

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

по дисциплине «Безопасность жизненного цикла»

Направление подготовки 09.04.04 – Программная инженерия.
Профиль подготовки – Разработка, мониторинг и управление жизненным циклом инженерных систем.
Квалификация выпускника - магистратура.
Очная форма обучения.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

БЛОК 1. SHIFT LEFT: ИНТЕГРАЦИЯ БЕЗОПАСНОСТИ В РАЗРАБОТКУ

Лабораторная работа №1. Интеграция SAST в CI/CD (Semgrep / CodeQL)
Лабораторная работа №2. Анализ зависимостей и формирование SBOM (SCA)
Лабораторная работа №3. Динамический анализ безопасности API (DAST)
Лабораторная работа №4. Безопасность контейнеров: сканирование образов и Pod Security Admission
Лабораторная работа №5. Compliance as Code с Open Policy Agent (OPA/Rego)

БЛОК 2. SUPPLY CHAIN SECURITY И БЕЗОПАСНОСТЬ СБОРКИ

Лабораторная работа №6. Подпись артефактов с помощью Cosign
Лабораторная работа №7. Формирование SBOM и интеграция с Dependency-Track
Лабораторная работа №8. Безопасность CI/CD пайплайна (Hardening)
Лабораторная работа №9. Анализ угроз автономной системы по методологии STRIDE

БЛОК 3. БЕЗОПАСНОСТЬ АВТОНОМНЫХ СИСТЕМ

Лабораторная работа №10. Анализ защищённости протокола MAVLink
Лабораторная работа №11. Настройка безопасности в ROS 2 (DDS Security)
Лабораторная работа №12. Безопасное OTA-обновление для БПЛА
Лабораторная работа №13. Безопасность наземной станции управления (GCS)
Лабораторная работа №14. Анализ безопасности цепочки поставок компонентов БПЛА

БЛОК 4. CONTINUOUS COMPLIANCE И ИТОГОВЫЙ ПРОЕКТ

Лабораторная работа №15. Continuous Compliance с OPA и Cloud Custodian
Лабораторная работа №16. Генерация отчётов для аудита (Compliance Reports)
Лабораторная работа №17. Оценка безопасности фреймворка автономной навигации
Лабораторная работа №18. Комплексный проект: Обеспечение безопасности жизненного цикла автономной системы

ПЕРЕЧЕНЬ ИСПОЛЬЗУЕМЫХ ИНСТРУМЕНТОВ И ТЕХНОЛОГИЙ

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА И ИСТОЧНИКИ

ВВЕДЕНИЕ

Настоящие методические указания предназначены для выполнения лабораторных работ по дисциплине «Безопасность жизненного цикла», изучаемой во 2-м семестре магистратуры по направлению 09.04.04 «Программная инженерия» (профиль «Разработка, мониторинг и управление жизненным циклом инженерных систем»).

Актуальность дисциплины. Современные инженерные системы — от облачных платформ до беспилотных летательных аппаратов (БПЛА) — характеризуются высокой сложностью, распределённой архитектурой и критическими требованиями к информационной безопасности. Традиционный подход, при котором безопасность обеспечивается на финальных стадиях разработки или после ввода в эксплуатацию, демонстрирует низкую эффективность и высокую стоимость устранения уязвимостей. Концепция Shift Left Security (сдвиг безопасности влево) предполагает интеграцию практик безопасной разработки на самых ранних этапах жизненного цикла, а методология DevSecOps делает безопасность неотъемлемой частью конвейера непрерывной интеграции и доставки (CI/CD).

Особую значимость безопасность жизненного цикла приобретает для автономных систем (БПЛА, робототехнические комплексы, беспилотный транспорт), где компрометация канала связи, подмена прошивки или атака на цепочку поставок компонентов может привести не только к финансовым потерям, но и к физическому ущербу, угрозе жизни людей и нарушению функционирования критической информационной инфраструктуры (КИИ).

Цель лабораторного практикума — формирование у обучающихся системных знаний и практических навыков по интеграции практик безопасности на всех этапах жизненного цикла разработки инженерных систем, обеспечению непрерывного соответствия (continuous compliance) требованиям безопасности и защите автономных систем от киберугроз.

Задачи лабораторного практикума:

1. Освоить методологию и инструменты DevSecOps для «сдвига безопасности влево» (Shift Left), включая SAST, DAST, IAST, SCA применительно к инженерным системам.
2. Освоить практики управления цепочкой поставок (Supply Chain Security): формирование SBOM, проверка целостности артефактов (Cosign), анализ уязвимостей зависимостей.
3. Научиться применять политики безопасности контейнеров (Pod Security Standards, Admission Controller) и сканирования образов в CI/CD.
4. Сформировать навыки проведения анализа угроз кибербезопасности для автономных систем (БПЛА, роботы) по методологии STRIDE.
5. Изучить методы защиты каналов связи (MAVLink, ROS 2 DDS) и безопасной OTA-доставки прошивок.
6. Освоить инструменты непрерывного соответствия (continuous compliance) для автоматизированного сбора доказательств и генерации отчётов.
7. Изучить правовые и регуляторные аспекты безопасности в РФ (152-ФЗ, приказы ФСТЭК) для инженерных систем.

Место дисциплины в образовательной программе. Дисциплина относится к вариативной части, формируемой участниками образовательных отношений. Она опирается на знания и навыки, полученные в 1-м семестре в рамках дисциплины «Безопасность жизненного цикла» (базовые понятия SAST/SCA/DAST, инструменты Semgrep, Trivy, OWASP ZAP), а также на компетенции,

сформированные в курсах «Архитектура и проектирование высоконагруженных систем» и «Системная интеграция и управление конфигурациями». Данная дисциплина не дублирует материал 1-го семестра, а развивает его в контексте сложных инженерных систем, безопасности цепочки поставок и уникальной проблематики защиты автономных систем.

Формируемые компетенции:

- ПК-8. Способен интегрировать практики безопасности на всех этапах ЖЦ, включая SAST, DAST, SCA, безопасность контейнеров и облачной инфраструктуры (ИД-5пк-8 — сканирует IaC конфигурации, ИД-8пк-8 — обеспечивает continuous compliance).
- ПК-11. Способен проводить анализ угроз кибербезопасности для автономных систем (БПЛА, роботы) и внедрять меры защиты (ИД-1пк-11 — проводит анализ угроз, ИД-3пк-11 — обеспечивает защиту каналов связи, ИД-5пк-11 — внедряет безопасные OTA-обновления).

Лабораторная работа №1

Интеграция SAST в CI/CD (Semgrep / CodeQL)

1. Цель

Сформировать практический навык встраивания статического анализа безопасности (SAST) в конвейер непрерывной интеграции (CI/CD), обеспечивающий автоматическое выявление уязвимостей на ранних этапах разработки и блокировку сборочной линии при обнаружении дефектов критического уровня (Shift Left Security).

2. Задачи

- Создать тестовое приложение (Python / JavaScript), содержащее типовые уязвимости (SQL-инъекция, жёстко закодированные секреты и др.).
- Написать конфигурацию CI/CD-пайплайна (GitLab CI) с шагом статического анализа.
- Внедрить инструмент Semgrep (или CodeQL) и выполнить сканирование кодовой базы.
- Настроить Quality Gate: автоматический провал пайплайна при обнаружении уязвимостей уровня HIGH или CRITICAL.
- Сформировать отчёт в формате SARIF и проанализировать его содержимое.

3. Теоретическое обоснование

SAST (Static Application Security Testing) – метод анализа исходного кода/байт-кода без фактического выполнения программы. Инструменты SAST (Semgrep, CodeQL, SonarQube и др.) ищут паттерны уязвимостей на основе сигнатур, абстрактных синтаксических деревьев (AST) и графов потока данных.

Shift Left – практика переноса контроля безопасности на максимально ранние стадии жизненного цикла (написание кода, commit, сборка), что позволяет сократить стоимость исправления дефектов.

Semgrep – легковесный инструмент с открытым исходным кодом, использующий декларативный язык правил для поиска шаблонов. Поддерживает Python, JavaScript, Java, Go и др. Интеграция в CI/CD сводится к вызову утилиты semgrep и анализу кода возврата.

Quality Gate – условие, при невыполнении которого CI/CD-конвейер помечается как неуспешный. Для SAST типичный критерий: отсутствие уязвимостей с severity $\geq$ HIGH. Semgrep позволяет задать --error для указанных уровней, что приводит к ненулевому коду выхода.

SARIF (Static Analysis Results Interchange Format) – стандарт представления результатов статического анализа (OASIS), поддерживаемый большинством SAST-инструментов и платформ (GitHub, GitLab, Azure DevOps).

4. Задания для индивидуального выполнения

Каждый студент выполняет вариант согласно последней цифре номера студенческого билета (или списка группы) после утверждения преподавателем. Допускается согласование замены инструмента и языка по желанию.

4.1. Базовый уровень (15 вариантов)

Выполните интеграцию SAST в GitLab CI с блокировкой на уровне HIGH. Язык, уязвимость и инструмент заданы вариантом.

Вариант	Язык	Тип уязвимости	Инструмент
1	Python	SQL injection	Semgrep
2	Python	Hardcoded secret (API key)	Semgrep
3	Python	Command injection	Semgrep
4	Python	XSS (Cross-Site Scripting)	Semgrep
5	JavaScript	SQL injection (NoSQL)	Semgrep
6	JavaScript	Hardcoded JWT secret	Semgrep
7	JavaScript	Prototype pollution	Semgrep
8	Python	Insecure deserialization (pickle)	Semgrep
9	JavaScript	Open redirect	Semgrep
10	Python	Path traversal	Semgrep
11	JavaScript	Arbitrary code eval (eval())	Semgrep
12	Python	Weak cryptography (MD5)	Semgrep
13	JavaScript	XPath injection	Semgrep
14	Python	LDAP injection	Semgrep
15	JavaScript	Sensitive data exposure in logs	Semgrep

Требования к результату базового уровня:

- Работающий CI/CD-пайплайн в GitLab (можно локальный раннер), падающий при наличии уязвимости HIGH.
- Сгенерированный SARIF-отчёт.
- Минимум одна подтверждённая уязвимость именно заданного типа.

4.2. Средний уровень (15 вариантов)

Расширьте задачу базового уровня написанием пользовательского правила Semgrep для поиска нетипового дефекта или интеграцией CodeQL.

Вариант	Задача
1	Создать YAML-правило Semgrep для поиска использования subprocess.call с shell=True в Python
2	Правило для обнаружения JavaScript-кода, добавляющего innerHTML без очистки
3	Правило для Python-функций, выполняющих eval() с конкатенацией строк
4	Правило для поиска os.popen в Python
5	Правило для обнаружения MongoDB-запроса, собранного через конкатенацию строк (NoSQL injection)
6	Правило для Python-функции exec() с параметром, поступающим от пользователя
7	Правило для JavaScript eval(variable)
8	Интегрировать CodeQL (GitHub Actions) для Python/Flask и показать отчет SARIF
9	Интегрировать CodeQL для JavaScript/Node.js и сравнить результаты с Semgrep
10	Правило Semgrep, детектирующее отсутствие параметризованных запросов в Django (raw SQL)
11	Правило для Python: использование pickle.loads на данных от пользователя
12	Правило для JavaScript: RegExp с пользовательским вводом без экранирования (ReDoS)
13	Правило для Python: использование yaml.load без SafeLoader
14	Интегрировать CodeQL с политикой блокировки на уровне error (severity $\geq$ high)

Вариант	Задача
15	Правило для Python: вызов <code>os.system</code> с форматированием строк

Требования: базовые условия + собственное правило / альтернативный движок, доказательство его срабатывания (ложные/истинные примеры), сохранение SARIF-отчёта.

4.3. Повышенный уровень (15 вариантов)

Углублённая настройка пайплайна, обработка ложных срабатываний, интеграция с внешними системами.

Вариант	Задача
1	Настройка подавления (suppression) ложных срабатываний через <code>.semgrepignore</code> и inline-комментарии <code># nosemgrep</code> . Продемонстрировать рабочий пайплайн.
2	Интеграция Semgrep с платформой DefectDojo (локальный или облачный инстанс) через загрузку SARIF.
3	Сравнительный отчёт: запустить Semgrep и CodeQL на одном проекте, собрать метрики (TP/FP) для заданных уязвимостей.
4	Настройка пайплайна с двумя SAST-инструментами (Semgrep + CodeQL) и объединение результатов в один SARIF.
5	Реализация кастомного Quality Gate: пайплайн проваливается только при количестве HIGH > 2.
6	Автоматическая генерация issue в GitLab при обнаружении уязвимости с severity HIGH.
7	Создание bash-скрипта для разбора SARIF и вывода сводки в читаемом виде (уязвимость, файл, строка).
8	Настройка incremental scanning (только изменённые файлы) с помощью Semgrep.
9	Сборка собственного Docker-образа с Semgrep и предустановленными кастомными правилами, использование его в CI.
10	Интеграция Semgrep с GitHub Advanced Security (Code Scanning) и отображение алертов в Security tab.

Вариант	Задача
11	Написание правила Semgrep на основе dataflow (pattern-inside, pattern-sinks) для обнаружения tainted-поточков.
12	Конфигурация пайплайна, который при провале отправляет уведомление в Telegram/Slack с деталями.
13	Реализация политики: если найдена уязвимость CRITICAL, откатывается merge request (через API GitLab).
14	Интеграция Semgrep в pre-commit хук (local) и в CI, сравнение поведения.
15	Разработка GitHub Actions workflow с CodeQL и Semgrep с созданием статус-отчёта на основе сравнения результатов.

Требования: полная автоматизация, демонстрация нестандартного сценария, анализ результатов. Варианты можно комбинировать при согласовании с преподавателем.

5. Контрольные вопросы

1. Что такое SAST и в чём его отличие от DAST?
2. Объясните принцип Shift Left и его преимущества в безопасной разработке.
3. Как Semgrep анализирует код? В чём особенности его движка по сравнению с традиционными анализаторами на базе AST?
4. Каково назначение формата SARIF? Приведите примеры его полей.
5. Чем Quality Gate в CI/CD отличается от простого выполнения сканирования? Как настроить аварийное завершение пайплайна при высоких уязвимостях?
6. Назовите ограничения статического анализа (ложные срабатывания, пропуск уязвимостей). Как можно уменьшить их влияние?
7. Какое основное правило (YAML) вы использовали для собственного детектирования? Опишите его структуру.
8. Чем CodeQL отличается от Semgrep в плане анализа потоков данных?
9. Как организовать безопасное хранение секретов в CI/CD, чтобы инструменты SAST не реагировали на тестовые переменные?
10. Объясните смысл параметра --error в Semgrep. Как он влияет на пайплайн?

6. Требования к отчёту

Отчёт оформляется в электронном виде (PDF) и должен содержать:

- Титульный лист (название ЛР, вариант, ФИО, группа).
- Цель и задачи работы.

- Краткое описание тестового приложения (листинг фрагментов с уязвимостями, пояснения).
- Конфигурация пайплайна (YAML-файл CI/CD).
- Скриншоты запуска пайплайна (успешный + проваленный из-за Quality Gate).
- Содержимое SARIF-отчёта (или его фрагмент с комментариями).
- Описание выполненного индивидуального задания (правило, интеграция, подавление и т.д.) с подтверждающими скриншотами.
- Ответы на контрольные вопросы (письменно).
- Выводы о проделанной работе.

7. Критерии оценивания

Оценка формируется по следующему принципу:

7.1. «Удовлетворительно» (базовый уровень):

- Тестовое приложение реализовано с заданной уязвимостью.
- Пайплайн GitLab CI включает шаг SAST с Semgrep и завершается ошибкой при HIGH.
- SARIF-отчёт сформирован.
- Отчёт содержит все обязательные разделы.
- Даны верные ответы не менее чем на 60% контрольных вопросов.

7.2. «Хорошо» (средний уровень):

- Выполнены все требования базового уровня.
- Реализовано индивидуальное задание среднего уровня: написано и протестировано кастомное правило Semgrep или произведена интеграция CodeQL.
- Продемонстрировано срабатывание правила на уязвимом коде и молчание на безопасном.
- Дан сравнительный анализ результатов (где применимо).

7.3. «Отлично» (повышенный уровень):

- Выполнены все требования среднего уровня.
- Реализовано одно из заданий повышенного уровня с полной демонстрацией работоспособности.
- Проведён анализ эффективности SAST в контексте Shift Left, сформулированы рекомендации по снижению ложных срабатываний.
- Ответы на контрольные вопросы полны и аргументированы.

7.4. Понижение оценки внутри каждого уровня:

- Отсутствие одного из обязательных элементов отчёта – снижение на 0.5 балла.
- Неполный ответ на контрольные вопросы (менее 60% для базового, менее 80% для среднего и повышенного) – оценка не выставляется, работа отправляется на доработку.

- Некорректная работа Quality Gate (пайплайн не падает при наличии HIGH) – несоответствие базовому уровню.

Успешное выполнение всех запланированных лабораторных работ цикла гарантирует уверенное владение компетенциями ПК-8 (в части интеграции практик SAST в конвейер) и формирует базу для последующих занятий по безопасности цепочки поставок и автономных систем.

Лабораторная работа №2

Анализ зависимостей и формирование SBOM (SCA)

1. Цель

Сформировать практические навыки выявления уязвимостей в сторонних компонентах (SCA – Software Composition Analysis), генерации спецификации программного обеспечения (SBOM) в формате CycloneDX и автоматической блокировки конвейера CI/CD при обнаружении зависимостей с критическими уязвимостями.

2. Задачи

- Создать тестовый проект на Python или Node.js, использующий библиотеки с известными уязвимостями.
- Выполнить анализ состава зависимостей с помощью Trivy (или OWASP Dependency-Check) и получить отчёт об обнаруженных CVE.
- Сгенерировать SBOM (Software Bill of Materials) в формате CycloneDX с помощью Syft (или Trivy).
- Настроить Quality Gate в CI/CD на основе CVSS-оценок: пайплайн должен завершаться ошибкой при наличии уязвимостей с $CVSS \geq 7.0$ (HIGH / CRITICAL).
- Проанализировать полученные отчёты и предложить меры по обновлению или замене опасных зависимостей.

3. Теоретическое обоснование

SCA (Software Composition Analysis) — класс инструментов, идентифицирующих сторонние компоненты (библиотеки, фреймворки) и проверяющих их на наличие известных уязвимостей (CVE) в базах данных NVD, OSV, GitHub Advisory Database и др.

SBOM (Software Bill of Materials) — машиночитаемая опись всех компонентов, входящих в программный продукт. Форматы: CycloneDX (OWASP) и SPDX (Linux Foundation). SBOM необходим для прозрачности цепочки поставок и быстрого реагирования на инциденты (например, Log4Shell).

Trivy — универсальный сканер безопасности (Aqua Security). Поддерживает сканирование образов контейнеров, файловых систем, репозиториях, а также генерацию SBOM.

OWASP Dependency-Check — инструмент SCA, работающий с Java, .NET, Python, Node.js и др. Использует собственную базу CVE.

Syft — инструмент от Anchore для генерации SBOM из образов контейнеров, файловых систем и архивов.

CVSS (Common Vulnerability Scoring System) — метрика опасности уязвимости (0–10). Градации:

- None (0.0)
- Low (0.1–3.9)
- Medium (4.0–6.9)
- High (7.0–8.9)
- Critical (9.0–10.0)

Quality Gate по CVSS ≥ 7.0 позволяет блокировать внедрение кода с опасными зависимостями на ранних этапах.

4. Задания для индивидуального выполнения

Студент выполняет вариант согласно последней цифре номера зачётной книжки/списка.
Допускается использование эквивалентных инструментов по согласованию с преподавателем.

4.1. Базовый уровень (15 вариантов)

Выполнить настройку SCA и генерацию SBOM. Язык, уязвимая библиотека и инструмент заданы вариантом.

Вар.	Язык	Пакет с CVE (пример)	Инструмент SCA	Генератор SBOM
1	Python	requests==2.21.0	Trivy	Syft
2	Python	flask==0.12.3	Trivy	Syft
3	Python	pillow==6.0.0	Trivy	Syft
4	Node.js	lodash==4.17.15	Trivy	Syft
5	Node.js	express==4.16.1	Trivy	Syft
6	Node.js	axios==0.19.0	Trivy	Syft
7	Python	django==2.0	Dependency-Check	Syft
8	Python	numpy==1.16.4	Dependency-Check	Syft
9	Node.js	mongoose==5.4.0	Dependency-Check	Syft
10	Python	pyyaml==5.1	Trivy	Trivy (встр.)
11	Node.js	handlebars==4.5.0	Trivy	Trivy
12	Python	lxml==4.3.0	Trivy	Syft
13	Node.js	jquery==3.3.1	Dependency-Check	Syft

Вар.	Язык	Пакет с CVE (пример)	Инструмент SCA	Генератор SBOM
14	Python	urllib3==1.24.1	Trivy	Syft
15	Node.js	minimist==1.2.0	Trivy	Syft

Требования к базовому уровню:

- Проект с зависимостями описан в стандартном манифесте (requirements.txt / package.json).
- Настроен CI/CD (GitLab CI или GitHub Actions), выполняющий SCA-шаг с падением при HIGH/CRITICAL.
- Артефактом сохранён SBOM в формате CycloneDX (JSON).
- В отчёте представлен перечень найденных CVE (минимум 1) и их CVSS-оценка.

4.2. Средний уровень (15 вариантов)

Дополнить базовое задание кастомизацией политик или сравнительным анализом.

Вар.	Задача
1	Настроить фильтрацию по CVSS с точным порогом (например, только CRITICAL) через параметры Trivy
2	Написать .trivyignore для подавления конкретного CVE и проверить, что пайплайн восстанавливается
3	Сравнение результатов Trivy и OWASP Dependency-Check на одном проекте (таблица CVE, их совпадение)
4	Использовать Trivy в режиме --format sarif и загрузить отчёт в GitLab как SAST (настроить код сканирования)
5	Генерация SBOM в формате SPDX (Syft) и сравнение содержимого с CycloneDX
6	Написание кастомного правила Rego для Trivy (политика блокировки по CVE ID из конкретного бюллетеня)
7	Интеграция SCA в GitHub Actions с использованием github/codeql-action export SBOM и Trivy
8	Настройка Trivy для сканирования не только манифеста, но и транзитивных

Вар.	Задача
	зависимостей (сборка в Docker)
9	Реализовать скрипт, который автоматически обновляет найденные уязвимые пакеты до безопасной версии (если возможно) и повторяет сканирование
10	Создать SBOM и проверить его целостность с помощью cosign sign (подпись SBOM)
11	Использовать Syft для сканирования Docker-образа с уязвимым приложением, сравнить SBOM с хостовой сборкой
12	Настроить CI, который при провале SCA создаёт GitLab issue с описанием уязвимости
13	Интегрировать Trivy в pre-commit hook (локально) и сравнить сработку с CI
14	Настроить пайплайн, который выводит сводку по Severity (количество HIGH, CRITICAL) в удобочитаемом виде
15	Провести анализ проекта, где зависимость указана в Pipfile/Poetry, а не requirements.txt; настроить Trivy

Требования: все условия базового уровня + выполненное расширенное задание с демонстрацией работоспособности.

4.3. Повышенный уровень (15 вариантов)

Автоматизация, оркестрация и расширенная безопасность цепочки поставок.

Вар.	Задача
1	Интеграция с Dependency-Track: настроить CI для автоматической загрузки SBOM (CycloneDX) в Dependency-Track и получения алертов
2	Реализовать пайплайн, который подписывает SBOM с помощью Cosign и сохраняет сигнатуру в реестре OCI
3	Построение комплексного пайплайна: SCA + SAST (Semgrep) + SBOM, агрегирование результата в единый отчёт
4	Настройка политики: блокировка merge request только если найдена CVE с известным

Вар.	Задача
	публичным эксплойтом (с использованием Trivy --exploitability)
5	Создание кастомного Docker-образа для SCA, включающего Trivy, Syft и jq для постобработки результатов
6	Разработка GitHub Action composite, который выполняет SCA и комментирует PR списком найденных уязвимостей
7	Настройка непрерывного мониторинга SBOM: периодический запуск Trivy и сравнение с предыдущим SBOM, уведомление об изменениях
8	Использование Rekor (Sigstore) для прозрачного журнала подписей SBOM (cosign sign + rekor upload)
9	Анализ рисков: добавление шага, который вычисляет суммарный критичный риск и формирует заключение (например, red/yellow/green)
10	Интеграция с VEX (Vulnerability Exploitability eXchange): генерация VEX-отчёта для известных CVE
11	Автоматическая проверка наличия обновлений для уязвимых пакетов и создание merge request с обновлением (Renovate/Dependabot)
12	Сканирование SBOM на соответствие лицензионным политикам (например, запрет GPLv3) с помощью tools (fossa-cli)
13	Развёртывание self-hosted Dependency-Track и полного цикла: загрузка SBOM → алерт → тикет в Jira
14	Интеграция с OpenVEX: сгенерировать VEX-документ для CI, показывающий статус эксплуатации уязвимостей
15	Реализация пайплайна, который в случае обнаружения CRITICAL блокирует не только текущий пайплайн, но и отклоняет связанный merge request через API

Требования: полная автоматизация, демонстрация нестандартных сценариев, связка с другими системами безопасности.

5. Контрольные вопросы

1. Что такое Software Composition Analysis (SCA) и чем он отличается от SAST?
2. Перечислите форматы SBOM. В чём ключевые различия CycloneDX и SPDX?
3. Какие источники данных об уязвимостях использует Trivy?
4. Объясните шкалу CVSS. Какие диапазоны соответствуют уровням High и Critical?
5. Какими способами можно заблокировать конвейер CI/CD при обнаружении опасных зависимостей?
6. Для чего нужен файл .trivyignore? Приведите пример синтаксиса.
7. Какая информация содержится в SBOM и какую пользу она приносит при реагировании на инциденты (например, Log4Shell)?
8. Чем инструмент Syft отличается от Trivy в контексте генерации SBOM?
9. Что такое транзитивные зависимости и почему их необходимо сканировать?
10. Как обеспечить целостность SBOM в процессе доставки?

6. Требования к отчёту

Отчёт оформляется в электронном виде (PDF) и включает:

- Титульный лист (название ЛР, вариант, ФИО, группа).
- Цель и задачи работы.
- Описание тестового проекта: листинг манифеста зависимостей, указание уязвимых версий.
- Конфигурация CI/CD с пояснениями.
- Скриншоты запуска пайплайна (успешный до исправления и падающий из-за Quality Gate).
- Фрагмент отчёта Trivy / Dependency-Check (JSON/таблица с CVE и CVSS).
- Фрагмент SBOM (JSON) с аннотацией полей.
- Описание индивидуальной части (правила, сравнения, интеграции) с подтверждающими скриншотами.
- Ответы на контрольные вопросы.
- Выводы по итогам работы.

7. Критерии оценивания

Оценка выставляется на основе уровня выполнения и качества отчёта.

7.1. «Удовлетворительно» (базовый уровень):

- Проект содержит хотя бы одну уязвимую зависимость с CVSS ≥ 7.0 .
- CI/CD-пайплайн включает SCA-шаг (Trivy/Dependency-Check) и завершается ошибкой при обнаружении HIGH/CRITICAL.
- SBOM сгенерирован в формате CycloneDX и приложен к отчёту.
- Отчёт содержит все обязательные разделы.
- Даны верные ответы минимум на 60% контрольных вопросов.

7.2. «Хорошо» (средний уровень):

- Выполнены все требования базового уровня.
- Реализовано задание среднего уровня (кастомная политика, сравнение инструментов, подпись SBOM и т.п.) с демонстрацией результата.
- Проведён краткий анализ полученных данных (сравнение CVE, объяснение игнорирования).

7.3. «Отлично» (повышенный уровень):

- Выполнены все требования среднего уровня.
- Реализовано одно из заданий повышенного уровня с полной демонстрацией автоматизированного процесса.
- Продемонстрирована непрерывная цепочка безопасности поставок (например, загрузка в Dependency-Track, подпись, уведомления).
- Ответы на контрольные вопросы полны и аргументированы, студент демонстрирует понимание роли SBOM в безопасности цепочки поставок.

7.4. Понижение оценки:

- Отсутствие одного из ключевых артефактов (SCA-отчёт, SBOM, скриншот) — работа не принимается.
- Quality Gate не срабатывает на заданном уровне — несоответствие базовому уровню.
- Ответы на контрольные вопросы неполны (менее 60% базовый, менее 80% повышенный) — отправляется на доработку.

Освоение данной лабораторной работы формирует компетенции ПК-8 в части управления цепочкой поставок и непрерывного соответствия требованиям безопасности, а также закладывает основу для практик защиты автономных систем, где прозрачность состава ПО критически важна.

Лабораторная работа №3

Динамический анализ безопасности API (DAST)

1. Цель

Сформировать у обучающихся практические навыки автоматического динамического анализа безопасности веб-приложений (DAST) путём встраивания OWASP ZAP в CI/CD-конвейер после развёртывания тестируемого API, настройки полноценного сканирования и создания отчётности.

2. Задачи

- Разработать простое веб-приложение (REST API) на Python (FastAPI/Flask) или Node.js (Express), содержащее намеренно внедрённые уязвимости (XSS, SQL-инъекция, Command Injection и др.).
- Развернуть приложение в контейнере Docker для обеспечения изолированной тестовой среды.
- Интегрировать в GitLab CI этап динамического сканирования с использованием образа `owasp/zap2docker-stable`.
- Настроить автоматический запуск OWASP ZAP (например, `zap-baseline.py` или `zap-api-scan.py`) для целевого API.
- Получить и проанализировать HTML/JSON-отчёт ZAP с обнаруженными алертами.
- Реализовать примитивный Quality Gate: завершить пайплайн с ошибкой при наличии алертов уровня High и выше.

3. Теоретическое обоснование

DAST (Dynamic Application Security Testing) — метод анализа безопасности приложения во время его выполнения, не имея доступа к исходному коду. Инструмент отправляет вредоносные запросы и анализирует ответы, выявляя уязвимости типа XSS, инъекций, неправильной конфигурации безопасности.

OWASP ZAP — один из самых распространённых инструментов DAST с открытым исходным кодом. Поддерживает несколько режимов сканирования:

- Baseline Scan — пассивное сканирование (без активных атак), быстрое, безопасное для продуктивных сред, ищет низко висящие фрукты (заголовки безопасности, отпечатки технологий, утечки информации).
- Full Scan — активное сканирование с применением атакующих полезных нагрузок, включая XSS, SQLi, Path Traversal. Требуется настройки и осторожности.
- API Scan — оптимизирован для REST API (OpenAPI/Swagger, GraphQL), выполняет активное сканирование на основе спецификации.

Интеграция в CI/CD достигается запуском ZAP в контейнере после шага деплоя приложения в динамическое окружение (Docker Compose или Kubernetes). Команда `zap-baseline.py` или `zap-api-scan.py` сканирует целевой URL и возвращает ненулевой код при наличии алертов заданного порога, что позволяет заблокировать пайплайн.

Качество шлюза (Quality Gate) для DAST может включать: отсутствие алертов с уровнем риска High/Critical, ограничение на количество алертов определённого типа, соответствие политике, определённой через ZAP Context.

4. Задания для индивидуального выполнения

Номер варианта выбирается по последней цифре студенческого билета или списка группы.

4.1. Базовый уровень (15 вариантов)

Выполнить интеграцию ZAP Baseline Scan в CI/CD и получить отчёт для API с заданной уязвимостью.

Вар.	Фреймворк API	Уязвимость в приложении
1	FastAPI	SQL Injection (в параметре запроса)
2	FastAPI	Command Injection (ping)
3	FastAPI	Reflected XSS (HTML-ответ)
4	Flask	SQL Injection
5	Flask	Path Traversal (отдача файлов)
6	Flask	Sensitive Data Exposure (пароль в ответе)
7	Express (Node)	SQL Injection (SQLite)
8	Express	NoSQL Injection (MongoDB)
9	Express	XSS via query parameter
10	FastAPI	Open Redirect
11	Flask	LDAP Injection (имитация)
12	Express	External XML Entity (XXE) через загрузку XML
13	Flask	Server-Side Template Injection (SSTI)

Вар.	Фреймворк API	Уязвимость в приложении
14	FastAPI	Insecure Deserialization (pickle)
15	Express	Arbitrary Code Execution (eval)

Требования к базовому уровню:

- Работающий CI/CD-пайплайн, в котором после деплоя приложения запускается OWASP ZAP.
- Сформирован HTML-отчёт ZAP.
- Пайплайн завершается ошибкой при обнаружении алертов High.
- В отчёте студент должен выделить как минимум одну подтверждённую уязвимость своего варианта.

4.2. Средний уровень (15 вариантов)

Расширить функциональность: активное сканирование, работа с ZAP API, спецификации OpenAPI.

