллер (для этой цели можно использовать *sealed secrets*).

Werf – это утилита для построения полного цикла доставки CI/CD с Kubernetes. Авторы werf предложили схожий с GitOps подход и назвали его гитерминизм. Он схож с GitOps в плане организации декларативной и версионируемой инфраструктуры. Однако для применения изменений вместо pull-модели (т.е. оператора- «синхронизатора», как это сделано у упомянутых Flux) используется push-модель.

1.3.2 Методология разработки DevSecOps

DevSecOps – это развитие концепции DevOps, где помимо автоматизации затрагиваются вопросы обеспечения качества и надёжности кода [34].

DevSecOps, как и DevOps, также можно назвать целой культурой. DevSecOps – это философия интеграции методов безопасности в процесс DevOps. DevSecOps инженеру также важна командная работа. DevSecOps с самого начала жизненного цикла приложения занимается его безопасностью, создавая и используя различные средства защиты [34].

Команды разработчиков ПО используют следующие инструменты DevSecOps для оценки недочетов в безопасности во время разработки ПО, их обнаружения и оповещения о них:

- Статическое тестирование безопасности приложений заключается в анализе исходного кода и обнаружении уязвимостей в нем.

- Анализ состава ПО включает автоматизированный анализ используемого в проекте ПО (в том числе с открытым исходным кодом) на известные уязвимости. Выполняется с целью управления рисками, обеспечения безопасности и соответствия лицензиям.
- Интерактивное тестирование безопасности приложений применяется для оценки потенциальных уязвимостей приложения в рабочей среде. Состоит из специальных мониторов безопасности, которые запускаются из приложения.
- Динамическое тестирование безопасности приложений имитирует действия злоумышленников.

Существует еще одна концепция безопасной разработки – SDL или SDLC (Security development lifecycle) от компании Microsoft. SDL позволяет снизить вероятность возникновения уязвимостей, максимально усложнить их эксплуатацию и ускорить исправление [35]. Требования по безопасности участвуют на каждом этапе разработки, от этапа разработки требований до выпуска окончательной версии приложения.

1.4 Лучшие практики для обеспечения киберустойчивости

1.4.1 Модель безопасности Нулевое доверие

Нулевое доверие (Zero Trust) – модель безопасности с нулевым доверием, также называемая архитектурой нулевого доверия, в основе которой лежит отсутствие доверия ко всем объектам IT-инфраструктуры организации. Zero Trust является целостным подходом обеспечения сетевой безопасности, который включает в себя несколько различных принципов и технологий [36].

ZTNA (Zero Trust Network Access) – это основная технология, связанная с архитектурой Zero Trust [37].

Традиционный подход к обеспечению сетевой безопасность основан на концепции «за́мок со рвом», в которой трудно осуществить несанкционированный доступ из-за пределов сети, но по умолчанию доверяют всем, кто находится внутри сетевого периметра.

Проблема традиционного подхода заключается в том, что, как только злоумышленник получает доступ к сети, он получает полную свободу действий внутри. Этот недостаток также

усугубляется тем, что компании больше не хранят свои данные в одном месте. Сегодня информация часто распространяется среди поставщиков облачных услуг, что затрудняет использование единого элемента управления безопасностью для всей сети.

Модель безопасности нулевым доверием означает, что по умолчанию никому не доверяют (ни внутри, ни за пределами сети) и требуют проверки от всех, кто пытается получить доступ к сетевым ресурсам. Этот дополнительный уровень безопасности снижает вероятность утечки данных. Исследования показали, что средняя стоимость одной утечки превышает три миллиона долларов. Учитывая данный факт, многие организации стремятся принять политику безопасности Zero Trust.

Философия Zero Trust лучше подходит для современных ИТ-систем, чем традиционные подходы безопасности. При таком большом количестве пользователей и устройств, имеющих доступ к внутренним данным, а также при наличии данных, хранящихся как внутри, так и вне сети (в облаке), гораздо безопаснее предположить, что ни один пользователь или устройство не заслуживает доверия, чем предположить, что превентивные меры безопасности сработали.

Основное преимущество применения принципа нулевого доверия заключается в том, что он позволяет уменьшить поверхность атаки. Он также минимизирует ущерб, ограничивая нарушение одной небольшой областью с помощью микросегментации, что снижает и стоимость восстановления. Zero Trust также уменьшает вероятность кражи учетных данных пользователя и фишинговых атак, требуя нескольких факторов аутентификации.

Проверяя каждый запрос, Zero Trust снижает риски, связанные с уязвимыми устройствами, в том числе устройствами IoT, которые часто трудно защитить и обновить.

Модель Zero Trust также может быть использована для поддержки удаленной работы, так как она расширяет безопасный контроль доступа к подключениям из любого места, в то время как виртуальные частные сети создают узкие места и могут снизить производительность удаленных сотрудников. Zero Trust позволяет быстро предоставить ограниченный доступ с минимальными привилегиями внешним сторонам, которые используют устройства, которыми не управляют внутренние ИТ-группы.

Отметим, что сеть с Zero Trust проверяет любой запрос, независимо от его источника или назначения. Это также может помочь снизить использование неавторизованных облачных сервисов, контролируя или блокируя использование несанкционированных приложений.

