2 ИЮНЯ 2026, МОСКВА, ЛОФТ ГОЭЛРО

БЕКОН LUNTRY '26

ЕДИНСТВЕННАЯ КОНФЕРЕНЦИЯ ПО БЕЗОПАСНОСТИ КОНТЕЙНЕРОВ И КОНТЕЙНЕРНЫХ СРЕД

Контроль целостности в Kubernetes по-взрослому

Максим Василенко | Флант

БЕКОН'26
КОНФЕРЕНЦИЯ ПО БЕЗОПАСНОСТИ КОНТЕЙНЕРОВ И КОНТЕЙНЕРНЫХ СРЕД

Deckhouse
ФЛАНТ



Контроль целостности в Kubernetes по-взрослому

Deckhouse
ФЛАНТ

Максим Василенко

Старший инженер архитектурных решений
Deckhouse Platform Security

План доклада

Поговорим о проблематике и подходе к реализации контроля целостности в Kubernetes:

- Как устроена цепь поставки контейнерных приложений, где в ней проблема и как от нее страдают
- Целостность чего нужно контролировать
- **Deep dive** в имплементацию
- Выводы

Вопрос

Зачем вообще нужен контроль целостности образов?

Все знают, что такое Kubernetes

Kubernetes — незаменимый инструмент для развёртывания, масштабирования и управления инфраструктурой

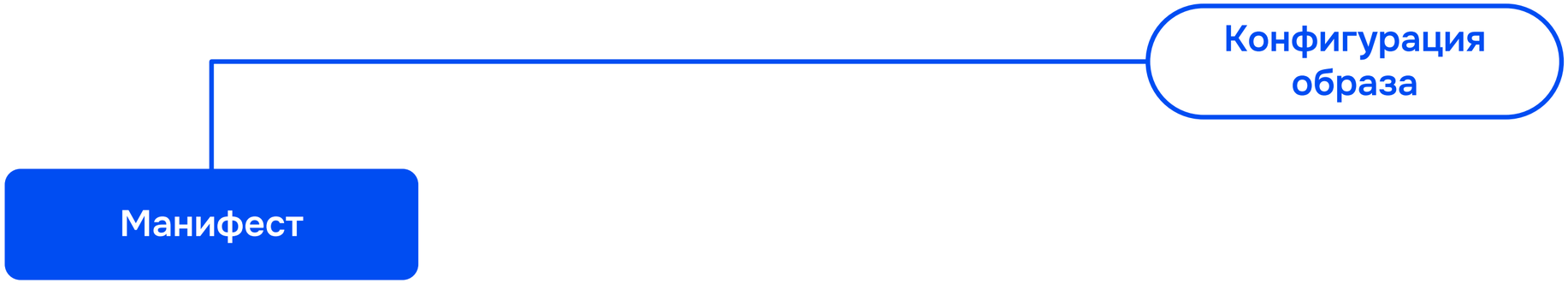
По мере того как стек технологий, запускаемых в K8s, становится всё изощрённее, **растёт и спектр угроз**, которые могут попасть в него через контейнерные образы

А что конкретно защищать?