Вар.	Задача
1	Настроить ZAP Full Scan с активным сканированием и сравнить результаты с Baseline
2	Сгенерировать OpenAPI-спецификацию для своего API и выполнить zap-api-scan.py
3	Интегрировать ZAP Context (файл .context) с исключением ложных срабатываний
4	Настроить аутентификацию в ZAP (например, токен в заголовке) через конфигурационный файл
5	Использовать ZAP API (через curl или zap-cli) для запуска сканирования и получения результатов в JSON
6	Разработать скрипт на Python, который анализирует JSON-отчёт ZAP и принимает решение о блокировке
7	Добавить шаг для проверки CSP-заголовков с помощью ZAP (политики Content-Security-Policy)
8	Настроить пассивное сканирование с последующим активным только для алертов уровня Medium+

Вар.	Задача
9	Интегрировать ZAP с GitHub Actions (как альтернатива GitLab CI)
10	Создать кастомный Docker-образ, включающий ZAP и предварительно загруженные скрипты для автоматизации
11	Сравнить результаты сканирования ZAP с результатами инструмента nikto (в Docker)
12	Автоматически загружать ZAP-отчёт как артефакт в GitLab Pages для удобного просмотра
13	Использовать ZAP с включённым AJAX Spider для одностраничных приложений (SPA)
14	Написать кастомное правило пассивного сканирования ZAP для обнаружения специфического заголовка
15	Произвести DAST приложения, работающего по HTTPS, с подгрузкой самоподписанного сертификата

Требования: выполнение условий базового уровня + реализация выбранного расширенного сценария с документальными подтверждениями (скриншоты, листинги).

4.3. Повышенный уровень (15 вариантов)

Оркестрация, визуализация, интеграция с системами управления уязвимостями и умная блокировка.

Вар.	Задача
1	Интегрировать ZAP с DefectDojo (загрузка результатов сканирования через API, создание engagement)
2	Реализовать "канареечное" тестирование: сравнение результатов сканирования текущего коммита с предыдущим и блокировка при новых алертах High
3	Настроить автоматическое создание Issue в GitLab с полным описанием алерта и рекомендациями при обнаружении CRITICAL
4	Создать рекуррентный пайплайн (ночной запуск) для DAST на staging-окружении и отправку уведомлений в Slack/Telegram

Вар.	Задача
5	Разработать интеграцию с OWASP DefectDojo и custom Quality Gate на основе общего количества уязвимостей Critical = 0 и High ≤ 3
6	Построить конвейер, который после сканирования автоматически создаёт исправленную версию API и повторно прогоняет тест (авто-верификация)
7	Использовать ZAP в режиме демона (daemon) и взаимодействовать с ним через REST API из разных шагов пайплайна
8	Внедрить ZAP как часть GitLab Ultimate DAST (использовать DAST.gitlab-ci.yml с кастомизацией)
9	Провести сканирование GraphQL API (если endpoint поддерживает) с помощью ZAP GraphQL add-on
10	Разработать систему, которая на основе SARIF-отчёта ZAP (конвертация), добавляет аннотации в Merge Request (Code Quality widget)
11	Настроить автоматический сбор метрик (количество алертов, тренд) и визуализацию в Grafana через InfluxDB
12	Интеграция с Kafka: ZAP-алерты направляются в очередь для дальнейшей обработки SIEM системой
13	Реализация политики: если найдена XSS Critical, автоматически откатывать деплой этого коммита
14	Создание собственного плагина/скрипта для ZAP, который проверяет специфическую бизнес-логику (например, IDOR)
15	Полная цепочка: SAST (Semgrep) → SCA (Trivy) → DAST (ZAP) с единым дашбордом и сводным отчётом

Требования: полная автоматизация процесса, интеграция с внешними системами, демонстрация глубокого понимания API ZAP и CI/CD.

5. Контрольные вопросы

1. В чём отличие динамического анализа (DAST) от статического (SAST)? Приведите примеры.

2. Какие режимы сканирования поддерживает OWASP ZAP, и в каких случаях какой следует применять?
3. Объясните, что такое пассивное и активное сканирование. Какие риски несёт активное сканирование?
4. Как работает zap-baseline.py? Каковы его ограничения?
5. Каким образом можно настроить Quality Gate при использовании OWASP ZAP в CI/CD?
6. Что такое ZAP Context и для чего он используется при интеграции в автоматизированные пайплайны?
7. Какими способами можно аутентифицироваться в ZAP для сканирования защищённых API?
8. Как ZAP обнаруживает SQL-инъекции и XSS? Каковы механизмы проверки?
9. Назовите основные артефакты (отчёты), которые генерирует ZAP, и их форматы.
10. В чём преимущество сканирования на основе OpenAPI-спецификации (zap-api-scan.py)?

6. Требования к отчёту

Структура отчёта (PDF):

- Титульный лист.
- Цель и задачи.
- Описание тестового приложения с указанием внедрённых уязвимостей (листинги ключевых фрагментов).
- Dockerfile и конфигурация CI/CD (YAML) с комментариями.
- Скриншоты успешного деплоя и этапа DAST (консоль, GitLab UI).
- Полученный отчёт ZAP (скриншот ключевой части или вложение HTML/JSON).
- Детальное описание индивидуальной части (конфигурации, скрипты, сравнения).
- Ответы на контрольные вопросы.
- Выводы (оценка эффективности DAST, анализ найденных уязвимостей, рекомендации по исправлению).

7. Критерии оценивания

7.1. «Удовлетворительно» (базовый уровень):

- Разработано API с целевой уязвимостью.
- Настроен GitLab CI с шагом OWASP ZAP (baseline), пайплайн падает при алертах High.
- Сгенерирован HTML-отчёт.
- Отчёт содержит обязательные разделы.
- Правильные ответы на 60% контрольных вопросов.

7.2. «Хорошо» (средний уровень):

- Все условия базового уровня.
- Выполнено задание среднего уровня (активное сканирование, API scan, Context, сравнение и т.д.).
- Качество сканирования улучшено: исключены ложные срабатывания, добавлена аутентификация (по смыслу варианта).
- Показан сравнительный анализ результатов (если применимо).

7.3. «Отлично» (повышенный уровень):

- Все условия среднего уровня.
- Реализована задача повышенного уровня с полной автоматизацией и интеграцией с внешними сервисами.
- Продемонстрировано глубокое понимание процесса: объяснены причины блокировок, предложены меры по снижению false positives.
- Ответы на контрольные вопросы полные и точные, студент способен обосновать выбор инструментов.

7.4. Понижение оценки:

- Отсутствие артефактов (отчёт, скриншоты) — возврат на доработку.
- Неправильная работа Quality Gate (пайплайн не падает при наличии алертов High) — не может быть оценен выше базового уровня.
- Менее 60% правильных ответов на вопросы — обязательная доработка.

Выполнение данной работы закрепляет практики DAST в составе DevSecOps, формируя компетенцию ПК-8 в части динамического тестирования и подготавливая студентов к более сложным сценариям защиты автономных систем, где динамический анализ API играет ключевую роль в проверке безопасности каналов связи.

1. Цель

Сформировать практические навыки выявления уязвимостей в Docker-образах с помощью Trivy и внедрения политик безопасности времени выполнения (Pod Security Admission) в Kubernetes для предотвращения запуска привилегированных и небезопасных подов.

2. Задачи

- Создать Docker-образ приложения, содержащий уязвимые пакеты ОС или языка.
- Интегрировать сканирование образа Trivy в CI/CD-пайплайн с блокировкой при наличии критических уязвимостей.
- Развернуть локальный кластер Kubernetes (minikube/kind) и применить к нему Pod Security Admission на уровне enforce: restricted.
- Продемонстрировать успешную блокировку подов, запрашивающих привилегированный режим (privileged: true), и разрешить запуск подов, удовлетворяющих ограничительной политике.
- Получить отчёт Trivy и задокументировать поведение системы безопасности кластера.

3. Теоретическое обоснование

Сканирование образов на уязвимости — обязательный этап DevSecOps для обеспечения безопасности контейнеризованных приложений. Инструмент Trivy (Aqua Security) проводит анализ слоёв образа, выявляя известные уязвимости в системных пакетах (Alpine, Debian, etc.) и зависимостях приложений (Python, Node.js). Trivy поддерживает форматы вывода HTML, JSON, SARIF и может возвращать ненулевой код при наличии уязвимостей заданного уровня.

Pod Security Admission (PSA) — встроенный в Kubernetes (с версии 1.25) механизм, контролирующий безопасность подов на основе стандартов Pod Security Standards (baseline, restricted). PSA пришёл на смену устаревшему PodSecurityPolicy (PSP) и работает путём назначения метки pod-security.kubernetes.io/<mode> на уровне неймспейса.

Pod Security Standards:

- Privileged — политика без ограничений, разрешён любые атрибуты безопасности (привилегированный режим, доступ к хостовым ресурсам и т.д.).
- Baseline — минимально рекомендуемые ограничения, предотвращающие известные эскалации привилегий. Запрещает hostNetwork, hostPID, добавление дополнительных capabilities без явного списка и т.д.
- Restricted — строгая политика, максимально жёстко ограничивающая под (обязан использовать непривилегированные контейнеры, read-only root filesystem, ограничения по привилегиям, seccomp и т.д.). Это современная лучшая практика для большинства рабочих нагрузок.

Режимы (Modes):

- enforce — нарушение политики приводит к запрету создания или обновления пода.

- audit — нарушения логируются, но не блокируются.
- warn — предупреждения пользователям при создании нарушающего пода.

Комбинация сканирования образов (сдвиг влево) и runtime-политик (PSA) образует многоуровневую защиту контейнерной среды, реализуя компетенцию ПК-8 в части безопасности облачной инфраструктуры и контейнеров.

4. Задания для индивидуального выполнения

Номер варианта соответствует последней цифре номера зачётки или списка. При необходимости студент согласует замену инструментов с преподавателем.

4.1. Базовый уровень (15 вариантов)

Каждый вариант включает базовый образ с уязвимостями и уровень enforce PSA для блокировки.

Вар.	Базовый образ Docker	Уязвимые пакеты (пример)	PSA уровень	Проверяемая характеристика пода
1	alpine:3.15.0	openssl, curl	restricted	privileged: true
2	ubuntu:18.04	libssl1.1, wget	restricted	hostPID: true
3	debian:9-slim	libc-bin, perl-base	restricted	hostNetwork: true
4	python:3.6-slim	libexpat, libssl	restricted	runAsUser: 0 (без runAsNonRoot)
5	node:12-alpine	libuv, openssl	restricted	capabilities: add: ["SYS_ADMIN"]
6	openjdk:8-jre-slim	glibc, zlib	restricted	allowPrivilegeEscalation: true (без запрета разрешать)
7	golang:1.12-alpine	musl, openssl	restricted	privileged: true + hostNetwork
8	python:3.7-alpine	expat, libbz2	baseline	hostPath volume (запрещено baseline)
9	ruby:2.5-slim	openssl, readline	restricted	readOnlyRootFilesystem: false (без капабилити)
10	nginx:1.17-alpine	alpinelinux:3.11	restricted	privileged: true

Вар.	Базовый образ Docker	Уязвимые пакеты (пример)	PSA уровень	Проверяемая характеристика пода
11	httpd:2.4.39-alpine	apk-tools, curl	restricted	capabilities: add: ["NET_ADMIN"]
12	node:10-jessie	openssl, libssl	restricted	privileged: true + hostPID
13	centos:7	kernel-headers, glibc	restricted	runAsUser: 0 + allowPrivilegeEscalation
14	alpine:3.14.2	zlib, busybox	restricted	hostNetwork: true + no seccomp profile
15	ubuntu:16.04	perl, libc6	restricted	privileged: true

Требования к базовому уровню:

- Собрать Docker-образ согласно варианту, запустить или сохранить в архив.
- Настроить GitLab CI шаг Trivy с блокировкой по HIGH.
- Получить HTML/JSON отчёт Trivy.
- Развернуть Kubernetes и применить заданный PSA enforce к неймспейсу.
- Продемонстрировать успешную блокировку пода с нарушающим параметром и успешный запуск безопасного пода.
- Отчёт должен содержать все обязательные разделы.

4.2. Средний уровень (15 вариантов)

Расширение базового сценария: модификация политик, использование дополнительных инструментов управления политиками, сканирование IaC.

Вар.	Задача
1	Добавить к PSA метку warn: baseline и продемонстрировать получение предупреждения при применении пода, не соответствующего baseline
2	Использовать OPA Gatekeeper (ConstraintTemplate) для реализации политики, запрещающей запуск образов с тегом latest
3	Настроить Kyverno политику, которая автоматически добавляет securityContext.runAsNonRoot: true, если он не задан

Вар.	Задача
4	Сканировать не только образ, но и Kubernetes манифесты (YAML) с помощью Checkov, добиться обнаружения некорректных securityContext
5	Реализовать CI-пайплайн, который после сканирования Trivy при обнаружении CRITICAL уязвимостей автоматически создаёт issue в GitLab с рекомендацией по обновлению
6	Настроить PSA в режиме audit для мониторинга с последующим логированием нарушений и разбором аудит-логов с помощью простого скрипта
7	Создать кастомный образ Trivy с заранее загруженной базой уязвимостей для офлайн-сканирования и использовать его в пайплайне
8	Настроить подпись образа cosign и политику проверки подписи на этапе деплоя в Kubernetes (Connaisseur / Kyverno)
9	Сравнить детекты уязвимостей Trivy и Snyk (или Docker Scout) на одном образе, представить сводный отчёт
10	Добавить в пайплайн генерацию SBOM для образа и его проверку на лицензионные ограничения (например, с помощью fossa-cli)
11	Разработать Admission Webhook (простой HTTP сервер), который отклоняет поды, не имеющие метки security: strict
12	Интегрировать Trivy в виде GitHub Actions и настроить аннотирование PR с найденными уязвимостями
13	Настроить PSA с параметром exemptions (если используется контроллер исключений) для определённых service accounts
14	Создать Helm-чарт для тестового приложения и добавить в него валидацию безопасности с помощью helm unittest и kubeval
15	Реализовать политику, которая блокирует создание подов, если в образе обнаружены уязвимости CRITICAL (использовать admission controller, интегрированный с Trivy через webhook)

Требования: все пункты базового уровня, плюс выполнение расширенного задания с доказательством работоспособности. Для вариантов, требующих интеграции с внешними сервисами, допускается использование открытых API или локальных инсталляций.

4.3. Повышенный уровень (15 вариантов)

Комплексная автоматизация, глубокая интеграция с Kubernetes и экосистемой безопасности.

Вар.	Задача
1	Построить полный конвейер: SAST → SCA → сборка образа → сканирование Trivy → подпись Cosign → деплой с проверкой подписи и PSA Restricted
2	Интегрировать Trivy в режиме server-client для централизованного сканирования в кластере Kubernetes
3	Настроить систему непрерывного мониторинга безопасности кластера: PSA (audit) + Falco для обнаружения нарушений времени выполнения, отправка инцидентов в Elasticsearch
4	Реализовать автоматическое исправление уязвимых образов: при обнаружении уязвимости HIGH создаётся PR с обновлённым Dockerfile и доказательством исправления
5	Использовать OPA Gatekeeper для реализации правила, запрещающего загрузку образов из непроверенных реестров (кроме заданного списка)
6	Создать кастомный CRD (Custom Resource Definition) SecurityPolicy и контроллер, который проверяет поды на соответствие этой политике, реализуя функционал, аналогичный PSA
7	Интегрировать Kubernetes Event-driven Autoscaling (KEDA) для динамической проверки безопасности: при неудачной политике под автоматически масштабируется для расследования
8	Создать визуализацию уязвимостей образа с помощью Grafana Loki/Promtail и скрипта, собирающего метрики из Trivy JSON
9	Разработать GitLab CI шаблон (component) для повторного использования пайплайна безопасности контейнеров в нескольких проектах
10	Настроить политику PSA, при которой для продакшн неймспейсов enforced restricted, а для staging – baseline, с автоматической валидацией на основе имени неймспейса

Вар.	Задача
11	Внедрить Kyverno с политикой, обеспечивающей автоматическое добавление securityContext и валидацию на основе меток образа
12	Разработать систему «канареечного» обновления, при которой новый образ сначала деплоится в песочницу, сканируется, и только после успешной проверки раскатывается в прод
13	Использовать Trivy Operator в Kubernetes для автоматического сканирования всех образов в кластере и создания отчётов VulnerabilityReports
14	Построить полный цикл управления уязвимостями контейнеров: от сканирования до автоматического закрытия тикетов после подтверждения исправления
15	Интеграция с Cosign и Rekor: подпись образа, запись в лог Rekor, проверка подписи в Admission Controller, все логируются в SIEM

Требования: автономная автоматизированная система, демонстрирующая зрелость безопасности контейнеров; детальное описание архитектуры и конфигурации в отчёте.

5. Контрольные вопросы

1. Какие компоненты контейнерного образа анализирует Trivy при сканировании?
2. В чём разница между Pod Security Admission (PSA) и PodSecurityPolicy (PSP)? Почему PSA заменил PSP?
3. Опишите уровни Pod Security Standards: Privileged, Baseline, Restricted.
4. Какие режимы работы PSA вы знаете? Чем enforce отличается от audit?
5. Какие атрибуты пода запрещены на уровне Restricted (приведите минимум 3 примера)?
6. Как с помощью меток неймспейса настроить PSA на режим enforce restricted?
7. Почему недостаточно только сканирования образов? Зачем нужны runtime-политики?
8. Каким образом можно автоматически блокировать CI/CD-пайплайн при обнаружении критической уязвимости в образе?
9. Что такое admission controller? Как встроенный PodSecurity admission интегрируется в API Kubernetes?
10. Какие ещё инструменты, кроме Trivy, можно использовать для сканирования образов? (Назовите 2-3)

6. Требования к отчёту

Отчёт в формате PDF должен содержать:

- Титульный лист.

- Цель и задачи.
- Описание используемого образа (Dockerfile, список уязвимых компонентов).
- Конфигурацию CI/CD (gitlab-ci.yml или аналогичный с комментариями).
- Скриншоты пайплайна (успешное сканирование и падение при уязвимостях), отчёт Trivy (фрагмент).
- Описание настройки Kubernetes (создание кластера, неймспейса, метки PSA).
- Демонстрация блокировки небезопасного пода (скриншот ошибки) и успешного запуска безопасного пода.
- Индивидуальная часть: описание реализованного расширения (скрипты, политики, скриншоты).
- Ответы на контрольные вопросы.
- Выводы и рекомендации.

7. Критерии оценивания

7.1. «Удовлетворительно» (базовый уровень):

- Собран Docker-образ согласно варианту.
- Пайплайн Trivy в CI: при наличии HIGH уязвимостей пайплайн аварийно останавливается.
- Кластер Kubernetes настроен с PSA (enforce restricted или baseline согласно варианту).
- Продемонстрирована блокировка пода с нарушениями и успешный запуск безопасного.
- Отчёт оформлен правильно, ответы на контрольные вопросы $\geq 60\%$.

7.2. «Хорошо» (средний уровень):

- Выполнены все требования базового уровня.
- Реализовано задание среднего уровня с демонстрацией работоспособности.
- Показаны дополнительные проверки (расширенные политики, инструменты).
- Ответы на контрольные вопросы $\geq 80\%$ корректных.

7.3. «Отлично» (повышенный уровень):

- Выполнены все требования среднего уровня.
- Полностью реализована задача повышенного уровня, демонстрирующая глубокую автоматизацию.
- Приведены аргументированные выводы и сравнительный анализ инструментов.
- Ответы на контрольные вопросы полные и точные.

7.4. Понижение оценки:

- Отсутствие подтверждающих скриншотов или артефактов (Trivy-отчёт, логи PSA) — доработка.

- Неправильная конфигурация Quality Gate (пайплайн не падает при наличии HIGH) — не выше базового.
- Недостаточное знание теории — отправляется на защиту повторно.

Данная лабораторная работа закладывает важные навыки защиты контейнерной инфраструктуры, напрямую связанные с компетенцией ПК-8 и готовящие студентов к безопасному развёртыванию инженерных систем, включая компоненты автономных устройств, где контейнеризация находит всё большее применение.

Лабораторная работа №5
Compliance as Code с Open Policy Agent (OPA/Rego)

1. Цель

Сформировать навыки реализации Compliance as Code путём написания политик на языке Rego, их тестирования в OPA Playground и проверки Kubernetes-манифестов с помощью утилиты Conftest, интегрируя автоматическую валидацию в CI/CD.

2. Задачи

- Изучить основы языка Rego и принципы работы Open Policy Agent.
- Написать хотя бы одну политику безопасности (запрет тегов latest, обязательные метки, запрет привилегированных подов и др.) в виде Rego-правила.
- Протестировать политику в Rego Playground и локально с помощью Conftest.
- Интегрировать проверку манифестов с Conftest в GitLab CI (или GitHub Actions).
- Настроить Quality Gate: пайплайн должен блокироваться при обнаружении нарушений политик.
- (Средний/повышенный уровень) Реализовать несколько политик, написать модульные тесты (opa test), линтинг Rego через Regal, или интегрировать политики в Kubernetes через Gatekeeper.

3. Теоретическое обоснование

Compliance as Code (CaC) — подход, при котором требования к безопасности и соответствию стандартам формализуются в виде программного кода. Это позволяет автоматизировать проверку соответствия на всех этапах жизненного цикла, в том числе в CI/CD, обеспечивая continuous compliance.

Open Policy Agent (OPA) — универсальный движок принятия решений, отделяющий логику политик от прикладного кода. Политики описываются на декларативном языке Rego, основанном на Datalog. OPA может внедряться как sidecar, в виде библиотеки или через Kubernetes admission controller (Gatekeeper).

Rego — язык запросов к данным (обычно JSON/YAML) с правилами. Типичное правило состоит из условия (body) и заключения (head). Например:

Conftest — утилита командной строки, позволяющая применять политики Rego к конфигурационным файлам (Helm-чарты, Kubernetes YAML, Dockerfile, Terraform и др.). Она оборачивает OPA и предоставляет удобный интерфейс для проверки.

Regal — линтер для политик Rego, проверяющий стилистику, идиоматичность и потенциальные ошибки.

Интеграция в CI/CD достигается запуском `conftest test <files>` на этапе верификации. При наличии нарушений (нарушены deny правила) Conftest возвращает ненулевой код, что останавливает пайплайн. Это реализует "сдвиг влево" для политик безопасности и соответствия.

4. Задания для индивидуального выполнения

Номер варианта определяется последней цифрой зачётки/списка. Допускается использование эквивалентных инструментов по согласованию.

4.1. Базовый уровень (15 вариантов)

Каждый вариант — написать указанную политику(и) на Rego, протестировать через Conftest и интегрировать в CI.

Вар.	Проверяемое требование (политика)
1	Запретить использование тега latest во всех контейнерах (любые ресурсы)
2	Требовать метку app во всех Pod/Deployment
3	Требовать метки app и version в селекторе Deployment
4	Запретить запуск привилегированных контейнеров (privileged: true)
5	Запретить использование hostNetwork: true
6	Требовать указания resources.requests и resources.limits для каждого контейнера
7	Запретить образ nginx:latest (конкретный образ, имя репозитория задаётся параметром)
8	Требовать readOnlyRootFilesystem: true для всех контейнеров
9	Требовать наличия livenessProbe и readinessProbe у каждого контейнера
10	Запретить монтирование томов типа hostPath (любой hostPath)
11	Требовать, чтобы Namespace имел метку environment (ресурс Namespace)
12	Запретить запуск контейнеров с runAsUser: 0 (root)
13	Требовать, чтобы в Deployment было настроено не менее 2 реплик
14	Запретить Ingress без секции tls
15	Требовать аннотацию owner в Deployment

Требования к базовому уровню:

- Написан файл политики Rego, реализующий указанное правило.
- Проведена локальная проверка манифестов с помощью Conftest (показан провал на "плохом" и успех на "хорошем").
- Настроен GitLab CI/GitHub Actions этап с Conftest, падающий при нарушениях.

- В отчёте приведены листинги политики, CI-конфигурации, скриншоты.

4.2. Средний уровень (15 вариантов)

Расширение базового задания: несколько политик, тестирование, линтинг, интеграция с Gatekeeper.

Вар.	Задача
1	Реализовать пакет из трёх политик (например, latest-ter, обязательные метки, запрет привилегированного режима) и написать unit-тесты Rego (opa test), обеспечив покрытие >80%.
2	Написать политику, параметризуемую через data.allowedRegistries (список допустимых реестров), и проверить с разными входными данными.
3	Использовать Regal для линтинга Rego-кода и исправить все замечания.
4	Интегрировать политики в Kubernetes через OPA Gatekeeper: создать ConstraintTemplate и Constraint (например, K8sAllowedRepos). Продемонстрировать блокировку пода.
5	Сравнить Conftest и Kyverno для одной и той же политики. Написать политику на обоих языках (Rego и YAML), провести проверку и сравнить выдачу.
6	Написать политику, которая проверяет не только структуру, но и пересечение с данными из внешнего источника (например, список запрещённых CVE в образах — заглушка).
7	Создать кастомный Docker-образ с Conftest и предварительно загруженными политиками для использования в CI.
8	Настроить проверку не только Kubernetes-манифестов, но и Dockerfile (например, запрет FROM latest, проверка наличия USER).
9	Реализовать политику, требующую, чтобы в метках присутствовал cost-center из заданного набора (список разрешённых значений).
10	Использовать conftest verify для тестирования политики напрямую без opa test.
11	Написать политику, проверяющую, что все Service имеют тип не NodePort (запрет NodePort для production namespace).
12	Создать пайплайн, который при провале проверки автоматически создаёт GitLab issue с перечнем нарушений.

Вар.	Задача
13	Разработать политику, которая проверяет наличие NetworkPolicy для каждого Namespace с меткой policy: required.
14	Написать политику, требующую аннотацию last-reviewed с датой не старше 90 дней.
15	Интегрировать проверки Conftest в pre-commit хук (локально) и в CI, сравнить поведение.

Требования: выполнение базового уровня + реализация расширенного задания с демонстрацией.

4.3. Повышенный уровень (15 вариантов)

Комплексная автоматизация, оркестрация, глубокая интеграция в процессы.

Вар.	Задача
1	Построить централизованную систему политик: Git-репозиторий с Rego-политиками, CI/CD для их тестирования и публикации в OCI-артефакт, используемый Conftest в проектах.
2	Разработать Kubernetes-контроллер на базе OPA, который динамически валидирует ConfigMap'ы, содержащие конфигурации приложений.
3	Интегрировать политики с системой сбора метрик (Prometheus): количество нарушений по типам, тренд. Визуализировать в Grafana.
4	Реализовать GitLab CI шаблон (CI component) для повторного использования Conftest проверок с возможностью включения/выключения отдельных политик.
5	Настроить Admission Control с Gatekeeper и реализовать политику, которая проверяет наличие подписи образа (Cosign) и соответствие реестру.
6	Создать кастомную политику проверки соответствия требованиям ФСТЭК для государственных информационных систем (модельный набор правил).
7	Разработать автоматическую генерацию политик Rego на основе высокоуровневого описания (DSL) с помощью шаблонизатора, интегрированного в CI.
8	Настроить систему непрерывного аудита: периодический прогон Conftest на всех активных манифестах кластера (получаемых через kubectl), уведомление о новых нарушениях.

Вар.	Задача
9	Реализовать политику, которая учитывает контекст окружения (имя namespace) и применяет разные правила для prod и dev (например, в prod запрещён latest, в dev — warning).
10	Интегрировать OPA с Kafka: результаты проверок манифестов отправлять в топик для дальнейшей обработки SIEM.
11	Создать веб-интерфейс (простой дашборд) для отображения результатов проверок Conftest в реальном времени (через API).
12	Настроить политику, которая использует внешние данные из Trivy (JSON-отчёт) для запрета развёртывания образов с HIGH уязвимостями.
13	Разработать библиотеку Rego функций для проверки формата меток (RFC 1123) и использовать её в разных проектах.
14	Построить мультисистемный пайплайн: SAST → SCA → контейнеры → Compliance, где последний этап — Conftest проверка всех сгенерированных манифестов.
15	Создать оператор Kubernetes, который при обнаружении нарушения политики автоматически создаёт исправленный манифест и предлагает Pull Request.

Требования: полная автоматизация, демонстрация работающей системы, подробное описание архитектуры.

5. Контрольные вопросы

1. Что такое Compliance as Code и какие преимущества даёт этот подход?
2. Опишите архитектуру Open Policy Agent (OPA). Как он отделяет логику политик от приложений?
3. Каковы основные компоненты правила Rego? Приведите пример простого deny-правила.
4. Чем Conftest отличается от прямого использования OPA?
5. Как можно параметризовать политики Rego, чтобы они были переиспользуемыми?
6. Что такое input в Rego, и как Conftest предоставляет данные из YAML-файлов?
7. Каким образом Conftest возвращает ненулевой код при нарушениях, и как это использовать для Quality Gate?
8. Для чего нужны unit-тесты для политик Rego? Как их запускать?
9. Что такое OPA Gatekeeper и в чём его отличие от Conftest?

10. Как можно организовать централизованное управление политиками для нескольких проектов?

6. Требования к отчёту

Отчёт (PDF) содержит:

- Титульный лист (номер ЛР, вариант, ФИО, группа).
- Цель и задачи.
- Листинг написанной Rego-политики с пояснениями.
- Описание тестовых манифестов (хороший и плохой примеры).
- Результаты локального запуска Conftest (скриншоты консоли).
- Фрагмент CI/CD-конфигурации (gitlab-ci.yml / github actions) с комментариями.
- Для среднего/повышенного уровня — описание дополнительного задания, листинги, скриншоты.
- Ответы на контрольные вопросы.
- Выводы.

7. Критерии оценивания

7.1. «Удовлетворительно» (базовый уровень):

- Написана и протестирована одна политика Rego, выполняющая требование варианта.
- Настроен CI с Conftest, который падает при нарушениях.
- Представлены скриншоты успешных и неуспешных проверок.
- Отчёт оформлен, ответы на контрольные вопросы верны не менее чем на 60%.

7.2. «Хорошо» (средний уровень):

- Выполнены все требования базового уровня.
- Реализовано задание среднего уровня (несколько политик, тесты, линтинг, сравнение и т.п.) с демонстрацией.
- Показано использование дополнительных инструментов (Regal, Gatekeeper, unit-тесты).
- Ответы на контрольные вопросы $\geq 80\%$.

7.3. «Отлично» (повышенный уровень):

- Выполнены все требования среднего уровня.
- Внедрена комплексная система политик с автоматизацией.
- Продемонстрировано глубокое понимание Rego и экосистемы OPA.
- Ответы на контрольные вопросы полны, аргументированы.

7.4. Понижение оценки:

- Отсутствие подтверждений работоспособности (скриншотов, логов) — возврат на доработку.

- Неправильная работа Quality Gate (CI не падает) — не выше базового уровня.
- Неполные ответы на контрольные вопросы — необходимо доработать.

Освоение данной лабораторной работы формирует компетенцию ПК-8 в части обеспечения continuous compliance и служит фундаментом для применения политик безопасности в инженерных и автономных системах, где строгое соответствие регламентам обязательно.

Лабораторная работа №6
Подпись артефактов с помощью Cosign

1. Цель

Сформировать практические навыки обеспечения целостности артефактов в цепочке поставок путём их криптографической подписи с помощью Cosign, включая keyless-подход на основе OIDC в CI/CD, а также настройку автоматической верификации перед развёртыванием.

2. Задачи

- Сгенерировать ключевую пару Cosign (или настроить keyless-подпись через OIDC-токен CI/CD).
- Подписать собранный Docker-образ после успешной сборки в пайплайне.
- Настроить шаг верификации подписи перед деплоем (блокировка при несовпадении).
- Опубликовать подпись в прозрачном журнале Rekor (опционально, в зависимости от уровня).
- Реализовать keyless-подпись с использованием эфемерного ключа, полученного через OIDC-токен в CI.

3. Теоретическое обоснование

Supply Chain Security на этапе сборки требует гарантий, что артефакт (образ, бинарный файл) получен из доверенного источника и не был изменён после сборки. Подпись артефактов позволяет проверить их целостность и авторство.

Cosign — инструмент из экосистемы Sigstore для подписи и верификации контейнерных образов, а также других артефактов (файлов, SBOM). Поддерживает два основных режима:

- Долгосрочный ключ (`cosign generate-key-pair`) — классический асимметричный ключ, приватный ключ хранится в CI-секретах.
- Keyless (безключевой) — эфемерный ключ генерируется "на лету", а идентификация подписывающего осуществляется с помощью OIDC-токена, выпущенного CI/CD платформой (GitHub, GitLab). Сертификат, привязанный к токenu, записывается в журнал Rekor. Такой подход исключает необходимость долгосрочного хранения приватного ключа.