1.4.2 Разбор инцидентов безопасности. Постмортем

Известно, что изменения в системе могут вносить нестабильность, которая вызывает инциденты. Внедрение DevOps позволяет ускорить выпуск релизов и не ожидать выпуска одного большого релиза, а поставлять результаты работы команды разработки небольшими порциями. Это снижает риск сбоев в конкретной версии. Однако отметим, что увеличение количества релизов не обязательно приведет к уменьшению количества инцидентов, на которые должны реагировать дежурные группы.

Основной обязанностью группы реагирования на инциденты является количественная оценка и, при необходимости, смягчение последствий. В результате сервис возвращается к нормальным условиям работы. При этом анализ первопричины и принятие мер предотвращения инцидента не относятся к этому процессу. Отметим, что если такой анализ не проводится, причины, повлекшие инцидент, остаются без лечения. Это приводит к тому, что инциденты начинают множиться, а каскадные ошибки становятся частью еженедельной рутины. В итоге количество времени, которое команда DevOps тратит на реагирование на инциденты, растет, а качество обслуживания постоянно снижается. Необходим ретроспективный анализ и разбор инцидентов.

Постмортем (postmortem) – это бизнес-процесс, который позволяет проектной команде, руководству проекта и другим заинтересованным сторонам просматривать и оценивать результаты в конце проекта или после разрешения инцидента [38]. Постмортем обычно связан с неудачей проекта или инцидентом, таким как сбой в работе ИТ-службы, приводящий к простою или другим последствиям для бизнеса.

Простыми словами, постмортем – это процесс разбора и анализа инцидента, завершающийся созданием документа [39], описывающего сам инцидент, его результат и меры, которые можно принять, чтобы избежать повторения [40].

Расследование должно запускаться всякий раз, когда инцидент требует ответа от дежурного инженера. Типичный постмортем начинается с регистрации объективных доказательств:

- что вызвало инцидент;
- какой урон он нанес;
- время его обнаружения и нейтрализации;
- принятые меры по нейтрализации;
- анализ причин.

На основании приведенных выше сведений следует провести анализ. Анализ обычно выполняется дежурным, который отреагировал на инцидент, и может включать других членов команды, которые либо помогли нейтрализовать последствия, либо проанализировать основную причину. В процессе анализа необходимо найти ответы на следующие вопросы:

- Триггер
    - Сколько оповещений об инциденте было получено?
    - Сработал ли триггер своевременно или существовала возможность зарегистрировать событие раньше?
- Импакт
    - Была ли угроза достаточной, чтобы спровоцировать инцидент? Или должна быть выполнена калибровка триггеров?
    - Были ли предприняты адекватные меры для смягчения воздействия? Если нет, необходимо ли улучшение руководящих принципов или дополнительное вложение средств в обучение?
    - Достаточно ли быстро было нейтрализовано воздействие? Может ли что-либо быть сделано, чтобы сократить время смягчения последствий?
- Первопричина
    - Возможно ли устранить причину инцидента?
    - Если причина должна быть устранена, что именно должно быть сделано для ее устранения?

На основе анализа должно быть составлен резюмирующий документ, включающий извлеченные уроки и последующие задачи,

зарегистрированные и расставленные по приоритетам. Последующие задачи обычно включают задачи для:

- инженеров по устранению основной причины;
- DevOps-инженеров по улучшению настройки мониторинга;
- менеджеров по улучшению процессов.

Внедрить процесс написания посмортемов в организации, которая ранее их не проводила, достаточно трудоемко. Как и в случае любого нового или изменяющегося процесса, внедрение изменений требует времени и усилий на всех уровнях.

Глава 2

Практические работы

2.1 Указания к выполнению и проверке практических работ

Практические работы выполняются обучающимися индивидуально или в группе в течение курса. Выполнение практической работы состоит в выполнении заданий, оформлении отчета и защиты в форме устного доклада. Задания необходимо выполнить с учетом указаний преподавателя. Оставшиеся невыполненными пункты задания практического занятия студент обязан доделать самостоятельно. На занятии студент готовит отчет о проделанной работе, фиксируя процесс выполнения работы. Представление отчета и защита работ проходят в конце курса. После проверки отчета преподаватель проводит устный опрос студентов для контроля усвоения ими основных умений и навыков (студенты должны знать смысл полученных ими результатов и ответы на контрольные вопросы). В случае, если оформление отчета и доклад обучающегося во время защиты соответствуют указанным ниже требованиям, обучающийся получает максимальную оценку.

Перечень заданий для практических работ

№ п/п	Наименование практической работы	Цель работы
1	Настройка системы контроля версий и автоматизация сборки приложений	Изучение системы контроля версий git и освоение автоматизации сборки приложений с использованием GitHub CI.

2	Изучение принципов передачи информации в сети	Изучение протоколов передачи данных в сети, инструментов и индикаторов активности.
3	Анализ защищенности k8s	Изучение принципов анализа защищенности k8s инфраструктуры.