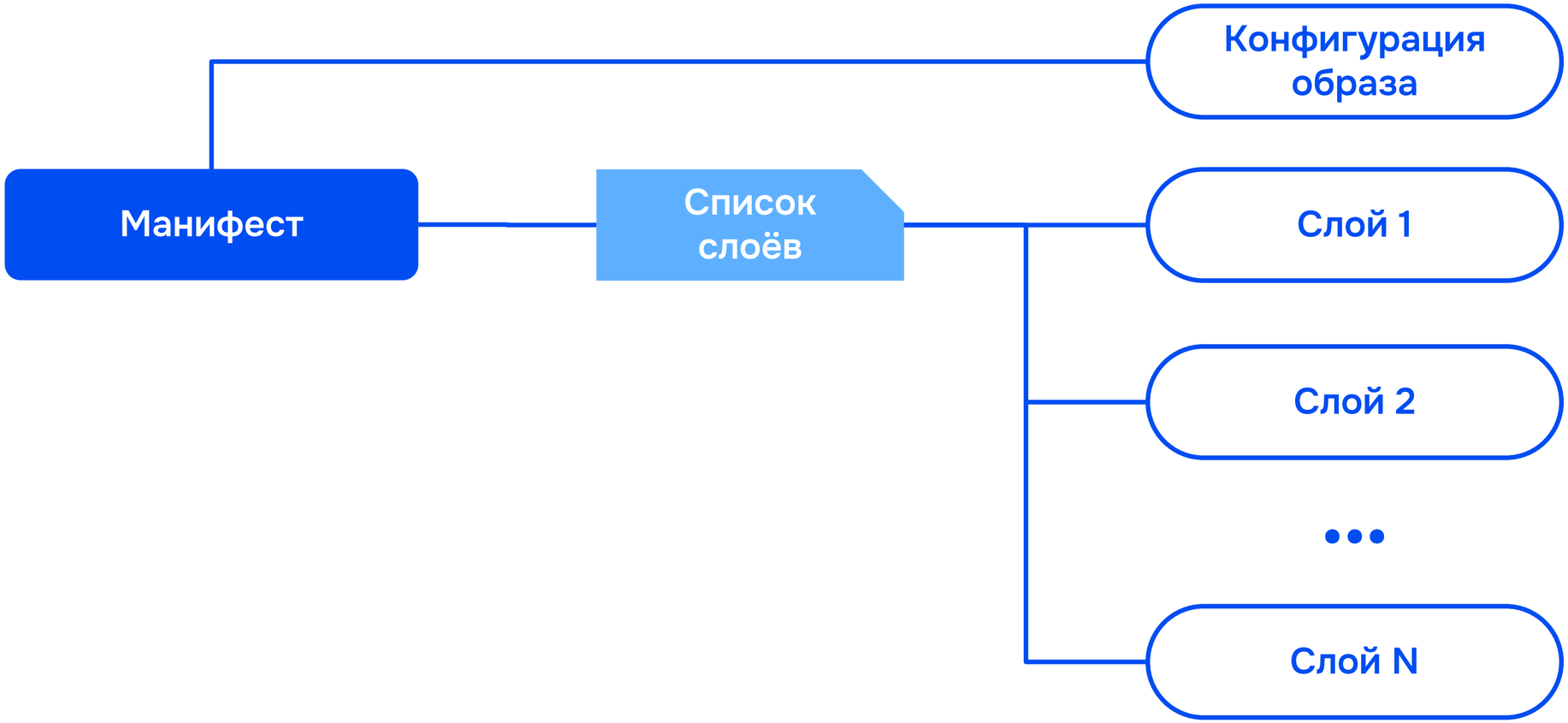
Как устроен типичный контейнер

Манифест

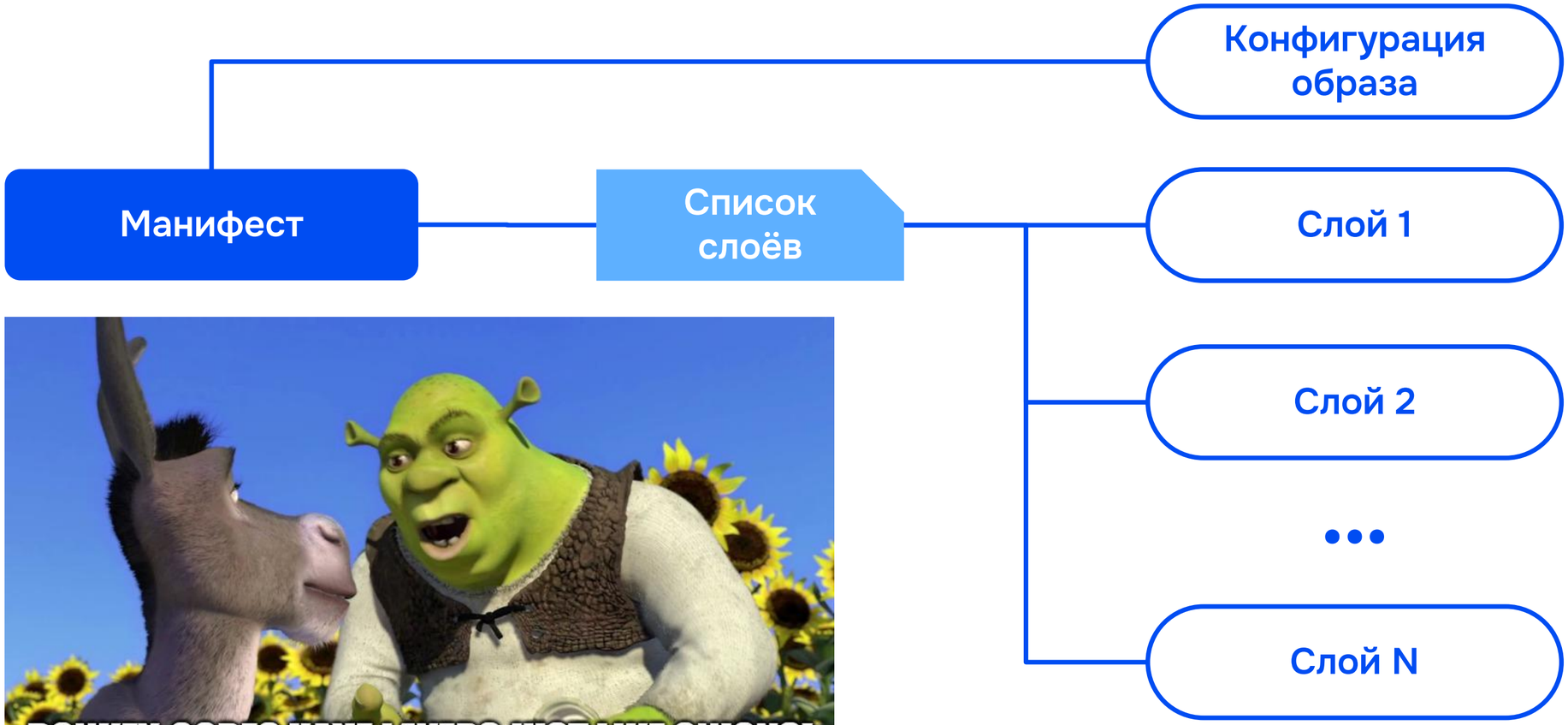
Как устроен типичный контейнер



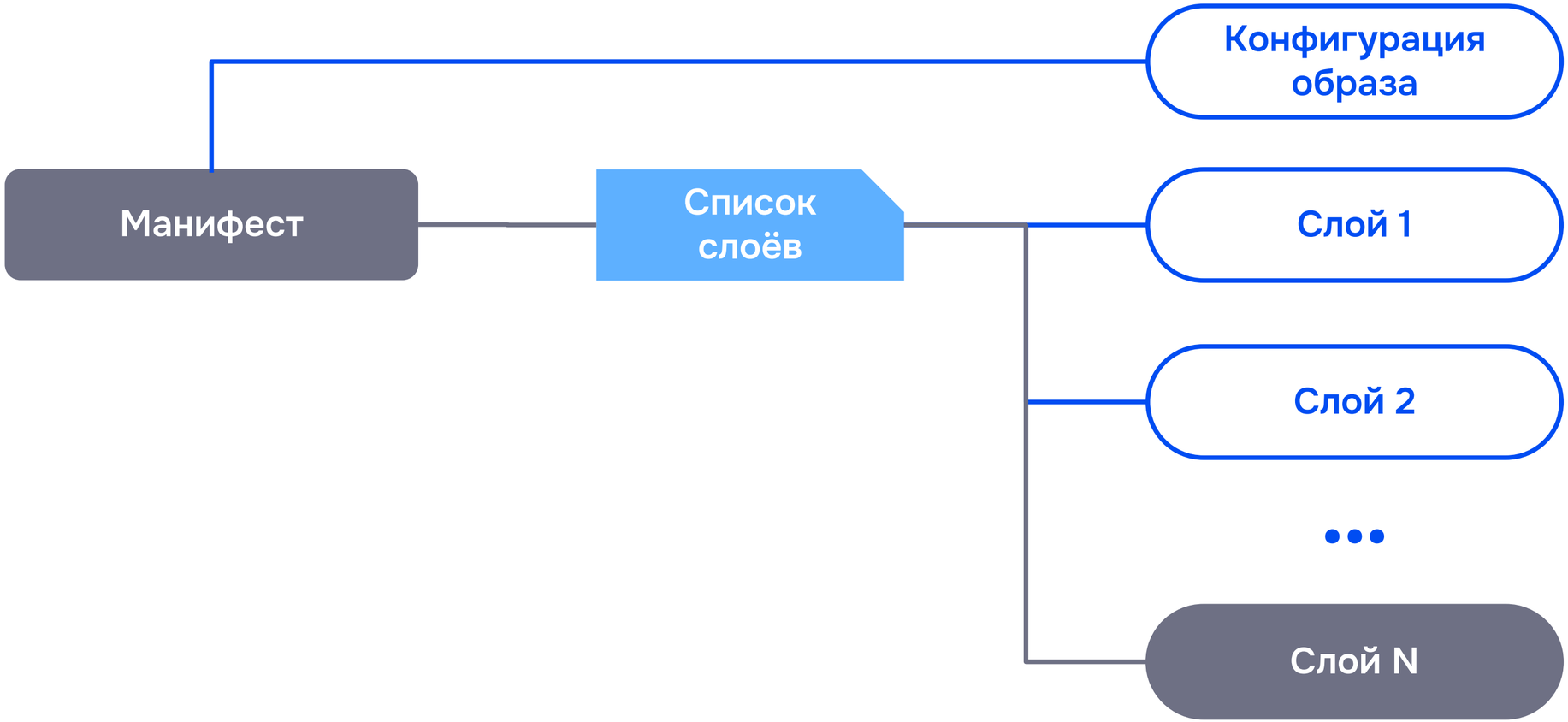
Как устроен типичный контейнер



Как устроен типичный контейнер



Как устроен типичный контейнер



Реальные инциденты показывают, насколько опасными могут быть такие уязвимости.

Недавние supply-chain-атаки включали в себя:

01 Редактирование открытых компонентов в репозиториях

Реальные инциденты показывают, насколько опасными могут быть такие уязвимости.

Недавние supply-chain-атаки включали в себя:

01 Редактирование открытых компонентов в репозиториях

02 Подмену Docker-образов

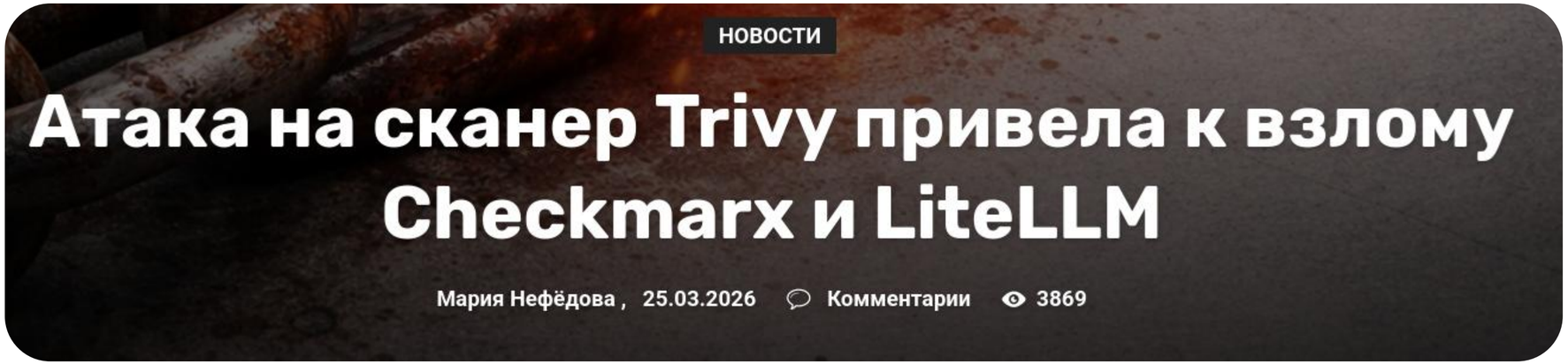
Реальные инциденты показывают, насколько опасными могут быть такие уязвимости.

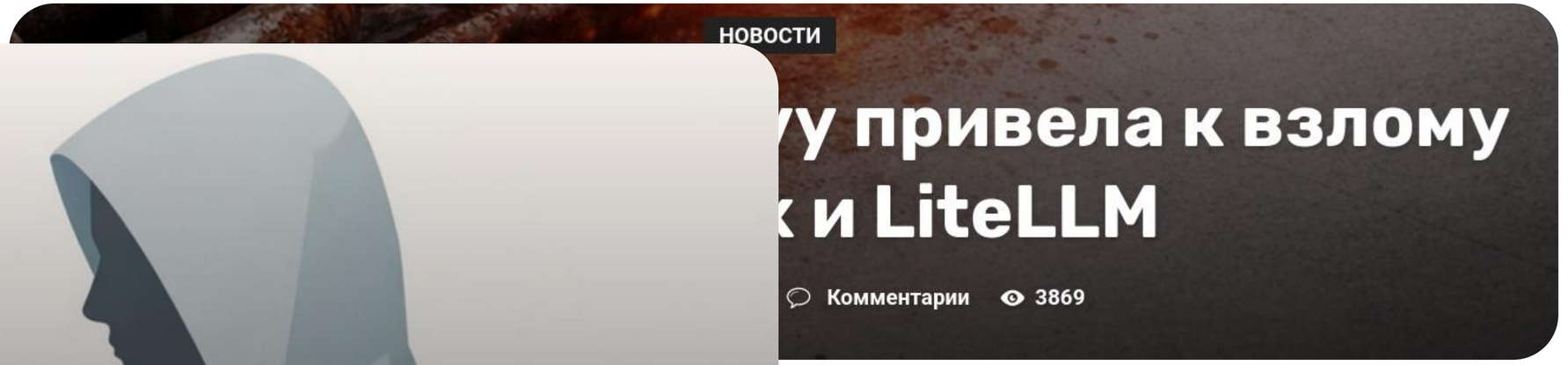
Недавние supply-chain-атаки включали в себя:

01 Редактирование открытых компонентов в репозиториях

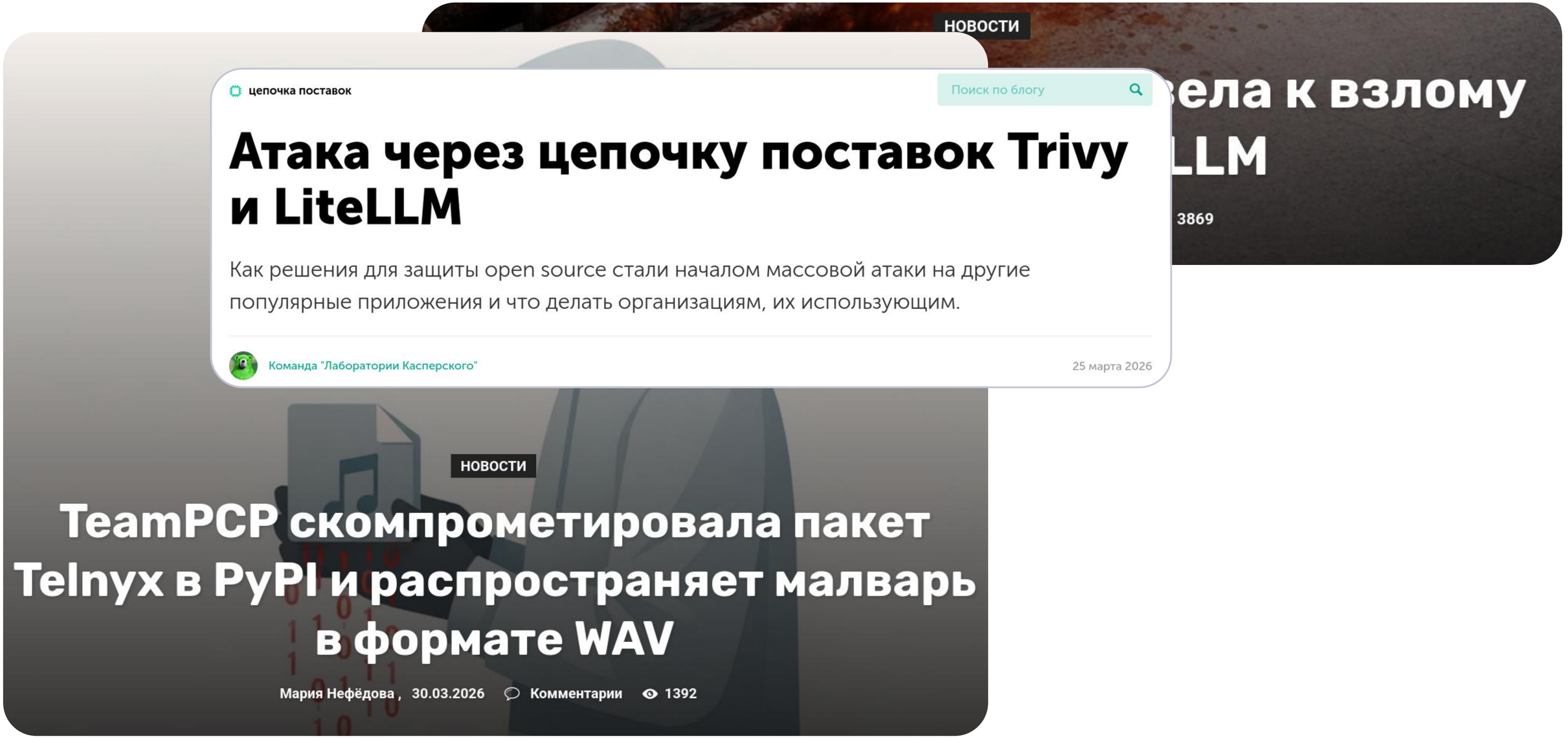
02 Подмену Docker-образов

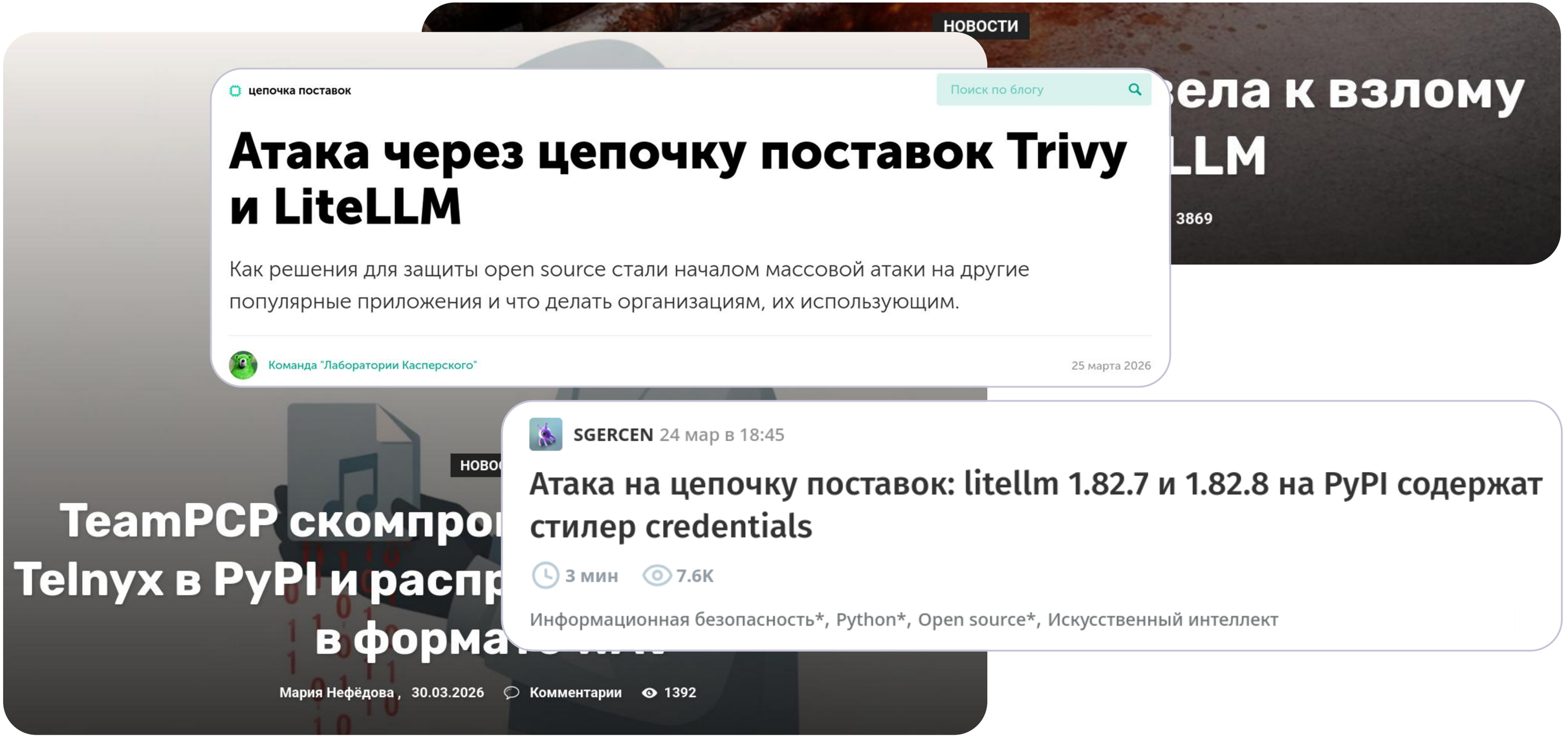
03 Внедрение троянов, ищущих и сливающих различные секреты из инфраструктуры