Rekor — прозрачный журнал (transparency log), в который записываются метаданные подписания. Обеспечивает невозможность "тихой" компрометации — факт подписания становится публично проверяемым.

Sigstore — проект, объединяющий Cosign, Fulcio (центр сертификации), Rekor. Обеспечивает доверие без управления секретными ключами.

Интеграция в CI/CD:

- После сборки образа в реестр (Docker Hub, GitLab Container Registry, etc.) выполняется `cosign sign <image>`.
- Верификация перед деплоем — `cosign verify --key <pubkey> <image>` (или с флагом `--certificate-identity` и `--certificate-oidc-issuer` для keyless). Если подпись отсутствует или некорректна, команда вернёт ненулевой код.
- Блокировка деплоя гарантирует, что в продакшн попадают только подписанные артефакты.

4. Задания для индивидуального выполнения

4.1. Базовый уровень (15 вариантов)

Студент выполняет вариант в соответствии с последней цифрой номера зачётки/списка. Во всех вариантах реализуется подпись образа и верификация с ключом.

Вар.	Тип подписываемого артефакта	Реестр для хранения образа (предпочтительный)	Механизм ключа
1	Docker-образ (Python приложение)	GitLab Container Registry	Генерация через cosign generate-key-pair, парольная защита
2	Docker-образ (Node.js приложение)	GitLab Container Registry	Генерация без пароля
3	OCI-образ (любой)	Docker Hub	Импорт существующей RSA-пары
4	SBOM-файл (CycloneDX JSON)	- (файл в репозитории)	Генерация ключа
5	Бинарный файл (Go-сборка)	- (артефакт сборки)	Генерация ключа
6	Docker-образ (Java-приложение)	GitHub Container Registry (GHCR)	Генерация ключа
7	Текстовый файл с конфигурацией	- (артефакт)	Генерация ключа
8	Docker-образ (минимальный alpine)	GitLab Container Registry	Генерация ключа с последующей проверкой через cosign.pub
9	Helm-чарт (пакет)	OCI-based реестр	Генерация ключа
10	Docker-образ (Python+Flask)	Docker Hub	Импорт ключа
11	Docker-образ (Express.js)	GitLab Container Registry	Генерация ключа, требование наличия аннотации app

Вар.	Тип подписываемого артефакта	Реестр для хранения образа (предпочтительный)	Механизм ключа
12	SBOM в формате SPDX (JSON)	- (файл)	Генерация ключа
13	Docker-образ (уязвимый)	GitLab Container Registry (с блокировкой при verify failure)	Генерация ключа
14	Бинарный файл (С-программа)	- (артефакт)	Генерация ключа
15	Docker-образ (любой)	GitLab Container Registry	Генерация ключа, верификация не только по public key, но и по дополнительным аннотациям (custom)

Требования к базовому уровню:

- Наличие тестового артефакта (образ или файл).
- Пайплайн (GitLab CI / GitHub Actions) с шагами: сборка/публикация → подпись → верификация.
- Пайплайн падает, если верификация не пройдена.
- Отчёт содержит скриншоты успешного подписания, неудачной верификации (можно подменить образ).
- Ответы на контрольные вопросы $\geq 60\%$.

4.2. Средний уровень (15 вариантов)

Расширение базового сценария: keyless подпись, подпись SBOM, интеграция с реестром политик.

Вар.	Задача
1	Реализовать keyless-подпись образа, используя OIDC-токен GitHub Actions (или GitLab JWT), настроить верификацию с проверкой identity.
2	Подписать SBOM (CycloneDX), созданный Syft, и добавить его как аттестацию (cosign attest) для образа.
3	Настроить подпись образа с аннотациями (например, git_commit, build_date) и проверять

Вар.	Задача
	их при verify.
4	Использовать cosign sign-blob для подписи бинарного файла, проверка в CI.
5	Настроить двухстороннюю подпись: разработчик подписывает коммит, CI проверяет подпись коммита и затем подписывает образ.
6	Сравнить подходы keyless и с долгосрочным ключом, реализовать оба варианта и показать разницу в конфигурации.
7	Интегрировать проверку подписи в кластер Kubernetes через Connaisseur или Kyverno (см. ЛР №4) — только верификация при деплое.
8	Подписать образ и затем использовать cosign verify-attestation для проверки SBOM аттестации.
9	Реализовать pipeline, который автоматически подписывает все образы, помеченные тегом release-*.
10	Настроить подпись образа с использованием аппаратного токена (имитация, например, softHSM) — теоретически описать, продемонстрировать генерацию пары.
11	Подписать образ в GitLab CI с использованием ключа, защищённого в HashiCorp Vault или внешнем KMS (демонстрация с заглушкой).
12	Добавить верификацию подписи как Quality Gate в CI на этапе деплоя в staging.
13	Реализовать подпись нескольких образов (мультиархитектурный манифест) и проверку целостности.
14	Создать кастомный скрипт на Python, использующий cosign CLI, для автоматического подписания и верификации с логированием.
15	Подписать образ ключом, переданным через OIDC (GitLab JWT или GitHub), и проверить его с помощью Rekor search.

Требования: выполнение базового уровня + реализация среднего задания с пояснениями. Для keyless показать скриншоты записей в Rekor (если используется).

4.3. Повышенный уровень (15 вариантов)

Комплексная автоматизация, интеграция с экосистемой Sigstore, проверка на прозрачность.

Вар.	Задача
1	Интегрировать Cosign с Rekor: при подписании записывать запись в Rekor, при верификации проверять наличие записи в логе Rekor (cosign verify с проверкой включения).
2	Настроить политику Kyverno/Gatekeeper, принудительно требующую подпись Cosign для всех образов, деплоимых в production namespace.
3	Реализовать подпись образов с помощью Fulcio (keyless) с кастомным OIDC-провайдером (например, Dex).
4	Создать систему "канареечного" деплоя, где неподписанный образ не проходит стадию canary и откатывается.
5	Разработать CLI-инструмент, оборачивающий Cosign, для упрощения подписи и верификации с метаданными.
6	Настроить интеграцию с Sigstore policy-controller: создание ClusterImagePolicy, которая проверяет подпись и identity.
7	Построить полный жизненный цикл артефакта: сборка → подпись (keyless) → загрузка SBOM → аттестация → верификация в Admission контроллере → развёртывание.
8	Автоматическая ротация ключей подписи через заданный период (имитация) и обновление публичных ключей в верификаторах.
9	Использование Rust/Python библиотеки sigstore для подписания артефакта напрямую без CLI, интеграция в кастомный билд-инструмент.
10	Настройка защиты от downgrade-атак: при верификации проверять, что подпись соответствует ключу, действительному на момент сборки (использование TUF).
11	Создание веб-дашборда для отображения статуса подписанности образов в реестре (сканер + Grafana).
12	Интеграция с OPA: политика, которая допускает деплой только если образ подписан и

Вар.	Задача
	идентификатор подписи соответствует разрешённым.
13	Настройка проверки подписи через CI при развёртывании в OpenShift (если доступен) с использованием встроенного механизма образов.
14	Реализовать мульти-подпись (несколько подписывающих) и политику, требующую, чтобы образ был подписан минимум 2 из 3 ключей.
15	Полная интеграция Sigstore: хранилище подписей в OCI, мониторинг Rekor, уведомления при обнаружении неподписанного образа в проде.

Требования: автоматизированная система, демонстрирующая полное доверие к цепочке поставок; детальное описание архитектуры.

5. Контрольные вопросы

1. Что такое Cosign и какую проблему безопасности цепочки поставок он решает?
2. В чём разница между подписью с долгосрочным ключом и keyless-подписью? Преимущества и недостатки.
3. Опишите процесс получения сертификата в keyless-режиме с помощью OIDC.
4. Что такое Rekor и как он обеспечивает прозрачность подписей?
5. Каким образом можно проверить, был ли образ подписан, перед его деплоем в кластер?
6. Как связаны Cosign и SBOM? Как обеспечить целостность SBOM с помощью подписи?
7. Что произойдёт, если приватный ключ Cosign будет скомпрометирован в классической схеме? Как минимизировать риски?
8. Как интегрировать проверку подписи в пайплайн CI/CD и предотвратить деплой неподписанных образов?
9. Какие метаданные можно прикрепить к подписи (аннотации) и зачем они нужны?
10. Какие альтернативы Cosign вы знаете для подписи артефактов?

6. Требования к отчёту

Отчёт (PDF) должен содержать:

- Титульный лист, цель, задачи.
- Описание созданного артефакта (Dockerfile, манифест) и реестра.
- Конфигурацию CI/CD с пояснением шагов sign/verify (YAML-листинг).
- Скриншоты успешной подписи и успешной верификации (включая консольный вывод).
- Демонстрацию провала пайплайна при подделке (например, верификация чужого образа).

- Для keyless — скриншот с информацией об эфемерном сертификате, ссылка на запись в Rekor (если доступна).
- Индивидуальная часть: описание выполненного задания, листинги, скриншоты.
- Ответы на контрольные вопросы.
- Выводы.

7. Критерии оценивания

7.1. «Удовлетворительно» (базовый уровень):

- Сгенерирован ключ (или использован keyless), выполнена подпись одного артефакта.
- Пайплайн включает верификацию, которая завершается ошибкой при неправильной подписи.
- Отчёт оформлен, контрольные вопросы $\geq 60\%$.

7.2. «Хорошо» (средний уровень):

- Все базовые требования.
- Реализовано среднее задание (keyless, аттестация SBOM, сравнение, интеграция с политикой).
- Показано, что студент понимает разницу между долгосрочным ключом и keyless.

7.3. «Отлично» (повышенный уровень):

- Все требования среднего уровня.
- Реализовано сложное задание: полная интеграция с Rekor, Admission контроллером, автоматическое управление ключами.
- Продемонстрировано глубокое понимание цепочки доверия и прозрачности.
- Ответы на контрольные вопросы аргументированы, возможно предложение улучшений.

7.4. Понижение оценки:

- Отсутствие работающего пайплайна или скриншотов — отправить на доработку.
- Верификация не приводит к падению пайплайна при нарушении — не выше базового.
- Неполные ответы (менее 60% для базового, менее 80% для повышенного) — отправляется на пересдачу.

Данная лабораторная работа формирует ключевые практики безопасности цепочки поставок (ПК-8), готовя студентов к дальнейшему изучению безопасности автономных систем, где обеспечение целостности прошивок и OTA-обновлений критически важно.

Лабораторная работа №7
Формирование SBOM и интеграция с Dependency-Track

1. Цель

Сформировать практические навыки непрерывного мониторинга безопасности цепочки поставок путём автоматического формирования SBOM (Software Bill of Materials) и его загрузки в платформу управления зависимостями OWASP Dependency-Track для централизованного анализа уязвимостей, управления рисками и соответствием политикам.

2. Задачи

- Развернуть OWASP Dependency-Track с помощью Docker Compose и получить доступ к веб-интерфейсу.

- Сгенерировать SBOM в формате CycloneDX (JSON) для тестового проекта с помощью Syft, Trivy или cdxgen.
- Настроить автоматическую загрузку SBOM в Dependency-Track через REST API с использованием ключа доступа.
- Проанализировать результаты сканирования в пользовательском интерфейсе: идентифицированные уязвимости, их серьёзность, затронутые компоненты.
- Настроить политики безопасности (оповещения, пороговые значения) и продемонстрировать их работу.
- (Для среднего/повышенного уровней) интегрировать загрузку SBOM в CI/CD-пайплайн, настроить уведомления (Webhook/Email), реализовать управление рисками через VEX.

3. Теоретическое обоснование

OWASP Dependency-Track — это платформа с открытым исходным кодом для непрерывного анализа состава программного обеспечения (SCA), которая позволяет организациям отслеживать и управлять рисками, связанными с использованием сторонних компонентов. Она принимает на вход SBOM (в формате CycloneDX или SPDX), анализирует их на наличие известных уязвимостей (CVE), нарушений лицензионной политики и устаревших версий, предоставляя централизованный дашборд для принятия решений.

Основные компоненты платформы:

- API Server — REST-интерфейс для загрузки SBOM, получения отчётов, управления политиками.
- Frontend (UI) — Angular-приложение для визуализации данных.
- База данных — поддерживается H2 (для разработки) и PostgreSQL (для production).
- Анализаторы — внутренние сервисы, сверяющие компоненты из SBOM с базами уязвимостей (NVD, OSS Index, GitHub Advisories и др.).

Процесс работы:

1. SBOM загружается через API с указанием проекта и версии.
2. Платформа разбирает SBOM, извлекает перечень компонентов (имя, версия, лицензия, хеш) и создаёт инвентаризацию.
3. Запускается анализ на уязвимости и нарушения лицензионных политик.
4. Результаты отображаются в UI, могут высылаться уведомления (Email, Webhook, Jira и т.д.).
5. Политики позволяют автоматически реагировать на изменения (например, назначать статус "не соответствует требованиям" при появлении критической уязвимости).

VEX (Vulnerability Exploitability eXchange) — стандарт, позволяющий документировать анализ применимости уязвимости в конкретном контексте (например, уязвимость не эксплуатируется, потому что уязвимый код не используется). Dependency-Track поддерживает импорт VEX-документов для уточнения статуса уязвимостей.

Интеграция с CI/CD осуществляется через вызов API в пайплайне после сборки. Это позволяет автоматизировать сбор данных о составе ПО на каждом коммите и блокировать развёртывание, если нарушены политики безопасности.

4. Задания для индивидуального выполнения

Номер варианта соответствует последней цифре номера зачётки или списка группы.

4.1. Базовый уровень (15 вариантов)

Создать тестовый проект, развернуть Dependency-Track, сгенерировать SBOM и загрузить его через API.

Вар.	Язык / пакетный менеджер	Инструмент генерации SBOM	Формат SBOM	Способ загрузки	Особенность
1	Python (pip)	Syft	CycloneDX JSON	curl + API-ключ	Включить хотя бы одну CVE критического уровня
2	Node.js (npm)	cdxgen	CycloneDX JSON	curl	Проект с уязвимым lodash
3	Java (Maven)	cdxgen	CycloneDX JSON	curl	Имитация уязвимого log4j-core
4	Go (go mod)	Syft	SPDX JSON	curl	Использовать старую версию crypto или net
5	Python (Poetry)	Trivy	CycloneDX JSON	UI (ручная загрузка)	Показать, что уязвимости появляются после ручной загрузки
6	.NET (NuGet)	cdxgen	CycloneDX JSON	curl	Уязвимый Newtonsoft.Json
7	PHP (Composer)	Syft	CycloneDX JSON	curl	Использовать уязвимую версию phpmailer
8	Ruby (Bundler)	Syft	SPDX JSON	curl	Уязвимый nokogiri

Вар.	Язык / пакетный менеджер	Инструмент генерации SBOM	Формат SBOM	Способ загрузки	Особенность
9	Rust (Cargo)	Syft	CycloneDX JSON	curl	Использовать cargo audit для предварительного сканирования
10	Docker-образ (Alpine с python)	Syft	CycloneDX JSON	curl	SBOM всего образа, включая системные пакеты
11	Мультиязыковой проект (Python + JS)	cdxgen	CycloneDX JSON	curl	Показать компоненты обоих языков в одном SBOM
12	Node.js (Yarn)	Syft	CycloneDX JSON	curl	Добавить уязвимый пакет minimist
13	Python (requirements.txt)	pip-audit + конвертация	CycloneDX JSON	curl	Сконвертировать вывод pip-audit в CycloneDX через скрипт
14	Go binary файл (собранный)	Syft	CycloneDX JSON	curl	Сканировать бинарный файл, показать встроенные библиотеки
15	Java (Gradle)	cdxgen	CycloneDX JSON	curl	Уязвимая версия Spring Boot

Требования к базовому уровню:

- Работающий экземпляр Dependency-Track (локальный или в CI).
- SBOM сгенерирован и загружен; в UI видно как минимум одну уязвимость.
- Продемонстрирована загрузка через API (скриншот ответа сервера).
- Отчёт содержит скриншоты ключевых экранов (проект, компоненты, список уязвимостей).

- Ответы на контрольные вопросы ($\geq 60\%$).

4.2. Средний уровень (15 вариантов)

Автоматизация загрузки в CI/CD, настройка политик, уведомлений и аналитики.

Вар.	Задача
1	Интегрировать загрузку SBOM в GitLab CI: после сборки генерировать SBOM и отправлять в Dependency-Track. При обнаружении уязвимости с $CVSS \geq 9.0$ пайплайн должен завершаться ошибкой (через API-проверку).
2	Настроить Webhook-уведомление в Slack/Telegram при появлении новой уязвимости с уровнем High и выше.
3	Создать политику в Dependency-Track, которая нарушается при наличии любого CVE с $CVSS \geq 8.0$, и проверить её через API в CI.
4	Использовать DT API для получения статистики (количество уязвимостей по уровням) и сгенерировать Markdown-отчёт.
5	Настроить автоматическое тегирование версий в Dependency-Track при загрузке SBOM из CI (добавлять тег окружения, например, staging, production).
6	Сравнить два SBOM (текущая и предыдущая версия) с помощью DT API: вывести список новых уязвимостей и устранённых.
7	Реализовать VEX: сгенерировать VEX-документ для известной уязвимости (например, CVE-2021-44228) с указанием, что приложение не подвержено, и загрузить в DT через <code>/api/v1/vex</code> . Показать, что статус уязвимости изменился.
8	Интеграция с GitHub Actions: при push в репозиторий автоматически создаётся/обновляется проект в DT и загружается SBOM.
9	Настроить автоматическую проверку лицензионной политики (запретить использование GPL-3.0) и получить оповещение через API при нарушении.
10	Использовать Dependency-Track Python Client для загрузки SBOM и получения отчётов.
11	Настроить экспорт метрик (количество уязвимостей) из DT API в Prometheus и отобразить в Grafana.

Вар.	Задача
12	Создать скрипт, который периодически опрашивает DT API на наличие новых уязвимостей и при их обнаружении создаёт issue в GitLab.
13	Настроить DT на MySQL (вместо PostgreSQL) через Docker Compose для демонстрации.
14	Реализовать динамическое создание проекта в DT при первом появлении нового репозитория (на основе имени).
15	Сравнить поведение Dependency-Track при загрузке SBOM в формате CycloneDX и SPDX (для одного и того же проекта).

Требования: базовый уровень + реализация автоматизированного сценария с подтверждением (логи CI, скриншоты уведомлений, выводы скриптов).

4.3. Повышенный уровень (15 вариантов)

Глубокая интеграция, управление рисками, оркестрация.

Вар.	Задача
1	Интеграция Dependency-Track с DefectDojo: автоматическая загрузка результатов в DefectDojo для централизованного менеджмента уязвимостей.
2	Настроить DT в кластере Kubernetes с использованием Helm, настроить Ingress и TLS (Let's Encrypt).
3	Разработать простой Java-плагин для Dependency-Track (сборка, установка), добавляющий кастомный источник данных об уязвимостях (например, локальный файл).
4	Создать конвейер, который при обнаружении уязвимости HIGH автоматически формирует Pull Request с обновлением зависимости (Renovate/Dependabot), проверяет, что обновление устраняет уязвимость, и закрывает тикет в DT.
5	Реализовать "Compliance as Code" для Dependency-Track: все политики (какие уровни уязвимостей допустимы, обязательные лицензии) описаны в YAML-файлах в репозитории и применяются через API при каждом деплое.
6	Организовать работу DT в режиме команды: несколько проектов, разделение прав на основе OIDC (например, GitLab OAuth), аудит действий.

Вар.	Задача
7	Интеграция с системой ServiceNow/Jira: при появлении критической уязвимости автоматически создаётся инцидент, а когда он закрывается, статус синхронизируется с DT.
8	Развернуть DT в высокодоступной конфигурации: несколько экземпляров API-сервера за балансировщиком, общая БД PostgreSQL, кеширование Redis.
9	Создать внешний сервис-посредник (API Gateway), который анализирует разрешение от DT перед деплоем: если в проекте есть неприемлемые риски, деплой блокируется.
10	Построить систему непрерывного мониторинга: SBOM для всех активных проектов загружаются каждую ночь, результаты агрегируются, формируется сводный отчет для руководства (PDF).
11	Разработать утилиту командной строки на Python (с использованием Typer/Click), которая позволяет разработчикам локально анализировать SBOM через DT API и получать рекомендации.
12	Интеграция с системой хранения артефактов (Nexus/Artifactory): автоматически загружать SBOM при публикации артефакта, а в DT отслеживать связи.
13	Использование ML-модели для автоматической оценки риска уязвимости на основе дополнительных данных (наличие эксплойта, тренды упоминаний в соцсетях) и интеграция с DT через custom risk score.
14	Организовать процесс управления уязвимостями по стандарту ISO 27001 с использованием Dependency-Track как инструмента: документирование лечения, сроки, эскалация.
15	Создание сервиса "SBOM as a Service" внутри компании: разработчик отправляет запрос, система генерирует SBOM, загружает в DT и возвращает краткий отчет с рекомендациями.

Требования: полная автоматизация, демонстрация зрелого процесса управления безопасностью цепочки поставок, описание архитектуры решения.

5. Контрольные вопросы

1. Что такое OWASP Dependency-Track и какие основные функции он выполняет?
2. Какие форматы SBOM поддерживает Dependency-Track?

3. Опишите процесс загрузки SBOM через REST API: какие эндпоинты и аутентификация используются?
4. Какие источники данных об уязвимостях использует Dependency-Track?
5. Что такое VEX и как он помогает уменьшить количество ложных срабатываний?
6. Каким образом можно настроить автоматические уведомления о появлении новых уязвимостей?
7. Как в Dependency-Track организована модель угроз и политик безопасности?
8. Какие преимущества даёт интеграция Dependency-Track в CI/CD конвейер?
9. Каковы основные компоненты архитектуры Dependency-Track (API, Frontend, DB) и зачем нужен каждый?
10. Какие инструменты помимо Syft можно использовать для генерации SBOM в формате CycloneDX?

6. Требования к отчёту

Отчёт (PDF) должен включать:

- Титульный лист.
- Цель и задачи.
- Описание тестового проекта и его зависимостей.
- Пошаговое развёртывание Dependency-Track и получение API-ключа (скриншоты).
- Листинг SBOM (фрагмент) и команды его генерации.
- Скриншоты загрузки SBOM через API и ответа сервера.
- Иллюстрации из UI: список проектов, детали компонентов, список уязвимостей.
- Для среднего/повышенного уровня: описание автоматизации, листинги CI-скриптов, скриншоты уведомлений, политик и VEX.
- Ответы на контрольные вопросы.
- Выводы (оценка эффективности инструмента, рекомендации по внедрению).

7. Критерии оценивания

7.1. «Удовлетворительно» (базовый уровень):

- Развёрнут Dependency-Track, через UI создан проект и загружен SBOM (или через API).
- Продемонстрировано отображение хотя бы одной уязвимости.
- Отчёт содержит обязательные разделы и скриншоты.
- Правильные ответы на 60% контрольных вопросов.

7.2. «Хорошо» (средний уровень):

- Все требования базового уровня.

- Выполнена автоматизированная загрузка SBOM в CI/CD с проверкой политик или уведомлениями.
- Продемонстрирована настройка уведомлений или политик (Webhook/Email/Portege).
- Ответы на контрольные вопросы $\geq 80\%$.

7.3. «Отлично» (повышенный уровень):

- Все требования среднего уровня.
- Реализован комплексный сценарий управления рисками: интеграция с внешними системами, VEX, управление политиками как кодом, дашборды.
- Представлена архитектурная схема решения и обоснование выбора технологий.
- Ответы на контрольные вопросы полны и развёрнуты.

7.4. Понижение оценки:

- Отсутствие работающего UI или скриншотов, подтверждающих уязвимости, — доработка.
- Нет демонстрации автоматической загрузки (для среднего и выше) — не выше базового.
- Контрольные вопросы слабо раскрыты — направляется на пересдачу.

Выполнение данной лабораторной работы закрепляет навыки управления безопасностью цепочки поставок (ПК-8) и является основой для внедрения комплексного процесса управления уязвимостями в инженерных и автономных системах, где состав программного обеспечения критически влияет на надёжность и безопасность.

Лабораторная работа №8

Безопасность CI/CD пайплайна (Hardening)

1. Цель

Сформировать у обучающихся практические навыки защиты CI/CD-конвейера от типовых угроз: утечки секретов, компрометации сборочного агента (раннера), избыточных привилегий и использования небезопасных статических ключей — путём внедрения механизмов маскирования, защищённых переменных, временных токенов и OIDC-аутентификации.

2. Задачи

- Изучить основные риски безопасности CI/CD в соответствии с OWASP Top 10 CI/CD Security Risks.
- Настроить переменные GitLab CI с флагами Protected и Masked, проверить маскирование в логах.
- Организовать безопасное взаимодействие между задачами и реестром с помощью CI_JOB_TOKEN.
- Реализовать аутентификацию в облачном провайдере (AWS/GCP/Azure) через OIDC, исключая хранение статических долгосрочных ключей.

- Проанализировать конфигурацию раннеров с точки зрения минимальных привилегий и рекомендовать улучшения.

3. Теоретическое обоснование

Безопасность CI/CD пайплайна охватывает меры по предотвращению компрометации самого процесса сборки, доставки и развёртывания. Основные угрозы включают:

- Утечка секретов в логи или артефакты (API-ключи, пароли, сертификаты).
- Компрометация раннера — получение злоумышленником контроля над сборочным агентом, что даёт доступ ко всем проектам, использующим этот раннер.
- Инъекции в скрипты сборки через пользовательские переменные или параметры.
- Недостаточное управление доступом к триггерам пайплайна, окружениям и секретам.

Защитные механизмы GitLab CI:

- Protected variables — переменные, доступные только для защищённых веток (main) или тегов, предотвращая их использование в незащищённых контекстах.
- Masked variables — значение переменной автоматически заменяется на [MASKED] при выводе в логи, снижая риск утечки.
- File variables — секреты, сохраняемые в виде временного файла во время работы задания, не раскрываются в интерфейсе и не попадают в логи.
- CI_JOB_TOKEN — временный токен, уникальный для каждого задания, который предоставляет ограниченные права на чтение ресурсов в рамках того же проекта (Container Registry, Packages, Terraform State). Его преимущество в том, что он не привязан к конкретному пользователю и имеет короткий срок жизни.

OIDC (OpenID Connect) — протокол аутентификации, позволяющий установить доверительные отношения между GitLab (как поставщиком удостоверений) и облачным провайдером. Вместо хранения статических AWS_ACCESS_KEY_ID / AWS_SECRET_ACCESS_KEY в переменных CI, пайплайн получает кратковременные облачные креды, обменивая JWT-токен задания (CI_JOB_JWT) на роль с доверительной политикой. Аналогичные механизмы существуют для GCP (Workload Identity Federation) и Azure (Workload Identity Federation).

Принцип минимальных привилегий требует, чтобы раннеры выполнялись в изолированной среде с ограниченными capabilities (например, без privileged: true в Docker, если это возможно), а сервисные аккаунты имели только необходимые разрешения.

4. Задания для индивидуального выполнения

Номер варианта соответствует последней цифре номера зачётной книжки / списка группы.

4.1. Базовый уровень (15 вариантов)

Реализовать механизм защиты секретов или безопасной аутентификации согласно заданию, продемонстрировать работоспособность в CI.

Вар.	Задача

Вар.	Задача
1	Настроить переменную DOCKER_PASSWORD как Masked и Protected, вывести её в лог (подтвердить маскировку) и показать, что она недоступна в ветке, отличной от main.
2	Использовать переменную типа File для хранения приватного SSH-ключа. Показать, что в скрипте можно скопировать файл и использовать, а в логах содержимое не отображается.
3	Создать пайплайн, выводящий значение переменной SECRET_TOKEN без маскировки (имитация утечки), затем включить маскировку и продемонстрировать, что утечка устранена.
4	Настроить доступ к GitLab Container Registry другого проекта с помощью CI_JOB_TOKEN: добавить проект в Members с ролью Reporter и выполнить docker pull в своём пайплайне.
5	Настроить OIDC-аутентификацию в AWS: создать минимальную роль (доступ только к s3:ListBucket в тестовом бакете), получить временные креды и выполнить aws s3 ls.
6	Аналогично варианту 5, но для Google Cloud (Workload Identity Federation): настроить пул и провайдера, получить кратковременный токен GCP и выполнить gcloud storage ls.
7	Настроить OIDC для Azure: создать federated credential, получить токен и показать az group list.
8	Комбинировать Protected и Masked: задать переменную DB_PASSWORD, доступную только для protected-веток и тегов, и показать, что в незащищённой ветке пайплайн завершается ошибкой, т.к. переменная не определена.
9	Использовать CI_JOB_TOKEN для доступа к GitLab Packages (например, опубликовать generic-пакет в одном проекте и скачать его в другом).
10	Настроить маскирование переменной, содержащей символы (спецсимволы), и проверить, что маскировка работает корректно (GitLab маскирует значения с длиной ≥ 8 символов без пробелов).
11	Создать две переменные: USER и PASS (Masked, Protected), использовать их для базовой аутентификации в curl-запросе, убедиться, что пароль не появляется в логах, а URL с \$USER:\$PASS маскирован.

Вар.	Задача
12	Внедрить проверку на наличие секретов в логах с помощью gitleaks в отдельном stage (после сборки), показать пример обнаружения случайного коммита с ключом.
13	Использовать динамические переменные окружения с помощью CI/CD variables и маскирование для передачи токена SonarQube, выполнив анализ кода (sonar-scanner) и проверив маскировку.
14	Настроить доступ к HashiCorp Vault через AppRole (статический секрет, хранящийся в переменной), показать получение динамического секрета и его использование без раскрытия.
15	Сравнить передачу секрета через обычную переменную и через masked variable на примере токена доступа к внешнему API – показать разницу в логах.

Требования к базовому уровню:

- Работающий пайплайн, в котором применяется заданный механизм защиты.
- Скриншоты логов с замаскированным секретом или подтверждение успешной OIDC-аутентификации без статического ключа.
- Отчёт с описанием настроек и ответами на вопросы $\geq 60\%$.

4.2. Средний уровень (15 вариантов)

Расширенная настройка безопасности пайплайна: OIDC с деталями, анализ прав раннера, аудит зависимостей, интеграция с секрет-менеджерами.

Вар.	Задача
1	Реализовать OIDC с AWS с запросом временных кредов и проверкой того, что роли имеют минимально необходимые права (например, только на запись в конкретный S3 bucket). Создать IAM политику с ограничением по IP (условно).
2	Настроить GitLab runner с Docker executor, но запретить использовать привилегированный режим (privileged = false). Продемонстрировать, что это препятствует запуску контейнеров с особыми capabilities (например, Docker-in-Docker).
3	Провести аудит безопасности пайплайна с помощью инструмента gitlab-sast (или kics для IaC) и исправить найденные проблемы в конфигурации CI.
4	Настроить защищённые переменные для разных окружений: STAGING_TOKEN доступен только для ветки develop, PROD_TOKEN —

Вар.	Задача
	только для main. Проверить, что при попытке деплоя из feature ветки переменные не определены.
5	Интегрировать HashiCorp Vault с аутентификацией по JWT от GitLab (через jwt auth backend). Вместо статического VAULT_TOKEN использовать CI_JOB_JWT для получения секретов. Показать, что статический секрет не хранится в переменных GitLab.
6	Использовать gitleaks для сканирования не только логов, но и артефактов сборки на предмет утечки секретов, и настроить Quality Gate (пайплайн падает при обнаружении).
7	Настроить OIDC для доступа к AWS из GitLab и продемонстрировать, что при попытке использовать чужой JWT (из другого проекта) доступ запрещён благодаря условиям в доверительной политике.
8	Реализовать сценарий: пайплайн в проекте А с помощью CI_JOB_TOKEN пушит образ в Container Registry проекта В, но только если проект В явно разрешил это в настройках. Показать ошибку при отсутствии разрешения и успех при его добавлении.
9	Настроить переменную типа File для хранения GPG-ключа для подписи артефактов. Продемонстрировать подпись файла без раскрытия приватного ключа в логах.
10	Создать шаблон (CI/CD component) "secure-pipeline", который автоматически настраивает маскирование и проверки на утечки секретов для любого включающего проекта.
11	Проанализировать безопасность раннера, выполняющегося в Kubernetes executor: ограничить его привилегии (securityContext), запретить монтирование хостовых путей. Показать инцидент при попытке монтирования hostPath.
12	Настроить мониторинг использования секретов: после завершения пайплайна отправлять Webhook с хешем использованных переменных (не сами значения) для аудита.
13	Сравнить использование статического ключа API и OIDC для доступа к облачному хранилищу. Провести тест на время получения кредов, показать, что OIDC не требует ручного обновления ключей.
14	Реализовать пайплайн, который динамически генерирует временные SSH-ключи (ssh-keygen), использует их для деплоя и уничтожает после завершения, не сохраняя в переменных.

