Университетский лицей №1523 Предуниверситария НИЯУ МИФИ

Проект ‹‹Безопасное развертывание кластеров в инфраструктуре››

Ученик Шпак Андрей Владимирович _____11_____

Москва - 2026

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящем отчете о проекте применяют следующие термины с соответствующими определениями:

Кластер Kubernetes — совокупность компонентов контрольной плоскости и рабочих узлов, обеспечивающих запуск и управление контейнеризованными приложениями.

Узел (Node) — физическая или виртуальная машина, на которой запускаются Pod-ы и компоненты контейнерного рантайма.

Под (Pod) — минимальная развертываемая единица Kubernetes; группа одного или нескольких контейнеров с общими сетевыми и дисковыми ресурсами.

Контейнер — изолированный процесс и файловая система, запускаемые контейнерным рантаймом на узле.

Сервисный аккаунт (ServiceAccount) — учетная запись Kubernetes, от имени которой Pod обращается к kube-API.

Мисконфигурация — формально корректная настройка Kubernetes-объектов, которая увеличивает риск нарушения конфиденциальности, целостности или доступности.

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящем отчете о проекте применяют следующие сокращения и обозначения:

K8s — Kubernetes.

RBAC — Role-Based Access Control; управление доступом на основе ролей.

SA — ServiceAccount; сервисный аккаунт.

CNI — Container Network Interface; плагин сетевого взаимодействия контейнеров.

CRI — Container Runtime Interface; интерфейс взаимодействия kubelet и контейнерного рантайма.

PSS/PSA — Pod Security Standards / Pod Security Admission; стандарты и механизм контроля безопасности Pod-ов.

IMDS — Instance Metadata Service; сервис метаданных облачной виртуальной машины/инстанса.

IAM — Identity and Access Management; управление идентичностями и правами в облаке.

eBPF — extended Berkeley Packet Filter; технология ядра Linux для наблюдаемости и контроля.

LSM — Linux Security Modules; подсистема безопасности ядра Linux.

YAML — текстовый формат описания манифестов Kubernetes.

JSON — формат машинно-читаемых отчетов.

SARIF — Static Analysis Results Interchange Format; формат обмена результатами статического анализа.

ВВЕДЕНИЕ

Информационные системы всё чаще разворачиваются в контейнерной среде, а Kubernetes стал де-факто стандартом оркестрации. Это повышает масштабируемость и скорость доставки, но одновременно расширяет поверхность атаки: ошибка в манифесте, утечка токена сервис-аккаунта, лишние Linux-capabilities или неверные сетевые политики способны привести не к падению одного сервиса, а к компрометации всего кластера. Поэтому тема практической проверки безопасности «кубера» критична для индустрии.

Классические меры (TLS, флаги HttpOnly/SameSite, базовые сканеры образов) полезны, но недостаточны. Угрозы современного кластера многослойны: XSS/утечка сессионных cookie может открыть доступ к API приложения; изнутри — чрезмерные RBAC-права или automount сервисного токена дадут злоумышленнику возможность перемещения по кластерам и пространствам имён; на уровне узла — слабые `sysctl`, доступ к cloud-metadata, привилегированные контейнеры и `hostPath` создают прямые пути к эскалации привилегий. Даже аккуратно собранные образы остаются рисковыми, если запущены с `runAsRoot`, без seccomp/AppArmor, с writable rootfs и `NET_RAW`.

Проблема усугубляется тем, что многие опасные конфигурации «легальны» для Kubernetes — кластер их принимает, а эксплуатация часто незаметна до инцидента. Требуется простой и воспроизводимый способ регулярно проверять контейнеры и узлы именно изнутри кластера, фиксировать признаки компрометации и немедленно подсвечивать мисконфиги, которые открывают путь к атаке.

Цель проекта — разработать in-cluster CLI-инструмент для Kubernetes, который автоматически проверяет контейнеры и узлы на уязвимости и небезопасные настройки, способные привести к компрометации кластера, формирует отчёт с приоритизацией рисков и выдаёт практические рекомендации по исправлению.