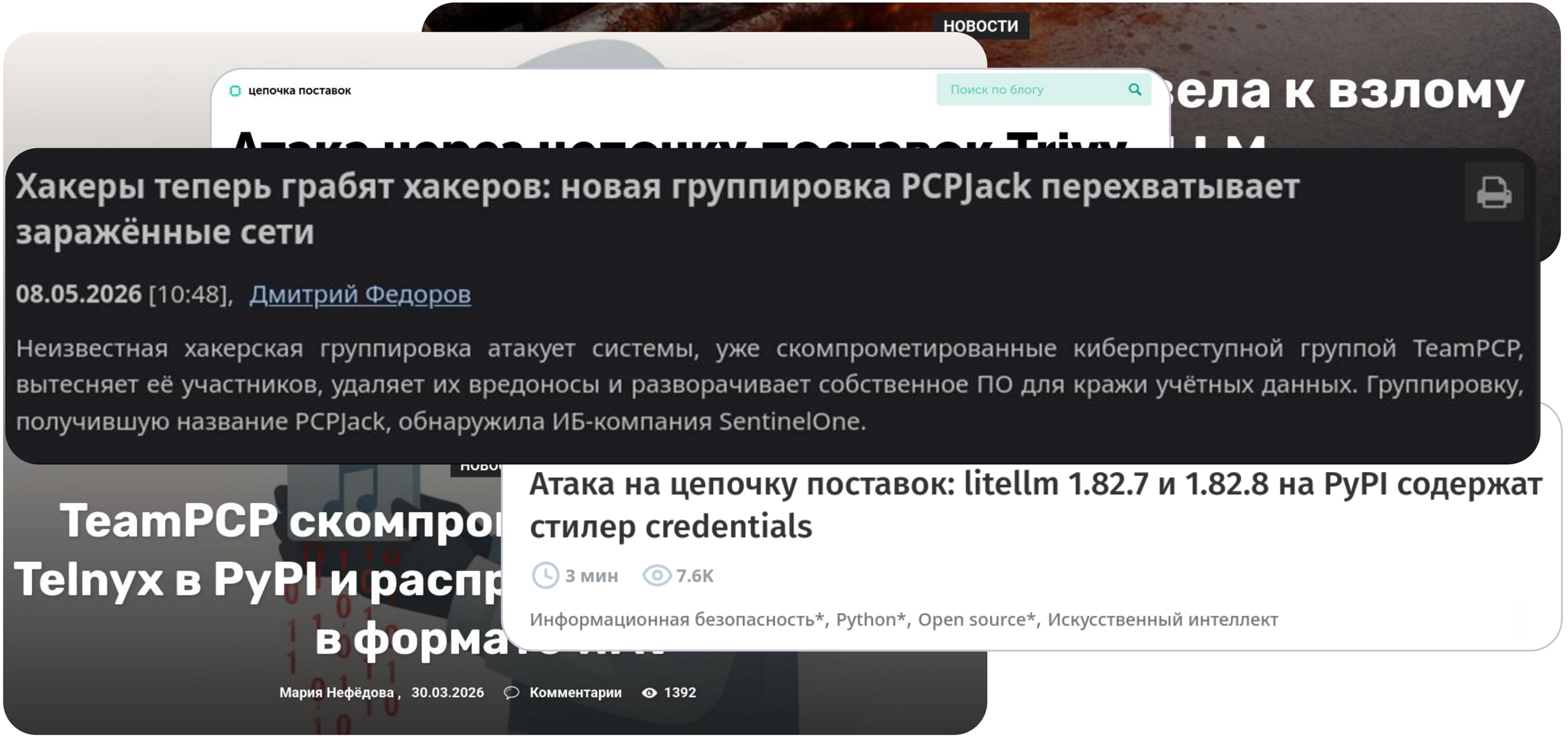

НОВОСТИ
Атака на сканер Trivy привела к взлому Checkmarx и LiteLLM
Мария Нефёдова , 25.03.2026
Комментарии
3869


НОВОСТИ
у привела к взлому
к и LiteLLM
Комментарии
3869


НОВОСТИ
TeamPCP скомпрометировала пакет Telnyx в PyPI и распространяет малварь в формате WAV
Мария Нефёдова , 30.03.2026
Комментарии
1392


НОВОСТИ
ела к взлому
LLM
3869
цепочка поставок
Поиск по блогу
Атака через цепочку поставок Trivy и LiteLLM
Как решения для защиты open source стали началом массовой атаки на другие популярные приложения и что делать организациям, их использующим.
Команда "Лаборатории Касперского"
25 марта 2026
НОВОСТИ
TeamPCP скомпрометировала пакет Telnyx в PyPI и распространяет малварь в формате WAV
Мария Нефёдова , 30.03.2026
Комментарии
1392

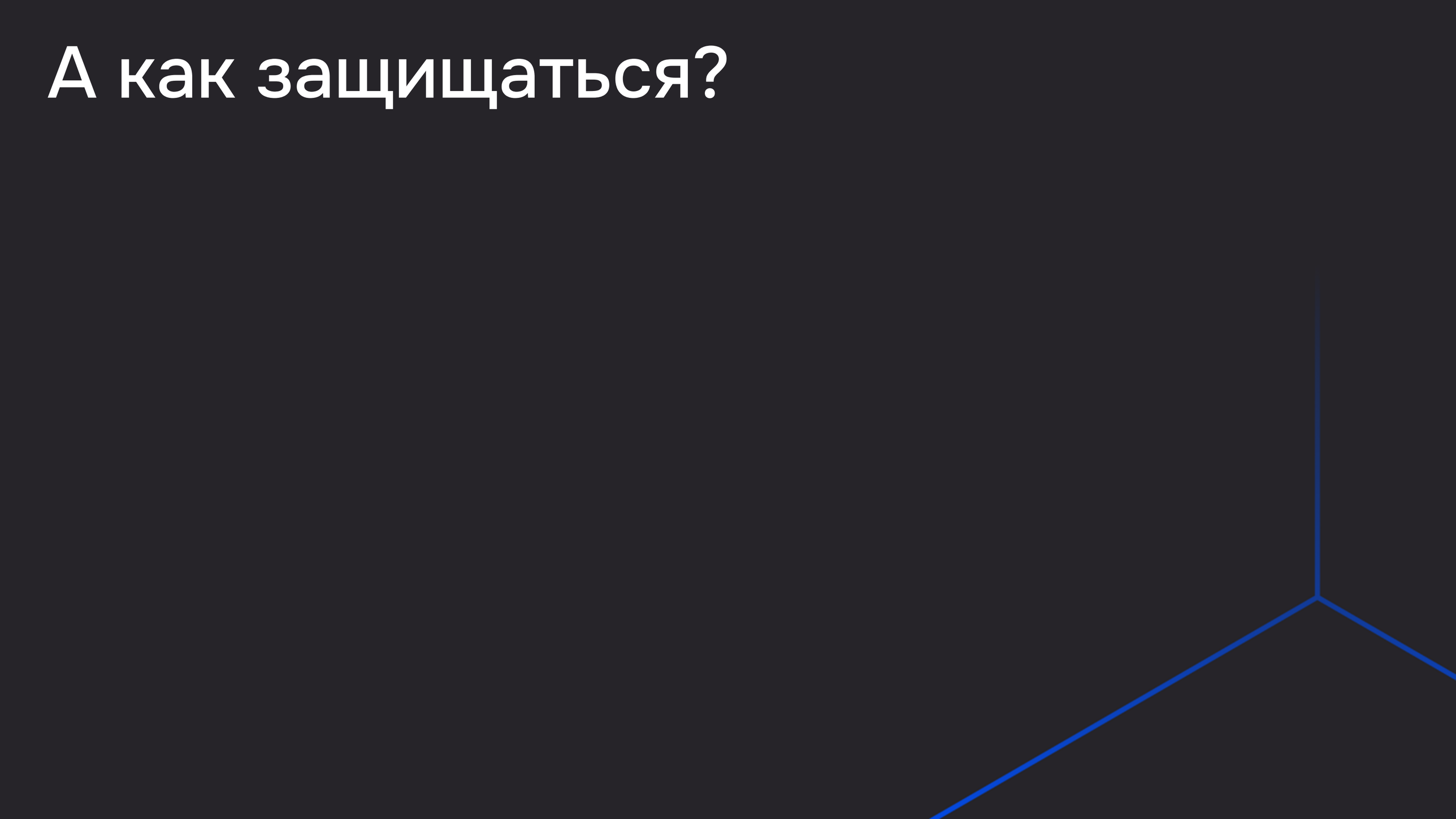

НОВОСТИ
ела к взлому
LLM
3869
цепочка поставок
Поиск по блогу
Атака через цепочку поставок Trivy и LiteLLM
Как решения для защиты open source стали началом массовой атаки на другие популярные приложения и что делать организациям, их использующим.
Команда "Лаборатории Касперского"
25 марта 2026
НОВО
TeamPCP скомпро
Telnyx в PyPI и распр
в форма
Мария Нефёдова , 30.03.2026
Комментарии
1392
SGERCEN 24 мар в 18:45
Атака на цепочку поставок: litellm 1.82.7 и 1.82.8 на PyPI содержат стилер credentials
3 мин
7.6K
Информационная безопасность*, Python*, Open source*, Искусственный интеллект


НОВОСТИ
ела к взлому
цепочка поставок
Поиск по блогу
Хакеры теперь грабят хакеров: новая группировка PCPJack перехватывает заражённые сети
08.05.2026 [10:48], Дмитрий Федоров
Неизвестная хакерская группировка атакует системы, уже скомпрометированные киберпреступной группой TeamPCP, вытесняет её участников, удаляет их вредоносы и разворачивает собственное ПО для кражи учётных данных. Группировку, получившую название PCPJack, обнаружила ИБ-компания SentinelOne.
TeamPCP скомпро
Telnyx в PyPI и распр
в форма
Мария Нефёдова , 30.03.2026 Комментарии 1392
Атака на цепочку поставок: litellm 1.82.7 и 1.82.8 на PyPI содержат стилер credentials
3 мин 7.6K
Информационная безопасность*, Python*, Open source*, Искусственный интеллект

Кто пострадал от TeamPCP

- AquaSecurity (Trivy)
- BerriAI (LiteLLM)
- Bitwarden
- SAP
- OwnCloud
- CheckMarx
- Telnyx
- Великое множество NPM-пакетов
- ...