Вар.	Задача
15	Использовать GitLab External Secrets Operator (на базе Kubernetes Secrets Store CSI Driver) для подстановки секретов из AWS Secrets Manager без переменных CI. Эмуляция локально через скрипт.

Требования: базовые условия + реализация расширенного задания с демонстрацией улучшения безопасности.

4.3. Повышенный уровень (15 вариантов)

Комплексная защита всей CI/CD инфраструктуры, автоматизация аудита и реагирования.

Вар.	Задача
1	Построить полностью "бесключевой" CI/CD: все аутентификации только через OIDC/Federation во все облачные сервисы, никаких статических секретов в GitLab. Проверить, что при компрометации переменных окружения облачных ключей нет.
2	Разработать систему автоматического аудита CI/CD пайплайнов: анализатор .gitlab-ci.yml на предмет небезопасных практик (curl в лог, не маскированные переменные) и генерация отчёта с рекомендациями.
3	Внедрить canary-раннер: при изменении конфигурации раннера сначала запускать тестовый пайплайн в изолированном окружении, проверять, не нарушена ли безопасность, и только затем обновлять основной раннер.
4	Реализовать систему обнаружения аномалий в поведении пайплайна: анализ логов на предмет попыток доступа к необычным ресурсам, резкое увеличение длительности, частые перезапуски, с уведомлением в SIEM.
5	Настроить сетевую изоляцию CI/CD раннера (Docker/Kubernetes): разрешить только необходимые исходящие соединения (реестры, API) через NetworkPolicy/Firewall, и показать, что попытка доступа к внешнему серверу блокируется.
6	Создать оператор безопасности пайплайнов (GitLab Pipeline Security Operator), который просматривает все запущенные пайплайны, проверяет их на соответствие политикам (наличие маскированных переменных, OIDC) и при нарушениях отправляет алерт.
7	Интеграция с системой управления событиями безопасности (ELK) для централизованного хранения и анализа логов раннеров с поиском инцидентов.
8	Реализовать "Ephemeral CI/CD": каждое задание выполняется в полностью новой

Вар.	Задача
	временной среде, которая уничтожается после завершения (Kubernetes pods с TTL), не оставляя артефактов на раннере. Показать, что следы сборки отсутствуют.
9	Внедрить политику "четырёх глаз" для секретных операций: изменение защищённых переменных требует подтверждения через MR с аппрувом от security-team, реализовать через API GitLab.
10	Разработать инструмент для "красной команды" CI/CD: имитирует атаку утечки секретов через подмену образа раннера или инъекцию в скрипт, затем предлагает меры защиты.
11	Настроить кэширование секретов из Vault с автоматической ротацией: в начале пайплайна запрашивается временный секрет, который действителен только на время сборки, и отзывается после завершения.
12	Построить децентрализованную систему подписи пайплайнов с помощью OIDC и Sigstore для подтверждения целостности определения пайплайна.
13	Реализовать блокировку деплоя, если пайплайн не удовлетворяет baseline безопасности (отсутствуют обязательные переменные, не используются OIDC и т.д.) через GitLab Approval Rules.
14	Разработать и задокументировать безопасный SDLC процесс для организации, включающий все изученные меры (секреты, OIDC, минимальные привилегии, аудит) и представить в виде регламента.
15	Полная интеграция всех компонентов: GitLab + Vault + OIDC (облако) + NetworkPolicy + мониторинг аномалий + автоматическая ротация секретов. Представить архитектурную схему и демо.

Требования: полное описание архитектуры, доказательство работоспособности, оценка эффективности защиты.

5. Контрольные вопросы

1. Какие основные категории рисков безопасности CI/CD описываются в OWASP CI/CD Security Risks?
2. Чем отличаются переменные GitLab с флагами Protected и Masked? Как они помогают предотвратить утечки?
3. Что такое CI_JOB_TOKEN и в чём его преимущество перед персональными токенами доступа?

4. Объясните принцип работы OIDC в GitLab: как пайплайн получает временные облачные креды без статического ключа?
5. Какие требования к переменной, чтобы её можно было замаскировать (Masked)?
6. Почему не рекомендуется запускать Docker-раннеры в привилегированном режиме (privileged: true)? Какие альтернативы?
7. Каким образом можно интегрировать внешний секрет-менеджер (например, HashiCorp Vault) с GitLab CI без хранения долгоживущих токенов?
8. Что такое утечка секретов в артефактах и как её обнаружить с помощью инструментов типа gitleaks?
9. Какие ограничения нужно накладывать на сетевую активность раннеров для минимизации риска эксфильтрации данных?
10. Сформулируйте принцип минимальных привилегий применительно к настройке CI/CD.

6. Требования к отчёту

Отчёт (PDF) должен содержать:

- Титульный лист, цель, задачи.
- Описание исходной (небезопасной) конфигурации и реализованных улучшений.
- Листинг .gitlab-ci.yml (фрагменты, относящиеся к безопасности).
- Скриншоты настроек переменных (с маскированием/защитой), логов пайплайна с демонстрацией маскировки/отсутствия секретов.
- Для OIDC: скриншот доверительной политики (облачный провайдер) и успешного выполнения команды с временными правами.
- Индивидуальная часть (средний/повышенный уровень): описание дополнительного сценария, подтверждения его работы.
- Ответы на контрольные вопросы.
- Выводы и рекомендации по дальнейшему ужесточению (hardening).

7. Критерии оценивания

7.1. «Удовлетворительно» (базовый уровень):

- Выполнено одно из базовых заданий: настроена защита секретов (маскирование, protected, CI_JOB_TOKEN) или реализована OIDC-аутентификация.
- Логи подтверждают отсутствие утечек.
- Отчёт содержит все обязательные разделы.
- Правильные ответы на 60% контрольных вопросов.

7.2. «Хорошо» (средний уровень):

- Все базовые требования.
- Реализован сценарий среднего уровня (анализ прав, интеграция с Vault по JWT, сравнение подходов, gitleaks и т.д.).

- Продемонстрировано понимание принципов минимизации привилегий и отказа от статических ключей.

7.3. «Отлично» (повышенный уровень):

- Все требования среднего уровня.
- Построена комплексная система защиты пайплайна, включающая несколько аспектов (секреты, аутентификация, мониторинг, сетевая изоляция).
- Проведён детальный анализ рисков и эффективности мер, представлены архитектурные решения.
- Ответы на контрольные вопросы полные, аргументированные.

7.4. Понижение оценки:

- Обнаружение открытого секрета в логах при заявленной маскировке — работа не принимается.
- Неправильная настройка доступа (например, OIDC не работает, а используется статический ключ) — не выше базового уровня.
- Пропуск обязательных разделов отчёта — отправляется на доработку.

Освоение данной лабораторной работы формирует компетенции ПК-8 в части безопасной сборки и управления секретами, что является неотъемлемой частью безопасности цепочки поставок и необходимо для развёртывания защищённых инженерных и автономных систем.

Лабораторная работа №9

Анализ угроз автономной системы по методологии STRIDE

1. Цель

Сформировать практические навыки проведения систематического анализа угроз для автономных инженерных систем (БПЛА / робот) по методологии Microsoft STRIDE, оценки рисков методом DREAD и разработки архитектурных мер защиты, документируемых в отчёте «Модель угроз и нарушителя».

2. Задачи

- Выбрать конкретную автономную систему (симулятор дрона, роботизированная платформа, ROS 2-система) в качестве объекта анализа.
- Построить диаграмму потоков данных (Data Flow Diagram, DFD) целевой системы, выделив доверенные границы (trust boundaries).
- Применить методологию STRIDE к каждому элементу DFD для идентификации угроз безопасности.
- Оценить выявленные угрозы по шкале DREAD (Damage, Reproducibility, Exploitability, Affected users, Discoverability) или упрощённой метрике риска.
- Разработать перечень контрмер и архитектурных улучшений, снижающих риски до приемлемого уровня.

- Оформить результаты в виде структурированного документа «Модель угроз и нарушителя».

3. Теоретическое обоснование

STRIDE — методология анализа угроз, предложенная Microsoft, классифицирующая угрозы по шести категориям, соответствующим первым буквам:

- Spoofing (подделка) — выдача себя за другого субъекта.
- Tampering (искажение) — несанкционированная модификация данных или кода.
- Repudiation (отказ от факта действия) — невозможность доказать совершение действия субъектом.
- Information Disclosure (разглашение информации) — утечка конфиденциальной информации.
- Denial of Service (отказ в обслуживании) — нарушение доступности системы.
- Elevation of Privilege (повышение привилегий) — получение несанкционированного доступа с более высокими правами.

Data Flow Diagram (DFD) — графическое представление движения данных в системе, включающее элементы:

- External Entity (Внешняя сущность) — источник или потребитель данных вне системы (оператор, облачный сервис).
- Process (Процесс) — логический компонент, преобразующий данные (полётный контроллер, модуль навигации).
- Data Store (Хранилище данных) — место хранения информации (логи полёта, база данных миссий).
- Data Flow (Поток данных) — канал передачи данных между элементами.
- Trust Boundary (Доверенная граница) — граница, разделяющая зоны с разным уровнем доверия (шина дрона vs. внешний радиоканал).

Применение STRIDE к элементам DFD:

Элемент DFD	S	T	R	I	D	E
External Entity	✓		✓			
Process	✓	✓	✓	✓	✓	✓
Data Store		✓	✓	✓	✓	
Data Flow		✓		✓	✓	

DREAD — методика количественной оценки риска угрозы (или упрощённый качественный High/Medium/Low):

- Damage — какой ущерб нанесёт реализация угрозы.
- Reproducibility — насколько легко воспроизвести атаку.
- Exploitability — требуемые ресурсы и квалификация.
- Affected users — доля затронутых пользователей/компонентов.
- Discoverability — насколько легко обнаружить уязвимость.

Автономные системы (БПЛА, роботы) имеют уникальные аспекты безопасности:

- Беспроводные каналы связи (MAVLink, ROS 2 DDS, WiFi/LTE) — уязвимы к глушению (jamming), перехвату, подделке команд.
- Бортовые вычислители с ограниченными ресурсами — затруднено применение классических криптографических мер.
- Физический доступ к устройству после падения или захвата — риск компрометации прошивки, извлечения ключей.
- Цепочка поставок компонентов (сенсоры, актуаторы, полётные стеки) — риск внедрения вредоносных модификаций на этапе производства.
- OTA-обновления — критический канал, через который возможно распространение вредоносной прошивки.

Компетенция ПК-11 прямо предписывает умение проводить анализ угроз и внедрять меры защиты для таких систем.

4. Задания для индивидуального выполнения

Номер варианта определяется последней цифрой зачётной книжки / списка группы.

4.1. Базовый уровень (15 вариантов)

Каждый вариант задаёт конкретную автономную систему и приоритетный контекст угроз для анализа.

Вар.	Система	Контекст / фокус угроз
1	Квадрокоптер с PX4 (симулятор)	Радиоканал управления MAVLink
2	Квадрокоптер с ArduPilot (симулятор)	OTA-обновление прошивки (через WiFi/LTE)
3	Робот-доставщик (ROS 2)	Каналы связи между узлами (DDS) внутри робота
4	Беспилотный автомобиль (модель)	Сенсорная подсистема (лидары, камеры) — подделка данных

Вар.	Система	Контекст / фокус угроз
5	Подводный дрон (BlueROV)	Гидроакустическая связь и управление
6	Автономный агро-дрон	GPS-спуфинг и его влияние на точность опрыскивания
7	Роботизированная рука (манипулятор)	Промышленный протокол (Modbus/EtherCAT) — подделка команд
8	Рой дронов (3 единицы)	Безопасность связи между дронами в рое (swarm)
9	Беспилотный летательный аппарат самолётного типа	Дальняя телеметрия через спутниковый канал
10	Робот-пылесос (умный)	Облачное управление и интеграция с голосовым помощником
11	Автономный марсоход (имитатор)	Защита от задержек и сбоя в канале "Земля-Марс" (DTN)
12	Гоночный FPV-дрон	Перехват и подмена видеопотока, влияние на пилота
13	Автономный патрульный корабль	AIS-спуфинг и подделка навигационных данных
14	Роботизированный складской погрузчик	Безопасность Wi-Fi сети склада, MITM
15	Дрон для поисково-спасательных операций	Утечка телеметрии и разглашение местоположения пострадавших

Требования к базовому уровню:

- DFD-диаграмма системы (минимум 3 процесса, 2 хранилища, 2 внешние сущности).
- Таблица STRIDE с не менее чем 10 идентифицированными угрозами, распределёнными по элементам DFD.
- Оценка риска для трёх наиболее критичных угроз (качественная High/Medium/Low или DREAD).

- Перечень контрмер для найденных угроз (минимум 5 конкретных мер).
- Отчёт соответствует структуре, ответы на контрольные вопросы $\geq 60\%$.

4.2. Средний уровень (15 вариантов)

Расширенный анализ с применением дополнительных инструментов и углублённой оценкой.

Вар.	Задача
1	Построить DFD в PlantUML/Mermaid с автоматической генерацией из текста. Дополнить STRIDE анализом дерева атак (Attack Tree) для одной критической угрозы.
2	Применить методологию STRIDE-per-element ко всем элементам DFD (как в таблице выше), суммарно не менее 20 угроз.
3	Провести количественную оценку DREAD для всех найденных угроз с обоснованием каждой оценки. Построить тепловую карту рисков.
4	Сравнить методологии STRIDE и PASTA (кратко описать PASTA, применить к одному элементу DFD и сравнить результаты).
5	Использовать инструмент Microsoft Threat Modeling Tool для моделирования системы и автоматической генерации отчёта. Сравнить с ручным STRIDE.
6	Построить модель нарушителя (Motivation, Capability, Opportunity) для двух различных профилей: script kiddie и APT. Оценить угрозы с точки зрения каждого.
7	Провести анализ угроз для системы, использующей ROS 2 с Fast DDS, с акцентом на безопасность middleware (аутентификация, шифрование DDS).
8	Дополнить модель угроз рассмотрением физической безопасности: что происходит при захвате дрона, извлечении SD-карты, JTAG-отладке.
9	Разработать проверочные сценарии (Test Cases) для верификации защиты от 3 наиболее критичных угроз после внедрения контрмер.
10	Построить "карту навигации атакующего" (Attack Navigator) — визуализацию последовательности шагов для реализации сложной атаки (Kill Chain).
11	Сравнить применение STRIDE к симулированной системе и к системе с реальным оборудованием (на основе документации на открытые проекты, например, ArduPilot).

Вар.	Задача
12	Провести анализ Software Supply Chain для выбранной автономной системы: какие сторонние библиотеки/прошивки используются и какие риски они несут (SBOM + STRIDE).
13	Применить STRIDE к каналам связи, рассматривая не только MAVLink, но и видео-канал, управляющий канал второго оператора (dual control).
14	Провести оценку рисков по NIST SP 800-30 (Likelihood × Impact) и сравнить с DREAD. Объяснить различия в результатах.
15	Интегрировать анализ угроз с общей архитектурой безопасности: показать на DFD, где должны размещаться контроли безопасности (security controls) — схема "to-be".

Требования: выполнение базового уровня + углублённый анализ согласно варианту с графическими материалами и расширенной документацией.

4.3. Повышенный уровень (15 вариантов)

Комплексное моделирование угроз с элементами автоматизации и практической проверки.

Вар.	Задача
1	Разработать скрипт (Python/Bash) для автоматической генерации матрицы STRIDE на основе формального описания DFD в JSON.
2	Провести симуляцию атаки "GPS Spoofing" на симуляторе PX4 SITL с использованием HackRF/имитатора, задокументировать влияние на полёт и контрмеры.
3	Реализовать MAVLink Signing на симуляторе и продемонстрировать, что подписанные пакеты отвергают подделку (mitmпроху / самописный скрипт).
4	Построить цифровой двойник угроз (Digital Threat Twin): модель системы в виде графа, где узлы — компоненты, а рёбра — угрозы, с расчётом метрик центральности для определения критических узлов.
5	Разработать автоматизированный генератор отчёта по Threat Modeling в формате Docx/PDF на основе заполненной таблицы STRIDE + DREAD с графиками рисков.
6	Интегрировать анализ угроз в DevSecOps: при изменении архитектуры (MR) автоматически запускается скрипт, проверяющий, не появились ли новые потоки данных

Вар.	Задача
	без контрмер.
7	Построить полную модель безопасности БПЛА, включая Secure Boot, OTA, шифрование каналов, физическую защиту, и продемонстрировать на симуляторе/эмуляторе работоспособность одного из механизмов (например, подпись OTA).
8	Сравнить угрозы для одиночного БПЛА и для роя (swarm): как меняется поверхность атаки и какие появляются дополнительные векторы.
9	Разработать плагин для draw.io/VS Code, который позволяет прямо на DFD назначать элементы STRIDE и генерирует таблицу угроз.
10	Провести анализ угроз с использованием формального метода (например, Alloy или TLA+) для проверки выполнимости контрмер.
11	Создать интерактивный дашборд (Streamlit / Jupyter) для визуализации угроз, позволяющий фильтровать по уровню риска, элементу DFD, типу STRIDE.
12	На основе анализа угроз разработать полный Security Requirements Specification для автономной системы в формате IEEE 830.
13	Применить методологию MITRE ATLAS (Adversarial Threat Landscape for AI) к системе с элементами AI/ML (например, дрон с автономным компьютерным зрением).
14	Провести A/B тестирование: сравнить угрозы и контрмеры для системы с классической RTOS и с микроядерной ОС (seL4) в качестве основы полётного контроллера.
15	Разработать корпоративный стандарт Threat Modeling для компании, производящей БПЛА, на основе STRIDE, включающий шаблоны документов, интеграцию в процесс разработки и критерии приёмки безопасности.

Требования: демонстрация автоматизации/практической проверки, глубокий аналитический отчёт, архитектурные схемы с контрмерами, возможно видео-демонстрация симуляции атак и защит.

5. Контрольные вопросы

1. Что такое методология STRIDE и для чего она применяется?
2. Какие элементы включает диаграмма потоков данных (DFD) и как они связаны с категориями STRIDE?
3. В чём разница между угрозой (threat) и уязвимостью (vulnerability)?

4. Опишите методику оценки риска DREAD. Какие недостатки у этого подхода?
5. Какие особенности автономных систем (БПЛА) делают их уязвимыми к кибератакам, в отличие от классических веб-приложений?
6. Что такое Trust Boundary и как её определение влияет на анализ угроз?
7. Перечислите не менее трёх специфических атак на канал связи БПЛА и возможные контрмеры.
8. Какие угрозы из категории Elevation of Privilege (повышение привилегий) наиболее вероятны в полётном контроллере?
9. Как обеспечить целостность OTA-обновлений прошивки дрона? Какие криптографические механизмы применяются?
10. Каким образом результаты Threat Modeling интегрируются в процесс безопасной разработки (SDL)?

6. Требования к отчёту

Итоговый документ «Модель угроз и нарушителя» (PDF) должен включать:

- Титульный лист.
- Описание системы: краткая архитектура, назначение, ключевые компоненты, технологии.
- Диаграмма потоков данных (DFD): графическое представление с легендой, указанием доверенных границ.
- Таблица анализа STRIDE: для каждого элемента DFD → категории угроз → описание угроз.
- Оценка рисков: методика (DREAD / качественная), оценки для наиболее критичных угроз с обоснованием.
- Модель нарушителя: описание одного или нескольких профилей (внешний/внутренний, уровень подготовки, мотивация, ресурсы).
- Контрмеры: таблица соответствия "угроза → контрмера → остаточный риск".
- Инструментарий: список использованных инструментов (если применимо).
- Ответы на контрольные вопросы.
- Выводы: критический анализ защищённости системы, рекомендации по дальнейшим исследованиям.

Для среднего/повышенного уровня добавляются дополнительные секции согласно варианту.

7. Критерии оценивания

7.1. «Удовлетворительно» (базовый уровень):

- DFD построен корректно, выделены доверенные границы.
- Идентифицировано не менее 10 угроз, покрывающих все элементы DFD и не менее 4 категорий STRIDE.
- Проведена оценка рисков для ключевых угроз, предложены контрмеры.

- Отчёт оформлен в соответствии с требованиями.
- Верные ответы на контрольные вопросы не менее 60%.

7.2. «Хорошо» (средний уровень):

- Выполнены все требования базового уровня.
- Углублён анализ: DREAD со всеми оценками, модель нарушителя, Attack Tree, сравнение методологий (в зависимости от варианта).
- Представлены симуляции или дополнительные диаграммы.
- Ответы на вопросы $\geq 80\%$ правильных и развёрнутых.

7.3. «Отлично» (повышенный уровень):

- Выполнены все требования среднего уровня.
- Реализована автоматизация генерации/анализа, практическая симуляция атаки или дашборд.
- Продемонстрировано системное понимание процесса Threat Modeling и его места в жизненном цикле.
- Ответы на контрольные вопросы полные, содержит примеры и ссылки на источники.

7.4. Понижение оценки:

- Слабое покрытие STRIDE (менее 10 угроз, не затронуты все элементы) — не выше базового.
- Отсутствие контрмер или остаточных рисков — отправляется на доработку.
- Неверное определение доверенных границ, приводящее к принципиально неверному анализу — работа не принимается.

Лабораторная работа №10

Анализ защищённости протокола MAVLink

1. Цель

Сформировать практические навыки анализа защищённости коммуникационного протокола MAVLink путём перехвата трафика, изучения структуры пакетов и проведения симулированной атаки подмены навигационных данных с оценкой реакции автономной системы (БПЛА).

2. Задачи

- Развернуть среду симуляции PX4 SITL (Software-in-the-Loop) совместно с наземной станцией управления QGroundControl.
- Настроить перехват и анализ трафика MAVLink между полётным контроллером и GCS с использованием Wireshark / tcpdump.
- Изучить структуру пакета MAVLink v1/v2, идентифицировать поля, критичные для безопасности.
- Реализовать симуляцию атаки подмены GPS-координат (GPS Spoofing) через инъекцию пакетов MAVLink GPS_INPUT или HIL_GPS и пронаблюдать реакцию дрона (изменение позиции, режим полёта, предупреждения).
- Задokumentировать наблюдаемые индикаторы компрометации и предложить механизмы защиты (MAVLink Signing).

3. Теоретическое обоснование

Протокол MAVLink (Micro Air Vehicle Link) — легковесный бинарный протокол связи, де-факто стандарт для взаимодействия между компонентами беспилотных систем (полётный контроллер, GCS, полезная нагрузка). Текущая версия — MAVLink v2 с улучшенной структурой пакета.

Структура пакета MAVLink v2:

Байты	Поле	Описание
0	STX	Стартовый байт (0xFD для v2)
1	Payload Length	Длина полезной нагрузки (0–255)
2	Incompat Flags	Флаги несовместимости (бит 0 — подпись)
3	Compat Flags	Флаги совместимости
4	Sequence	Порядковый номер пакета
5	System ID	Идентификатор системы-отправителя
6	Component ID	Идентификатор компонента-отправителя
7–9	Message ID	Тип сообщения (24 бита)
10–..	Payload	Данные сообщения
..	Checksum (CRC16)	Контрольная сумма
..	Signature (13B)	Подпись (опционально, при incompat_flags & 0x01)

Основные уязвимости MAVLink без подписи:

- Отсутствие аутентификации: нет встроенного механизма проверки отправителя. Злоумышленник, имеющий доступ к радиоканалу, может инжектировать пакеты с любым System ID/Component ID.
- Отсутствие шифрования: вся телеметрия (координаты, состояние батареи, параметры) передаётся в открытом виде.
- Отсутствие проверки целостности: CRC16 защищает от случайных ошибок, но не от злонамеренной модификации (легко пересчитывается).
- Подверженность replay-атакам: без MAVLink Signing (или одноразовых номеров в криптографической подписи) перехваченные команды можно воспроизвести.

MAVLink Signing добавляет 13-байтовую подпись (SHA-256 + секретный ключ) к каждому пакету, обеспечивая аутентификацию и целостность. Однако требует управления ключами и увеличивает накладные расходы.

Фреймы MAVLink, используемые для GPS Spoofing:

- GPS_INPUT (#232) — позволяет внешнему источнику вводить GPS-данные в полётный контроллер.
- HIL_GPS (#113) — сообщение Hardware-in-the-Loop для симуляции GPS.
- GLOBAL_POSITION_INT (#33) — сообщение с текущей оценкой позиции (можно наблюдать эффект спуфинга).

4. Задания для индивидуального выполнения

Номер варианта соответствует последней цифре зачётки / списка группы.

4.1. Базовый уровень (15 вариантов)

Для каждого варианта указан специфический тип пакетов MAVLink, на который направлена симулированная атака.

Вар.	Атакуемый тип сообщения MAVLink	Механизм атаки	Ожидаемый эффект
1	GPS_INPUT / HIL_GPS	Инъекция поддельных координат (Цюрих → Москва)	Скачок позиции, срыв режима удержания
2	ATTITUDE_TARGET	Подмена целевых углов (roll/pitch)	Резкий наклон, возможно падение
3	SET_MODE	Принудительное переключение режима в STABILIZE/RTL	Потеря управления, уход в RTL без команды оператора
4	COMMAND_LONG (CMD=DO_SET_SERVO)	Инъекция команды сброса полезной нагрузки	Преждевременный сброс, нарушение миссии
5	COMMAND_LONG (CMD=NAV_LAND)	Принудительная посадка в заданной точке	Дрон начинает незапланированную посадку
6	PARAM_SET	Изменение параметра (например, EKF2_AID_MASK)	Внесение ошибки в алгоритмы навигации

Вар.	Атакуемый тип сообщения MAVLink	Механизм атаки	Ожидаемый эффект
7	MANUAL_CONTROL	Подмена сигналов стиков (roll/pitch/yaw/throttle)	Неуправляемое движение дрона
8	MISSION_ITEM	Добавление произвольной точки маршрута	Изменение плана полёта, увод в заданную точку
9	MISSION_CLEAR_ALL	Очистка миссии в полёте	Дрон теряет задание, переходит в режим ожидания
10	HEARTBEAT	Подмена HEARTBEAT от другого компонента (например, GCS)	Дезориентация системы, двойное управление
11	SYS_STATUS	Подмена статуса батареи (критически низкий заряд)	Аварийная посадка
12	ESTIMATOR_STATUS	Инъекция ложных флагов (ошибка GPS, ошибка IMU)	Срабатывание failsafe, переход в аварийный режим
13	RC_CHANNELS_OVERRIDE	Перехват управления через RC-канал	Полный контроль над движением дрона
14	GPS_RTCM_DATA	Инъекция ложных RTCM-поправок (DGNS)	Искажение GPS-решения
15	COMMAND_LONG (CMD=DO_DIGICAM_CONTROL)	Несанкционированное фотографирование	Нарушение конфиденциальности миссии

Требования к базовому уровню:

- Развёрнута связка PX4 SITL + QGroundControl.
- Выполнен захват трафика Wireshark (скриншот с разбором пакета MAVLink).

- Реализована одна симулированная атака согласно варианту с демонстрацией эффекта (скриншот до/после, лог).
- Предложены меры противодействия (MAVLink Signing, rate limiting, и т.д.).
- Ответы на контрольные вопросы $\geq 60\%$.

4.2. Средний уровень (15 вариантов)

Вар.	Задача
1	Включить MAVLink Signing в симуляции (PX4 SITL), продемонстрировать, что инъекция неподписанных пакетов отвергается.
2	Реализовать Replay Attack: записать легитимные пакеты управления и воспроизвести их, показав, что дрон выполняет команды повторно (без signing).
3	Написать сниффер на Python (Scapy/pymavlink), который детектирует аномалии в потоке MAVLink (например, резкие скачки координат) и выдаёт алерт.
4	Сравнить задержки и накладные расходы при включении/выключении MAVLink Signing (ping, время цикла).
5	Провести атаку Man-in-the-Middle (MITM), модифицирующую координаты в реальном времени (прозрачный прокси с mavproxy).
6	Реализовать атаку GPS Spoofing с плавным уводом координат (имитация "угона"), оценить, как EKF реагирует на медленный drift по сравнению с резким скачком.
7	Интегрировать в симуляцию инструмент mavlink-warden и показать, как он может блокировать вредоносные пакеты на уровне маршрутизатора MAVLink.
8	Провести анализ трафика MAVLink между реальным полётным контроллером (если доступен) и GCS, сравнить с SITL.
9	Разработать скрипт автоматического мониторинга и оповещения (Webhook/Slack) при обнаружении инъекции пакетов MAVLink в сеть.
10	Исследовать защищённость QGroundControl: можно ли подменить IP-адрес GCS и перехватить видеопоток?
11	Настроить протоколирование всех команд MAVLink с временными метками и подписью для последующего аудита.

Вар.	Задача
12	Проверить, как PX4 обрабатывает некорректные CRC пакетов: генерирует пакеты с неправильной контрольной суммой и анализирует логи.
13	Использовать SDR (HackRF/имитатор) для передачи поддельных пакетов MAVLink по радиоканалу (если оборудование доступно).
14	Настроить MAVLink Router (или MAVP2P) и показать, что неподписанные пакеты блокируются на уровне маршрутизатора.
15	Интегрировать MAVLink Signing с CI/CD: автоматически проверять, что прошивка поддерживает signing и ключи корректны.

Требования: базовый уровень + выполнение расширенного задания с демонстрацией.

4.3. Повышенный уровень (15 вариантов)

Вар.	Задача
1	Реализовать полноценный MAVLink IDS (Intrusion Detection System) на основе анализа последовательности пакетов и машинного обучения.
2	Разработать систему управления ключами MAVLink для флота дронов: автоматическая ротация, распределение pre-shared keys.
3	Интегрировать MAVLink с Transport Layer Security (TLS) через прокси (mavlink-over-tcp), сравнить с нативным MAVLink Signing.
4	Создать "red team / blue team" сценарий: одна команда атакует дрона (GPS Spoof, Command Injection), другая защищает с помощью MAVLink Signing и IDS.
5	Провести фаззинг MAVLink-парсера PX4 (используя AFL / libFuzzer), найти потенциальные уязвимости в обработке некорректных пакетов.
6	Разработать кастомный MAVLink-сообщение с шифрованием полезной нагрузки (AES-GCM) и интегрировать в PX4/QGC.
7	Построить honeypot-ловушку для БПЛА: эмулировать MAVLink-интерфейс, привлекающий атакующих, и логировать их действия.

Вар.	Задача
8	Исследовать безопасность MAVLink v2 с подписью против атаки с понижением версии (downgrade to v1).
9	Разработать модуль для PX4, реализующий challenge-response аутентификацию для критических команд (например, ARM, LAND).
10	Сравнить безопасность MAVLink и ROS 2 DDS в контексте управления БПЛА: провести аналогичную атаку на обе системы, сравнить результаты.
11	Разработать расширение для Wireshark (диссектор), которое автоматически подсвечивает подозрительные паттерны MAVLink (скачки координат, необычные команды).
12	Реализовать протокол согласования ключей (ECDH) поверх MAVLink для динамического установления сессионного ключа подписи.
13	Провести аудит безопасности полного стека управления дроном (PX4 + MAVSDK + GCS) с автоматизированным инструментом и задокументировать все найденные уязвимости.
14	Разработать микроядерный прокси безопасности, который фильтрует MAVLink-трафик на уровне бизнес-логики (белый список команд, rate limit, гео-фенсинг).
15	Построить систему доверенной загрузки и аттестации для полётного контроллера, интегрированную с MAVLink (передача evidence о целостности прошивки в GCS).

Требования: демонстрация работы комплексной защиты, отчёт с архитектурой, код/конфигурации.

5. Контрольные вопросы (ЛР №10)

1. Опишите структуру пакета MAVLink v2. Какие поля влияют на безопасность?
2. В чём разница между MAVLink v1 и v2 с точки зрения безопасности?
3. Какие основные уязвимости протокола MAVLink при отсутствии подписи пакетов?
4. Что такое MAVLink Signing и как он защищает от спуфинга и подмены данных?
5. Почему CRC16 недостаточно для обеспечения целостности пакетов?
6. Какие типы сообщений MAVLink наиболее опасны при компрометации радиоканала?
7. Как фильтр Калмана (EKF) в PX4 может частично противостоять GPS Spoofing?
8. Какие ещё протоколы, кроме MAVLink, используются в БПЛА и требуют защиты (видео, управление полезной нагрузкой)?
9. Как можно организовать мониторинг и обнаружение атак на MAVLink?

10. Какие ограничения MAVLink Signing (производительность, управление ключами) важно учитывать при развёртывании на реальных БПЛА?

Лабораторная работа №11

Настройка безопасности в ROS 2 (DDS Security)

1. Цель

Сформировать практические навыки защиты публикационно-подписочного обмена сообщениями в робототехнической операционной системе ROS 2 путём настройки встроенных механизмов безопасности DDS (Data Distribution Service): аутентификации, шифрования и контроля доступа на основе SROS 2.