В рамках проекта:

- формализуются ключевые угрозы для k8s: утечки секретов и токенов, эскалация через RBAC и привилегии контейнеров, небезопасные сетевые границы, уязвимые узлы и доступ к метаданным облака;

- анализируются существующие подходы и ограничения (сканеры образов, admission-политики, Pod Security Standards) применительно к задачам постфактум-аудита в рантайме;
- проектируется и реализуется CLI с двумя режимами запуска (Job/Pod с минимальными RBAC-правами и DaemonSet-агент для глубоких node-чеков), источниками данных из kube-API, CRI и `/proc`;
- разрабатывается набор детектов: опасные `securityContext` и монтирования, automount SA-токена, секреты в env/файловой системе, отсутствие seccomp/AppArmor/PSa(PSS), открытые NodePort/ingress, доступ к сервису метаданных облака (Instance Metadata Service, IMDS) узла/BM, откуда можно получить данные об инстансе и временные облачные учётные данные (IAM-роли/токены), слабые `sysctl` и лишние capabilities;
- готовятся человекочитаемые и машинные отчёты (таблица + JSON/SARIF) с уровнями риска и YAML-патчами/советами по фиксам

Практическая ценность — показать, что даже при частичном успехе атакующего (например, краже сессионного токена или доступе в Pod) дальнейшая эксплуатация может быть остановлена благодаря сегментации сети, строгому RBAC, безопасным параметрам запуска, контролю доступа к узлам и выявлению опасных артефактов в контейнерах. Это повышает стоимость атаки и снижает её вероятность.

Ожидаемый результат — воспроизводимый прототип (бинарь + Helm-чарт), чек-лист проверок с идентификаторами и ссылками на исправления, а также отчёт о применении инструмента к тестовому кластеру с метриками обнаружений и снижением риска после внедрения рекомендаций.

1 ИССЛЕДОВАНИЕ ПРОБЛЕМЫ

1.1 Предметная область и базовые понятия

Современные сервисы всё чаще работают в контейнерах и управляются Kubernetes. Это даёт масштабируемость и скорость релизов, но усложняет безопасность: поверх «классической» ИБ появляются уровни контейнерного рантайма (CRI), оркестратора, сетевой плоскости CNI, реестров образов и облачной инфраструктуры (IMDS, IAM).

Kubernetes валидирует схему манифестов, а не риск, поэтому «легальные, но опасные» конфигурации (privileged, runAsRoot, hostPath, automount сервисного токена, отсутствие PSS/seccomp/AppArmor, открытый egress/NodePort, слабый hardening узлов) попадают в прод. Достаточно foothold в одном Pod (уязвимость приложения, утёкший токен), чтобы выполнить боковое перемещение по кластеру, получить секреты/токены и, при наличии выхода наружу, обратиться к сервису метаданных облака и расширить компрометацию

1.2 Описание проблемы

В Kubernetes безопасность часто строится на предположении «манифест валиден — значит безопасен». Кластер принимает множество рискованных конфигураций (привилегированные контейнеры, automount сервисного токена, `hostPath`, отсутствие сетевых политик, слабый hardening узлов). Эти настройки не считаются ошибками оркестратором, но в сумме образуют **реальные пути компрометации кластера**: от утечки секретов и эскалации привилегий до выхода за периметр в облако.

Типичный сценарий. В продакшене запускаются Pods с удобными, но опасными параметрами и узлы без жёстких ограничений. Проверки на CI и admission-политики (если есть) не отражают фактическое состояние рантайма. В результате атакующий, получив внутри одного Pod минимальную foothold (через уязвимость приложения или утёкший токен), может перемещаться по кластеру и/или получить доступ к внешним ресурсам.

Где возникает (классы источников риска):

1. **Манифесты Pods/Deployments.**

- `privileged`, `runAsRoot`, `allowPrivilegeEscalation`, `hostNetwork/hostPID`, монтирование `hostPath`, writable rootfs, `NET_RAW`;
- automount сервисного токена; секреты/ключи в env-переменных и слоях FS; отсутствие seccomp/AppArmor/Pod Security Standards.