Шкала оценивания и критерии оценки

К практической работе предъявляются следующие требования:

1. *к выполнению заданий* – в работе должны быть:
    - представлены в логической последовательности основные этапы исследования или решения,
    - указаны используемые теоретические положения и методы,
    - получены точные результаты и требуемые графические изображения (снимки экрана основных шагов выполнения работы);
2. *к оформлению отчета* – отчет должен быть представлен в печатном или электронном виде в форматах doc, docx, pdf и содержать:
    - титульный лист (название работы, ФИО исполнителей, номера групп, ФИО проверяющего),
    - условия всех заданий,
    - решение (исследование), его теоретическое обоснование, численные результаты,
    - графики или рисунки, иллюстрирующие решение каждой задачи работы,
    - ссылки на теоретический материал, используемый при исследовании и решении,
    - выводы;
3. *к докладу* – для доклада отводится от 3 до 5 минут. Во время доклада оценивается качество устного изложения материала и ответы на вопросы по теме работы. Доклад должен содержать:
    - постановку задачи,
    - изложение основных этапов решения, их результаты и оценку,
    - выводы.

Основаниями для снижения оценки являются:

- небрежное, непоследовательное выполнение, неполнота и нерациональность решения (исследования),

- необоснованное и некорректное применение методов решения,
- некорректность результатов, некорректная обработка результатов измерений,
- низкое качество графического материала (в том числе неверный выбор масштаба чертежей, отсутствие указания единиц измерения на графиках),
- неполнота отчета,
- низкое качество оформления отчета,
- низкая содержательность и низкое качество устного изложения материала
- некорректность и неполнота ответа на дополнительные вопросы,
- отсутствие необходимых разделов.

Оценка / Уровень	Критерий
«5» (отлично) / Высокий	выполнены все задания, обучающийся четко и без ошибок ответил на все контрольные вопросы
«4» (хорошо) / Средний	выполнены все задания, обучающийся ответил на все контрольные вопросы с замечаниями
«3» (удовлетворительно) / Низкий	выполнены все задания с замечаниями, обучающийся ответил на все контрольные вопросы с замечаниями
«2» (неудовлетворительно) / Неудовлетворительный	обучающийся не выполнил или выполнил задания неправильно, ответил на контрольные вопросы с ошибками или не ответил на контрольные вопросы

2.2 Настройка системы контроля версий и автоматизация сборки приложений

Цель работы

изучение системы контроля версий git и освоение автоматизации сборки приложений с использованием GitHub CI.

Задачи

1. изучить и установить инструменты для работы с системой контроля версий git;
2. создать репозиторий и опубликовать код минимального приложения с использованием Docker;
3. изучить принципы использования GitHub CI;
4. создать GitHub Action для автоматической сборки приложения.

Описание

Работа выполняется индивидуально. Включает в себя написание кода, настройку инструментов и экспериментальное изучение.

Ход выполнения

1. Зарегистрироваться на GitHub.
2. Установить git.
3. Сделать форк репозитория типового приложения, выданного преподавателем.
4. На основе данного приложения доработать функционал в соответствии с вариантом задания.
5. Изучить документацию GitHub Actions.
6. Добавить Action, который будет собирать Docker-образ приложения.

Варианты заданий

Вариант задания формируется из случайной комбинации следующих условий:

- Триггер срабатывания: push, pull request.

- Использование следующих возможностей: переменные (variables), сохранение данных между запусками, вычисляемые выражения (Expressions), использование функций проверки статуса, зависимости.
- Стратегии сборки: OC, node.
- Использование секретов или переменных.
- Язык программирования: Python, Go, Node.js.

2.3 Изучение принципов передачи информации в сети

Цель работы

изучение протоколов передачи данных в сети, инструментов и индикаторов активности.

Задачи

1. выполнить набор действий по передаче информации и файлов между несколькими виртуальными машинами с использованием протоколов и инструментов: SSH, Telnet, FTP, SFTP, SMB;
2. проанализировать передачи с использованием анализатора трафика WireShark с целью выявления подозрительной активности и извлечения данных.

Описание

Работа выполняется индивидуально. Включает в себя написание кода, настройку инструментов и экспериментальное изучение.

Студент последовательно выполняет набор действий по передаче информации и файлов между несколькими виртуальными машинами с использованием протоколов и инструментов: SSH, Telnet, FTP, SFTP, SMB;

Студент анализирует подобные передачи с использованием анализатора трафика WireShark с целью выявления подозрительной активности и извлечения данных.

Ход выполнения

1. передать файл, содержащий требуемую строку, с АРМ на Windows сервер:
    - по протоколу HTTP с использованием PowerShell на стороне сервера;
    - по протоколу HTTP с использованием CertUtil на стороне сервера.
2. передать большой бинарный файл с использованием подключения RDP с Windows сервера на АРМ. Буфер обмена для подключения должен быть отключен;
3. передать файл, содержащий требуемую строку, по SMB протоколу с Windows сервера на АРМ;

4. передать файл, содержащий требуемую строку с АРМ на Linux сервер:
    - по протоколу SCP;
    - по протоколу FTP через FTP клиент;
    - по протоколу TFTP через TFTP клиент;
    - через netcat.
5. По каждому из предыдущих шагов изучить запись трафика в WireShark и письменно зафиксировать следующие данные:
    - IP адрес клиента;
    - md5 сумма передаваемых бинарных файлов (по мере возможности) ;
    - содержимое передаваемых текстовых файлов (по мере возможности).

Реализация практической работы

Стенд содержит 2 общие для всех виртуальные машины. Каждый студент также получает 1 персональную рабочую машину.

Студент работает с персональным Kali Linux. У каждого студента есть сетевой доступ к двум отдельным машинам (Windows, Linux).