2. Задачи

- Развернуть среду ROS 2 (Humble/Iron/Rolling) с двумя узлами, обменивающимися сообщениями через топик (например, chatter / listener из demo_nodes_cpp).
- Настроить рабочее пространство безопасности (Security Enclaves) с помощью утилит SROS 2 (ros2 security).
- Сгенерировать пары ключей и сертификатов для необходимых компонентов (CA, Identity, Permissions).
- Сконфигурировать XML-политики управления доступом (Permissions), определяющие разрешённые действия для каждого узла.
- Запустить узлы с включёнными DDS-защитами (AUTH, ENCRYPT, ACCESS_CONTROL) и убедиться в успешном безопасном обмене.
- Провести анализ защищённого трафика с помощью Wireshark для подтверждения шифрования.

3. Теоретическое обоснование

ROS 2 (Robot Operating System 2) использует стандарт DDS (Data Distribution Service) в качестве связующего ПО (middleware) для коммуникации между узлами. В отличие от ROS 1, где безопасность практически отсутствовала, ROS 2 предоставляет встроенные механизмы

безопасности, основанные на спецификации DDS Security (DDS-SEC), и инструменты для удобной настройки — SROS 2.

DDS Security включает следующие механизмы:

- Authentication (Аутентификация): Проверка подлинности участников домена DDS (DomainParticipants) с использованием PKI (X.509 сертификатов) и протокола DDS:Auth:PKI-DH. Взаимная аутентификация на основе challenge-response с использованием закрытых ключей.
- Access Control (Контроль доступа): Проверка прав участника на выполнение определённых операций (создание топиков, чтение/запись) на основе подписанных XML-файлов разрешений (Permissions), выданных центром сертификации (CA). Файлы включают разрешённые действия (allow rule) с указанием топика, домена, допустимых операций.
- Cryptography (Криптография): Прозрачное шифрование полезной нагрузки сообщений (AES-GCM или AES-CMAC) с использованием сессионных ключей, получаемых во время аутентификации. Защита обеспечивается на уровне DDS без изменения кода узлов.

SROS 2 — набор инструментов командной строки, упрощающих создание и управление инфраструктурой безопасности:

bash

```
ros2 security create_keystore <keystore_dir>    # создание хранилища ключей
```

```
ros2 security create_key <keystore_dir> <identity> # генерация ключей для конкретного identity
```

```
ros2 security create_permission <keystore_dir> <identity> <policy_xml> # настройка прав
```

Структура Keystore:

keystore/

public/ # сертификаты CA

private/ # приватный ключ CA

enclaves/

/talker/ # identity узла

cert.pem, permissions.p7s, key.pem, permissions.xml

/listener/

...

В переменной окружения ROS_SECURITY_ENCLAVE_OVERRIDE=/path/to/enclaves/talker или через --ros-args --enclave /talker указывается путь к папке identity.

Принцип работы:

1. CA создаёт корневой сертификат и ключи.
2. Для каждого узла генерируются Identity-сертификат и ключевая пара, подписанные CA.
3. Создаются Permissions.xml и подписываются CA (файл .p7s).

4. При запуске узлы загружают свои identity, обмениваются сертификатами, проверяют подлинность СА, и устанавливают зашифрованный обмен, проверяя права доступа.

4. Задания для индивидуального выполнения (ЛР №11)

Номер варианта соответствует последней цифре зачётки / списка группы.

4.1. Базовый уровень (15 вариантов)

Вар.	Конфигурация узлов	Политика безопасности	Особенность
1	talker + listener (cpp)	talker: publish chatter; listener: subscribe chatter	Базовый обмен с шифрованием
2	publisher + subscriber (python)	publish + subscribe один топик	Python-версия узлов, проверка совместимости
3	service server + client	service: reply; client: request	Безопасность сервисов в дополнение к топикам
4	два talker-узла	оба могут писать в chatter	Проверка, что оба talker успешно аутентифицируются
5	talker + listener + дополнительный узел без ключей	только два узла имеют ключи, третий пытается подписаться	Блокировка неподписанного узла
6	talker с доступом к топику /chatter, listener — к /different	разные топики в политиках	Проверка, что listener не может подписаться на /chatter
7	узлы в разных доменах DDS (domain 0 и domain 1)	политики с разными domain id	Проверка изоляции доменов
8	talker (только publish), listener (только subscribe), отдельный узел без прав	узел без прав пытается publish	Блокировка при нарушении контроля доступа
9	sensor_node (публикует /scan) + processor_node (подписывается /scan)	имитация лидара и обработчика	Практическая симуляция защищённого сенсорного потока

Вар.	Конфигурация узлов	Политика безопасности	Особенность
10	command_node (публикует /cmd_vel) + robot_node (подписывается /cmd_vel)	политика, запрещающая другим узлам управлять роботом	Демонстрация критичности контроля доступа к управлению
11	3 узла: data_source, aggregator, viewer	цепочка publish-subscribe, viewer только subscribe	Проверка сквозной безопасности через несколько узлов
12	talker с permissions с истекшим сроком действия (not_after в прошлом)	listener с валидным сертификатом	Проверка блокировки при просроченном сертификате
13	talker с несовпадающим Subject Name (CN=/fake)	CA подписывает /talker, но используется CN=/fake	Проверка аутентификации по CN
14	генерация ключей через кастомный shell-скрипт без SROS 2 CLI	ручная настройка через openssl	Демонстрация, что можно настроить без ros2 security CLI
15	использование DDS-имплементации Fast DDS вместо CycloneDDS	сравнение процесса настройки безопасности	Проверка, что механизм работает одинаково

Требования к базовому уровню:

- Узлы успешно обмениваются сообщениями с включённым шифрованием.
- Wireshark/tcpdump подтверждает, что трафик зашифрован (скриншот).
- Продемонстрирована хотя бы одна блокировка при нарушении политики.
- Ответы на контрольные вопросы $\geq 60\%$.

4.2. Средний уровень (15 вариантов)

Вар.	Задача
1	Настроить использование отдельного CA для разных роботов (собственный keystore для каждого) и проверить, что узлы из разных CA не могут общаться.
2	Реализовать сценарий: узлу разрешён доступ только на чтение определённых полей в

Вар.	Задача
	сообщении (Content Filtering).
3	Настроить ротацию ключей: сгенерировать новый Identity-сертификат, перезапустить узел без остановки системы (dynamic reload).
4	Сравнить производительность (latency, throughput) защищённого и незащищённого обмена через ros2 topic pub и ros2 topic echo.
5	Написать Python-скрипт для автоматизации генерации keystore и permissions для произвольного графа узлов, заданного в YAML.
6	Развернуть ROS 2 Discovery Server с безопасностью и показать, что discovery-информация также защищена.
7	Интегрировать SROS 2 с Docker: создать образы с pre-baked keystore и продемонстрировать безопасный обмен между контейнерами.
8	Исследовать использование Hardware Security Module (HSM) для хранения приватных ключей (эмуляция через SoftHSM).
9	Настроить политику с ограничением по времени суток (validity с точным временем) и проверить, что доступ блокируется вне разрешённого окна.
10	Реализовать multi-robot сценарий: три робота, у каждого свой namespace и свои политики доступа (например, robot1 может писать только в /robot1/cmd).
11	Настроить промежуточный проху-узел (ros2 relay), который имеет права на чтение и запись и пересылает сообщения между изолированными доменами.
12	Сравнить реализации безопасности в Fast DDS и Cyclone DDS: есть ли различия в файлах политик и поведении.
13	Провести атаку на узел без включённой проверки подлинности сертификатов (без CA verification) и показать важность проверки цепочки доверия.
14	Настроить логирование безопасности: включить audit логи, показывающие успешные и неудачные попытки доступа.

Вар.	Задача
15	Реализовать сценарий с временным доступом: узлу выдаётся сертификат на 5 минут, после чего он теряет возможность публикации.

Требования: базовый уровень + выполнение расширенного задания с демонстрацией.

4.3. Повышенный уровень (15 вариантов)

Вар.	Задача
1	Построить PKI для флота роботов: корневой CA, промежуточные CA для цехов, автоматическая выдача сертификатов узлам при регистрации.
2	Интегрировать ROS 2 Security с аппаратным Secure Element (ATECC608A) для хранения ключей. Эмулировать через скрипт.
3	Разработать сервис выдачи кратковременных удостоверений (как Vault) для динамически подключаемых узлов ROS 2 на основе OIDC/JWT.
4	Реализовать DDS Security Interoperability Test: проверить, могут ли узлы с разными DDS-вендорами общаться с шифрованием.
5	Разработать модуль для ROS 2, который автоматически переключает топики на незашифрованные при обнаружении высокой нагрузки (по аналогии с HDLC).
6	Построить систему обнаружения вторжений (IDS) для ROS 2, которая анализирует аномалии в зашифрованном трафике по метаданным (частота, размеры).
7	Провести анализ безопасности SROS 2 с использованием формальных методов (модель безопасности DDS-SEC).
8	Разработать CLI-инструмент, который аудирует все permissions.xml в проекте и находит противоречия или избыточные права.
9	Реализовать федерацию доменов: несколько ROS 2 доменов, маршрутизатор между ними с проверкой прав на межсетевом экране.
10	Интегрировать ROS 2 с DDS Security в систему управления флотом (Mission Control), где центр управления подписывает миссии и узлы проверяют подписи.

Вар.	Задача
11	Провести сравнительный анализ безопасности ROS 2 DDS Security и ROS 1 (безопасность через VPN). Написать отчёт с рекомендациями по миграции.
12	Создать конфигурацию "Zero Trust" для ROS 2: все узлы должны аутентифицироваться, весь трафик шифруется, доступ по белому списку.
13	Настроить интеграцию SROS 2 keystore с HashiCorp Vault PKI: автоматическая выдача и отзыв сертификатов для узлов.
14	Разработать дашборд Grafana, отображающий состояние безопасности ROS 2 системы (какие узлы подключены, какие топики защищены, какие алерты).
15	Написать мастер-диссертацию по безопасности ROS 2: сравнение с MAVLink, предложения по улучшению стандарта DDS-SEC для робототехники (теоретическая работа).

Требования: комплексное решение, демонстрирующее зрелый подход к безопасности коммуникаций в робототехнических системах.

5. Контрольные вопросы (ЛР №11)

1. Какие механизмы безопасности предоставляет спецификация DDS Security?
2. Что такое SROS 2 и какую роль он играет в настройке безопасности ROS 2?
3. Объясните структуру Keystore: какие компоненты там хранятся и для чего они нужны.
4. Чем отличаются Permissions.xml от Governance.xml в DDS Security?
5. Как обеспечивается аутентификация узлов в ROS 2 с использованием DDS Security?
6. Какие алгоритмы шифрования используются для защиты данных сообщений?
7. Что произойдёт, если злоумышленник попытается подключиться к топику, не имея правильных ключей?
8. Какие накладные расходы (latency, throughput) добавляет SROS 2? В каких сценариях это критично?
9. Как можно автоматизировать управление ключами для большого числа узлов?
10. Сравните безопасность MAVLink Signing и DDS Security. В чём их различия и где какой подход предпочтителен?

Лабораторная работа №10
Анализ защищённости протокола MAVLink

1. Цель

Сформировать практические навыки анализа защищённости коммуникационного протокола MAVLink путём перехвата трафика, изучения структуры пакетов и проведения симулированной атаки подмены навигационных данных с оценкой реакции автономной системы (БПЛА).

2. Задачи

- Развернуть среду симуляции PX4 SITL (Software-in-the-Loop) совместно с наземной станцией управления QGroundControl.
- Настроить перехват и анализ трафика MAVLink между полётным контроллером и GCS с использованием Wireshark / tcpdump.
- Изучить структуру пакета MAVLink v1/v2, идентифицировать поля, критичные для безопасности.
- Реализовать симуляцию атаки подмены GPS-координат (GPS Spoofing) через инъекцию пакетов MAVLink GPS_INPUT или HIL_GPS и пронаблюдать реакцию дрона (изменение позиции, режим полёта, предупреждения).
- Задokumentировать наблюдаемые индикаторы компрометации и предложить механизмы защиты (MAVLink Signing).

3. Теоретическое обоснование

Протокол MAVLink (Micro Air Vehicle Link) — легковесный бинарный протокол связи, де-факто стандарт для взаимодействия между компонентами беспилотных систем (полётный контроллер, GCS, полезная нагрузка). Текущая версия — MAVLink v2 с улучшенной структурой пакета.

Структура пакета MAVLink v2:

Байты	Поле	Описание
0	STX	Стартовый байт (0xFD для v2)
1	Payload Length	Длина полезной нагрузки (0–255)

Байты	Поле	Описание
2	Incompat Flags	Флаги несовместимости (бит 0 — подпись)
3	Compat Flags	Флаги совместимости
4	Sequence	Порядковый номер пакета
5	System ID	Идентификатор системы-отправителя
6	Component ID	Идентификатор компонента-отправителя
7–9	Message ID	Тип сообщения (24 бита)
10–..	Payload	Данные сообщения
..	Checksum (CRC16)	Контрольная сумма
..	Signature (13B)	Подпись (опционально, при incompat_flags & 0x01)

Основные уязвимости MAVLink без подписи:

- Отсутствие аутентификации: нет встроенного механизма проверки отправителя. Злоумышленник, имеющий доступ к радиоканалу, может инжектировать пакеты с любым System ID/Component ID.
- Отсутствие шифрования: вся телеметрия (координаты, состояние батареи, параметры) передаётся в открытом виде.
- Отсутствие проверки целостности: CRC16 защищает от случайных ошибок, но не от злонамеренной модификации (легко пересчитывается).
- Подверженность replay-атакам: без MAVLink Signing (или одноразовых номеров в криптографической подписи) перехваченные команды можно воспроизвести.

MAVLink Signing добавляет 13-байтовую подпись (SHA-256 + секретный ключ) к каждому пакету, обеспечивая аутентификацию и целостность. Однако требует управления ключами и увеличивает накладные расходы.

Фреймы MAVLink, используемые для GPS Spoofing:

- GPS_INPUT (#232) — позволяет внешнему источнику вводить GPS-данные в полётный контроллер.
- HIL_GPS (#113) — сообщение Hardware-in-the-Loop для симуляции GPS.

- GLOBAL_POSITION_INT (#33) — сообщение с текущей оценкой позиции (можно наблюдать эффект спуфинга).

4. Задания для индивидуального выполнения

Номер варианта соответствует последней цифре зачётки / списка группы.

4.1. Базовый уровень (15 вариантов)

Для каждого варианта указан специфический тип пакетов MAVLink, на который направлена симулированная атака.

Вар.	Атакуемый тип сообщения MAVLink	Механизм атаки	Ожидаемый эффект
1	GPS_INPUT / HIL_GPS	Инъекция поддельных координат (Цюрих → Москва)	Скачок позиции, срыв режима удержания
2	ATTITUDE_TARGET	Подмена целевых углов (roll/pitch)	Резкий наклон, возможно падение
3	SET_MODE	Принудительное переключение режима в STABILIZE/RTL	Потеря управления, уход в RTL без команды оператора
4	COMMAND_LONG (CMD=DO_SET_SERVO)	Инъекция команды сброса полезной нагрузки	Преждевременный сброс, нарушение миссии
5	COMMAND_LONG (CMD=NAV_LAND)	Принудительная посадка в заданной точке	Дрон начинает незапланированную посадку
6	PARAM_SET	Изменение параметра (например, EKF2_AID_MASK)	Внесение ошибки в алгоритмы навигации
7	MANUAL_CONTROL	Подмена сигналов стиков (roll/pitch/yaw/throttle)	Неуправляемое движение дрона
8	MISSION_ITEM	Добавление произвольной точки маршрута	Изменение плана полёта, увод в заданную точку

Вар.	Атакуемый тип сообщения MAVLink	Механизм атаки	Ожидаемый эффект
9	MISSION_CLEAR_ALL	Очистка миссии в полёте	Дрон теряет задание, переходит в режим ожидания
10	HEARTBEAT	Подмена HEARTBEAT от другого компонента (например, GCS)	Дезориентация системы, двойное управление
11	SYS_STATUS	Подмена статуса батареи (критически низкий заряд)	Аварийная посадка
12	ESTIMATOR_STATUS	Инъекция ложных флагов (ошибка GPS, ошибка IMU)	Срабатывание failsafe, переход в аварийный режим
13	RC_CHANNELS_OVERRIDE	Перехват управления через RC-канал	Полный контроль над движением дрона
14	GPS_RTCM_DATA	Инъекция ложных RTCM-поправок (DGNS)	Искажение GPS-решения
15	COMMAND_LONG (CMD=DO_DIGICAM_CONTROL)	Несанкционированное фотографирование	Нарушение конфиденциальности миссии

Требования к базовому уровню:

- Развёрнута связка PX4 SITL + QGroundControl.
- Выполнен захват трафика Wireshark (скриншот с разбором пакета MAVLink).
- Реализована одна симулированная атака согласно варианту с демонстрацией эффекта (скриншот до/после, лог).
- Предложены меры противодействия (MAVLink Signing, rate limiting, и т.д.).
- Ответы на контрольные вопросы $\geq 60\%$.

4.2. Средний уровень (15 вариантов)

Вар.	Задача
------	--------

Вар.	Задача
1	Включить MAVLink Signing в симуляции (PX4 SITL), продемонстрировать, что инъекция неподписанных пакетов отвергается.
2	Реализовать Replay Attack: записать легитимные пакеты управления и воспроизвести их, показав, что дрон выполняет команды повторно (без signing).
3	Написать сниффер на Python (Scapy/pymavlink), который детектирует аномалии в потоке MAVLink (например, резкие скачки координат) и выдаёт алерт.
4	Сравнить задержки и накладные расходы при включении/выключении MAVLink Signing (ping, время цикла).
5	Провести атаку Man-in-the-Middle (MITM), модифицирующую координаты в реальном времени (прозрачный прокси с mavproxy).
6	Реализовать атаку GPS Spoofing с плавным уводом координат (имитация "угона"), оценить, как EKF реагирует на медленный drift по сравнению с резким скачком.
7	Интегрировать в симуляцию инструмент mavlink-warden и показать, как он может блокировать вредоносные пакеты на уровне маршрутизатора MAVLink.
8	Провести анализ трафика MAVLink между реальным полётным контроллером (если доступен) и GCS, сравнить с SITL.
9	Разработать скрипт автоматического мониторинга и оповещения (Webhook/Slack) при обнаружении инъекции пакетов MAVLink в сеть.
10	Исследовать защищённость QGroundControl: можно ли подменить IP-адрес GCS и перехватить видеопоток?
11	Настроить протоколирование всех команд MAVLink с временными метками и подписью для последующего аудита.
12	Проверить, как PX4 обрабатывает некорректные CRC пакетов: генерирует пакеты с неправильной контрольной суммой и анализирует логи.
13	Использовать SDR (HackRF/имитатор) для передачи поддельных пакетов MAVLink по радиоканалу (если оборудование доступно).

Вар.	Задача
14	Настроить MAVLink Router (или MAVP2P) и показать, что неподписанные пакеты блокируются на уровне маршрутизатора.
15	Интегрировать MAVLink Signing с CI/CD: автоматически проверять, что прошивка поддерживает signing и ключи корректны.

Требования: базовый уровень + выполнение расширенного задания с демонстрацией.

4.3. Повышенный уровень (15 вариантов)

Вар.	Задача
1	Реализовать полноценный MAVLink IDS (Intrusion Detection System) на основе анализа последовательности пакетов и машинного обучения.
2	Разработать систему управления ключами MAVLink для флота дронов: автоматическая ротация, распределение pre-shared keys.
3	Интегрировать MAVLink с Transport Layer Security (TLS) через прокси (mavlink-over-tcp), сравнить с нативным MAVLink Signing.
4	Создать "red team / blue team" сценарий: одна команда атакует дрона (GPS Spoof, Command Injection), другая защищает с помощью MAVLink Signing и IDS.
5	Провести фаззинг MAVLink-парсера PX4 (используя AFL / libFuzzer), найти потенциальные уязвимости в обработке некорректных пакетов.
6	Разработать кастомный MAVLink-сообщение с шифрованием полезной нагрузки (AES-GCM) и интегрировать в PX4/QGC.
7	Построить honeypot-ловушку для БПЛА: эмулировать MAVLink-интерфейс, привлекающий атакующих, и логировать их действия.
8	Исследовать безопасность MAVLink v2 с подписью против атаки с понижением версии (downgrade to v1).
9	Разработать модуль для PX4, реализующий challenge-response аутентификацию для критических команд (например, ARM, LAND).

Вар.	Задача
10	Сравнить безопасность MAVLink и ROS 2 DDS в контексте управления БПЛА: провести аналогичную атаку на обе системы, сравнить результаты.
11	Разработать расширение для Wireshark (диссектор), которое автоматически подсвечивает подозрительные паттерны MAVLink (скачки координат, необычные команды).
12	Реализовать протокол согласования ключей (ECDH) поверх MAVLink для динамического установления сессионного ключа подписи.
13	Провести аудит безопасности полного стека управления дроном (PX4 + MAVSDK + GCS) с автоматизированным инструментом и задокументировать все найденные уязвимости.
14	Разработать микроядерный прокси безопасности, который фильтрует MAVLink-трафик на уровне бизнес-логики (белый список команд, rate limit, гео-фенсинг).
15	Построить систему доверенной загрузки и аттестации для полётного контроллера, интегрированную с MAVLink (передача evidence о целостности прошивки в GCS).

Требования: демонстрация работы комплексной защиты, отчёт с архитектурой, код/конфигурации.

5. Контрольные вопросы

1. Опишите структуру пакета MAVLink v2. Какие поля влияют на безопасность?
2. В чём разница между MAVLink v1 и v2 с точки зрения безопасности?
3. Какие основные уязвимости протокола MAVLink при отсутствии подписи пакетов?
4. Что такое MAVLink Signing и как он защищает от спуфинга и подмены данных?
5. Почему CRC16 недостаточно для обеспечения целостности пакетов?
6. Какие типы сообщений MAVLink наиболее опасны при компрометации радиоканала?
7. Как фильтр Калмана (EKF) в PX4 может частично противостоять GPS Spoofing?
8. Какие ещё протоколы, кроме MAVLink, используются в БПЛА и требуют защиты (видео, управление полезной нагрузкой)?
9. Как можно организовать мониторинг и обнаружение атак на MAVLink?
10. Какие ограничения MAVLink Signing (производительность, управление ключами) важно учитывать при развёртывании на реальных БПЛА?

Лабораторная работа №11

Настройка безопасности в ROS 2 (DDS Security)

1. Цель

Сформировать практические навыки защиты публикационно-подписочного обмена сообщениями в робототехнической операционной системе ROS 2 путём настройки встроенных механизмов безопасности DDS (Data Distribution Service): аутентификации, шифрования и контроля доступа на основе SROS 2.

2. Задачи

- Развернуть среду ROS 2 (Humble/Iron/Rolling) с двумя узлами, обменивающимися сообщениями через топик (например, chatter / listener из demo_nodes_cpp).
- Настроить рабочее пространство безопасности (Security Enclaves) с помощью утилит SROS 2 (ros2 security).
- Сгенерировать пары ключей и сертификатов для необходимых компонентов (CA, Identity, Permissions).
- Сконфигурировать XML-политики управления доступом (Permissions), определяющие разрешённые действия для каждого узла.
- Запустить узлы с включёнными DDS-защитами (AUTH, ENCRYPT, ACCESS_CONTROL) и убедиться в успешном безопасном обмене.
- Провести анализ защищённого трафика с помощью Wireshark для подтверждения шифрования.

3. Теоретическое обоснование

ROS 2 (Robot Operating System 2) использует стандарт DDS (Data Distribution Service) в качестве связующего ПО (middleware) для коммуникации между узлами. В отличие от ROS 1, где безопасность практически отсутствовала, ROS 2 предоставляет встроенные механизмы безопасности, основанные на спецификации DDS Security (DDS-SEC), и инструменты для удобной настройки — SROS 2.

DDS Security включает следующие механизмы:

- Authentication (Аутентификация): Проверка подлинности участников домена DDS (DomainParticipants) с использованием PKI (X.509 сертификатов) и протокола

DDS:Auth:PKI-DH. Взаимная аутентификация на основе challenge-response с использованием закрытых ключей.

- Access Control (Контроль доступа): Проверка прав участника на выполнение определённых операций (создание топиков, чтение/запись) на основе подписанных XML-файлов разрешений (Permissions), выданных центром сертификации (CA). Файлы включают разрешённые действия (allow rule) с указанием топика, домена, допустимых операций.
- Cryptography (Криптография): Прозрачное шифрование полезной нагрузки сообщений (AES-GCM или AES-CMAC) с использованием сессионных ключей, получаемых во время аутентификации. Защита обеспечивается на уровне DDS без изменения кода узлов.

SROS 2 — набор инструментов командной строки, упрощающих создание и управление инфраструктурой безопасности:

```
ros2 security create_keystore <keystore_dir>    # создание хранилища ключей
```

```
ros2 security create_key <keystore_dir> <identity> # генерация ключей для конкретного identity
```

```
ros2 security create_permission <keystore_dir> <identity> <policy_xml> # настройка прав
```

Структура Keystore:

keystore/

public/ # сертификаты CA

private/ # приватный ключ CA

enclaves/

/talker/ # identity узла

cert.pem, permissions.p7s, key.pem, permissions.xml

/listener/

...

В переменной окружения ROS_SECURITY_ENCLAVE_OVERRIDE=/path/to/enclaves/talker или через --ros-args --enclave /talker указывается путь к папке identity.

Принцип работы:

1. CA создаёт корневой сертификат и ключи.
 2. Для каждого узла генерируются Identity-сертификат и ключевая пара, подписанные CA.
 3. Создаются Permissions.xml и подписываются CA (файл .p7s).
 4. При запуске узлы загружают свои identity, обмениваются сертификатами, проверяют подлинность CA, и устанавливают зашифрованный обмен, проверяя права доступа.
4. Задания для индивидуального выполнения

Номер варианта соответствует последней цифре зачётки / списка группы.

4.1. Базовый уровень (15 вариантов)

Вар.	Конфигурация узлов	Политика безопасности	Особенность
1	talker + listener (cpp)	talker: publish chatter; listener: subscribe chatter	Базовый обмен с шифрованием
2	publisher + subscriber (python)	publish + subscribe один топик	Python-версия узлов, проверка совместимости
3	service server + client	service: reply; client: request	Безопасность сервисов в дополнение к топикам
4	два talker-узла	оба могут писать в chatter	Проверка, что оба talker успешно аутентифицируются
5	talker + listener + дополнительный узел без ключей	только два узла имеют ключи, третий пытается подписаться	Блокировка неподписанного узла
6	talker с доступом к топику /chatter, listener — к /different	разные топики в политиках	Проверка, что listener не может подписаться на /chatter
7	узлы в разных доменах DDS (domain 0 и domain 1)	политики с разными domain id	Проверка изоляции доменов
8	talker (только publish), listener (только subscribe), отдельный узел без прав	узел без прав пытается publish	Блокировка при нарушении контроля доступа
9	sensor_node (публикует /scan) + processor_node (подписывается /scan)	имитация лидара и обработчика	Практическая симуляция защищённого сенсорного потока
10	command_node (публикует /cmd_vel) + robot_node (подписывается /cmd_vel)	политика, запрещающая другим узлам управлять роботом	Демонстрация критичности контроля доступа к управлению
11	3 узла: data_source, aggregator, viewer	цепочка publish-subscribe, viewer только	Проверка сквозной безопасности через

Вар.	Конфигурация узлов	Политика безопасности	Особенность
		subscribe	несколько узлов
12	talker с permissions с истекшим сроком действия (not_after в прошлом)	listener с валидным сертификатом	Проверка блокировки при просроченном сертификате
13	talker с несовпадающим Subject Name (CN=/fake)	CA подписывает /talker, но используется CN=/fake	Проверка аутентификации по CN
14	генерация ключей через кастомный shell-скрипт без SROS 2 CLI	ручная настройка через openssl	Демонстрация, что можно настроить без ros2 security CLI
15	использование DDS-имплементации Fast DDS вместо CycloneDDS	сравнение процесса настройки безопасности	Проверка, что механизм работает одинаково

Требования к базовому уровню:

- Узлы успешно обмениваются сообщениями с включённым шифрованием.
- Wireshark/tcpdump подтверждает, что трафик зашифрован (скриншот).
- Продемонстрирована хотя бы одна блокировка при нарушении политики.
- Ответы на контрольные вопросы $\geq 60\%$.

4.2. Средний уровень (15 вариантов)

Вар.	Задача
1	Настроить использование отдельного CA для разных роботов (собственный keystore для каждого) и проверить, что узлы из разных CA не могут общаться.
2	Реализовать сценарий: узлу разрешён доступ только на чтение определённых полей в сообщении (Content Filtering).
3	Настроить ротацию ключей: сгенерировать новый Identity-сертификат, перезапустить узел без остановки системы (dynamic reload).
4	Сравнить производительность (latency, throughput) защищённого и незащищённого обмена

Вар.	Задача
	через <code>ros2 topic pub</code> и <code>ros2 topic echo</code> .
5	Написать Python-скрипт для автоматизации генерации keystore и permissions для произвольного графа узлов, заданного в YAML.
6	Развернуть ROS 2 Discovery Server с безопасностью и показать, что discovery-информация также защищена.
7	Интегрировать SROS 2 с Docker: создать образы с pre-baked keystore и продемонстрировать безопасный обмен между контейнерами.
8	Исследовать использование Hardware Security Module (HSM) для хранения приватных ключей (эмуляция через SoftHSM).
9	Настроить политику с ограничением по времени суток (validity с точным временем) и проверить, что доступ блокируется вне разрешённого окна.
10	Реализовать multi-robot сценарий: три робота, у каждого свой namespace и свои политики доступа (например, robot1 может писать только в /robot1/cmd).
11	Настроить промежуточный проху-узел (ros2 relay), который имеет права на чтение и запись и пересылает сообщения между изолированными доменами.
12	Сравнить реализации безопасности в Fast DDS и Cyclone DDS: есть ли различия в файлах политик и поведении.
13	Провести атаку на узел без включённой проверки подлинности сертификатов (без CA verification) и показать важность проверки цепочки доверия.
14	Настроить логирование безопасности: включить audit логи, показывающие успешные и неудачные попытки доступа.
15	Реализовать сценарий с временным доступом: узлу выдаётся сертификат на 5 минут, после чего он теряет возможность публикации.

Требования: базовый уровень + выполнение расширенного задания с демонстрацией.

4.3. Повышенный уровень (15 вариантов)

Вар.	Задача
1	Построить PKI для флота роботов: корневой CA, промежуточные CA для цехов, автоматическая выдача сертификатов узлам при регистрации.
2	Интегрировать ROS 2 Security с аппаратным Secure Element (ATECC608A) для хранения ключей. Эмулировать через скрипт.
3	Разработать сервис выдачи кратковременных удостоверений (как Vault) для динамически подключаемых узлов ROS 2 на основе OIDC/JWT.
4	Реализовать DDS Security Interoperability Test: проверить, могут ли узлы с разными DDS-вендорами общаться с шифрованием.
5	Разработать модуль для ROS 2, который автоматически переключает топики на незашифрованные при обнаружении высокой нагрузки (по аналогии с HDLC).
6	Построить систему обнаружения вторжений (IDS) для ROS 2, которая анализирует аномалии в зашифрованном трафике по метаданным (частота, размеры).
7	Провести анализ безопасности SROS 2 с использованием формальных методов (модель безопасности DDS-SEC).
8	Разработать CLI-инструмент, который аудирует все permissions.xml в проекте и находит противоречия или избыточные права.
9	Реализовать федерацию доменов: несколько ROS 2 доменов, маршрутизатор между ними с проверкой прав на межсетевом экране.
10	Интегрировать ROS 2 с DDS Security в систему управления флотом (Mission Control), где центр управления подписывает миссии и узлы проверяют подписи.
11	Провести сравнительный анализ безопасности ROS 2 DDS Security и ROS 1 (безопасность через VPN). Написать отчёт с рекомендациями по миграции.
12	Создать конфигурацию "Zero Trust" для ROS 2: все узлы должны аутентифицироваться, весь трафик шифруется, доступ по белому списку.
13	Настроить интеграцию SROS 2 keystore с HashiCorp Vault PKI: автоматическая выдача и отзыв сертификатов для узлов.

Вар.	Задача
14	Разработать дашборд Grafana, отображающий состояние безопасности ROS 2 системы (какие узлы подключены, какие топики защищены, какие алерты).
15	Написать мастер-диссертацию по безопасности ROS 2: сравнение с MAVLink, предложения по улучшению стандарта DDS-SEC для робототехники (теоретическая работа).

Требования: комплексное решение, демонстрирующее зрелый подход к безопасности коммуникаций в робототехнических системах.