2. **RBAC и сервисные аккаунты.**
   - Чрезмерные роли/cluster-role binding'и, токены с широкими правами, отсутствие ограничений по namespace/verb/resource.
3. **Сеть.**
   - Нет deny-by-default NetworkPolicy; открытые NodePort/ingress; разрешён произвольный egress, в т.ч. к **сервису метаданных облака (IMDS)** — источнику временных облачных учётных данных.
4. **Узлы и рантайм (Node/CRI).**
   - Небезопасные `sysctl`, лишние capabilities у демонов, доступность облачных метаданных с узла, слабая изоляция контейнеров.
5. **Цепочка поставки (supply chain).**
   - Неаудированные образы, отсутствие подписей/SBOM, уязвимости в слоях, но запуск с завышенными привилегиями усиливает последствия.

1.3 Актуальность проблемы

Kubernetes стал стандартом для веб-, data- и финтех-платформ. Рост числа микросервисов и кластеров приводит к тому, что ошибка конфигурации затрагивает не один сервер, а целые контуры. На практике часто встречается:

- **Долгоживущие и широкие права** сервис-аккаунтов, automount токенов и избыточные RBAC-роли.
- **Опасные параметры запуска подов:** `privileged`, `runAsRoot`, `hostPath`, `NET_RAW`, writable rootfs, отсутствие seccomp/AppArmor и PSS.
- **Отсутствие сетевой изоляции:** нет deny-by-default egress/ingress, открытые NodePort/ingress; доступ к **сервису метаданных облака (IMDS)** из рабочего пода.
- **Слабый hardening узлов:** небезопасные `sysctl`, лишние capabilities рантайма, общий доступ к файловой системе узла.

- **Разрыв между сборкой и рантаймом:** сканеры образов и admission-политики не покрывают уже запущенные поды/ноды и их фактические артефакты (секреты в FS, реальные сетевые пути).

В совокупности это создаёт **реальные пути компрометации кластера**: утечка секретов и токенов, эскалация привилегий, боковое перемещение по namespace/кластеру и выход за периметр в облако через IMDS. Инциденты приводят к простоям, потере данных и финансовым/репутационным рискам.

С учётом массового внедрения Kubernetes и требований комплаенса (минимизация привилегий, Zero-Trust, регулярный аудит конфигураций) актуальность темы оправдывается и требованиями олимпиады, и реальной ситуацией на рынке ИБ.

1.4 Анализ аналогичных решений

Аудит конфигураций/кластера

- **kube-bench** — прогоняет CIS-бенчмарк против кластера/узлов (контрольные проверки безопасности Kubernetes). Не ищет секреты в FS контейнеров, фокус на соответствие CIS.
- **kubeaudit (Shopify)** — CLI для поиска мисконфигов: privileged, runAsRoot, capabilities, readonly FS и т.п.; работает по манифестам и «вживую» по кластеру. Лёгкий, но без глубокой узловой проверки.
- **Polaris (Fairwinds)** — валидатор best-practices, есть CLI/дашборд, умеет авторемедиацию по политикам. Больше про «гигиену» манифестов.
- **KubeLinter (StackRox/Red Hat)** — статический линтер YAML/Helm на предмет небезопасных настроек. Не смотрит рантайм и узлы.

In-cluster сканеры уязвимостей/мисконфигов

- **Trivy Operator** — оператор, который автоматически сканит Pods/Images и складывает отчёты в CRD (vuln/config/compliance). Хорош для «потока», меньше внимания узловым sysctl/IMDS.
- **Kubescape** — платформа для риска/комплаенса (NSA-CISA, CIS, MITRE), сканирует манифесты и живые кластеры, есть приоритизация. Фокус шире (политики/комплаенс), не специализирован на узловых низкоуровневых проверках.