На общих машинах необходимо создать отдельных пользователей для каждого студента.

Все персональные логины и пароли должны быть доступны студентам (можно разместить их на рабочих столах в TXT файле).

Требования к рабочей машине студента:

- Операционная система Kali Linux (2022.2)
- Привилегии на установку ПО
- Требования к общему Windows Server
    - Операционная система Windows Server (2019)
    - Настроенный RDP доступ
    - Установленный Powershell
    - Установленный Certutil
- Требования к общему Linux Server

- Операционная система Ubuntu Server 20.04
- Установленный SSH с доступом по логину/паролю
- Установленный FTP сервер(ftp)
- Установленный SMB клиент (smbclient)

2.4 Анализ защищенности k8s

Цель работы

изучение принципов анализа защищенности k8s инфраструктуры.

Задачи

1. изучить основы концепций k8s и Docker;
2. найти типовые ошибки конфигурации k8s;
3. изучить методы автоматизации анализа безопасности манифестов k8s;
4. освоить методы эксплуатации уязвимостей небезопасного кластера k8s.

Описание

Работа выполняется в группах по 2–3 человека. Включает в себя выполнение поискового исследования, категоризацию и экспериментальное изучение.

Ход выполнения

1. Установить инструментарий для работы с k8s при помощи команд:
    - `curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"`
    - `sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl`
    - `kubectl get nodes`
2. Скачать и установить пакетный менеджер Helm, который можно использовать для загрузки готовых зависимостей (RabbitMQ, MongoDB) и для упаковки приложения[4].
    - Скачать готовый релиз https://get.helm.sh/helm-v3.8.0-darwin-amd64.tar.gz
        - Распаковать бинарный файл командой `tar -zxvf helm-v3.8.0-darwin-amd64.tar.gz`
        - Устанавить бинарный файл командой `mv linux-amd64/helm /usr/local/bin/helm`

[4] Для установки на ОС, отличной от MacOS, необходимо выбрать соответствующий релиз (например, заменить darwin на Linux).

- Проверить версию через helm version
- Загрузить Lens https://k8slens.dev/
- Подключиться к кластеру, используя kubectl и , выполнив команды
  - `docker ps`
  - `docker exec -it kind-control-plane bash`
- Настроить Lens и изучить окружение кластера
- Пробросить порт и подключиться с правами user:user, выполнив команды
  - `kubectl port-forward ssh-server 9027:2222`
  - `ssh -p 9027 user@localhost`
  - `k8s kubectl get pods --namespace default`
- Подключиться к ssh-server и осуществить поиск токенов и enumeration.
- Подключиться к dnsutils и изучить DNS пространство кластера, выполнив команды:
  - `cd /var/run/secrets/kubernetes.io/serviceaccount`
  - `ls -larth`
  - `curl --cacert ${CACERT} --header "Authorization: Bearer ${TOKEN}" -X GET ${APISERVER}/api`
  - `curl --cacert ${CACERT} --header "Authorization: Bearer ${TOKEN}" -X GET ${APISERVER}/api/v1/secrets`
  - `curl --cacert ${CACERT} --header "Authorization: Bearer ${TOKEN}" -X GET ${APISERVER}/api/v1/namespaces/${NAMESPACE}/secrets`

Варианты заданий и настройка

Рекомендуемое расположение строчек-ответов (флагов), которые необходимо обнаружить при выполнении:

- в pod ssh-server
- в env pod dnsutils
- в git репозитории в узле кластера
- в секрете, доступном через сервисный аккаунт
- в mounted директории

- в соседнем ssh-server контейнере, в который можно подключиться, если знать его DNS и набрутить пароль
- в DNS PTR записи
- в ключе ETCD

В качестве референсных уязвимых конфигураций рекомендуется ориентироваться на следующие примеры:

- https://github.com/madhuakula/kubernetes-goat/tree/master/scenarios
- https://madhuakula.com/kubernetes-goat/docs/

Приложение А. Тест для оценки входных знаний

Обучающимся предлагается выполнить входное тестирование, которое позволяет узнать начальный уровень знаний и определить наличие необходимых знаний перед изучением курса. Тест включает вопросы с выбором ответа из числа предложенных.

Входное тестирование

1. Какой идентификатор присваивают уязвимости в каком-либо конкретном продукте или системе?
    a. cwe
    b. cve
    c. mitre
    d. cvss
2. Какой из способов проверки системы защиты информации представляет собой поиск уязвимостей без их дальнейшей эксплуатации?
    a. пентест
    b. аудит
    c. анализ защищенности
    d. red team
3. Подход к проверке безопасности системы, при котором проверяющему неизвестно ее внутреннее устройство
    a. blackbox
    b. redbox
    c. greybox

d. greenbox

4. Угрозой какой составляющей информационной безопасности является DDoS-атака?

   a. конфиденциальность
   b. целостность
   c. доступность
   d. актуальность

5. Что из этого является риском информационной безопасности?

   a. Получение злоумышленником доступа в локальную сеть
   b. Удаленное подключение с паролем по умолчанию
   c. Раскрытие паролей доступа из-за возможности атаки SMB Relay

6. Что из перечисленного является видом интернет-мошенничества, цель которого – получить данные пользователей?

   a. Фишинг
   b. Спуфинг
   c. Трекинг
   d. Хакинг

7. Какой регулятор осуществляет лицензирование деятельности по контролю защищенности информации от несанкционированного доступа в РФ?

   a. ФСБ
   b. ФСТЭК
   c. МВД
   d. ФСО

Приложение Б. Вопросы для самоконтроля и тест для оценки выходных знаний

В данном разделе представлены дополнительные вопросы тестового формата, которые могут использоваться для самоконтроля освоения материалов курса. В конце пособия приведены правильные ответы.