А как защищаться?



Мы обречены?

На самом деле — **нет**.

Применительно к Kubernetes нас спасёт следование лучшим практикам **РБПО**

Как защищаться от атак на образы контейнеров

Нужно:

01 Контролировать, что и из чего на самом деле собирается

Как защищаться от атак на образы контейнеров

Нужно:

01 Контролировать, что и из чего на самом деле собирается

02 Проверять содержимое каждого слоя образа, а не только digest

Как защищаться от атак на образы контейнеров

Нужно:

01 Контролировать, что и из чего на самом деле собирается

02 Проверять содержимое каждого слоя образа, а не только digest

03 Использовать для хранения скачанных слоёв решения, которые by design обеспечивают неизменяемость содержимого образов

Как защищаться от атак на образы контейнеров

Нужно:

01 Контролировать, что и из чего на самом деле собирается

02 Проверять содержимое каждого слоя образа, а не только digest

03 Использовать для хранения скачанных слоёв решения, которые by design обеспечивают неизменяемость содержимого образов

04 Контролировать целостность образов и их контейнеров не только at rest, но и в runtime

Как защищаться от атак на образы контейнеров

Таким образом, хотелось бы получить от контроля целостности следующее:

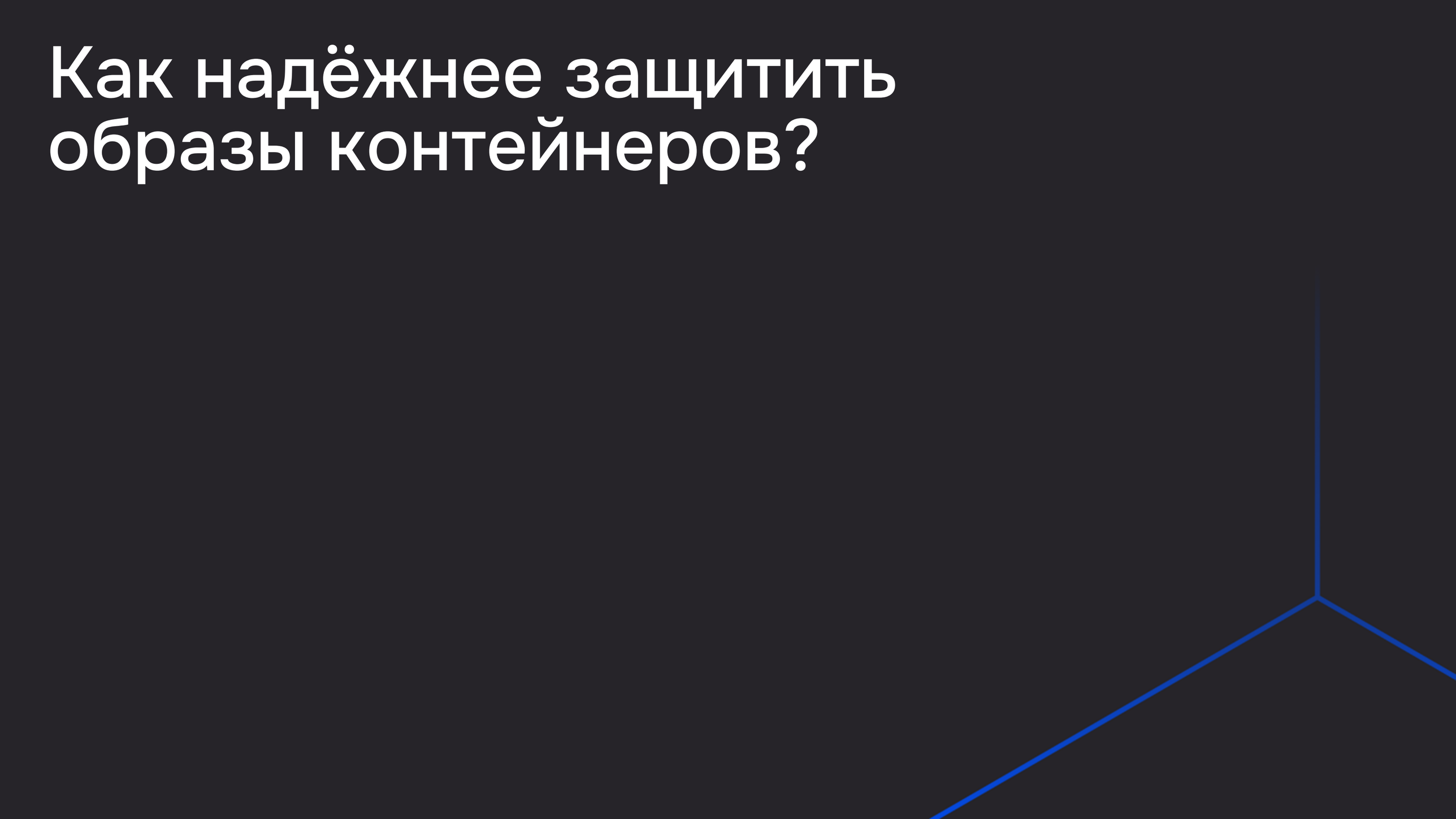
- Уверенность в том, что локальная копия манифеста контейнера и слоёв соответствует тому, что лежит в registry

Как защищаться от атак на образы контейнеров

Таким образом, хотелось бы получить от контроля целостности следующее:

- Уверенность в том, что локальная копия манифеста контейнера и слоёв соответствует тому, что лежит в registry
- Уверенность в том, что вмешательство в данные уже запущенного контейнера как минимум надёжно отслеживается, а в идеале предотвращается

Как надёжнее защитить образы контейнеров?



Мы думали, что это будет просто

Решение вопроса можно разбить на две большие задачи:

01 Цифровая подпись образа контейнера с возможностью аутентификации всего его содержимого (не только манифеста)

Мы думали, что это будет просто

Решение вопроса можно разбить на две большие задачи:

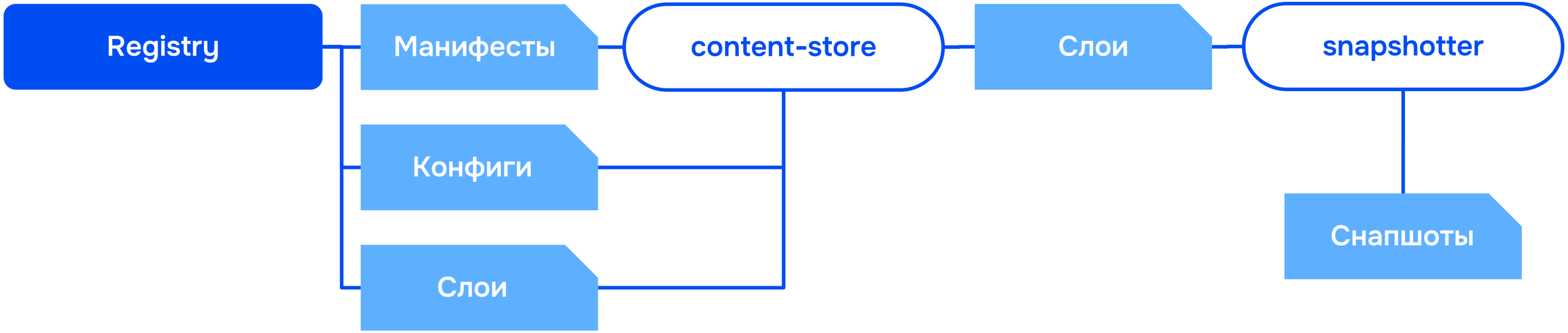
01 Цифровая подпись образа контейнера с возможностью аутентификации всего его содержимого (не только манифеста)

02 Обеспечение неизменности контейнера в процессе его запуска и работы

Цифровые подписи образов

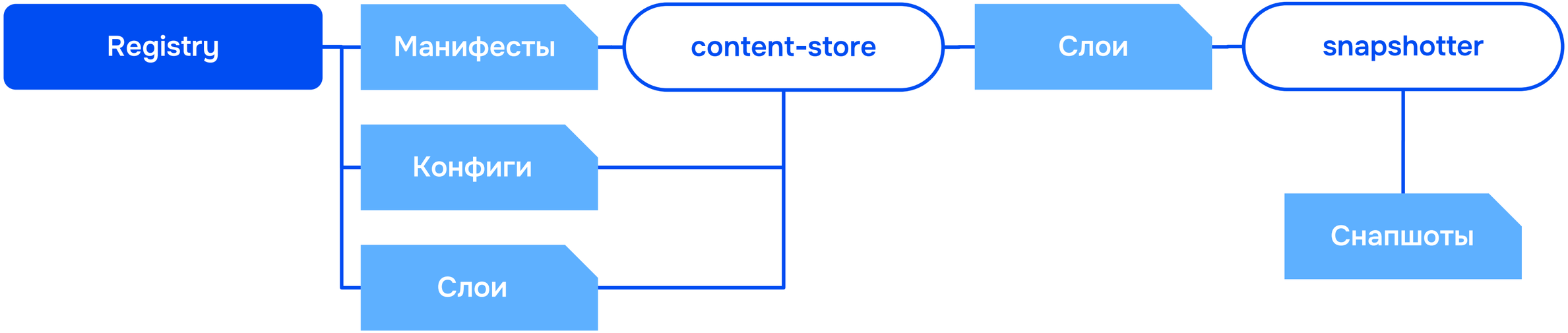
Как устроен containerd

В процессе скачивания образа в containerd участвуют 2 основные абстракции: **content-store** и **snapshotter**