«Оборонительная разведка» и периметр-тест

- **kube-hunter** — активное «пентестирование» кластера (снаружи/изнутри Pod) для поиска открытых сервисов/дырок. Про уязвимости периметра, не про системный аудит Pod/Node.

Рантайм-детекция/энфорсмент

- **Falco** — eBPF/драйвер-агент для детекции аномалий в реальном времени по системным вызовам/событиям Kubernetes. Не делает «скан отчёт», а ловит поведение.
- **Cilium Tetragon** — eBPF-наблюдаемость и рантайм-энфорсмент: события процессов/файлов/сети, реакции в реальном времени.
- **KubeArmor** — рантайм-политики через eBPF+LSM (AppArmor/SELinux), телеметрия и блокировки поведения.
- **RunTime Radar** - https://runtimeradar.com/

Итог. Уже есть множество средств для защиты Kubernetes, но многие из них громоздкие или дорогие, узкоспециализированные и не дают цельной картины риска. Предлагаемый инструмент — лёгкий **in-cluster CLI одним бинарём** с **узловыми проверками** (sysctl, capabilities демонов, настройки IMDS), **поиском секретов в файловых системах контейнеров** и **явными YAML-фикспатчами**. Он включает **специальные детекты доступа к сервису метаданных облака (IMDS)** и **egress-рисков**, которые у большинства аудиторских решений покрываются частично. *Выход — **единый risk-centric отчёт** (цепочка exploit-пути → приоритет → готовый патч), вместо россыпи несвязанных сигналов*. Инструмент открытый, гибко настраиваемый, разворачивается поверх любого кластера (без vendor lock-in) и даёт быстрый практический эффект.

1.5 Аудитория проекта

Проект будет актуален для следующей аудитории: разработчики, SRE/DevOps, специалисты ИБ

2 РЕШЕНИЕ ПРОБЛЕМЫ

2.1 Формализация проблемы

Пусть K — кластер Kubernetes, N — множество узлов, P — множество подов. Для каждого p ∈ P зададим вектор признаков конфигурации x_p (privileged, runAsRoot, caps, hostPath, seccomp, AppArmor, tokenMount, netPolicy и др.). Введём набор детектов $D = \{d_i\}$, где $d_i(x_p) \in \{0,1\}$ сигнализирует о риске/мисконфиге, и веса $w(d_i) \geq 0$. Риск пода: $R(p) = \sum w(d_i) \cdot d_i(x_p)$; риск узла R(n) определяется по вектору y_n (sysctl, доступ к IMDS, демоны с правами и пр.). Совокупный риск кластера: $R(K) = \sum_p R(p) + \sum_n R(n)$. Цель — минимизация R(K) через набор патчей Δ к манифестам/политикам и харденинг узлов при сохранении работоспособности сервисов.

2.2 Описание алгоритма решения проблемы

1) Сбор фактов: kube-API (Pods/Deployments/SA/RBAC/NetworkPolicy), CRI и /proc в контейнерах (секреты, capabilities, writable rootfs), узлы (sysctl, reachability IMDS).

2) Нормализация и графы: RBAC-граф (SA→Role/ClusterRole→verbs/resources), матрица сетевой достижимости (ingress/egress, пути к IMDS), унификация securityContext.

3) Детектирование и скоринг: прогон наборов D, расчёт R(p), R(n), построение цепочек эксплуатации (foothold → RBAC/секреты → узел/облако).

4) Рекомендации: для каждого детекта — YAML-патч/совет (seccomp/AppArmor, automountServiceAccountToken=false, egress-политики, ограничение RBAC и т. п.), отчёты HTML/JSON.

5) Верификация: повторный прогон на тестовом неймспейсе и сравнение R(K) до/после.

Псевдокод:

```
facts = collect_from_api_and_runtime()
graphs = build_rbac_and_network_graphs(facts)
findings = run_detectors(facts, graphs)
scores   = score(findings, context=graphs)
patches  = map_findings_to_patches(findings)
report(scores, findings, patches)
```

2.3 Демонстрация интерфейса приложения

CLI + HTML-отчёт.