Общие вопросы

- Что такое DevOps и какие принципы он подразумевает?
- Какие основные инструменты вы используете для автоматизации развертывания и управления инфраструктурой?
- Какие преимущества относительно безопасности предлагает DevSecOps по сравнению с традиционным подходом к разработке ПО?
- Каковы основные цели DevSecOps и как они отличаются от целей DevOps?
- Каким образом DevSecOps влияет на облачные технологии и микросервисную архитектуру?
- Как вы управляете доступом ваших разработчиков к production среде?
- Какой процесс обнаружения и исправления багов или уязвимостей существует в рамках DevSecOps?
- Какие типы тестирования допустимы в DevOps-подходе?
- Какую роль играет непрерывная интеграция и непрерывная доставка в DevOps и DevSecOps?

- Какие метрики и инструменты вы используете для мониторинга и анализа производительности ваших систем?
- Как вы управляете версиями и контролируете изменения кода и инфраструктуры в DevOps-проектах?
- Как вы обеспечиваете безопасность ваших клиентов, работая в среде DevOps?
- Какие основные преимущества и вызовы сопряжены с внедрением DevOps в больших организациях?
- Каким образом DevSecOps способствует снижению затрат на безопасность и повышению эффективности?

Выходное тестирование

1. Что такое уязвимость нулевого дня?
    a. уязвимость, о которой еще никому не известно
    b. уязвимость, связанная с неверной обработкой даты и времени
    c. уязвимость, для которой не существует исправления
    d. уязвимость, для которой выпущен патч или обновление безопасности
2. Что из нижеперечисленного является базой типов уязвимостей?
    a. exploit-db
    b. cvss
    c. cwss
    d. cwe
3. На каком уровне модели OSI работает протокол UDP?
    a. Транспортный
    b. Сетевой
    c. Канальный
    d. Сеансовый
4. Что является результатом атаки CAM-overflow?
    a. Получение доступа ко всем VLAN сетям на trunk порту
    b. Перенаправление сетевого трафика с легитимного узла на узел атакующего
    c. Исчерпание пула ip-адресов у DHCP-сервера
    d. Коммутатор входит в режим с пропусканием трафика и отправляет все кадры всем устройствам в сети
5. Какой маске подсети соответствует префикс /23?

a. 255.255.255.0
b. 255.255.254.0
c. 255.255.192.0
d. 255.255.255.128

6. Зачем нужна программа Chisel?

a. Для настройки прокси сервера
b. Для организации DNS/ICMP туннеля
c. Для организации TCP/UDP туннеля над протоколом HTTP
d. Для организации скрытого HTTP туннеля над ICMP протоколом

7. Какой из протоколов поддерживает Proxychains?

a. OSCP
b. UDP
c. ICMP
d. TCP

8. Для чего Sshuttle наиболее пригоден?

a. для сокрытия оригинального IP-адреса
b. для маскировки VPN туннеля
c. для доступа в закрытую сеть с целью администрирования
d. для передачи данных поверх UDP протокола

9. Для чего используется ForwardAgent yes в настройках SSH?

a. для входа без пароля
b. для проброса вашего ключа на сервер, на который вы зашли по ключу
c. для увеличения анонимности соединения
d. для проброса всех портов

10. Какой уровень будет у уязвимости с CVSS вектором: AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H ?

a. 10
b. 7
c. 5
d. 1

11. Как называются метрики, которые вносят в общую оценку поправку на полноту имеющейся информации об уязвимости, наличие кода эксплойта и доступность исправлений?

a. Базовые
b. Общие
c. Временные
d. Контекстные

12. Инструмент для проверки различных уязвимостей JWT:

a. hashcat
b. JWTtool
c. JWTcrack
d. hydra

13. Что произойдет при некорректной конфигурации и передаче в параметр redirect_uri значения host.com/callback&redirect_uri=evil-user.net?

a. После завершения OAUTH пользователь будет переадресован на вредоносный сайт
b. В случае, если OAUTH был инициирован на веб-сайте host.com с незащищенным подключением, токен будет отправлен на evil-user.net
c. authorization токен попадет на evil-user.net
d. authorization токен будет выдан для доступа к данным пользователя evil-user

Приложение В. Перечень вопросов и рекомендации для проведения промежуточной аттестации

Рекомендации по методике оценивания

Промежуточная аттестация по дисциплине осуществляется с помощью следующих оценочных средств: собеседование по билетам.

- формат проведения – устный зачет в формате ответов на вопросы билета;
- время, отводимое на подготовку ответа – 1 час;
- требования к ответу – полное раскрытие темы вопросов, наличие и разбор практического примера по теме каждого вопроса;
- порядок формирования билета – два случайных вопроса из перечня вопросов (1-й вопрос – с 1 по 11 вопрос из перечня вопросов, 2-й вопрос – с 12 по 35 вопрос.).