5. Контрольные вопросы (ЛР №11)

1. Какие механизмы безопасности предоставляет спецификация DDS Security?
2. Что такое SROS 2 и какую роль он играет в настройке безопасности ROS 2?
3. Объясните структуру Keystore: какие компоненты там хранятся и для чего они нужны.
4. Чем отличаются Permissions.xml от Governance.xml в DDS Security?
5. Как обеспечивается аутентификация узлов в ROS 2 с использованием DDS Security?
6. Какие алгоритмы шифрования используются для защиты данных сообщений?
7. Что произойдёт, если злоумышленник попытается подключиться к топику, не имея правильных ключей?
8. Какие накладные расходы (latency, throughput) добавляет SROS 2? В каких сценариях это критично?
9. Как можно автоматизировать управление ключами для большого числа узлов?
10. Сравните безопасность MAVLink Signing и DDS Security. В чём их различия и где какой подход предпочтителен?

7. Требования к отчётам

Отчёты для ЛР №10 и ЛР №11 оформляются как отдельные документы (PDF), каждый содержит:

- Титульный лист.
- Цель и задачи.
- Описание стенда (симулятор / ROS 2 окружение, версии, сетевые настройки).
- Подробный ход выполнения с листингами команд, конфигураций и скриншотами.
- Для ЛР №10: разбор пакета MAVLink (скриншоты Wireshark), результат атаки подмены (скриншоты симулятора), анализ реакции системы, предложенные контрмеры.
- Для ЛР №11: структура keystore, фрагмент permissions.xml, скриншоты запуска узлов с безопасностью и без, сравнение трафика в Wireshark, демонстрация блокировки.
- Индивидуальная часть (средний/повышенный уровень): описание задания, реализация, доказательство работоспособности.

- Ответы на контрольные вопросы.
- Выводы.

7. Критерии оценивания

7.1. «Удовлетворительно» (базовый уровень):

- Лабораторный стенд развёрнут, базовая атака (ЛР №10) или шифрование (ЛР №11) выполнены и задокументированы.
- Отчёт содержит все обязательные разделы и скриншоты.
- Ответы на контрольные вопросы $\geq 60\%$.

7.2. «Хорошо» (средний уровень):

- Все требования базового уровня.
- Реализовано расширенное задание (MAVLink Signing, Replay Attack, сравнение производительности, автоматизация keystore и т.п.).
- Ответы на контрольные вопросы $\geq 80\%$.

7.3. «Отлично» (повышенный уровень):

- Все требования среднего уровня.
- Внедрена комплексная защита или разработаны инструменты / проведено исследование.
- Продемонстрировано глубокое понимание протоколов безопасности коммуникаций автономных систем.
- Ответы на контрольные вопросы полные и развёрнутые.

7.4. Понижение оценки:

- Отсутствие скриншотов трафика (до/после атаки или шифрования) — доработка.
- Непродемонстрирована блокировка (несанкционированный пакет/узел не отвергнут) — не выше базового уровня.
- Слабые ответы на контрольные вопросы — направляется на пересдачу.

Комплекс лабораторных работ №10 и №11 формирует ключевые компетенции ПК-11 в части защиты каналов связи и настройки безопасности в автономных системах (БПЛА и робототехника), что является фундаментом для выполнения итоговой работы по комплексной защите автономной инженерной системы.

Лабораторная работа №12

Безопасное OTA-обновление для БПЛА

1. Цель

Сформировать практические навыки организации защищённого конвейера доставки обновлений прошивки (Over-the-Air, OTA) для автономных систем (БПЛА) с обеспечением криптографической целостности, аутентичности источника и защитой от downgrade-атак.

2. Задачи

- Создать имитацию процесса сборки прошивки (бинарный артефакт).
- Сгенерировать ключевую пару (GPG или Cosign) для подписи обновлений.
- Подписать сборочный артефакт и сформировать манифест обновления (версия, хеш, подпись, целевое устройство).
- Разработать скрипт симулятора бортового клиента OTA, который загружает обновление, проверяет подпись и устанавливает прошивку только при успешной верификации.
- Продемонстрировать блокировку установки при несовпадении подписи, повреждении файла или попытке отката на старую версию.

3. Теоретическое обоснование

OTA-обновление (Over-the-Air) — критически важный компонент безопасности жизненного цикла автономных систем. Через канал OTA злоумышленник может внедрить вредоносную прошивку, что приведёт к полной компрометации устройства. Безопасное OTA должно обеспечивать:

- Целостность (Integrity): Прошивка не была изменена при передаче или хранении.
- Аутентичность источника (Authenticity): Прошивка исходит от доверенного разработчика.
- Свежесть (Freshness): Защита от rollback-атак (установка устаревшей уязвимой версии).
- Конфиденциальность (опционально): Защита прошивки от обратного инжиниринга при передаче.

Архитектура типового безопасного OTA:

1. Сервер обновлений: Хранит подписанные прошивки, манифесты и метаданные (TUF/Notary).
2. Манифест обновления: Содержит версию, хеш (SHA-256), подпись, список поддерживаемых устройств, номер последовательности (sequence number) для защиты от rollback.
3. Канал связи: Может быть защищён TLS, но критически важна end-to-end подпись — даже при компрометации сервера или MITM на канале прошивка не пройдёт проверку.

4. Бортвой клиент: Загружает манифест, проверяет подпись с помощью предустановленного публичного ключа, сверяет хеш, проверяет версию (не ниже текущей), и только затем устанавливает.

Методы подписи:

- GPG (GNU Privacy Guard): Классическая асимметричная подпись. Приватный ключ хранится у разработчика, публичный — вшит в устройство.
- Cosign (Sigstore): Подпись контейнеров и бинарных артефактов. Может использовать keyless-механизм через OIDC.
- Ed25519 (через OpenSSL/libsodium): Современный, быстрый алгоритм подписи, популярный в embedded-системах.

The Update Framework (TUF) — фреймворк для построения безопасных систем обновлений, защищающий от компрометации сервера обновлений, replay-атак и других угроз. Используется во многих проектах (Uptane для automotive, Docker Content Trust).

Стандарт Uptane — адаптация TUF для автомобильной индустрии, применим и к БПЛА. Определяет роли (Director, Image Repository), двойную подпись и защиту даже при частичной компрометации инфраструктуры.

4. Задания для индивидуального выполнения

Номер варианта соответствует последней цифре зачётки / списка группы.

4.1. Базовый уровень (15 вариантов)

Вар.	Метод подписи	Формат манифеста	Защита от rollback	Особенность
1	GPG (Ed25519)	JSON	Номер последовательности	Базовая реализация с detached signature
2	GPG (RSA-2048)	JSON	Номер последовательности	RSA-ключ большего размера
3	Cosign (ключ)	JSON	Номер последовательности	Подпись через cosign sign-blob
4	OpenSSL (Ed25519)	JSON	Номер последовательности	Использование openssl pkeyutl для подписи
5	Minisign	JSON	Номер последовательности	Использование утилиты minisign

Вар.	Метод подписи	Формат манифеста	Защита от rollback	Особенность
6	GPG (Ed25519)	YAML	Номер последовательности	Манифест в YAML вместо JSON
7	GPG (Ed25519)	JSON	Временная метка (timestamp)	Защита от rollback через проверку свежести по времени
8	Cosign (keyless)	JSON	Номер последовательности	Keyless-подпись с эфемерным ключом
9	GPG (Ed25519)	JSON	Комбинированная (seq + time)	Двойная защита: и номер, и временная метка
10	GPG (Ed25519)	CBOR/Protobuf	Номер последовательности	Бинарный манифест для экономии трафика
11	Ed25519 (PyNaCl)	JSON	Номер последовательности	Чисто Python-реализация подписи
12	GPG (Ed25519)	JSON	Хеш-цепочка версий	Каждая версия ссылается на хеш предыдущей
13	GPG (Ed25519)	JSON + JWS	Номер последовательности	Манифест как JSON Web Signature
14	GPG (Ed25519)	JSON	Номер последовательности	Множественные подписи (dev + security team)
15	GPG (Ed25519)	JSON	Номер последовательности	Проверка через встроенный публичный ключ в самом клиенте (без файла)

Требования к базовому уровню:

- Сгенерированы ключи, подписан бинарный артефакт.
- Создан манифест обновления с метаданными и подписью.
- Реализован клиентский скрипт, выполняющий проверку подписи, хеша и версии.

- Продемонстрирована успешная установка и блокировка при нарушении (подделка, rollback, повреждение).
- Ответы на контрольные вопросы $\geq 60\%$.

4.2. Средний уровень (15 вариантов)

Вар.	Задача
1	Реализовать OTA-сервер на Flask/FastAPI, который отдаёт манифест и прошивку по HTTPS. Клиент подключается через TLS и выполняет загрузку.
2	Интегрировать TUF (The Update Framework) на стороне сервера и клиента с использованием python-tuf.
3	Добавить шифрование прошивки (AES-GCM) с обёртыванием ключа (key wrapping) публичным ключом дрона.
4	Реализовать дифференциальные обновления (delta updates) с помощью bsdiff / xdelta3. Подписать дельту.
5	Настроить Uptane-подобную систему с ролями Director и Image Repository (имитация).
6	Сравнить накладные расходы (размер подписи, время верификации) для Ed25519, RSA-2048 и ECDSA на целевом устройстве (RPi/эмуляторе).
7	Использовать аппаратный Secure Element (эмуляция SoftHSM) для хранения ключей подписи на сервере.
8	Реализовать OTA через MQTT с подписанными сообщениями (подпись в payload).
9	Добавить механизм "A/B slot" с fallback: клиент проверяет новую прошивку в неактивном слоте, и только при успехе переключается.
10	Разработать скрипт автоматической инвалидации уязвимых версий: при обнаружении CVE сервер помечает версию как "запрещённую", клиент отказывается от неё.
11	Настроить Cosign keyless с записью в Rekor transparency log, клиент проверяет наличие записи в Rekor.
12	Реализовать OTA клиент как Docker-контейнер, который монтирует том с текущей прошивкой и выполняет обновление.

Вар.	Задача
13	Разработать CI/CD-пайплайн (GitLab CI), который автоматически собирает, подписывает и публикует новую прошивку при коммите в main.
14	Интегрировать проверку SBOM прошивки перед установкой: клиент получает SBOM и сверяет с Dependency-Track API.
15	Сравнить безопасность OTA с GPG и с MAVLink Signing: какой подход и в каком сценарии предпочтительнее.

Требования: базовый уровень + реализация расширенного задания с демонстрацией.

4.3. Повышенный уровень (15 вариантов)

Вар.	Задача
1	Реализовать полноценную Uptane-совместимую OTA-систему для PX4 (эмуляция ролей, двойная подпись, защита от компрометации сервера).
2	Построить OTA-сервер с проверкой целостности через Merkle Tree: клиент получает корень дерева и может верифицировать отдельные блоки прошивки (streaming verification).
3	Разработать систему поэтапного развёртывания (canary OTA): прошивка сначала устанавливается на 1% устройств, и только при успехе — на остальные.
4	Интегрировать Secure Boot + OTA: прошивка подписывается ключом, который проверяется U-Boot/GRUB на устройстве (эмуляция через QEMU).
5	Реализовать систему отзыва ключей (key revocation): если ключ скомпрометирован, сервер публикует CRL/список отозванных ключей, клиент проверяет.
6	Разработать OTA с формальной верификацией: клиент на основе TLA+/Alloy модели проверяет, что обновление не нарушает инварианты безопасности.
7	Создать дашборд мониторинга OTA (Grafana): статистика установки, версии, ошибки верификации, алерты на аномалии.
8	Интегрировать OTA с блокчейном (Hyperledger/ Ethereum) для неизменяемого аудит-лога всех обновлений в флоте.

Вар.	Задача
9	Разработать peer-to-peer OTA между дронами в рое (swarm): дроны обмениваются прошивкой через mesh-сеть с проверкой подписей.
10	Реализовать квантово-устойчивую подпись (SPHINCS+ / Dilithium) для OTA и сравнить производительность с классическими алгоритмами.
11	Провести threat modeling (STRIDE) для всей OTA-инфраструктуры и реализовать контрмеры для каждого найденного вектора.
12	Разработать аппаратный OTA-клиент на базе ESP32/Arduino (или эмуляцию), который выполняет проверку подписи на микроконтроллере с ограниченными ресурсами.
13	Построить систему с гарантированной доставкой обновлений через Satellite Iridium/SBD с адаптивной компрессией и подписью.
14	Провести red team exercise: одна команда атакует OTA-инфраструктуру, другая защищает. Задokumentировать результаты.
15	Разработать открытый стандарт безопасного OTA для БПЛА (проект спецификации) на основе Uptane, TUF и NIST SP 800-53, с референсной реализацией.

Требования: комплексная система с глубокой проработкой, архитектурные схемы, отчёт с анализом угроз и контрмер.

5. Контрольные вопросы (ЛР №12)

1. Какие угрозы безопасности существуют при доставке OTA-обновлений для БПЛА?
2. Чем отличается проверка целостности (хеш) от проверки аутентичности (подпись)?
3. Что такое rollback-атака и как от неё защититься?
4. Опишите архитектуру типовой безопасной OTA-системы. Какие компоненты за что отвечают?
5. В чём преимущества Ed25519 перед RSA для подписи прошивок?
6. Что такое TUF (The Update Framework) и какие проблемы он решает?
7. Почему недостаточно полагаться только на TLS для защиты OTA?
8. Что такое A/B слоты и как они повышают надёжность обновлений?
9. Как организовать отзыв скомпрометированного ключа подписи в системе OTA?
10. Каковы особенности OTA-обновлений для роя дронов по сравнению с одиночным устройством?

Лабораторная работа №13

Безопасность наземной станции управления (GCS)

1. Цель

Сформировать практические навыки анализа защищённости наземной станции управления (Ground Control Station, GCS) беспилотных систем, выявления типовых уязвимостей в конфигурации и хранении секретных данных, а также настройки защищённого канала связи между GCS и БПЛА с применением TLS.

2. Задачи

- Провести аудит конфигурации QGroundControl (или аналога) на предмет безопасных настроек.
- Проанализировать механизм хранения паролей, токенов и ключей GCS в локальной файловой системе.
- Симулировать атаку на GCS: подмена сертификата, перехват сохранённых учётных данных.
- Настроить защищённое TLS-соединение между GCS и симулятором дрона.
- Разработать рекомендации по усилению безопасности GCS (hardening guide).

3. Теоретическое обоснование

Наземная станция управления (GCS) — программно-аппаратный комплекс, обеспечивающий интерфейс оператора с беспилотной системой. Компрометация GCS может привести к:

- Перехвату управления дроном (в том числе флотом).
- Раскрытию маршрутов, телеметрии, видео.
- Краже учётных данных облачных сервисов.
- Внедрению вредоносных миссий.

Типовые уязвимости GCS:

- Хранение секретов в открытом виде: Пароли для Wi-Fi, ключи API облачных сервисов, MAVLink signing keys часто сохраняются в конфигурационных файлах без шифрования.
- Отсутствие проверки сертификатов: Некоторые GCS по умолчанию принимают любые TLS-сертификаты (или игнорируют ошибки сертификата), что делает возможной MITM-атаку.
- Небезопасные настройки по умолчанию: Открытые порты, автоматическое подключение к любым сетям, lack of host firewall.
- Уязвимости в зависимостях: GCS — это сложное ПО (Qt, C++, множество библиотек), содержащее потенциальные CVE.

- Отсутствие разделения привилегий: GCS обычно запускается с правами пользователя, имеющего доступ ко всем его данным.

Стандартный стек защиты GCS:

- Шифрование хранилища секретов (OS keychain, KWallet, DPAPI).
- Настройка TLS с проверкой сертификатов (взаимный TLS при возможности).
- Применение принципа наименьших привилегий (запуск GCS в sandbox, отдельный пользователь).
- Брандмауэр хоста, ограничивающий входящие соединения.
- Обновление ПО и зависимостей.
- Безопасная конфигурация: отключение ненужных протоколов, настройка строгих cipher suites для TLS.

QGroundControl (QGC) — одна из самых распространённых GCS с открытым исходным кодом. Хранит настройки в JSON/SQLite, использует Qt, поддерживает MAVLink и видео-поток. Типовой объект для аудита в лабораторной работе.

Альтернативы: Mission Planner, APM Planner, UgCS, proprietary GCS.

4. Задания для индивидуального выполнения

Номер варианта соответствует последней цифре зачётки / списка группы.

4.1. Базовый уровень (15 вариантов)

Вар.	Целевая GCS	Фокус аудита	Симулируемая атака / настройка
1	QGroundControl	Хранение паролей Wi-Fi и ключей MAVLink	Поиск секретов в QGroundControl.ini
2	QGroundControl	Настройки TLS/SSL для видео-потока	Подмена сертификата через mitmproxy, перехват видео
3	Mission Planner	Файлы конфигурации (Windows)	Поиск токенов облачных сервисов (Azure, AWS)
4	QGroundControl	Файл базы данных SQLite (telemetry logs)	Извлечение телеметрии из логов, анализ конфиденциальных данных
5	APM Planner	Хранение настроек соединений (COM порты)	Обнаружение открытых паролей в конфигах
6	QGroundControl	Кэш карт (Offline Maps)	Извлечение координат баз, постоянных точек взлёта

Вар.	Целевая GCS	Фокус аудита	Симулируемая атака / настройка
7	QGroundControl	Права доступа к файлам конфигурации	Проверка, может ли другой пользователь ОС прочитать конфиги
8	QGroundControl	Настройка mTLS для MAVLink Bridge	Генерация сертификатов, настройка двустороннего TLS
9	MAVSDK Server	Безопасность gRPC-интерфейса	Настройка TLS для gRPC, проверка сертификатов клиентом
10	QGroundControl	Экспорт миссий (plan files)	Анализ .plan файлов на наличие чувствительных данных
11	QGroundControl	Настройки брандмауэра хоста	Сканирование портов (nmap), закрытие лишних
12	QGroundControl	Обновление ПО и зависимостей	Проверка версий библиотек Qt, OpenSSL на CVE
13	UgCS (демо)	Безопасность REST API (если доступен)	Анализ токенов, настройка HTTPS
14	QGroundControl	Симуляция физического доступа к компьютеру с GCS	Извлечение секретов при доступе к незаблокированному экрану
15	QGroundControl	Анализ сетевого трафика GCS (Wireshark)	Идентификация протоколов, передающих данные в открытом виде

Требования к базовому уровню:

- Проведён аудит конфигурации GCS, найдены хотя бы 3 проблемы безопасности.
- Выполнена симуляция одной атаки (подмена сертификата / извлечение секретов).
- Настроено защищённое соединение (TLS) или предложена конкретная мера защиты.
- Разработан краткий Hardening Guide (не менее 5 рекомендаций).
- Ответы на контрольные вопросы $\geq 60\%$.

4.2. Средний уровень (15 вариантов)

Вар.	Задача
------	--------

Вар.	Задача
1	Настроить взаимный TLS (mTLS) между QGC и MAVLink-прокси. Клиент также должен предъявить сертификат.
2	Настроить хранение ключей MAVLink Signing в OS Keychain (GNOME Keyring / KDE Wallet / Windows Credential Manager) вместо открытого файла.
3	Разработать скрипт автоматического аудита GCS (Python/Bash), который проверяет наличие секретов в открытом виде, права доступа, версии библиотек.
4	Провести аудит безопасности кода QGC (статический анализ, хотя бы ручной) на предмет использования exes, system, путей к файлам.
5	Настроить VPN (WireGuard) между GCS и дроном как дополнительный слой защиты, сохраняя MAVLink без изменений.
6	Реализовать систему оповещения (алерт) при обнаружении подключения к GCS с нового IP или при необычной активности.
7	Провести аудит видео-подсистемы QGC (RTSP/H.264), проверить, шифруется ли видео, можно ли его перехватить.
8	Интегрировать QGC с аппаратным токеном (YubiKey) для аутентификации оператора перед взлётом.
9	Настроить AppArmor/SELinux профиль для QGroundControl, ограничивающий доступ к файловой системе и сети.
10	Сравнить безопасность QGC, Mission Planner и MAVSDK Server по одинаковым критериям. Составить сводную таблицу.
11	Разработать политику паролей и секретов для операторов GCS: минимальная длина, ротация, запрет повторного использования.
12	Настроить централизованное логирование (syslog/ELK) событий безопасности GCS: запуск, подключения, ошибки сертификатов.
13	Реализовать проверку целостности исполняемых файлов GCS (через хеш-суммы / подпись) перед каждым запуском.

Вар.	Задача
14	Проанализировать уязвимости в зависимостях QGC с помощью OWASP Dependency-Check или Trivy (файловая система).
15	Разработать процедуру безопасного восстановления GCS после инцидента (Incident Response Playbook).

Требования: базовый уровень + выполнение расширенного задания.

4.3. Повышенный уровень (15 вариантов)

Вар.	Задача
1	Разработать изолированную среду для GCS на базе Qubes OS или специализированного Live-USB с Linux. Полный аудит безопасности такого решения.
2	Реализовать систему обнаружения компрометации GCS на основе анализа поведения (аномалии в телеметрии, необычные команды).
3	Построить инфраструктуру PKI для флота: CA, автоматическая выдача сертификатов GCS при регистрации, отзыв при компрометации.
4	Разработать протокол аутентификации оператора, устойчивый к shoulder surfing (графический пароль, комбинация с аппаратным токеном).
5	Провести полный пентест GCS по методологии OWASP/PTES с отчётом, включая фаззинг сетевых протоколов и анализ памяти.
6	Реализовать систему управления доступом (RBAC) в GCS: разные операторы имеют разные права (только просмотр телеметрии, управление, администрирование).
7	Создать кастомную сборку QGroundControl с усиленной безопасностью: удаление ненужных модулей, hardened compiler flags, статическая линковка.
8	Разработать систему непрерывного мониторинга безопасности GCS в флоте (SIEM-интеграция с Wazuh/Elastic).
9	Реализовать тайм-аут сессии оператора с автоматической блокировкой GCS при бездействии и повторной строгой аутентификацией.

Вар.	Задача
10	Провести анализ защищённости GCS на предмет наличия бэкдоров (supply chain security) — проверка зависимостей, origin кода.
11	Разработать модуль для QGC, который автоматически проверяет подпись MAVLink-сообщений и визуально предупреждает оператора при сбое верификации.
12	Настроить доверенный канал "GCS ↔ Drone" с использованием квантового распределения ключей (эмуляция QKD).
13	Создать "красную кнопку" для GCS: при обнаружении аномалии оператор нажимает кнопку, которая отправляет подписанную команду RTL с максимальным приоритетом.
14	Разработать стандарт безопасности GCS для государственных нужд на основе ГОСТ и требований ФСТЭК.
15	Интегрировать GCS с SIEM и SOAR: автоматическое реагирование на инциденты (изоляция GCS, блокировка дрона, уведомление SOC).

Требования: комплексное решение, отчёт с архитектурой, демонстрация, анализ эффективности.

5. Контрольные вопросы

1. Какие основные компоненты наземной станции управления (GCS) и какие угрозы наиболее вероятны для каждого?
2. Где QGroundControl хранит конфигурационные файлы и секреты? Как они защищены по умолчанию?
3. Что такое MITM-атака на GCS и как защититься от неё с помощью TLS?
4. В чём разница между односторонним TLS и взаимным TLS (mTLS)? Когда следует применять mTLS?
5. Какие криптографические протоколы и cipher suites рекомендуются для защиты канала GCS-дрон?
6. Почему хранение ключей MAVLink Signing в открытом виде опасно? Какие альтернативы?
7. Какие меры защиты хоста (ОС), на котором запущена GCS, вы можете предложить?
8. Что такое аппаратный токен (YubiKey) и как его можно использовать для аутентификации оператора GCS?
9. Как организовать безопасное обновление самой GCS, чтобы не допустить установки вредоносной версии?
10. Какие нормативные требования (например, ФСТЭК России) применимы к защите наземных станций управления БПЛА?

7. Требования к отчётам

Отчёты для ЛР №12 и ЛР №13 оформляются как отдельные документы (PDF), каждый содержит:

- Титульный лист.
- Цель и задачи.
- Описание стенда.
- Ход выполнения с листингами, скриншотами.
- Для ЛР №12: команды генерации ключей, содержимое манифеста, код клиента, скриншоты успешной верификации и блокировки при атаке.
- Для ЛР №13: расположение конфигурационных файлов, найденные секреты (замаскированные), скриншот MITM-атаки, настройка TLS, Hardening Guide.
- Индивидуальная часть.
- Ответы на контрольные вопросы.
- Выводы.

7. Критерии оценивания

7.1. «Удовлетворительно» (базовый уровень):

- ЛР №12: Подписан бинарник, создан манифест, клиент проверяет подпись и блокирует подделку.
- ЛР №13: Проведён аудит GCS, выявлены секреты, настроен TLS или проведена атака, предложены меры защиты.
- Отчёты оформлены, ответы на вопросы $\geq 60\%$.

7.2. «Хорошо» (средний уровень):

- Все требования базового уровня.
- Реализовано расширенное задание с демонстрацией.
- Ответы на вопросы $\geq 80\%$.

7.3. «Отлично» (повышенный уровень):

- Все требования среднего уровня.
- Комплексное решение / исследование с глубоким анализом.
- Ответы полные и развёрнутые.

7.4. Понижение оценки:

- Отсутствие скриншотов, неработающий механизм блокировки — доработка.
- Секреты раскрыты в отчёте без маскировки (для ЛР №13) — снижение оценки на балл.
- Слабые ответы на вопросы — пересдача.

Лабораторные работы №12 и №13 завершают блок безопасности автономных систем (ПК-11), формируя у обучающихся полное понимание процесса защиты БПЛА от сборки прошивки и доставки обновлений до защиты наземной инфраструктуры оператора.

Лабораторная работа №14

Анализ безопасности цепочки поставок компонентов БПЛА

1. Цель

Сформировать практические навыки анализа безопасности цепочки поставок (Supply Chain Security) для автономных инженерных систем на примере компонентов беспилотного летательного аппарата: выявление уязвимых зависимостей в прошивке и сторонних библиотеках, формирование SBOM, поиск CVE и разработка политики управления уязвимостями.

2. Задачи

- Выбрать конкретный программный или аппаратно-программный компонент БПЛА (полётный контроллер, библиотека, операционная система реального времени).
- Провести инвентаризацию зависимостей выбранного компонента и сгенерировать SBOM (Software Bill of Materials).
- Выполнить поиск известных уязвимостей (CVE) в идентифицированных зависимостях с использованием национальных и международных баз данных (BDU ФСТЭК, NVD, OSV).
- Провести анализ влияния найденных уязвимостей на безопасность БПЛА в контексте его эксплуатации.
- Разработать проект политики управления уязвимостями для цепочки поставок БПЛА (включая критерии приемлемого риска, сроки устранения, порядок мониторинга).

3. Теоретическое обоснование

Цепочка поставок БПЛА включает множество программных и аппаратных компонентов:

- RTOS / ОС: NuttX (используется в PX4), ChibiOS, Linux (в дополнительных компьютерах), FreeRTOS.
- Middleware: MAVLink, ROS 2 (DDS), внутренние протоколы обмена.
- Библиотеки: Библиотеки управления (PID, EKF), криптографические библиотеки (mbedtls, OpenSSL, wolfSSL), библиотеки работы с сенсорами (Bosch, InvenSense драйверы).
- Инструменты сборки: GCC, CMake, скрипты на Python, загрузчики (bootloaders).
- Аппаратные компоненты: Микроконтроллеры (STM32, NXP), GPS-модули (u-blox), IMU, радио-модули.

Угрозы цепочки поставок (Supply Chain Threats):

- Заражение на этапе сборки (compile-time injection): Вредоносный код, внедрённый через скомпрометированный компилятор или сборочный скрипт (аналогично атаке XZ Utils 2024).
- Уязвимые зависимости: Использование библиотек с известными CVE (например, уязвимости в mbedtls, затрагивающие TLS-коммуникации).
- Компрометация репозитория: Подмена исходного кода в upstream-репозитории.

- Аппаратные закладки: Модификация микроконтроллеров или сенсоров на этапе производства.
- Нарушение целостности при доставке: Подмена прошивки на этапе OTA-обновления (см. ЛР №12).

Нормативная база РФ:

- Приказ ФСТЭК России № 239 — требования по обеспечению безопасности значимых объектов КИИ.
- ГОСТ Р 56939-2016 — защита информации при разработке ПО (включая требования к анализу сторонних компонентов).
- Методический документ ФСТЭК — меры защиты информации в государственных информсистемах.
- Банк данных угроз (BDU) ФСТЭК России — национальная база уязвимостей, обязательная к использованию для сертификации и аттестации.

Процесс анализа цепочки поставок:

1. Инвентаризация компонентов: Построение SBOM для прошивки и сопутствующего ПО.
2. Поиск уязвимостей: Автоматизированное сопоставление компонентов SBOM с базами CVE (BDU ФСТЭК, NVD/NIST, OSV).
3. Анализ влияния (Impact Analysis): Оценка, как конкретная уязвимость влияет на безопасность БПЛА (доступность, целостность, конфиденциальность).
4. Оценка риска: Количественная или качественная оценка (CVSS, с учётом контекста эксплуатации).
5. Разработка мер: Обновление, замена компонента, компенсирующие меры (WAF, firewall, ограничение функциональности).

4. Задания для индивидуального выполнения

Номер варианта соответствует последней цифре зачётки / списка группы.

4.1. Базовый уровень (15 вариантов)

Вар.	Целевой компонент БПЛА	Источник для анализа	Фокус CVE
1	PX4 Autopilot (v1.14)	GitHub: PX4/PX4-Autopilot	Python build scripts
2	ArduPilot (v4.4)	GitHub: ArduPilot/ardupilot	Python dependencies + mavlink
3	NuttX RTOS (из состава PX4)	GitHub: PX4/NuttX	C-библиотеки, сетевая подсистема

Вар.	Целевой компонент БПЛА	Источник для анализа	Фокус CVE
4	mavlink C library (v2)	GitHub: mavlink/mavlink	CVE в библиотеке + зависимости
5	QGroundControl (v4.2)	GitHub: mavlink/qgroundcontrol	Qt, OpenSSL, картографические библиотеки
6	MAVSDK (C++ библиотека)	GitHub: mavlink/MAVSDK	gRPC, protobuf, libcurl
7	ROS 2 Humble (используется на дроне)	GitHub: ros2/ros2	Fast DDS, CycloneDDS, Python packages
8	Румavlink (Python библиотека)	PyPI + GitHub	Python-зависимости, уязвимости парсинга
9	Betaflight (полётный контроллер)	GitHub: betaflight/betaflight	STM32 HAL, FreeRTOS
10	OpenHD (видео-система для дронов)	GitHub: OpenHD/OpenHD	Kernel modules, WiFi-драйверы
11	MAVProxy (MAVLink прокси)	GitHub: ArduPilot/MAVProxy	Python-зависимости
12	Mission Planner (C# GCS)	GitHub: ArduPilot/MissionPlanner	.NET зависимости, Mono
13	DroneKit-Python	GitHub: dronekit/dronekit-python	Python-зависимости
14	APSync (образ для Raspberry Pi на дроне)	GitHub: ArduPilot/MAVProxy + образ	Системные пакеты (Debian)
15	Micro Air Vehicle Link (MAVLink) роутер	GitHub: mavlink-router/mavlink-router	C-зависимости, конфликтующие библиотеки

Требования к базовому уровню:

- Сгенерирован SBOM для выбранного компонента (формат CycloneDX или SPDX).
- Проведён поиск CVE (минимум 3 реальных уязвимости) в базах NVD и BDU ФСТЭК.
- Выполнен анализ влияния каждой уязвимости на безопасность БПЛА.
- Разработан проект политики управления уязвимостями (основные разделы).
- Ответы на контрольные вопросы $\geq 60\%$.

4.2. Средний уровень (15 вариантов)

Вар.	Задача
1	Сравнить уязвимости между PX4 v1.13 и v1.14: составить отчёт, какие CVE устранены, какие появились новые.
2	Интегрировать SBOM в Dependency-Track и настроить автоматический поиск CVE с оповещением при Critical.
3	Разработать скрипт на Python, который на основе SBOM обращается к BDU ФСТЭК API (или парсит открытые данные) и выводит сводку применимых уязвимостей.
4	Провести анализ цепочки поставок аппаратного компонента (например, STM32F7, GPS u-blox M8) — какие прошивки и библиотеки используются, какие уязвимости известны.
5	Выполнить сравнительный анализ уязвимостей для PX4 и ArduPilot, выявить общие зависимости и различия.
6	Проанализировать безопасность сборочного окружения (Dockerfile / CI) проекта PX4: есть ли в нём уязвимые пакеты.
7	Реализовать автоматическую проверку свежести зависимостей (наличие обновлений) и генерацию рекомендаций по обновлению.
8	Провести анализ лицензионных рисков в цепочке поставок: какие лицензии используются компонентами, есть ли конфликты GPL с проприетарным кодом.
9	Сравнить базы уязвимостей: найти CVE, присутствующие в NVD, но отсутствующие в BDU, и наоборот. Объяснить причину.
10	Разработать методику оценки риска уязвимости с учётом специфики БПЛА (лётная безопасность, физический ущерб) и применить к найденным CVE.