• Запуск в кластере (Job): kubectl apply -f audit-job.yaml

• Глубокая проверка узлов (DaemonSet): helm install cluster-audit charts/daemonset

• Выгрузка результатов: kubectl cp <pod>:/out/report.html ./report.html

2.4 Описание существующих алгоритмов

• Правил-бэйз проверки securityContext (privileged, runAsNonRoot, allowPrivilegeEscalation, writable rootfs, hostPath, NET_RAW, seccomp/AppArmor).

• Графовый анализ RBAC и выявление избыточных прав.

• Анализ сетевой достижимости (ingress/egress), детект путей к IMDS.

• Детекция секретов (env/файлы/метаданные образов).

• Аддитивный риск-скоринг с усилением для опасных сочетаний (например, privileged + hostPath).

2.5 Нормативно-правовые аспекты решения проблемы

• Соответствие внутренним политикам и бенчмаркам (минимальные привилегии, Zero-Trust, локальные профили безопасности).

• Учёт требований к персональным данным и журналированию; отчёты не содержат лишних персонализирующих сведений.

• Отчётность оформляется по учебным требованиям (структура разделов, демонстрационные материалы, список литературы).

3 ОПИСАНИЕ РЕАЛИЗАЦИИ

3.1 Техническое задание

Разрабатываемый продукт — in-cluster инструмент аудита безопасности Kubernetes-кластера (далее — Инструмент), поставляемый в виде:

• статически собранного CLI-бинаря (Linux amd64/arm64);

• контейнерного образа для запуска внутри кластера;

• Helm-чарта (и набора манифестов), обеспечивающего воспроизводимое развертывание.

Цель разработки: снизить риск компрометации кластера из-за ошибок конфигурации, обеспечив регулярный автоматизированный аудит состояния кластера и выдачу практических рекомендаций по исправлению.

Функциональные требования:

1) Сбор данных из Kubernetes API (client-go): Namespaces, Pods, Deployments/StatefulSet/DaemonSet, Jobs/CronJobs, Services, Ingress, NetworkPolicy, ServiceAccounts, Roles/ClusterRoles, RoleBinding/ClusterRoleBinding, Nodes (при наличии прав).

2) Проверка конфигураций Pod/контейнеров (securityContext): privileged, runAsNonRoot/runAsUser, allowPrivilegeEscalation, readOnlyRootFilesystem, hostNetwork/hostPID/hostIPC, hostPath, Linux capabilities, seccompProfile, AppArmor, автомонтирование токена ServiceAccount.

3) Анализ RBAC: построение графа эффективных прав (SA → bindings → roles → verbs/resources) и выявление опасных комбинаций (wildcard *, доступ к secrets, pods/exec, nodes, rolebindings/clusterrolebindings, возможность создавать privileged Pods и т. п.).

4) Сетевые проверки: наличие deny-by-default (ingress/egress) на уровне namespace/кластера, обнаружение открытых NodePort/Ingress, оценка потенциально опасного egress (включая доступ к IMDS при его наличии).

5) Узловые проверки (опционально, в режиме DaemonSet): базовый hardening узла (ключевые sysctl/параметры), доступность IMDS с узла и из Pod-ов, параметры kubelet и рантайма (по доступным файлам конфигураций).

6) Детекция «секретов» без раскрытия значений: поиск признаков хранения паролей/токенов в env, ConfigMap, аргументах команд, файлах внутри контейнера (на основе шаблонов/ключевых слов) с маскированием чувствительных данных в отчете.

7) Приоритизация: каждому срабатыванию присваивается уровень риска (LOW/MEDIUM/HIGH/CRITICAL) и описывается сценарий эксплуатации.

8) Генерация отчетов: HTML (для человека), JSON и/или XML (для интеграций), SARIF (для CI-пайплайнов статанализа). Отчет должен содержать идентификатор проверки, объект, доказательство (evidence), риск и конкретную рекомендацию (в т. ч. пример YAML-патча).