Шкала оценивания и критерии оценки

Критерии оценки	Минимальное количество баллов	Максимальное количество баллов
1. Уровень усвоения материала, предусмотренного программой	3	5
2. Умение выполнять задания, предусмотренные программой	3	5

3. Уровень раскрытия причинно-следственных связей	1,2	2
4. Уровень раскрытия междисциплинарных связей	1,2	2
5. Качество ответа (его общая композиция, логичность, общая эрудиция)	1,2	2
6. Изложение: полнота, аргументированность	1,8	3
7. Деловые и волевые качества: ответственное отношение к работе, стремление к достижению высоких результатов.	0,6	1
Итого:	**12**	**20**

Оценка		**Минимальное количество баллов**	**Максимальное количество баллов**
«5» (отлично)	Зачтено	18	20
«4» (хорошо)		15	17
«3» (удовлетворительно)		12	14
«2» (неудовлетворительно)	Не зачтено	0	11

Знания, умения и навыки обучающихся при промежуточной аттестации определяются оценками «зачтено (отлично)», «зачтено (хорошо)», «зачтено (удовлетворительно)», «не зачтено (неудовлетворительно)».

«Зачтено (отлично)» – обучающийся глубоко и прочно усвоил весь программный материал, исчерпывающе, последовательно, грамотно и логически стройно его излагает, не затрудняется с ответом при видоизменении задания, свободно справляется с задачами и практическими заданиями, правильно обосновывает принятые решения, умеет самостоятельно обобщать и излагать материал, не допуская ошибок.

«Зачтено (хорошо)» – обучающийся твердо знает программный материал, грамотно и по существу излагает его, не допускает существенных неточностей в ответе на вопрос, может правильно применять теоретические положения и владеет необходимыми умениями и навыками при выполнении практических заданий.

«Зачтено (удовлетворительно)» – обучающийся усвоил только основной материал, но не знает отдельных деталей, допускает неточности, недостаточно правильные формулировки, нарушает последовательность в изложении программного материала и испытывает затруднения в выполнении практических заданий.

«Не зачтено (неудовлетворительно)» – обучающийся не знает значительной части программного материала, допускает существенные ошибки, с большими затруднениями выполняет практические задания, задачи.

Перечень вопросов

1. Жизненный цикл веб-приложений
2. Понятие и основные принципы обеспечения кибербезопасности и киберустойчивости
3. Инструменты логирования и анализа событий
4. Уязвимости и угрозы, оценка киберустойчивости и киберучения
5. Cloud native
6. Методология 12 factor apps
7. Упущенные факторы из методологии 12 factor app
8. DevOps
9. DevSecOps
10. Постмортем
11. Принцип нулевого доверия
12. Основные типы аудита безопасности информационных систем
13. Матрица MITRE ATT&CK, понятие killchain
14. Классификация уязвимостей и методы их оценки
15. Методы анализа исходных данных об исследуемом объекте
16. Сбор информации с использованием DNS
17. Методы сбора информации о персонале организации
18. Методы сканирования периметра

19. Основные протоколы прикладного уровня и их эксплуатация на периметре организации
20. Парольная политика и методы защиты от атак методом перебора
21. Анализ безопасности конфигураций веб-сервера
22. Анализ структуры веб-сервиса, предположение векторов атаки
23. Типичные уязвимости веб-сервисов
24. Автоматизированные средства эксплуатации уязвимостей
25. Типы операционных систем и их особенности
26. Виды удаленного доступа и способы его получения
27. Особенности закрепления в различных ОС
28. Способы сокрытия присутствия
29. Привилегии и методы их повышения
30. Туннелирование для перемещения по корпоративной сети
31. Прокси и принципы его функционирования
32. Основные методы атак с использованием социальной инженерии
33. Методы защиты от атак с использованием социальной инженерии
34. Tor и анонимизация в сети
35. Методы сокрытия тестирования на проникновение

Приложение Г. Ответы к тестам

Входное тестирование

№ вопроса	Правильный ответ
1.	b
2.	c
3.	a
4.	c
5.	c
6.	a
7.	b

Выходное тестирование

№ вопроса	Правильный ответ
1.	c
2.	d
3.	a
4.	d
5.	b
6.	c
7.	d
8.	c
9.	b
10.	a
11.	c
12.	b
13.	c

Список использованных источников и рекомендованная литература

Список использованных источников

1. Безкоровайный М.М., Татузов А.Л. Кибербезопасность подходы к определению понятия // Вопросы кибербезопасности.– 2014.– №1(2).– С. 22-27.
2. Алпеев А.С. Терминология безопасности: кибербезопасность, информационная безопасность // Вопросы кибербезопасности.– 2014.– №5(8) .– С. 39-42.
3. Котенко И.В. Оценка киберустойчивости компьютерных сетей на основе моделирования кибератак методом преобразования стохастических сетей // Информатика и автоматизация.– 2017.– №6(55) .– С. 160-184.
4. *Социологический словарь. — М.*: *Экономика. Н. Аберкромби, С. Хилл, Б.С. Тернер. 2004.*
5. ГОСТ Р 53114-2008 Защита информации. Обеспечение информационной безопасности в организации. Основные термины и определения [Электронный ресурс]. Режим доступа: https://docs.cntd.ru/document/1200075565 (дата обращения: 25.01.2024).
6. ГОСТ Р 50922-2006 Национальный стандарт Российской Федерации. Защита информации. Основные термины и определения [Электронный ресурс]. Режим доступа: https://docs.cntd.ru/document/1200058320 (дата обращения: 25.01.2024).