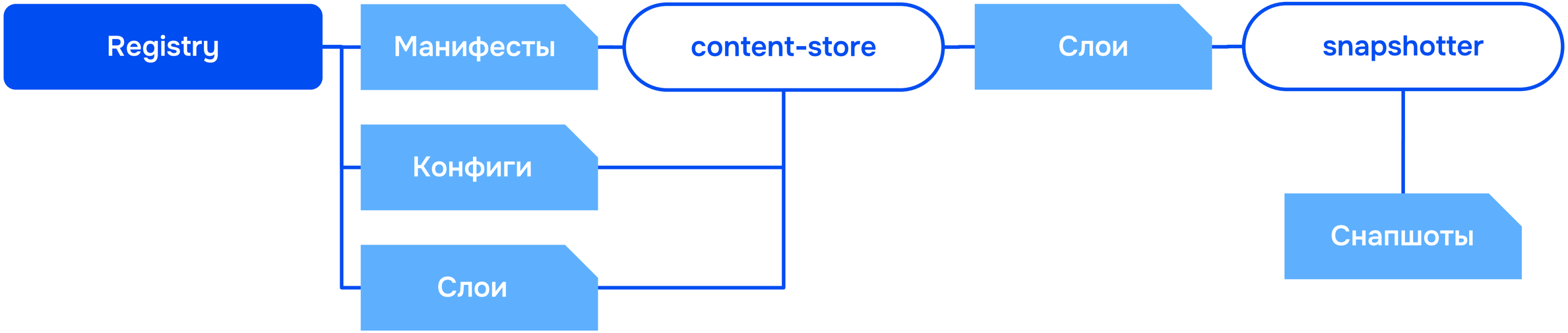
Как устроен containerd

В content-store просто хранятся скачанные артефакты



Как устроен containerd

Снапшоттеры нужны для преобразования tar-слоев в формат, пригодный для использования в работающем контейнере.



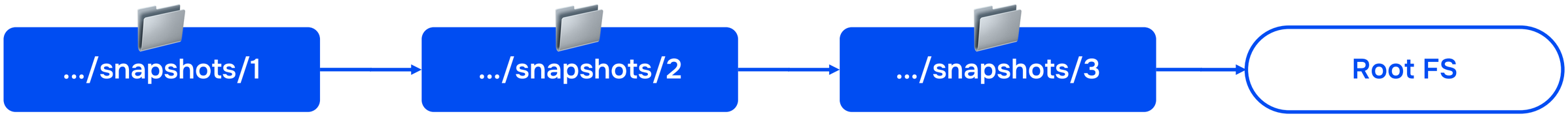
Как устроен containerd

По умолчанию они представляют собой просто **директории, в которые распаковываются слои образов**

```
/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/
├ metadata.db
└ snapshots
  └ 1
    ├ fs      < Файлы слоя лежат тут
    └ work
  └ 2
    ├ fs
    └ work
    ...
```

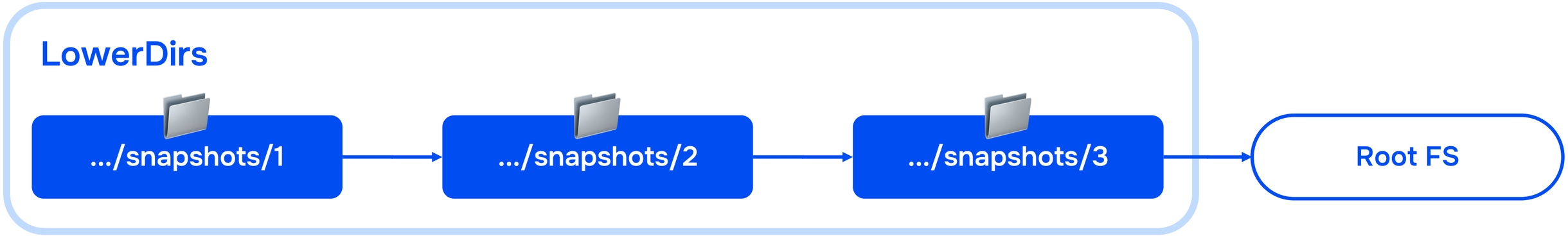
Как устроен containerd

Все каталоги слоёв, составляющие образ, при помощи драйвера **OverlayFS** собираются в один цельный каталог, который станет основой для корневой файловой системы контейнера



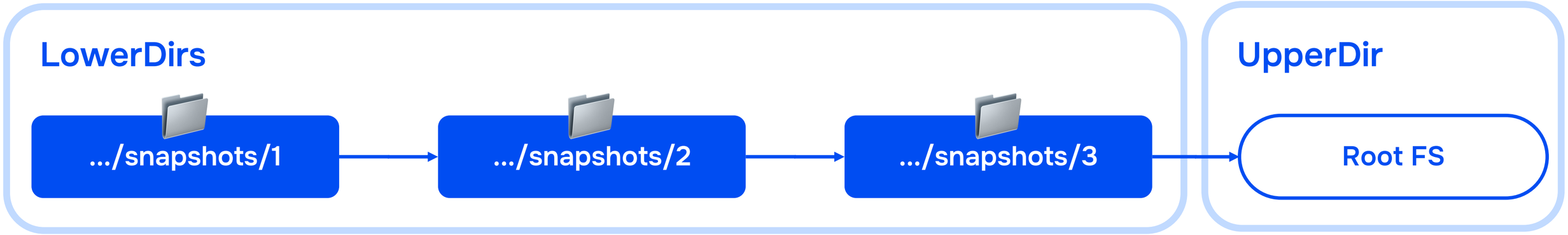
Как устроен containerd

В терминологии OverlayFS такие составляющие каталоги называются **LowerDirs**



Как устроен containerd

Также в каталог, который станет RootFS контейнера, автоматически **добавляются директории и файлы, смонтированные в контейнер с хоста** в качестве **UpperDir** OverlayFS



Как устроен containerd

UpperDir монтируется в mount-неймспейс контейнера в качестве его rootfs.
По умолчанию это происходит с опцией RW — **UpperDir доступна для записи**

```
overlay
  on /run/containerd/io.containerd.runtime.v2.task/k8s.io/4c..ba/rootfs
  type overlay (
    rw, relatime, seclabel,
    lowerdir=/run/containerd/.../7:/run/containerd/.../6:...,
    upperdir=/var/lib/containerd/io.containerd.snapshotter.v1.erofs/snapshots/667/fs
    ...
)
```

Проблемой такого подхода является то,
что **с хост-системы становится возможным подменить как файлы распакованных слоёв, так и файлы в итоговой rootfs контейнера**

Строгая интеграция EROFS и dm-verity

Для решения этой проблемы можно перейти на использование встроенного в Containerd v2.1 по умолчанию снапшоттера на базе драйвера **Enhanced Read-Only File System (EROFS)**

Строгая интеграция EROFS и dm-verity

Этот снапшоттер, кроме прочего, как раз и предназначался для **обеспечения иммутабельности снапшотов на диске**

Строгая интеграция EROFS и dm-verity

В такой схеме при скачивании образов каждый слой **конвертируется** в образ файловой системы EROFS, которая **by design является read-only:** драйвер ядра не предусматривает операций записи, а сама ФС является иммутабельной.

Для внесения изменений в слой потребуется полная пересборка образа ФС

Строгая интеграция EROFS и dm-verity

Но ведь одна только EROFS не обеспечивает полноценного контроля целостности!

Строгая интеграция EROFS и dm-verity

Злоумышленник всё ещё может заменить файл
с образом снапшота, и никто ничего не узнает.

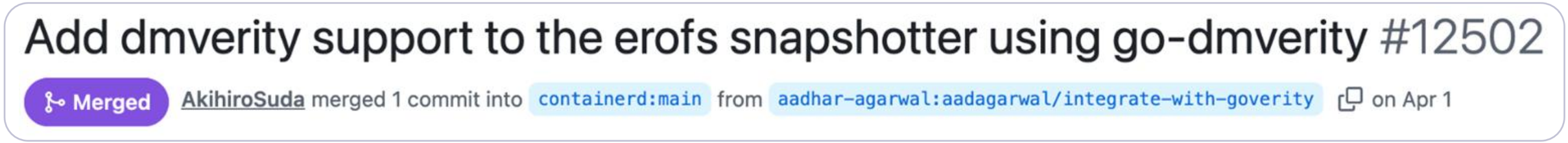
Как это предотвратить?