Требования к интерфейсу и эксплуатации:

• Инструмент запускается как внутри кластера (Job/Pod), так и с рабочей станции при наличии kubeconfig.

• CLI должен поддерживать: выбор namespace, порог риска для ненулевого exit-code, включение/отключение детектов, вывод в указанную директорию, режим «read-only» по умолчанию.

• Инструмент не должен изменять объекты в кластере без явного флага (--apply-patch=false ,по умолчанию).

Требования к безопасности и правам:

• Принцип минимально необходимых привилегий: отдельные роли для режима «классический аудит» и режима «node-checks».

• В отчете запрещено выводить реальные значения секретов; допускается выводить только факт обнаружения и место (путь/переменная) с маскированием.

Нефункциональные требования:

• Совместимость с актуальными версиями Kubernetes; использование стабильных API-групп (по возможности).

• Эффективность: ограничение потребления CPU/RAM, линейная сложность по числу объектов, таймауты на сетевые проверки (например, IMDS).

• Надежность: корректная работа при частичных ошибках доступа (например, нет прав на Nodes) — с пометкой в отчете, какие проверки пропущены.

Критерии приемки:

• Развертывание через Helm в тестовом кластере без ручных донастроек;

• Генерация отчета в заданных форматах;

• Обнаружение набора заранее подготовленных мисконфигураций на стенде и выдача рекомендаций по исправлению;

• Отсутствие влияния на работоспособность кластера (read-only режим, отсутствие деградации).

3.2 Описание идеи проекта

Идея проекта состоит в создании простого и воспроизводимого «аудита изнутри кластера», который помогает DevOps/SRE и специалистам ИБ находить опасные настройки там, где они реально эксплуатируются — в рантайме и в текущем состоянии кластера.

Ключевой принцип — «risk-centric» подход: Инструмент не просто перечисляет отдельные нарушения, а объясняет, почему они опасны (какой путь атаки открывают), и предлагает конкретное исправление (патч/рекомендацию) с учетом контекста (RBAC + сеть + параметры запуска Pod/Node).

В отличие от чисто статических линтеров манифестов, Инструмент ориентирован на проверку фактического состояния: какие Pods реально запущены, какие ServiceAccount используются, какие роли фактически назначены, и есть ли сетевые пути к критичным точкам (например, к метаданным облака — IMDS).

3.3 Используемые технологии

Для реализации выбраны следующие технологии:

• Go (Golang) — основной язык разработки. Причины: нативная экосистема Kubernetes, высокая производительность, удобная конкурентность, возможность статической сборки одного бинаря для Linux, простота доставки в air-gapped средах.

• client-go — официальный SDK Kubernetes для доступа к kube-API (List/Watch), получения объектов и метаданных.

• Helm — упаковка и развертывание Инструмента в кластер (Job/DaemonSet, RBAC-манифесты, ConfigMap с настройками).

• HTML templates (html/template) — генерация человекочитаемого отчета без внешних зависимостей.

• JSON/XML/SARIF — форматы машинных отчетов для интеграции с пайплайнами и другими системами.

• Минимальные зависимости рантайма: без внешних БД и без требований к интернет-доступу.

Тестирование и стенд:

• kind/minikube — локальный стенд Kubernetes для воспроизводимых тестов;

• набор «плохих» манифестов и намеренно небезопасных workload’ов для проверки детектов;

• unit-тесты правил (детектов) и «golden-files» для проверки стабильности отчетов.

3.4 Принцип работы инструмента

Архитектура решения (логические компоненты):

1) Collector — сбор фактов из kube-API (объекты кластера, конфигурации workload’ов, RBAC, сеть).

2) Node-Agent (опционально) — DaemonSet-агент для узловых проверок: sysctl/конфиги/доступность IMDS и др.

3) Detector Engine — модуль набора проверок (правила), который на вход получает факты и графы и возвращает список находок (findings).