7. Cybersecurity. Guidelines for Internet security [Электронный ресурс]. Режим доступа: https://www.iso.org/ru/standard/76070.html (дата обращения: 25.01.2024).
8. Что такое киберриски и как застраховать свой бизнес [Электронный ресурс]. Режим доступа: https://ir.alfastrah.ru/posts/271 (дата обращения: 25.01.2024).
9. Кибербезопасность, киберустойчивость, киберучения [Электронный ресурс]. Режим доступа: https://www.securityvision.ru/blog/razbor-terminologii-kiberbezopasnost-kiberustoychivost-kiberucheniya-chto-eto/ (дата обращения: 25.01.2024).
10. Сбербанк. Кибрарий Интерактивный словарь [Электронный ресурс]. Режим доступа: http://www.sberbank.ru/ru/person/kibrary/vocabulary/kiberataka (дата обращения: 25.01.2024).
11. Что такое кибербезопасность [Электронный ресурс]. Режим доступа: https://aws.amazon.com/ru/what-is/cybersecurity/ (дата обращения: 25.01.2024).
12. GDPR [Электронный ресурс]. Режим доступа: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679 (дата обращения: 25.01.2024).
13. Федеральный закон от 14.07.2022 № 266-ФЗ «О внесении изменений в Федеральный закон О персональных данных»
14. Кибербезопасность [Электронный ресурс]. Режим доступа: https://www.ptsecurity.com/ru-ru/research/knowledge-base/chto-takoe-kiberbezopasnost/ (дата обращения: 25.01.2024).
15. Методика оценки угроз безопасности информации [Электронный ресурс]. Режим доступа: https://fstec.ru/dokumenty/vse-dokumenty/spetsialnye-normativnye-dokumenty/metodicheskij-dokument-ot-5-fevralya-2021-g (дата обращения: 25.01.2024).
16. Cyber Kill Chain [Электронный ресурс]. Режим доступа: https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html (дата обращения: 25.01.2024).
17. Мельников, А. В. Алгоритм оценки относительного уровня опасности совместной эксплуатации уязвимостей информационной безопасности на основе CVSS / А. В. Мельников, В. Е. Чирков // Вестник Воронежского института МВД России.– 2019.– № 1.– С. 37-44.
18. Федеральная служба по техническому и экспортному контролю. Банк данных угроз безопасности информации

[Электронный ресурс]. Режим доступа: https://bdu.fstec.ru/ (дата обращения: 25.01.2024).
19. OWASP [Электронный ресурс]. Режим доступа: https://owasp.org/ (дата обращения: 25.01.2024).
20. MITRE [Электронный ресурс]. Режим доступа: https://www.mitre.org/ (дата обращения: 25.01.2024).
21. MITRE ATT&CK [Электронный ресурс]. Режим доступа: https://attack.mitre.org/ (дата обращения: 25.01.2024).
22. NIST Releases SP 800-160 Vol. 2: Developing Cyber Resilient Systems – A Systems Security Engineering Approach [Электронный ресурс]. Режим доступа: https://csrc.nist.gov/News/2019/sp-800-160-vol2-developing-cyber-resilient-systems (дата обращения: 25.01.2024).
23. Принцип минимальных привилегий: что это и зачем он нужен [Электронный ресурс]. Режим доступа: https://www.kaspersky.ru/blog/what-is-the-principle-of-least-privilege/36808/ (дата обращения: 25.01.2024).
24. Домбровская Лариса Александровна, Яковлева Наталья Александровна, Стахно Роман Евгеньевич Современные подходы к защите информации, методы, средства и инструменты защиты // Наука, техника и образование. 2016. №4 (22). https://cyberleninka.ru/article/n/sovremennye-podhody-k-zaschite-informatsii-metody-sredstva-i-instrumenty-zaschity (дата обращения: 26.03.2024).
25. Cyber Resiliency Engineering Framework [Электронный ресурс]. Режим доступа: https://www.mitre.org/news-insights/publication/cyber-resiliency-engineering-framework (дата обращения: 25.01.2024).
26. CERT Resilience Management Model (CERT-RMM) Version 1.2 [Электронный ресурс]. Режим доступа: https://insights.sei.cmu.edu/library/cert-resilience-management-model-cert-rmm-version-12/ (дата обращения: 25.01.2024).
27. What is Cloud Native [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/definition (дата обращения: 25.01.2024).
28. Immutable infrastructure [Электронный ресурс]. Режим доступа: https://www.techtarget.com/searchitoperations/definition/immutable-infrastructure (дата обращения: 25.01.2024).
29. The twelve-factor app [Электронный ресурс]. Режим доступа: https://12factor.net/ru/ (дата обращения: 25.01.2024).