Вар.	Задача
11	Проанализировать submodule-зависимости PX4: построить граф зависимостей и определить критические узлы (при обновлении которых ломается многое).
12	Создать CI/CD-пайплайн (GitLab CI), который при каждом коммите в main анализирует зависимости и блокирует сборку при появлении новых HIGH CVE.
13	Исследовать влияние CVE в инструментах сборки (GCC, CMake) на безопасность конечной прошивки.
14	Провести анализ уязвимостей в драйверах сенсоров (MPU9250, ICM-20689) и их влияния на точность навигации.
15	Разработать карту поставщиков (Supplier Map) для цепочки поставок БПЛА: кто производит ключевые компоненты, каков уровень их безопасности.

Требования: базовый уровень + выполнение расширенного задания с демонстрацией результатов.

4.3. Повышенный уровень (15 вариантов)

Вар.	Задача
1	Разработать систему непрерывного мониторинга цепочки поставок БПЛА с интеграцией Dependency-Track, ежедневным обновлением CVE и автоматической блокировкой выпуска при критических уязвимостях.
2	Провести формальную верификацию безопасности цепочки поставок с использованием модели Alloy/TLA+ для критического пути: сборка прошивки → подпись → доставка → установка.
3	Разработать методику оценки доверенности поставщиков (Trust Score) на основе анализа их истории уязвимостей, скорости исправления, прозрачности SBOM. Применить к трём реальным поставщикам.
4	Исследовать возможность атаки на цепочку поставок через компрометацию сборочных скриптов Python (confused.py, dependency confusion) в контексте проекта PX4. Реализовать защиту.
5	Построить "Supply Chain Risk Dashboard" (Grafana) для визуализации состояния безопасности цепочки поставок флота дронов.

Вар.	Задача
6	Провести red team exercise: попытка внедрить вредоносный код в сборку PX4 через компрометацию внешней зависимости (эмуляция). Задokumentировать методы обнаружения и предотвращения.
7	Разработать стандарт безопасности цепочки поставок для отечественных БПЛА с учётом требований ФСТЭК (Приказы №31, 239, ГОСТ Р ИСО/МЭК 27036).
8	Создать инструмент для автоматического аудита цепочки поставок (на Python) с генерацией сводного отчёта по форме ФСТЭК.
9	Исследовать возможности аппаратной компрометации (hardware trojan) в микроконтроллерах STM32/H7, используемых в полётных контроллерах. Предложить методы детектирования.
10	Проанализировать цепочку поставок криптографической библиотеки mbedTLS (используется в PX4): какие CVE были, как быстро исправляются, кто поддерживает.
11	Разработать прототип системы обнаружения аномалий в поведении дрона после установки новой прошивки, сравнивая с эталонной моделью (firmware behavior baseline).
12	Построить граф зависимостей для всего стека БПЛА (PX4 + NuttX + mavlink + QGC + MAVSDK) и провести анализ центральности для выявления наиболее критичных компонентов.
13	Интегрировать проверку цепочки поставок в процесс DevSecOps: автоматический анализ SBOM, CVE, лицензий на каждом MR, блокировка до исправления.
14	Провести сравнительный анализ безопасности цепочки поставок гражданского БПЛА (PX4) и военного БПЛА (на основе открытых источников), выявить различия в подходах.
15	Разработать полный пакет документов для сертификации БПЛА по требованиям ФСТЭК в части анализа уязвимостей цепочки поставок (SBOM, анализ CVE, план управления уязвимостями, отчёт об оценке рисков).

Требования: комплексное исследование или разработка, отчёт с архитектурой, код/документация, анализ эффективности.

5. Контрольные вопросы (ЛР №14)

1. Что такое цепочка поставок программного обеспечения (Software Supply Chain) и какие основные угрозы для неё существуют?
2. Какие компоненты БПЛА наиболее критичны с точки зрения безопасности цепочки поставок?
3. Что такое SBOM и почему он важен для анализа цепочки поставок?
4. Какие базы данных уязвимостей вы знаете? В чём разница между NVD (NIST) и BDU ФСТЭК России?
5. Как оценить влияние CVE в библиотеке на безопасность конечного продукта (БПЛА)?
6. Что такое Dependency Confusion и как защититься от этой атаки?
7. Какие нормативные документы РФ регулируют безопасность цепочки поставок ПО (в том числе для КИИ)?
8. Как организовать непрерывный мониторинг уязвимостей в цепочке поставок?
9. Почему уязвимость в инструменте сборки (например, в библиотеке Jinja2 для генерации кода) может быть так же опасна, как уязвимость в рантайм-библиотеке?
10. Какие меры защиты от компрометации репозитория исходного кода вы можете предложить?

1. Цель

Сформировать практические навыки реализации Continuous Compliance — автоматизированного отслеживания соответствия облачных ресурсов требованиям безопасности — с помощью политик как кода на языке Rego (OPA) и инструмента Cloud Custodian для облачных платформ (AWS, Yandex Cloud, GCP).

2. Задачи

- Изучить концепцию Compliance as Code и инструменты её реализации (OPA/Rego, Cloud Custodian).
- Написать политику(и) на языке Rego для проверки конфигураций облачных ресурсов (например, запрет публичных S3/OS бакетов, обязательное шифрование, проверка тегов).
- Развернуть и настроить Cloud Custodian (или custodian-cdk) для управления облачными ресурсами.
- Выполнить запуск политик в режиме dry-run для выявления несоответствующих ресурсов.
- Настроить автоматическую отправку уведомлений (Email/Slack) при обнаружении несоответствий.
- (Средний/Повышенный уровень) Реализовать автоматическое исправление (auto-remediation) или интеграцию в CI/CD.

3. Теоретическое обоснование

Continuous Compliance — практика непрерывного мониторинга и автоматического приведения облачной инфраструктуры и процессов в соответствие с требованиями безопасности, отраслевых стандартов (CIS, NIST, PCI DSS) и внутренними политиками. Ключевой принцип: политики формулируются как код (Policy as Code / Compliance as Code), что позволяет версионировать, тестировать и автоматически применять их.

Open Policy Agent (OPA) — движок принятия решений, использующий язык Rego для описания политик (см. также ЛР №5). В облачном контексте OPA может проверять Terraform-планы, Kubernetes-манифесты, конфигурации облачных ресурсов.

Cloud Custodian — open-source инструмент (CNCF Sandbox) для управления облачными ресурсами через политики, описанные в YAML. Поддерживает AWS, Azure, GCP, а также Yandex Cloud (через плагины). Возможности:

- Режимы работы: dry-run (только отчёт), notify (уведомление), remediate (автоматическое исправление).
- Фильтры: По типу ресурса, тегам, времени создания, региону, значениям свойств.
- Действия: Остановка/запуск/удаление ресурсов, отправка уведомлений, вызов Lambda/Cloud Functions.
- Формат политик: YAML-файлы с описанием ресурсов, фильтров и действий.

Сравнение OPA и Cloud Custodian:

Характеристика	OPA (Rego)	Cloud Custodian
Язык политик	Rego (декларативный, Datalog)	YAML
Область применения	Широкая (K8s, API, Terraform)	Облачные ресурсы
Сложность	Высокая (требует изучения Rego)	Низкая (YAML-конфигурация)
Основное применение	Admission control, API authZ	Cloud governance, cost, security
Интеграция с CI/CD	Confest (см. ЛР №5)	c7n CLI, AWS Lambda

Типовые политики безопасности облака (CIS benchmarks для AWS):

- S3 Buckets не должны быть публично доступны (Block Public Access).
- EBS-тома должны быть зашифрованы.
- Security Groups не должны иметь правил с 0.0.0.0/0 на порты SSH/RDP (22, 3389).
- IAM-пользователи должны иметь MFA.
- CloudTrail должен быть включен во всех регионах.

Интеграция в инженерные системы: Для облачных компонентов инженерных систем (сервера сбора телеметрии БПЛА, облачные ОТА-серверы, аналитические платформы) критически важно обеспечивать compliance в реальном времени, особенно если они входят в периметр КИИ.

4. Задания для индивидуального выполнения

Номер варианта соответствует последней цифре зачётки / списка группы.

4.1. Базовый уровень (15 вариантов)

Вар.	Облачная платформа	Тип ресурса для проверки	Политика безопасности
1	AWS	S3 Bucket	Бакет не должен быть публичным (BlockPublicAccess: true)
2	AWS	Security Group	Запрет входящего трафика 0.0.0.0/0 на порт 22 (SSH)
3	AWS	EBS Volume	Все тома должны быть зашифрованы

Вар.	Облачная платформа	Тип ресурса для проверки	Политика безопасности
4	Yandex Cloud	Object Storage Bucket	Бакет не должен быть публичным
5	Yandex Cloud	Compute Instance	ВМ должна иметь метку env=production
6	AWS	RDS Instance	База данных должна иметь включённое шифрование (storage_encrypted: true)
7	GCP	Cloud Storage Bucket	Бакет не должен быть публичным
8	GCP	Compute Engine Instance	Инстанс должен использовать определённый образ (OS Login enabled)
9	AWS	IAM User	Все IAM-пользователи должны иметь MFA
10	Yandex Cloud	Managed Kubernetes Cluster	Кластер должен иметь включённый Network Policy и audit logs
11	AWS	CloudTrail	CloudTrail должен быть включён во всех регионах
12	Terraform (любое облако)	План Terraform (JSON)	Проверка через OPA/Conftest: S3 бакет не имеет acl = public-read
13	Terraform	План Terraform	Проверка: security group не открывает порт 3389 на 0.0.0.0/0
14	Yandex Cloud	VPC Network	Запрет подсетей с маршрутом 0.0.0.0/0 в интернет без NAT Gateway
15	AWS	Lambda Function	Lambda не должна использовать устаревший runtime (Python 3.7 и ниже)

Требования к базовому уровню:

- Установлен Cloud Custodian (или OPA/Conftest) и успешно запущен.
- Написана одна политика согласно варианту.

- Выполнен запуск политики в dry-run (или test для Conftest) и выявлены несоответствующие ресурсы (или показан провал на тестовом нарушающем ресурсе).
- Отчёт содержит листинг политики и скриншоты запуска.
- Ответы на контрольные вопросы $\geq 60\%$.

4.2. Средний уровень (15 вариантов)

Вар.	Задача
1	Настроить Cloud Custodian в режиме notify с отправкой оповещений в Slack/Email через SMTP при нахождении нарушений.
2	Реализовать политику Cloud Custodian с автоматическим исправлением (auto-remediation): например, автоматически шифровать незашифрованные EBS-тома.
3	Сравнить OPA (Rego) и Cloud Custodian: реализовать одну и ту же политику (публичный S3) в обоих инструментах, сравнить сложность и возможности.
4	Интегрировать Cloud Custodian в CI/CD (GitLab CI/GitHub Actions): при каждом коммите в инфраструктурный репозиторий запускается проверка политик.
5	Написать сложную политику Rego для Terraform, проверяющую несколько взаимосвязанных условий (например, если используется AWS RDS, то обязательно есть subnet group и шифрование).
6	Настроить Cloud Custodian для работы с Yandex Cloud через yc CLI, продемонстрировать работу с кастомным провайдером.
7	Реализовать политику, которая проверяет теги ресурсов: все ресурсы должны иметь теги owner, project, cost-center.
8	Настроить периодический запуск Cloud Custodian через AWS Lambda / Cloud Functions и логирование результатов в CloudWatch / Yandex Monitoring.
9	Разработать дашборд (Grafana или простая HTML-страница), который отображает текущий статус compliance облачных ресурсов.
10	Использовать c7n-org для запуска политик на несколько аккаунтов/облачных проектов одновременно. Показать сводный отчёт.
11	Написать политику, которая проверяет, что все S3-бакеты имеют включённое

Вар.	Задача
	логирование (access logs) и версионирование.
12	Реализовать compliance-проверку для Kubernetes-кластера (Yandex Managed K8s): наличие NetworkPolicy, RBAC, audit logs.
13	Интегрировать OPA с Terraform Cloud/Enterprise: настроить автоматическую проверку планов перед применением (Policy Sets).
14	Провести сравнительный анализ Cloud Custodian, OPA/Conftest, Checkov и TFLint для задачи compliance-проверки облачных ресурсов.
15	Настроить Cloud Custodian с хранением истории проверок в базе данных (SQLite/PostgreSQL) и визуализацией тренда compliance.

Требования: базовый уровень + выполнение расширенного задания с демонстрацией.

4.3. Повышенный уровень (15 вариантов)

Вар.	Задача
1	Построить централизованную систему compliance monitoring для мультиоблачной среды: AWS + Yandex Cloud через единый Cloud Custodian конфиг.
2	Разработать собственную DSL/обёртку над Cloud Custodian для описания политик на русском языке в машиночитаемом формате с автоматической генерацией YAML.
3	Интегрировать Cloud Custodian с ITSM-системой (Jira/ServiceNow): при обнаружении несоответствия автоматически создаётся инцидент с SLA на устранение.
4	Реализовать систему автоматического подсчёта Compliance Score (% соответствия) и настроить оповещение при его падении ниже порога.
5	Разработать микросервис (FastAPI), который принимает webhook от Cloud Custodian, анализирует нарушения и назначает ответственных на основе тегов ресурсов.
6	Интегрировать compliance-проверки в процесс CI/CD: при деплое инфраструктурного кода автоматически запускается Custodian и OPA, и только при 100% compliance деплой продолжается.

Вар.	Задача
7	Разработать оператор Kubernetes, который периодически запускает Cloud Custodian для проверки облачных ресурсов и отображает статус в Custom Resources (CRD).
8	Настроить автоматическую генерацию отчётов для аудиторов (PDF) на основе результатов Cloud Custodian политик, с графиками и трендами.
9	Провести threat modeling (STRIDE) для системы Continuous Compliance и разработать меры защиты от атак на саму систему мониторинга.
10	Реализовать политики, соответствующие требованиям конкретного стандарта (CIS AWS Foundations Benchmark 1.4.0) в количестве не менее 10 штук.
11	Разработать систему "Compliance as a Service" для нескольких проектов: мультитенантный доступ, кастомные политики, отчётность.
12	Интегрировать Cloud Custodian с SIEM (ELK/Wazuh) для корреляции событий compliance с событиями безопасности.
13	Создать DSL на Python для декларативного описания compliance-правил, который транслируется и в Cloud Custodian, и в OPA одновременно.
14	Разработать политики compliance для Yandex Cloud, соответствующие требованиям 152-ФЗ и приказам ФСТЭК (обезличивание, сертификация СКЗИ).
15	Построить архитектуру "Policy Decision Point" (PDP) и "Policy Enforcement Point" (PEP) для облачных ресурсов на основе OPA с принятием решений в реальном времени.

Требования: комплексное решение с архитектурой, кодом, демонстрацией, анализом эффективности.

5. Контрольные вопросы (ЛР №15)

1. Что такое Continuous Compliance и чем он отличается от периодического аудита?
2. Какие преимущества даёт подход Compliance as Code по сравнению с ручной проверкой?
3. Сравните Open Policy Agent и Cloud Custodian: когда какой инструмент выбирать?
4. Что такое dry-run в Cloud Custodian и зачем он нужен?
5. Какие типовые политики безопасности облака вы знаете (CIS benchmarks)?
6. Как интегрировать проверку compliance в CI/CD-пайплайн для предотвращения развёртывания несоответствующих ресурсов?

7. Что такое auto-remediation и в каких случаях его следует применять, а в каких — нет?
8. Какие источники требований (стандарты, законы) могут быть автоматизированы через Compliance as Code?
9. Как устроен язык Rego для облачных политик? Приведите пример проверки Terraform-плана.
10. Какие особенности Yandex Cloud необходимо учитывать при написании политик (по сравнению с AWS)?

7. Требования к отчётам

Отчёты для ЛР №14 и ЛР №15 оформляются как отдельные документы (PDF), каждый содержит:

- Титульный лист.
- Цель и задачи.
- Для ЛР №14: описание выбранного компонента, листинг SBOM (фрагмент), таблица найденных CVE, анализ влияния на БПЛА, разработанная политика управления уязвимостями.
- Для ЛР №15: листинг политик (YAML/Rego), скриншоты запуска в dry-run с выявленными нарушениями, настройка уведомлений (если применимо).
- Индивидуальная часть: описание реализованного расширенного задания, листинги, скриншоты.
- Ответы на контрольные вопросы.
- Выводы и рекомендации.

7. Критерии оценивания

7.1. «Удовлетворительно» (базовый уровень):

- ЛР №14: SBOM сгенерирован, найдены CVE, проведён базовый анализ, предложена политика управления уязвимостями.
- ЛР №15: Написана политика, инструмент запущен в dry-run (или проверен через Conftest), выявлены нарушения.
- Отчёты оформлены, ответы на вопросы $\geq 60\%$.

7.2. «Хорошо» (средний уровень):

- Все требования базового уровня.
- Выполнено расширенное задание: интеграция с Dependency-Track, автоматизация, уведомления, auto-remediation.
- Ответы на вопросы $\geq 80\%$.

7.3. «Отлично» (повышенный уровень):

- Все требования среднего уровня.
- Комплексное решение, архитектура, демонстрация масштабируемой системы compliance.
- Ответы на вопросы полные, аргументированные.

7.4. Понижение оценки:

- Отсутствие SBOM или списка CVE (для ЛР №14) — доработка.
- Политика не находит нарушений на тестовом примере — не выше базового уровня.
- Ответы на вопросы неудовлетворительны — направляется на пересдачу.

Лабораторная работа №16
Автоматизированная генерация отчётов для аудита (Compliance Reports)

1. Цель

Сформировать практические навыки автоматизации сбора доказательств соответствия требованиям безопасности (compliance evidence) из различных источников (SAST, SCA, DAST, облачный аудит) и генерации консолидированного отчёта для внутреннего или внешнего аудита с использованием шаблонизации и визуализации.

2. Задачи

- Провести аудит облачной инфраструктуры (или локальной эмуляции) на соответствие CIS Benchmarks с помощью Prowler / ScoutSuite.
- Собрать результаты SAST, SCA и DAST из инструментов, изученных ранее (Semgrep, Trivy, OWASP ZAP, Dependency-Track) через их API или путём парсинга артефактов.
- Разработать Python-скрипт для агрегации всех данных в единую структуру (JSON).
- Создать HTML-шаблон (Jinja2) для визуализации сводного отчёта — compliance dashboard.
- (Средний/Повышенный уровни) Настроить периодическую генерацию отчётов, экспорт в PDF, интеграцию с Grafana, создание уведомлений по отклонениям.

3. Теоретическое обоснование

Compliance-отчётность — обязательный элемент системы управления информационной безопасностью (СМИБ) согласно ISO 27001, PCI DSS, требованиям ФСТЭК России. Аудиторам необходимо предоставлять доказательства того, что:

- Активы (ресурсы, приложения, образы) регулярно проверяются на соответствие политикам.
- Найденные несоответствия отслеживаются и устраняются в установленные сроки.
- Процесс сбора доказательств автоматизирован, воспроизводим и не зависит от человеческого фактора.

Источники данных для compliance-отчёта:

Категория	Инструменты	Данные
Cloud Security	Prowler, ScoutSuite, Cloud Custodian	Нарушения CIS Benchmarks, публичные ресурсы
SAST	Semgrep, CodeQL	Количество уязвимостей по severity, файл/строка
SCA	Trivy, Dependency-Track API	CVE в зависимостях, компоненты, SBOM

Категория	Инструменты	Данные
DAST	OWASP ZAP API	Алерты, типы уязвимостей, URL
Контейнеры	Trivy (image scan)	Уязвимости в образах, base image
K8s Security	kube-bench	Нарушения CIS Kubernetes Benchmark

Инструменты автоматизации сбора:

- Prowler (AWS, Azure, GCP) — опенсорсный инструмент для аудита облака на соответствие CIS, GDPR, PCI DSS и другим стандартам. Вывод в JSON, CSV, HTML.
- ScoutSuite (мультиоблачный) — аналогичный инструмент с HTML-отчётом.
- Dependency-Track API — программный доступ к результатам анализа SCA, проектам и уязвимостям.
- DefectDojo API — агрегатор результатов всех видов тестирования безопасности.

Технологии агрегации и визуализации:

- Python (Jinja2, WeasyPrint/xhtml2pdf) — генерация HTML → PDF отчётов из шаблонов.
- Grafana — дашборды, подключаемые к источникам данных (JSON API, Prometheus, Elasticsearch).
- Elasticsearch + Kibana — централизованное хранение результатов и визуализация.

Структура итогового отчёта (пример):

1. Резюме для руководства (Executive Summary): общий compliance score, тренд, ключевые риски.
2. Методология и периодичность сбора данных.
3. Результаты по категориям:
 - Облачная инфраструктура (CIS compliance %).
 - Безопасность приложений (SAST/DAST).
 - Безопасность цепочки поставок (SCA/SBOM).
 - Безопасность контейнеров.
4. Детальные находки (Findings): несоответствия, severity, статус устранения.
5. Рекомендации и план устранения.

Соответствие нормативным требованиям РФ:

- Приказ ФСТЭК № 239 (КИИ) — требует регулярного контроля соответствия.
- Методика оценки угроз (ФСТЭК) — требует документирования результатов аудита.

- 152-ФЗ «О персональных данных» — требует отчётности по мерам защиты.

4. Задания для индивидуального выполнения

Номер варианта соответствует последней цифре зачётки / списка группы.

4.1. Базовый уровень (15 вариантов)

Вар.	Основной источник аудита	Дополнительный источник 1	Дополнительный источник 2	Формат вывода
1	Prowler (AWS CIS)	Semgrep (JSON)	—	HTML (Jinja2)
2	Prowler (GCP CIS)	Trivy (JSON)	—	HTML
3	ScoutSuite (AWS)	OWASP ZAP JSON API	—	HTML
4	Prowler (Azure CIS)	Semgrep + Trivy	—	HTML
5	kube-bench (K8s CIS)	Trivy image scan	—	HTML
6	Prowler (AWS)	Dependency-Track API (CVE)	—	JSON + HTML
7	Cloud Custodian (dry-run JSON)	Semgrep	OWASP ZAP	HTML
8	Prowler (AWS)	Trivy fs scan	—	Markdown
9	kube-bench	Semgrep + OWASP ZAP	—	HTML
10	ScoutSuite (GCP)	Trivy	—	HTML
11	Prowler (AWS + K8s)	Semgrep	Trivy	JSON → HTML
12	Prowler (AWS)	SAST + SCA (через DefectDojo API)	—	HTML

Вар.	Основной источник аудита	Дополнительный источник 1	Дополнительный источник 2	Формат вывода
13	Локальный скрипт (CIS Docker)	Trivy image scan	Semgrep	HTML
14	Prowler (AWS)	SAST + SCA + DAST (все 3)	—	HTML
15	Ручной JSON (моковые данные)	Semgrep mock + Trivy mock	OWASP ZAP mock	HTML

Требования к базовому уровню:

- Выполнен аудит хотя бы одного источника (Prowler/kube-bench/ScoutSuite или парсинг данных SAST/SCA).
- Создан единый JSON-файл с агрегированными данными.
- Сгенерирован HTML-отчёт с использованием Jinja2 (или аналога), содержащий сводную таблицу находок и хотя бы один график/таблицу по каждому источнику.
- Ответы на контрольные вопросы $\geq 60\%$.

4.2. Средний уровень (15 вариантов)

Вар.	Задача
1	Интегрировать экспорт отчёта в PDF с помощью WeasyPrint или аналога. Отчёт должен содержать титульную страницу и оглавление.
2	Создать Grafana дашборд с метриками compliance (compliance score, критические находки, тренд по датам).
3	Настроить автоматическую рассылку отчётов по email (SMTP) после завершения пайплайна в GitLab CI.
4	Сравнить результаты Prowler и Cloud Custodian на одних и тех же ресурсах AWS. Объяснить расхождения.
5	Реализовать сбор данных из DefectDojo API (все findings по продукту) и агрегацию в единый отчёт.
6	Разработать интерактивный HTML-отчёт с фильтрацией по severity и источнику на чистом

Вар.	Задача
	JavaScript.
7	Настроить хранение метрик в Elasticsearch (через API), визуализацию в Kibana.
8	Реализовать сравнение двух отчётов (текущий vs предыдущий): подсчёт новых, устранённых и оставшихся находок.
9	Создать compliance-отчёт в формате, приближенном к требованиям ФСТЭК (РФ) — с указанием мер защиты и ссылками на нормативные акты.
10	Интегрировать сбор данных из Dependency-Track API (список проектов, уязвимостей) и отобразить в отчёте.
11	Настроить сбор метрик из Prometheus (количество алертов) и включение их в compliance-отчёт.
12	Разработать Python CLI-инструмент (с Click/Typer), который собирает данные из разных источников и генерирует отчёт.
13	Создать кастомный шаблон отчёта для конкретного стандарта (PCI DSS / 152-ФЗ) с маппингом находок на пункты стандарта.
14	Настроить автоматическую загрузку отчёта в GitLab Pages для публикации (доступ только для аудиторов).
15	Реализовать систему scoring (подсчёт баллов соответствия) на основе матрицы рисков и severity находок.

Требования: базовый уровень + выполнение расширенного задания с демонстрацией.

4.3. Повышенный уровень (15 вариантов)

Вар.	Задача
1	Построить централизованную систему compliance-отчётности для нескольких проектов/облаков (multi-tenant). Автоматический сбор, единый дашборд, ролевой доступ.
2	Разработать compliance-оператор Kubernetes (Operator pattern), который регулярно запускает сбор данных из Prowler/Trivy/Semgrep и обновляет CRD с результатами.

Вар.	Задача
3	Интегрировать систему отчётности с SIEM (Wazuh/ELK): данные compliance используются для корреляции с событиями безопасности.
4	Реализовать систему прогнозирования compliance score на основе исторических данных (ARIMA / Prophet).
5	Разработать REST API сервис (FastAPI), который принимает запросы на генерацию отчёта, запускает сбор данных и возвращает HTML/PDF.
6	Создать "Compliance as Code" платформу: политики описаны в YAML, движок автоматически собирает доказательства и генерирует отчёт.
7	Провести аудит безопасности самой системы compliance-отчётности (угрозы подмены данных, несанкционированный доступ).
8	Интегрировать систему с ITSM (Jira/ServiceNow): автоматическое создание тикетов на основе находок в отчёте, трекинг устранения.
9	Разработать модуль для анализа трендов: визуализация изменения compliance-показателей за год, прогноз до конца года.
10	Реализовать блокчейн-аудит для compliance-отчётов: хеш отчёта записывается в смарт-контракт (Ethereum/Hyperledger) для обеспечения неизменности.
11	Разработать систему алертинга: при падении compliance score ниже порога автоматически создаётся инцидент и уведомляется security team.
12	Создать полный пакет документов для сертификации по ГОСТ Р 57580.1-2017 (ФСТЭК) на основе автоматически собранных данных.
13	Интегрировать систему с ОРА: автоматическая проверка, что процессы (пайплайны) соответствуют требованиям compliance перед деплоем.
14	Разработать "политику хранения доказательств": как долго хранить отчёты, как организовать резервное копирование, как обеспечить конфиденциальность.
15	Создать цифровой двойник compliance-процесса: симуляция изменений в инфраструктуре, прогнозирование влияния на compliance score.

Требования: полная автоматизация, архитектурная схема, код, демонстрация работающей системы.

5. Контрольные вопросы (ЛР №16)

1. Что такое compliance-отчёт и для каких целей он используется в организации?
2. Какие основные источники данных для compliance-отчёта вы знаете? Приведите примеры для облачной инфраструктуры.
3. Какие инструменты автоматизации аудита облака вы использовали? В чём их различия?
4. Как агрегировать данные из разных источников в единый формат?
5. В чём преимущества использования шаблонизаторов (Jinja2) для генерации отчётов?
6. Что такое CIS Benchmarks и как они связаны с compliance-проверками?
7. Как настроить автоматическую периодическую генерацию отчётов в CI/CD?
8. Какие метрики compliance наиболее важны для руководства? Как их визуализировать?
9. Какие требования к отчётности содержатся в российских нормативных актах (ФСТЭК, 152-ФЗ)?
10. Как обеспечить целостность и неизменность compliance-отчёта для аудиторов?

Лабораторная работа №17

Комплексный анализ безопасности фреймворка автономной навигации (ROS 2 Navigation2)

1. Цель

Сформировать практические навыки проведения комплексного анализа безопасности открытого фреймворка автономной навигации (ROS 2 Navigation2) с применением методов SAST, SCA, анализа конфигурации безопасности по умолчанию и формулирования обоснованных выводов о применимости фреймворка в инженерных системах различного уровня критичности.

2. Задачи

- Выполнить статический анализ безопасности исходного кода Navigation2 (SAST) с помощью Semgrep и/или CodeQL.
- Провести анализ зависимостей фреймворка (SCA) с помощью Trivy, сгенерировать SBOM и оценить CVE в контексте робототехнических систем.
- Изучить и проанализировать конфигурацию безопасности по умолчанию: параметры DDS (Fast DDS / CycloneDDS), настройки SROS 2 Security Enclaves, права доступа узлов.
- Оценить архитектуру безопасности фреймворка: доверенные границы, аутентификация, авторизация, защита данных in-transit.
- Сформулировать выводы о пригодности Navigation2 для использования в системах с повышенными требованиями к безопасности (КИИ, АСУ ТП, военные применения).
- Разработать итоговый документ «Анализ безопасности ROS 2 Navigation2».

3. Теоретическое обоснование

ROS 2 Navigation2 (Nav2) — основной фреймворк для автономной навигации мобильных роботов в экосистеме ROS 2. Включает:

- BT Navigator Server — сервер деревьев поведения (Behavior Trees).
- Planner Server — глобальный планировщик пути (NavFn, Smac Planner).
- Controller Server — локальный контроллер движения (DWB, MPPI).
- Recoveries Server — сервер восстановления после сбоев.
- Costmap 2D — карты стоимости для планирования.
- AMCL / Slam Toolbox — локализация и SLAM.

Почему безопасность Nav2 критична:

- Фреймворк управляет движением робота в физическом мире — ошибки или вредоносные воздействия могут привести к столкновениям, травмам, материальному ущербу.
- Используется в промышленных, медицинских, сельскохозяйственных роботах, потенциально — в КИИ.
- Основан на ROS 2, чьи механизмы безопасности (SROS 2, DDS Security) являются опциональными и не всегда включены по умолчанию.
- Имеет сложную архитектуру с множеством зависимостей, что расширяет поверхность атаки.

Типовые угрозы для фреймворков навигации (STRIDE, см. ЛР №9):

- Spoofing: Подмена узла планировщика, подделка данных одометрии.
- Tampering: Модификация карты стоимости (costmap) для направления робота в препятствие.
- Repudiation: Отсутствие логов о том, какой узел дал команду движения.
- Information Disclosure: Утечка карты помещения через незащищённый топик /map.

- Denial of Service: Перегрузка планировщика запросами, срыв навигации.
- Elevation of Privilege: Эксплуатация уязвимости в плагине для выполнения кода на работе.

Инструменты анализа:

Метод	Инструмент	Что проверяется
SAST	Semgrep (C++/Python правила)	Интъекции, переполнения буфера, использование небезопасных функций
SAST	CodeQL (C++)	Memory corruption, use-after-free, double free
SCA	Trivy	CVE в зависимостях (Fast DDS, Boost, Eigen, libcurl)
SBOM	Syft	Инвентаризация компонентов фреймворка
Config	Ручной аудит + Conftest/OPA	Настройки DDS, права доступа, параметры безопасности по умолчанию

Ключевые аспекты анализа конфигурации:

- DDS Security: Включена ли аутентификация, шифрование, контроль доступа для топиков Nav2?
- Качество кода: Используются ли небезопасные функции C++ (strcpy, sprintf, raw pointers)?
- Обработка ошибок: Как обрабатываются некорректные данные сенсоров?
- Логирование: Логируются ли критические действия (изменение параметров, вызов сервисов)?
- Привилегии: Требуется ли root-права для запуска узлов Nav2?

4. Задания для индивидуального выполнения

Номер варианта соответствует последней цифре зачётки / списка группы.