4) Scoring — модуль приоритизации (severity) с учетом контекста (например, privileged + hostPath → выше риск).

5) Report Generator — формирование HTML/JSON/XML/SARIF отчетов и рекомендаций по исправлению.

Общий алгоритм (упрощенно):

```
facts = collect_from_api_and_agents()
graphs = build_rbac_graph(facts) + build_network_model(facts)
findings = run_detectors(facts, graphs)
findings = enrich_with_context_and_score(findings, graphs)
report = render_reports(findings)
```

Особенность обхода кластера: проверки выполняются в режиме «best-effort». Если доступ к некоторым ресурсам отсутствует (например, Nodes), Инструмент фиксирует это в отчете и продолжает работу по доступным данным.

Структура одной находки (finding) в JSON (пример):

{
"checkId": "K8S-RBAC-001",
"severity": "HIGH",
"resource": {"kind": "ServiceAccount", "namespace": "dev", "name": "build-bot"},
"title": "Избыточные права: доступ к secrets",
"evidence": "ClusterRoleBinding ... предоставляет verbs=get,list на secrets",
"risk": "Компрометация Pod → чтение secrets → эскалация в кластере",
"recommendation": "Ограничить роль по namespace и убрать доступ к secrets. Пример патча: ..."
}

3.5 Доказательство работоспособности и методика тестирования

Работоспособность Инструмента подтверждается двумя видами верификации: логической (формальной на уровне правил) и экспериментальной (на стенде).

1) Верификация правил (детектов):

• каждая проверка формализуется как предикат над фактами из kube-API/узла (например: privileged=true ∨ hostPath!=∅);

• для каждого правила разрабатываются unit-тесты на заранее заданных объектах (манифестах) и проверяется, что правило срабатывает/не срабатывает корректно;

• для генератора отчетов используются «golden-tests»: фиксируется эталонный JSON/SARIF и проверяется отсутствие регрессий.

2) Экспериментальное тестирование на тестовом контуре:

• разворачивается локальный кластер (например, kind) с несколькими узлами;

• создаются контролируемые мисконфигурации: privileged Pods, hostPath, automount токена, отсутствующие NetworkPolicy, избыточные RoleBinding/ClusterRoleBinding, открытые NodePort;

• запускается Инструмент в режиме Job и (при необходимости) DaemonSet;

• результаты сравниваются с «истиной» (списком внедренных мисконфигов) и инструментами-аналогами (kube-bench, trivy/kubescape) для перекрестной проверки.

Метрики эффективности (планируемые к фиксации в отчете):

• полнота (recall): доля обнаруженных внедренных мисконфигураций;

• точность (precision): доля корректных срабатываний среди всех срабатываний;

• время выполнения и потребление ресурсов;

• снижение суммарного риск-скора R(K) после применения рекомендаций и повторного запуска.

3.6 Перспективы развития проекта

Перспективы развития проекта:

• расширение базы детектов (в т. ч. под конкретные облака и CNI), добавление профилей (baseline/strict) и кастомных правил;

• интеграция с CI/CD: автоматический запуск, публикация SARIF в систему анализа кода, «quality gate» по порогу риска;

• режим непрерывного контроля: периодический запуск (CronJob) и хранение истории результатов;

• интеграция с admission-контроллерами (PSA/OPA Gatekeeper/Kyverno) для предотвращения появления опасных конфигураций;

• улучшение анализа сетевой достижимости (учет IngressClass, сервис-mesh) и построение цепочек атаки;

• UI-панель (дашборд) и экспорт в SIEM/лог-платформы (например, через JSON-выгрузку).

ЗАКЛЮЧЕНИЕ

В ходе работы была исследована предметная область безопасности Kubernetes-кластеров и выделены типовые пути компрометации, связанные с мисконфигурациями Pod/Node, избыточным RBAC и отсутствием сетевой сегментации.

Сформулированы цель и задачи проекта, предложена модель оценки риска R(K) и описан алгоритм работы инструмента: сбор фактов из kube-API и узлов, построение контекстных графов (RBAC/сеть), выполнение набора детектов и генерация отчетов с рекомендациями.