30. 7 Missing Factors from 12-Factor Applications [Электронный ресурс]. Режим доступа: https://www.ibm.com/blog/7-missing-factors-from-12-factor-applications/ (дата обращения: 25.01.2024).
31. Что такое DevOps и зачем он нужен разработчикам [Электронный ресурс]. Режим доступа: https://education.yandex.ru/journal/chto-takoe-devops-i-zachem-on-nuzhen-razrabotchikam (дата обращения: 25.01.2024).
32. Что такое GitLab CI Pipeline? [Электронный ресурс]. Режим доступа: https://stepik.org/lesson/878067/step/1?unit=882548 (дата обращения: 25.01.2024).
33. Энциклопедия Руниверсалис. Dependency hell [Электронный ресурс]. Режим доступа: https://руни.рф/Dependency_hell (дата обращения: 25.01.2024).
34. Какая разница между DevOps и DevSecOps [Электронный ресурс]. Режим доступа: https://pvs-studio.ru/ru/blog/posts/0710/ (дата обращения: 25.01.2024).
35. Как внедрить Secure Development Lifecycle и не поседеть. Рассказ Яндекса на ZeroNights 2017 [Электронный ресурс]. Режим доступа: https://habr.com/ru/companies/yandex/articles/346426/ (дата обращения: 25.01.2024).
36. Что такое архитектура нулевого доверия? [Электронный ресурс]. Режим доступа: https://www.trendmicro.com/ru_ru/what-is/what-is-zero-trust/zero-trust-architecture.html (дата обращения: 25.01.2024).
37. Yan X., Wang H. Survey on zero-trust network security //Artificial Intelligence and Security: 6th International Conference, ICAIS 2020, Hohhot, China, July 17–20, 2020, Proceedings, Part I 6.– Springer Singapore, 2020.– С. 50-60.
38. Управление инцидентами для высокоскоростных команд [Электронный ресурс]. Режим доступа: https://www.atlassian.com/ru/incident-management/handbook/postmortems (дата обращения: 25.01.2024).
39. How to structure a Postmortem document after an incident [Электронный ресурс]. Режим доступа: https://www.pixelmatters.com/blog/how-to-structure-a-post-mortem-document-after-an-incident (дата обращения: 25.01.2024).

40. Shaaban A., Abdelbaki N. Comparison study of digital forensics analysis techniques; findings versus resources //Procedia Computer Science.– 2018.– Т. 141.– С. 545-551.

Рекомендованная литература

1. Шаньгин, В.Ф. Информационная безопасность [Электронный ресурс]: учебное пособие / В.Ф. Шаньгин.– Электрон. дан.– Москва : ДМК Пресс, 2014.– 702 с.
2. Hoffman, A. Web Application Security: Exploitation and Countermeasures for Modern Web Applications: O'Reilly Media.– 2020.
3. Бегаев А.Н., Бегаев С.Н., Федотов В.А. Тестирование на проникновение: Учебное пособие / Реценз. : Кременчуцкий А.Л., проф., к.т.н.– Санкт-Петербург: Университет ИТМО, 2018.– 43 с.
4. Браницкий А. А., Котенко И. В. Открытые программные средства для обнаружения и предотвращения сетевых атак //Защита информации. Инсайд.– 2017.– №. 3.– С. 58-66.
5. Игнатьев, Е. Б. Основы криптографии: учебное пособие / Е. Б. Игнатьев.– Иваново: ИГЭУ, 2020.– 88 с.
6. Торстейнсон, П. Криптография и безопасность в технологии .NET / П. Торстейнсон, Г. А. Ганеш; под редакцией С. М. Молявко ; перевод с английского В. Д. Хорева.– 4-е изд.– Москва: Лаборатория знаний, 2020.– 482 с.– ISBN 978-5-00101-700-4.
7. Введение в теоретико-числовые методы криптографии: учебное пособие для спо / М. М. Глухов, И. А. Круглов, А. Б. Пичкур, А. В. Черемушкин.– Санкт-Петербург: Лань, 2021.– 396 с.– ISBN 978-5-8114-6926-0.
8. OWASP Testing Guide [Электронный ресурс]: учеб. пособие – 224 с.– Режим доступа: https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Testing_Guide_v4.pdf (дата обращения: 17.01.2024)
9. Bielska A. Open Source Intelligence Tools and Resources Handbook, 2018.
10. Илякова И.Е. Конкурентная разведка / 2-е изд. Учебное пособие для вузов, 2022.
11. OSINT или разведка по открытым источникам [Электронный ресурс] Режим доступа:

https://habr.com/ru/company/deiteriylab/blog/595801/. (дата обращения: 17.01.2024)
12. Херинг, М. DevOps для современного предприятия : учебное пособие / М. Херинг ; перевод с английского М. А. Райтмана.– Москва : ДМК Пресс, 2020.– 232 с.– ISBN 978-5-97060-836-4.
13. Wediman G. Penetration Testing: A Hands-On Introduction to Hacking.– No Stretch Press, 2014.– 528 p.– Режим доступа: https://repo.zenk-security.com/Magazine%20E-book/Penetration%20Testing%20-%20A%20hands-on%20introduction%20to%20Hacking.pdf (дата обращения: 17.01.2024)
14. Бирюков А.А. Информационная безопасность: защита и нападение / Бирюков А.А. Издательство "ДМК Пресс" 2017.– 434 стр.
15. Оголюк А.А. Защита приложений от модификации. Дополнительные материалы / Оголюк А.А. Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики 2014.– 122 стр.

Менщиков Александр Алексеевич

Заколдаев Данил Анатольевич

Воробьева Алиса Андреевна

Введение в кибербезопасность и киберустойчивость

Учебно-методическое пособие

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49, литер А