4.1. Базовый уровень (15 вариантов)

Вар.	Фреймворк для анализа	Основной метод анализа	Фокус проверки безопасности
1	ROS 2 Navigation2 (Humble)	SAST (Semgrep)	C++ компоненты: planner, controller

Вар.	Фреймворк для анализа	Основной метод анализа	Фокус проверки безопасности
2	ROS 2 Navigation2 (Humble)	SAST (Semgrep)	Python скрипты: bringup, launch, тесты
3	ROS 2 Navigation2 (Humble)	SCA (Trivy + Syft)	Все зависимости, генерация SBOM
4	ROS 2 Navigation2 (Humble)	SCA + проверка лицензий	Поиск CVE и проверка совместимости лицензий
5	ROS 2 Navigation2 (Humble)	Конфигурационный аудит	Проверка настроек Fast DDS, SROS 2 по умолчанию
6	ROS 2 Nav2 (Rolling)	SAST (Semgrep)	Сравнение с Humble: новые находки
7	MoveIt 2 (манипуляторы)	SAST (Semgrep)	Анализ кода, сравнение с Nav2
8	ROS 2 Nav2 + Slam Toolbox	SAST + SCA	Совместный анализ навигации и SLAM
9	ArduPilot (навигационная часть)	SAST (Semgrep)	Анализ C++ кода планировщика ArduPilot
10	PX4 Navigator (модуль)	SAST (Semgrep)	Анализ модуля навигации PX4
11	ROS 2 Nav2 (Humble)	STRIDE только	Полная модель угроз для Nav2
12	ROS 2 Nav2 (Humble)	SAST + конфиг	Semgrep + проверка дефолтных параметров
13	ROS 2 Nav2 (Humble)	SCA + конфиг	Trivy + проверка дефолтных параметров
14	ROS 2 Nav2 (Humble)	Ручной аудит кода	Проверка обработки ошибок, границ массивов, raw pointers

Вар.	Фреймворк для анализа	Основной метод анализа	Фокус проверки безопасности
15	ROS 2 Nav2 (Humble)	SAST + SCA + STRIDE (lite)	Комплексный, но базовый анализ

Требования к базовому уровню:

- Выполнен SAST и/или SCA анализ выбранного фреймворка.
- Идентифицированы уязвимости или потенциально опасные паттерны (минимум 5 находок).
- Проведён анализ конфигурации безопасности по умолчанию.
- Сформулированы выводы и рекомендации.
- Документ «Анализ безопасности...» оформлен в соответствии с требованиями.
- Ответы на контрольные вопросы $\geq 60\%$.

4.2. Средний уровень (15 вариантов)

Вар.	Задача
1	Выполнить SAST с помощью CodeQL (C++) для Nav2, сравнить результаты с Semgrep, объяснить различия.
2	Провести полный SCA с загрузкой SBOM в Dependency-Track, настроить мониторинг уязвимостей.
3	Настроить SROS 2 для Nav2 и проверить, защищены ли топики /cmd_vel, /map, /scan. Показать атаку без SROS 2.
4	Провести сравнительный анализ безопасности Nav2 и ArduPilot навигации (SAST + STRIDE).
5	Построить полную DFD-диаграмму Nav2 и применить STRIDE-per-element ко всем элементам (не менее 25 угроз).
6	Выполнить анализ уязвимостей по CWE Top 25 2024 года применительно к Nav2 (какие CWE релевантны для робототехники).
7	Провести фаззинг одного из узлов Nav2 (например, контроллера) с помощью AFL / libFuzzer. Задokumentировать краши.

Вар.	Задача
8	Реализовать автоматический анализ безопасности Nav2 в GitLab CI: Semgrep + Trivy + SROS 2 audit = единый отчёт.
9	Проанализировать безопасность поведенческих деревьев (Behavior Trees): могут ли вредоносные узлы изменить логику навигации?
10	Сравнить безопасность Nav2 на Fast DDS и на CycloneDDS (наличие уязвимостей, особенности конфигурации).
11	Провести анализ приватности: какие данные передаются в топиках Nav2 в открытом виде, как это нарушает конфиденциальность миссии?
12	Разработать скрипт автоматического аудита конфигурации Nav2 (наличие SROS 2, проверка прав узлов, параметры DDS).
13	Сравнить уязвимости в зависимостях Nav2 для Humble и Rolling: какие исправлены, какие появились.
14	Провести анализ безопасности Nav2 для мульти-роботных систем (несколько роботов в одном DDS домене).
15	Разработать рекомендации по внедрению Nav2 в КИИ на основе проведённого анализа (со ссылками на ФСТЭК).

Требования: базовый уровень + выполнение расширенного задания с демонстрацией.

4.3. Повышенный уровень (15 вариантов)

Вар.	Задача
1	Провести полный аудит безопасности Nav2 по методологии OWASP MASVS (Mobile Application Security) адаптированной к робототехнике.
2	Разработать модуль для ROS 2, который автоматически проверяет подписи сообщений Nav2 и блокирует некорректные (реализация IDS).
3	Построить "Security Hardened Nav2": собственная сборка с исправлением найденных уязвимостей и включением всех механизмов защиты по умолчанию.

Вар.	Задача
4	Исследовать возможность атаки на Nav2 через вредоносные данные сенсоров (adversarial examples для costmap). Проверить на симуляторе.
5	Разработать Threat Model для Nav2 в формате, пригодном для сертификации по ISO 21434 (автомобильная безопасность).
6	Провести пентест Nav2 в симуляции: атакующий узел пытается нарушить навигацию, защищённая система противостоит.
7	Разработать систему формальной верификации критических участков кода Nav2 (контроллер) с использованием Frama-C или аналогичного инструмента.
8	Создать CI/CD пайплайн, который автоматически обновляет SBOM Nav2, ищет CVE, запускает SAST, и при Critical блокирует использование фреймворка в проекте.
9	Разработать "Security Benchmark" для фреймворков навигации (по аналогии с CIS), включая проверку прав, шифрования, логирования.
10	Провести исследование на тему "квантово-устойчивая криптография для Nav2": какие алгоритмы можно применить, как это повлияет на производительность.
11	Разработать архитектуру "Zero Trust Navigation": все узлы Nav2 аутентифицируются, весь трафик шифруется, доступ — по временным токенам.
12	Исследовать возможность использования eBPF для мониторинга безопасности Nav2 в реальном времени (отслеживание системных вызовов узлов).
13	Разработать оператор безопасности для ROS 2 (Kubernetes-like), который автоматически применяет политики безопасности к Nav2 узлам.
14	Провести экономический анализ безопасности: сколько стоит внедрение защиты Nav2 vs потенциальный ущерб от компрометации (на примере промышленного робота).
15	Разработать стандарт безопасной разработки для ROS 2 навигационных фреймворков на основе проведённого анализа (Best Practices).

Требования: глубокое исследование, архитектурные решения, код/документация, демонстрация.

5. Контрольные вопросы

1. Какие основные компоненты входят в состав ROS 2 Navigation2 и какие угрозы безопасности для них наиболее критичны?
2. Какие категории CWE наиболее релевантны для робототехнического ПО и почему?
3. В чём преимущества и недостатки использования Semgrep по сравнению с CodeQL для анализа C++ кода?
4. Какие уязвимости в зависимостях Nav2 могут привести к компрометации робота? Приведите примеры.
5. Что такое SROS 2 и почему его включение критически важно для Nav2?
6. Как проанализировать конфигурацию безопасности DDS по умолчанию для ROS 2 узлов?
7. Какие нормативные документы РФ (ФСТЭК) применимы к оценке безопасности робототехнических систем?
8. Чем отличается анализ безопасности Nav2 для одиночного робота и для мульти-роботной системы?
9. Какие меры защиты вы предложили бы для внедрения Nav2 в КИИ?
10. Как организовать автоматический аудит безопасности фреймворка навигации в CI/CD?

7. Требования к отчётам

Отчёты для ЛР №16 и ЛР №17 оформляются как отдельные документы (PDF), каждый содержит:

- Титульный лист.
- Цель и задачи.
- Для ЛР №16: описание источников данных, листинги скриптов сбора, структура JSON, шаблон Jinja2 (фрагмент), скриншоты сгенерированного HTML-отчёта.
- Для ЛР №17: описание фреймворка и версии, методология анализа, таблицы находок SAST с листингами кода, таблицы CVE с анализом влияния, результаты конфигурационного аудита, DFD-диаграмма (если применимо), STRIDE (если применимо).
- Индивидуальная часть: описание выполненного расширенного задания, листинги, скриншоты.
- Ответы на контрольные вопросы.
- Выводы и рекомендации.

7. Критерии оценивания

7.1. «Удовлетворительно» (базовый уровень):

- ЛР №16: Собраны данные минимум из 2 источников, создан агрегированный JSON, сгенерирован HTML-отчёт по шаблону.
- ЛР №17: Выполнен SAST и/или SCA анализ, идентифицированы уязвимости, проведён аудит конфигурации, оформлен итоговый документ.
- Отчёты оформлены, ответы на вопросы $\geq 60\%$.

7.2. «Хорошо» (средний уровень):

- Все требования базового уровня.
- Выполнено расширенное задание: сравнение инструментов, интеграция с Dependency-Track/DefectDojo, Grafana-дашборд, автоматизация в CI/CD.
- Ответы на вопросы $\geq 80\%$.

7.3. «Отлично» (повышенный уровень):

- Все требования среднего уровня.
- Проведено комплексное исследование / разработана автоматизированная система, архитектура, продемонстрирована работа.
- Ответы на вопросы полные и развёрнутые.

7.4. Понижение оценки:

- Отсутствие сгенерированного отчёта или итогового документа — работа не принимается.
- Недостаточное покрытие анализа (менее 5 находок для ЛР №17) — не выше базового уровня.
- Слабые ответы на вопросы — направляется на пересдачу.

Лабораторная работа №18

Комплексный проект: Обеспечение безопасности жизненного цикла инженерной системы

1. Цель

Продemonстрировать системное владение компетенциями ПК-8 и ПК-11 путём сквозного применения всех изученных в дисциплине практик безопасности для построения защищённого конвейера CI/CD и комплексного анализа защищённости автономной системы (симулятор БПЛА) с формированием итогового отчёта «Security Lifecycle Assurance».

2. Задачи

- Спроектировать и реализовать единый secure CI/CD пайплайн, интегрирующий SAST, SCA, DAST, сканирование контейнеров, подпись артефактов, формирование SBOM и compliance-проверки.
- Провести комплексный анализ защищённости симулятора автономной системы (БПЛА на базе PX4 SITL + ROS 2) с применением методологии STRIDE и инструментальных проверок.
- Автоматизировать сбор доказательств безопасности и сгенерировать сводный отчёт о соответствии требованиям (compliance report).
- Разработать итоговый документ «Security Lifecycle Assurance Report», объединяющий результаты всех этапов.

3. Архитектура комплексного проекта

Комплексный проект объединяет все лабораторные работы семестра в единую систему, охватывающую два ключевых направления:

Направление А: Secure CI/CD Pipeline (ПК-8)

Охватывает безопасность процесса разработки и сборки:

- SAST (Semgrep/CodeQL) — статический анализ кода.
- SCA (Trivy) — анализ зависимостей, формирование SBOM.
- DAST (OWASP ZAP) — динамический анализ API.
- Сканирование образов (Trivy) — проверка Docker-образов.
- Подпись артефактов (Cosign) — обеспечение целостности.
- Compliance as Code (OPA/Conftest) — проверка манифестов.
- Защита секретов (Masked variables, OIDC) — безопасность пайплайна.
- Автоматическая отчётность — агрегация результатов.

Направление Б: Защита автономной системы (ПК-11)

Охватывает безопасность целевой инженерной системы:

- Анализ угроз (STRIDE) — модель угроз для БПЛА.
- Защита каналов связи — MAVLink Signing и/или SROS 2 DDS Security.
- Безопасность OTA — подпись прошивки, защита от rollback.
- Анализ цепочки поставок — SBOM компонентов БПЛА, CVE в прошивке.
- Аудит безопасности GCS — проверка конфигурации, хранение секретов.
- Оценка безопасности фреймворка навигации — SAST/SCA для Nav2 (опционально).

Итоговый отчёт объединяет результаты обоих направлений и демонстрирует полный цикл обеспечения безопасности жизненного цикла инженерной системы.

4. Критерии оценивания итогового проекта

Проект является экзаменационной работой и оценивается по 100-балльной шкале с переводом в оценку.

4.1. Оценка «Удовлетворительно» (минимальный порог — 60 баллов):

- Реализован CI/CD пайплайн, включающий минимум 5 этапов безопасности (SAST + SCA + DAST + image scan + SBOM).
- Выполнен базовый анализ защищённости БПЛА (STRIDE, минимум 10 угроз, описание MAVLink-уязвимости).
- Сгенерирован итоговый отчёт, содержащий основные разделы.
- Продемонстрирована работа пайплайна (скриншоты) и анализа угроз.

4.2. Оценка «Хорошо» (75 баллов):

- Все требования базового уровня выполнены.
- Пайплайн включает подпись артефактов (Cosign), compliance-проверки (Conftest), защиту секретов (OIDC).
- Анализ БПЛА включает MAVLink Signing (демонстрация блокировки атаки), безопасное OTA (подпись + anti-rollback).
- Отчёт содержит сравнение инструментов, анализ CVE цепочки поставок, Hardening Guide для GCS.
- Качество оформления высокое, все разделы заполнены.

4.3. Оценка «Отлично» (90+ баллов):

- Все требования среднего уровня выполнены.
- Пайплайн полностью автоматизирован, включает auto-remediation, централизованный мониторинг (Grafana/ELK).
- Анализ БПЛА включает STRIDE с DREAD-оценкой, демонстрацию нескольких атак, формальную верификацию (или фаззинг), SROS 2 (если применимо).
- SBOM интегрирован с Dependency-Track, настроены уведомления.
- Отчёт содержит архитектурные схемы высокого качества, количественные метрики, прогнозы, маппинг на нормативку РФ (ФСТЭК, ГОСТ).
- Проект презентован с демонстрацией работающей системы.

4.4. Штрафы и вычеты:

- Отсутствие работающего CI/CD пайплайна — максимум 40 баллов.
- Отсутствие анализа угроз БПЛА — максимум 50 баллов.
- Нет доказательств (скриншотов, логов) для заявленных результатов — минус 20% за каждый раздел.
- Отчёт оформлен небрежно, отсутствуют обязательные разделы — минус 10–20 баллов.
- Код/конфигурации не приложены — минус 15 баллов.

4.5. Перевод баллов в оценку:

Баллы	Оценка
90–100	Отлично (5)
75–89	Хорошо (4)
60–74	Удовлетворительно (3)
<60	Неудовлетворительно (2)

5. Контрольные вопросы к защите проекта

1. Обоснуйте архитектуру вашего secure CI/CD пайплайна. Почему этапы расположены именно в таком порядке?
2. Какие наиболее критичные уязвимости были найдены на каждом этапе (SAST, SCA, DAST) и как они были устранены?
3. Как работает MAVLink Signing и почему он критически важен для безопасности БПЛА? Прдемонстрируйте разницу в захвате трафика.
4. Каков процесс безопасного OTA-обновления в вашей реализации? Как защищены от rollback-атаки?
5. Какие CVE были найдены в цепочке поставок прошивки БПЛА? Оцените их влияние на лётную безопасность.
6. Какие compliance-политики были реализованы? Покажите маппинг на требования стандартов (CIS, ФСТЭК).
7. Где хранятся секреты в вашем пайплайне? Как обеспечена их защита?
8. Объясните, как работает OIDC-аутентификация в вашем пайплайне. Почему это лучше статических ключей?
9. Какие меры по усилению безопасности наземной станции управления (GCS) вы предложили?
10. Оцените остаточные риски для вашей системы. Что ещё можно улучшить?

6. Организационные аспекты выполнения

- Формат работы: Индивидуальный или командный (до 2 человек). При командной работе вклад каждого должен быть явно отмечен в отчёте.
- Срок выполнения: 6 недель (с 8-й по 14-ю неделю семестра).
- Репозиторий проекта: Обязательно размещение в GitLab/GitHub с настроенным CI/CD и историей коммитов.
- Консультации: Еженедельные очные консультации с преподавателем.
- Защита проекта: Публичная презентация (15 минут доклад + 10 минут вопросы/демонстрация).

- Допуск к экзамену: Сдача всех 17 лабораторных работ + pre-defense (черновик отчёта) за 2 недели до защиты.

7. Примерные темы для индивидуальных проектов (вместо стандартного БПЛА)

Для обеспечения разнообразия студенты могут предложить альтернативные инженерные системы:

Вар.	Целевая система	Уникальный аспект безопасности
1	Роботизированный складской погрузчик	Безопасность ROS 2 + Industrial IoT
2	Беспилотное надводное судно (USV)	Защита гидроакустической связи, AIS-спуфинг
3	Рой дронов (3 единицы)	Безопасность swarm-коммуникаций, консенсус
4	Автономный медицинский робот	Защита персональных данных пациентов (152-ФЗ)
5	БПЛА для КИИ (энергетика)	Compliance: Приказ ФСТЭК №239, категорирование
6	Подводный дрон (ROV)	Защита оптического кабеля, гидроакустики
7	Автономный агро-дрон	Защита геоданных полей, целостность рецептур распыления
8	Беспилотный автомобиль (модель)	Automotive SPICE, ISO 21434
9	Марсоход (имитатор)	Защита от задержек связи (DTN), автономное принятие решений
10	Дрон для доставки грузов	Безопасность логистической платформы, защита адресов

Итоговый комплексный проект является кульминацией дисциплины «Безопасность жизненного цикла», позволяя студентам продемонстрировать системное владение всеми изученными практиками и готовность к реальной работе по обеспечению безопасности инженерных систем на всех этапах их жизненного цикла.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по организации и проведению самостоятельной работы по дисциплине
«Безопасность жизненного цикла»
для студентов направления подготовки
09.04.04 Программная инженерия Направленность (профиль)
«Разработка, мониторинг и управление жизненным циклом инженерных
систем»

Ставрополь 2026

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1.1. Цель самостоятельной работы

1.2. Формируемые компетенции (ПК-8, ПК-11)

1.3. Трудоёмкость самостоятельной работы

1.4. Планируемые результаты обучения

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

2.1. Темы для самостоятельного изучения

2.2. Рекомендуемая литература (доступная в РФ)

2.3. Ресурсы информационно-телекоммуникационной сети «Интернет»

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

3.1. Технологический стек (разрешенное в РФ ПО)

3.2. Порядок выполнения лабораторных работ

3.3. Рекомендации по выполнению сквозного проекта (ЛР №18)

4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ

5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

5.1. Критерии оценивания лабораторных работ

5.2. Критерии оценивания сквозного проекта (ЛР №18)

ВВЕДЕНИЕ

Методические указания для самостоятельной работы по дисциплине «Безопасность жизненного цикла» предназначены для студентов магистратуры, обучающихся по направлению подготовки 09.04.04 «Программная инженерия» (профиль «Разработка, мониторинг и управление жизненным циклом инженерных систем»).

Дисциплина «Безопасность жизненного цикла» является одной из ключевых дисциплин вариативной части образовательной программы и направлена на формирование у магистрантов системных знаний и практических навыков в области интеграции практик безопасности на всех этапах жизненного цикла разработки программных и программно-аппаратных систем, включая наземные и автономные (БПЛА, робототехника), обеспечение непрерывного соответствия (continuous compliance) требованиям безопасности и защиту автономных систем от киберугроз.

Целью самостоятельной работы является закрепление и углубление теоретических знаний, полученных на лекционных занятиях, а также формирование практических навыков применения инструментов и методологий безопасности (SAST, DAST, SCA, Supply Chain Security, OPA, анализ угроз для БПЛА) в рамках выполнения индивидуального сквозного проекта.

Настоящие методические указания разработаны в соответствии с:

Федеральным государственным образовательным стандартом высшего образования по направлению подготовки 09.04.04 «Программная инженерия»;

Учебным планом направления подготовки 09.04.04 «Программная инженерия» (профиль «Разработка, мониторинг и управление жизненным циклом инженерных систем»);

Рабочей программой дисциплины «Безопасность жизненного цикла»;

Положением об организации образовательного процесса на основе рейтинговой системы оценки знаний студентов в ФГАОУ ВО «СКФУ»;

Требованиями к использованию программного обеспечения, разрешенного на территории Российской Федерации.

Дисциплина базируется на знаниях, полученных при изучении курсов «Безопасность жизненного цикла» (1 семестр) - базовые понятия SAST/SCA/DAST, инструменты Semgrep, Trivy, OWASP ZAP, а также «Архитектура и проектирование высоконагруженных систем» и «Системная интеграция и управление конфигурациями».

1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1.1. Цель самостоятельной работы

Целью самостоятельной работы является формирование у магистрантов компетенций в области безопасной разработки, анализа уязвимостей, управления цепочками поставок и защиты автономных систем (БПЛА) через выполнение практических заданий и сквозного проекта.

1.2. Формируемые компетенции

В ходе выполнения самостоятельной работы и защиты сквозного проекта студент должен обладать следующими компетенциями и индикаторами:

ПК-8. Способен интегрировать практики безопасности на всех этапах ЖЦ

Код индикатора	Формулировка индикатора	Уровень освоения
ИД-5пк-8	Сканирует конфигурации инфраструктуры как кода (IaC) на соответствие политикам безопасности	Высокий
ИД-6пк-8	Интегрирует инструменты безопасности в CI/CD-пайплайны, настраивая автоматические шлюзы качества	Высокий
ИД-8пк-8	Обеспечивает непрерывное соответствие (continuous compliance) систем требованиям стандартов безопасности	Высокий

ПК-11. Способен проводить анализ угроз кибербезопасности для автономных систем

Код индикатора	Формулировка индикатора	Уровень освоения
ИД-1пк-11	Проводит анализ угроз кибербезопасности для автономных систем с использованием методологии STRIDE	Высокий
ИД-3пк-11	Обеспечивает защиту каналов связи автономных систем на основе анализа протоколов	Высокий

Код индикатора	Формулировка индикатора	Уровень освоения
ИД-5пк-11	Внедряет механизмы безопасных OTA-обновлений с проверкой целостности и подлинности прошивок	Высокий

1.3. Трудоёмкость самостоятельной работы

Вид работы	Часы
Изучение теоретического материала	24
Подготовка к лабораторным работам (№1-17)	30
Выполнение сквозного проекта (Лабораторная работа №18)	24
Подготовка к экзамену	12
Итого	90

1.4. Планируемые результаты обучения

	Задача	Формируемые компетенции
	Интегрировать SAST/SCA/DAST в CI/CD-пайплайн с Quality Gate	ПК-8 (ИД-6пк-8)
	Формировать SBOM и управлять цепочкой поставок	ПК-8 (ИД-8пк-8)

	Задача	Формируемые компетенции
	Настраивать безопасность контейнеров (Pod Security Standards)	ПК-8 (ИД-5ПК-8)
	Писать политики безопасности на Rego (OPA)	ПК-8 (ИД-8ПК-8)
	Проводить анализ угроз для БПЛА/роботов по STRIDE	ПК-11 (ИД-1ПК-11)
	Анализировать защищенность протоколов MAVLink/DDS	ПК-11 (ИД-3ПК-11)
	Организовывать безопасные ОТА-обновления с подписью	ПК-11 (ИД-5ПК-11)
	Настраивать continuous compliance для cloud-ресурсов	ПК-8 (ИД-8ПК-8)

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

2.1. Темы для самостоятельного изучения

№	Тема	Литература/ Ресурсы	Часы
	DevSecOps и Shift Left	[1, 2], GitLab CI Docs	3
	Supply Chain Security и SBOM	[3], Syft, CycloneDX Docs	3
	SAST/DAST инструменты	Semgrep, OWASP ZAP	2

№	Тема	Литература/ Ресурсы	Часы
		Docs	
	Безопасность контейнеров (PSA, Trivy)	Kubernetes Docs, Trivy Docs	2
	Compliance as Code (OPA/Regal)	OPA, Gatekeeper Docs	2
	Анализ угроз автономных систем (STRIDE)	[4], NIST SP 800-160	4
	Безопасность протоколов MAVLink и DDS	MAVLink Security, ROS 2 DDS Security	4
	Безопасные OTA-обновления	Cosign, TUF Docs	2
	Регуляторные требования в РФ	152-ФЗ, приказы ФСТЭК	2
	Итого		24

2.2. Рекомендуемая литература (доступная в РФ)

2.2.1. Перечень основной литературы:

1. Лаврищева, Е. М. Программная инженерия и технологии программирования сложных систем : учебник для вузов / Е. М. Лаврищева. — 2-е изд. — Москва : Юрайт, 2023. — 432 с. — ISBN 978-5-534-12866-6. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: <https://urait.ru/bcode/511473> (дата обращения: 15.04.2026).
2. Казарин, О. В. Надежность и безопасность программного обеспечения : учебное пособие для вузов / О. В. Казарин, И. Б. Шубинский. — Москва : Юрайт, 2024. — 342 с. — ISBN 978-5-

534-14565-8. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: <https://urait.ru/bcode/541253> (дата обращения: 15.04.2026).

2.2.2. Перечень дополнительной литературы:

1. Ехлаков, Ю. П. Управление жизненным циклом программных продуктов: модели и алгоритмы : монография / Ю. П. Ехлаков, Д. Н. Бараксанов, Е. А. Янченко. — Томск : ТУСУР, 2013. — 196 с. — ISBN 978-5-86889-661-3. — Текст : электронный // IPR SMART : [сайт]. — URL: <https://www.iprbookshop.ru/72135.html> (дата обращения: 15.04.2026). — Режим доступа: для авторизир. пользователей.
2. Буканин, В. А. Вопросы безопасности программного обеспечения приборов и систем : учебное пособие / В. А. Буканин, А. Н. Иванов, В. Н. Павлов, А. О. Трусов. — Санкт-Петербург : СПбГЭТУ «ЛЭТИ», 2020. — 111 с. — ISBN 978-5-7629-2641-6. — Текст : непосредственный.

2.3. Ресурсы информационно-телекоммуникационной сети «Интернет»

№	Наименование ресурса	Адрес	Описание
1	Научная электронная библиотека eLIBRARY.RU	https://www.elibrary.ru	Научные публикации
2	ЭБС «Университетская библиотека онлайн»	http://www.biblioclub.ru	Электронные учебные издания

№	Наименование ресурса	Адрес	Описание
3	ЭБС IPR SMART	http://www.iprbookshop.ru	Электронные учебные издания
4	GitLab Documentation (зеркало)	https://docs.gitlab.ru	Документация GitLab CI/CD
5	Semgrep Documentation	https://semgrep.dev/docs	Статический анализ
6	Trivy Documentation	https://aquasecurity.github.io/trivy	Сканирование уязвимостей
7	Open Policy Agent (OPA)	https://www.openpolicyagent.org/docs	Политики как код
8	PX4 Development Guide	https://docs.px4.io/main/en/	Разработка автопилота
9	ROS 2 Security (SROS 2)	https://docs.ros.org/en/rolling/Tutorials/Security.html	Безопасность ROS 2
10	Habr (DevOps / Security)	https://habr.com/ru/hub/devops/	Профильные статьи

№	Наименование ресурса	Адрес	Описание
			на русском

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

3.1. Технологический стек (разрешенное в РФ ПО)

№	Наименование ПО	Лицензия и статус в РФ	Назначение
1	Git	Open Source (GNU GPL v2)	Система контроля версий
2	GitLab Community Edition	Open Source (MIT)	CI/CD-платформа (self-hosted)
3	Docker CE	Freemium (базовый функционал — Apache 2.0)	Контейнеризация
4	Minikube / Kind	Open Source (Apache 2.0)	Локальный Kubernetes
5	Semgrep	LGPL 2.1	Статический анализ (SAST)
6	Trivy	Apache 2.0	Сканирование уязвимостей

	Наименование ПО	Лицензия и статус в РФ	Назначение
7	OWASP ZAP	Apache 2.0	Динамический анализ (DAST)
8	Syft / CycloneDX	Apache 2.0	Генерация SBOM
9	Cosign (Sigstore)	Apache 2.0	Подпись артефактов
10	OPA/ Conftest	Apache 2.0	Политики как код
11	PX4 SITL / Gazebo	BSD	Симуляция БПЛА
12	Python 3.x	OSI-approved	Язык программирования
13	Visual Studio Code	Free	Среда разработки
14	GitFlic	Бесплатный отечественный сервис	Альтернатива GitHub/GitLab Cloud
15	«Графиня»	В отечественного ПО	Альтернатива Grafana (опционально)

3.2. Порядок выполнения лабораторных работ

1. Ознакомиться с теоретическим обоснованием работы.
2. Выбрать вариант задания в соответствии с указанием преподавателя.
3. Разработать решение (скрипты, конфигурации, Docker Compose).
4. Выполнить проверку в тестовой среде (локально или в CI).

5. Оформить отчёт в соответствии с требованиями.
6. Защитить работу перед преподавателем (демонстрация работы, ответы на вопросы).

3.3. Рекомендации по выполнению сквозного проекта (ЛР №18)

Сквозной проект выполняется на протяжении всего семестра параллельно с лабораторными работами. Каждая лабораторная работа добавляет новый компонент в проект.

Тема проекта: «Разработка secure CI/CD-пайплайна и анализ безопасности автономной системы».

Требования к проекту:

1. Интеграция SAST (Semgrep), SCA (Trivy), DAST (OWASP ZAP) в CI/CD.
2. Формирование SBOM (CycloneDX) и подпись артефактов (Cosign).
3. Настройка политик безопасности контейнеров (Pod Security Standards).
4. Написание политик Rego для проверки манифестов.
5. Проведение анализа угроз для симулятора дрона/робота (STRIDE).
6. Оценка защищенности каналов связи (MAVLink/DDS).
7. Формирование итогового отчёта «Security Lifecycle Assurance».

4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ

1. Что такое DevSecOps и «Shift Left»? Какие инструменты реализуют этот подход?
2. Что такое SBOM? Какие форматы (SPDX, CycloneDX) и инструменты генерации вы знаете?
3. Чем SAST отличается от DAST? Какие инструменты используются для каждого типа?

4. Как настроить Quality Gate в CI/CD на основе результатов SAST/SCA?
5. Какие политики безопасности контейнеров существуют в Kubernetes (Pod Security Standards)?
6. Что такое Open Policy Agent (OPA)? На каком языке пишутся политики?
7. Какие типичные уязвимости существуют в протоколах MAVLink и DDS?
8. Как защитить канал связи БПЛА от атак типа replay и подмены данных?
9. Что такое безопасное ОТА-обновление? Какие механизмы обеспечивают целостность и подлинность?
10. Как работает подпись артефактов с помощью Cosign?
11. Каковы основные требования 152-ФЗ к защите информации?
12. Что такое методология STRIDE? Какие угрозы она описывает?
13. Как составить модель угроз для автономной системы (БПЛА)?
14. Какие инструменты используются для continuous compliance (Cloud Custodian)?
15. Как интегрировать проверки безопасности в CI/CD без статических ключей (OIDC)?

5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

5.1. Критерии оценивания лабораторных работ

Оценка	Критерии
5 (отлично)	Задание выполнено полностью, инструменты безопасности интегрированы корректно, пайплайн работает без ошибок, студент демонстрирует понимание уязвимостей и обосновывает выбор

Оценка	Критерии
	мер защиты
4 (хорошо)	Задание выполнено полностью, но есть небольшие замечания (неоптимальная конфигурация, неполный отчёт, не все уязвимости проанализированы)
3 (удовлетворительно)	Задание выполнено с помощью преподавателя, есть ошибки, исправляемые с наводящими вопросами
2 (неудовлетворительно)	Задание не выполнено или выполнено неверно, студент не понимает принципов безопасности

5.2. Критерии оценивания сквозного проекта (ЛР №18)

Критерий	Вес, %	Отлично (5)	Хорошо (4)	Удовл. (3)	Неуд. (2)
Интеграция SAST/SCA/DAST	20 %	Все 3 типа + Quality Gate	2 типа	1 тип	Нет
Supply Chain Security	15 %	SBOM + подпись (Cosign)	Только SBOM	Только подпись	Нет
Безопасность контейнеров	10 %	Trivy + PSA	Только Trivy	Только PSA	Нет

Критерий	Вес, %	Отлично (5)	Хорошо (4)	Удовл. (3)	Неуд. (2)
Compliance as Code (OPA)	10 %	3+ политики	1-2 политик и	Заготовка	Нет
Анализ угроз (STRIDE)	15 %	Полная модель угроз	DFD + STRIDE	Только список угроз	Нет
Безопасность каналов связи	10 %	Анализ MAVLink/ DDS	Только анализ	Общее описание	Нет
Безопасные OTA-обновления	10 %	Подпись + верификация	Только подпись	Только эмуляция	Нет
Качество отчёта и ответы	10 %	Полный, структ., скриншоты	Небольшие недочёты	Минимальный	Отсутствует

Шкала перевода баллов в оценку:

Суммарный балл	Оценка
90-100%	5 (отлично)
75-89%	4 (хорошо)

Суммарный балл	Оценка
60-74%	3 (удовлетворительно)
< 60%	2 (неудовлетворительно)