Подготовлено техническое задание на разработку in-cluster CLI-инструмента, определены режимы развертывания (Job/DaemonSet), форматы отчетов (HTML/JSON/XML/SARIF) и требования к минимальным RBAC-правам.

Ограничения текущего решения: часть узловых проверок требует DaemonSet-агента с доступом к хост-файлам; возможны ложные срабатывания при использовании нестандартных политик безопасности и сетевых реализаций; инструмент не заменяет полноценные IDS/EDR и должен использоваться как аудит/комплаенс-контроль.

Перспективность проекта определяется практической применимостью: инструмент может быть встроен в регулярные проверки кластеров, снижая вероятность инцидентов и ускоряя исправление опасных настроек.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Kubernetes Documentation. Security Context for a Pod or Container: [Электронный ресурс]. URL: https://kubernetes.io/docs/tasks/configure-pod-container/security-context/ (дата обращения: 28.12.2025).

2 Kubernetes Documentation. RBAC Authorization: [Электронный ресурс]. URL: https://kubernetes.io/docs/reference/access-authn-authz/rbac/ (дата обращения: 28.12.2025).

3 Kubernetes Documentation. Network Policies: [Электронный ресурс]. URL: https://kubernetes.io/docs/concepts/services-networking/network-policies/ (дата обращения: 28.12.2025).

4 Kubernetes Documentation. Pod Security Standards / Pod Security Admission: [Электронный ресурс]. URL: https://kubernetes.io/docs/concepts/security/pod-security-standards/ (дата обращения: 28.12.2025).

5 CIS. CIS Kubernetes Benchmark: [Электронный ресурс]. URL: https://www.cisecurity.org/benchmark/kubernetes (дата обращения: 28.12.2025).

6 Aqua Security. Trivy Operator Documentation: [Электронный ресурс]. URL: https://aquasecurity.github.io/trivy-operator/ (дата обращения: 28.12.2025).

7 Fairwinds. Polaris (Kubernetes best practices): [Электронный ресурс]. URL: https://github.com/FairwindsOps/polaris (дата обращения: 28.12.2025).

8 Shopify. kubeaudit: [Электронный ресурс]. URL: https://github.com/Shopify/kubeaudit (дата обращения: 28.12.2025).

9 Aqua Security. kube-bench: [Электронный ресурс]. URL: https://github.com/aquasecurity/kube-bench (дата обращения: 28.12.2025).

10 MITRE. ATT&CK for Containers / Cloud (tactics and techniques): [Электронный ресурс]. URL: https://attack.mitre.org/ (дата обращения: 28.12.2025).

11 Falco (CNCF). Documentation: [Электронный ресурс]. URL: https://falco.org/docs/ (дата обращения: 28.12.2025).

12 Cilium. Tetragon Documentation: [Электронный ресурс]. URL: https://docs.cilium.io/en/stable/observability/tetragon/ (дата обращения: 28.12.2025).

13 Симоненко А.В. Актуальные вопросы подготовки кадров для противодействия киберпреступности // Вестник Краснодарского университета МВД России. 2021. № 4 (54). — (дата обращения: 28.12.2025).

ПРИЛОЖЕНИЯ

Приложение А — Пример структуры отчета (JSON)

```
{
  "cluster": {"name": "demo", "kubernetesVersion": "v1.30"},
  "generatedAt": "2025-12-28T00:00:00Z",
  "summary": {"critical": 0, "high": 2, "medium": 3, "low": 5},
  "findings": [
    {
      "checkId": "K8S-POD-001",
      "severity": "HIGH",
      "resource": {"kind": "Pod", "namespace": "dev", "name": "debug"},
      "title": "Privileged container",
      "evidence": "spec.containers[0].securityContext.privileged=true",
      "recommendation": "Убрать privileged, включить runAsNonRoot и
seccompProfile"
    }
  ]
}
```