Строгая интеграция EROFS и dm-verity

Такие проверки удобно реализуются за счёт использования модулей **Device Mapper** (dm) и **Device Mapper Verity** (dm-verity)

Все дороги ведут к dm-verity

Спустя около полугода после начала разработки нашего решения
в GitHub-репозитории containerd появилось это:



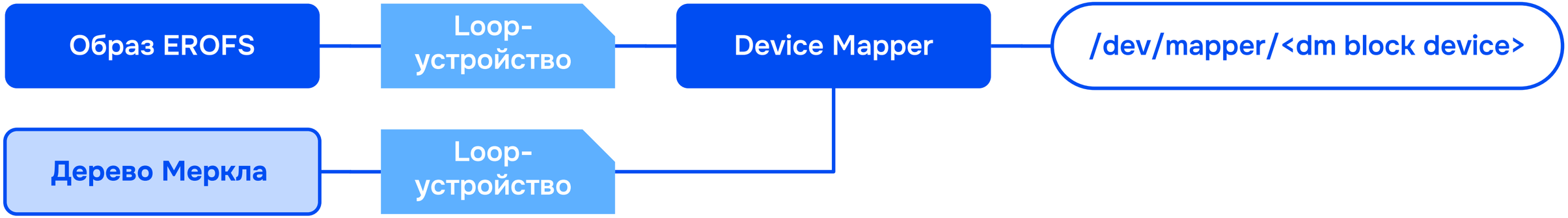
Строгая интеграция EROFS и dm-verity

Такие проверки удобно реализуются за счёт использования модулей **Device Mapper** (dm) и **Device Mapper Verity** (dm-verity)



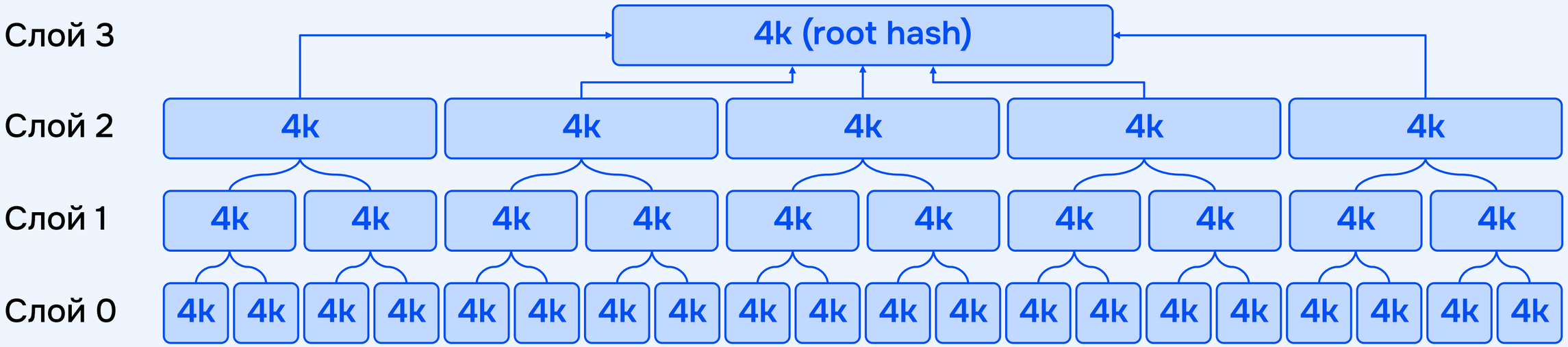
Строгая интеграция EROFS и dm-verity

Такие проверки удобно реализуются за счёт использования модулей **Device Mapper** (dm) и **Device Mapper Verity** (dm-verity)



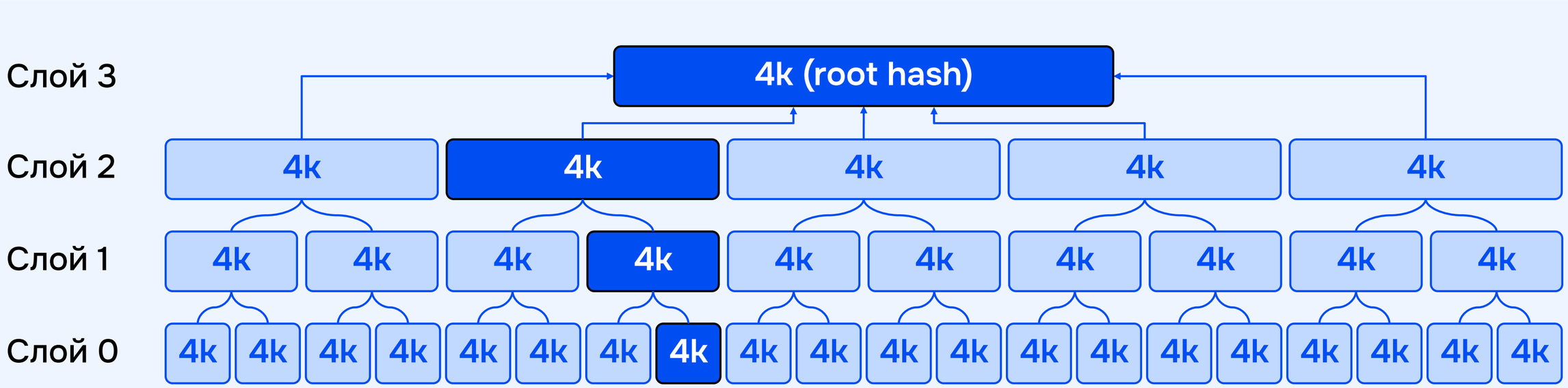
Дерево Меркла

Также называемое деревом хешей, это структура данных,
которая обобщает большой массив информации в один уникальный хеш



Дерево Меркла

Изменение даже одного байта в исходных данных
полностью меняет корень дерева



Строгая интеграция EROFS и dm-verity

Любое чтение данных слоя проходит
через **проверку целостности на уровне ядра.**

Это делает невозможными «скрытую» модификацию
образа без её обнаружения или запуск
скомпрометированного контейнера

Строгая интеграция EROFS и dm-verity

Полученные в результате конвертации снапшоты состоят уже из файлов с образами EROFS, для которых посчитаны деревья Меркла

```
/var/lib/containerd/io.containerd.snapshotter.v1.erofs/snapshots/20
├ layer.erofs
└ layer.erofs.verity
```

```
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
  "config": {...},
  "layers": [{
      "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
      ...
      "annotations": {
        "io.deckhouse.delivery-kit.build-timestamp": "1750791050",
        "io.deckhouse.delivery-kit.dm-verity-root-hash": "d2c...61f"
      }
    }, {...}, {...} ],
  "annotations": {
    "io.deckhouse.delivery-kit.cert": "LS0...LQo=",
    "io.deckhouse.delivery-kit.chain": "LS0...LQo=",
    "io.deckhouse.delivery-kit.signature": "AIb...bZQ=="
  }
}
```

Мы думали, что это будет просто

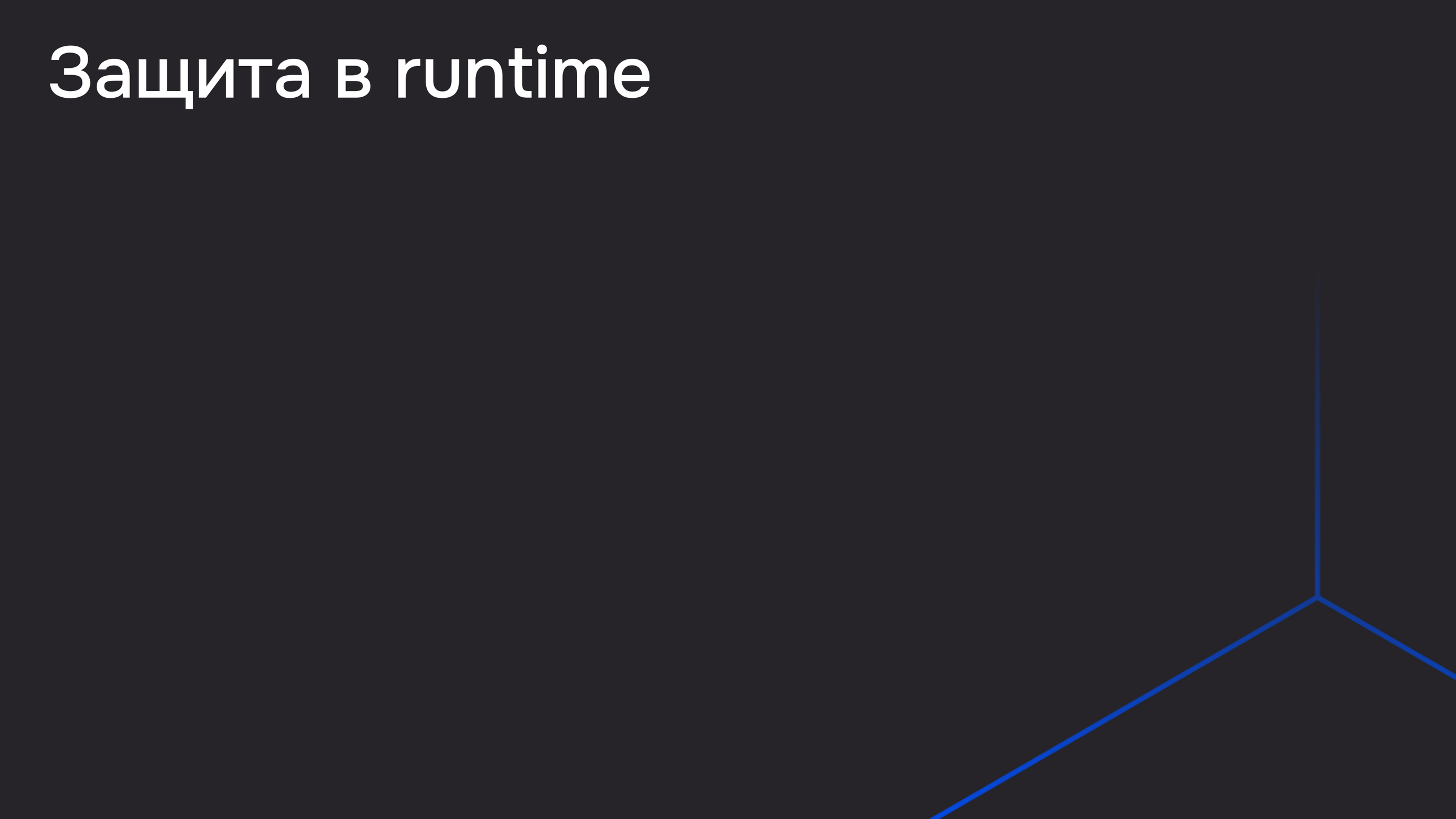
Решение вопроса можно разбить на две большие задачи:

01 Цифровая подпись образа контейнера с возможностью аутентификации всего его содержимого (не только манифеста)



02 Обеспечение неизменности контейнера в процессе его запуска и работы

Защита в runtime



Контроль целостности контейнеров в runtime

В процессе экспериментов мы заметили, что файлы в контейнере можно подменять, накладывая собственный mount поверх существующего либо где-то внутри него

Контроль целостности контейнеров в runtime

Для предотвращения таких вмешательств мы сделаем 2 вещи:

01 Блокировку бесконтрольной запись в UpperDir

02 Контроль маунтов в контейнере

Mount namespace защищенного контейнера выглядит примерно так

```
overlay
  on /.../io.containerd.runtime.v2.task/k8s.io/c13...010/rootfs
  type overlay (
    ro,
    lowerdir=/.../t/53283/4:/.../t/53283/3:/.../t/53283/2:/../t/53283/1,
  )
/dev/mapper/14361
  on /run/containerd/io.containerd.mount-manager.v1.bolt/t/53283/4
  type erofs (ro, ...)
/dev/mapper/14243
  on /run/containerd/io.containerd.mount-manager.v1.bolt/t/53283/3
  type erofs (ro, ...)
...
...
...
```

Контроль целостности контейнеров в runtime

Здесь видно, что UpperDir OverlayFS находится в режиме **read-only,**
а UpperDir не используется.

В ванильном containerd это не так, и UpperDir монтируется в режиме RW

```
overlay
  on /.../io.containerd.runtime.v2.task/k8s.io/c13...010/rootfs
  type overlay (
    ro,
    lowerdir=/.../t/53283/4:/.../t/53283/3:/.../t/53283/2:/../t/53283/1
```

Контроль целостности контейнеров в runtime

Это необходимо для того, чтобы при монтировании в контейнер файлов или директорий под них можно было **автоматически создать точки монтирования** для bind-mount

Контроль целостности контейнеров в runtime

То есть UpperDir в режиме **RW дает как возможность прокидывать в контейнер файлы и директории хоста, так и возможность обойти наш контроль целостности.**

Получилась палка о двух концах.

Контроль целостности контейнеров в runtime

Поскольку **точки монтирования задаются в момент создания контейнера,** мы заранее не знаем, какие пути могут понадобиться образу.

Решение этой проблемы на первый взгляд радикально, но на самом деле является хорошей практикой и позволяет разработчику образа управлять тем, что и куда в нём может быть смонтировано

Контроль целостности контейнеров в runtime

Мы создали «стандартный» слой, где собраны общеупотребимые точки монтирования (/tmp, /etc/resolv.conf и т. п.), и монтируем этот слой в режиме read-only автоматически ко всем создаваемым контейнерам

Контроль целостности контейнеров в runtime

Dm-verity-сумма этого образа заранее известна
и захардкожена в containerd, так что проверку целостности
контейнеры с этим слоем проходят без проблем.

**Read-Write UpperDir же в такой схеме не подключается
в контейнер вовсе**

Контроль целостности контейнеров в runtime

Самым существенным ограничением этого подхода является необходимость создания **всех сторонних точек монтирования в образе во время его сборки.**

Но это даёт и плюсы в виде строгого контроля допустимых точек монтирования разработчиком образа

К слову о точках монтирования

Мы говорили о возможности переписать содержимое контейнера наложив свой mount поверх rootfs или вложенных директорий

К слову о точках монтирования

Для предотвращения таких атак мы проводим периодическую проверку существующих маунтов в контейнерах и проверяем что не появилось лишних, без dm-verity которой мы доверяем.

Мы думали, что это будет просто

Решение вопроса можно разбить на две большие задачи:

01 Цифровая подпись образа контейнера с возможностью аутентификации всего его содержимого (не только манифеста)

02 Обеспечение неизменности контейнера в процессе его запуска и работы

Не всё так просто

01 Цифровая подпись образа контейнера с возможностью аутентификации всего его содержимого (не только манифеста)



02 Обеспечение неизменности контейнера в процессе его запуска и работы



Не всё так просто

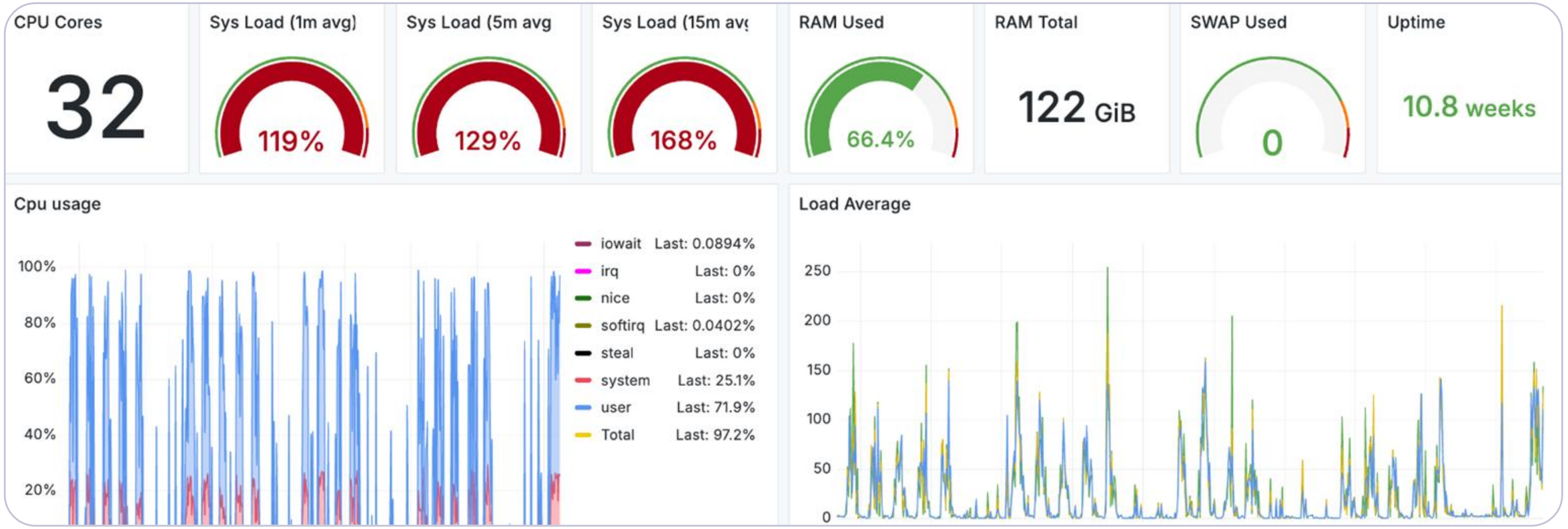
В процессе тестов мы наткнулись на интересные проблемы,
о паре из них тоже хотелось бы рассказать для полноты картины:

01 Периодически CPU и диски узлов загружались на 100 %

02 Остановка контейнеров фантомно съедала дисковое пространство

Тестовым кластерам иногда было больно

Тестовые кластеры периодически присылали алерты
о 100%-ной загрузке CPU или высоком iowait



Нагрузка на кластер неожиданно высока

Потребление на криптографию при периодических проверках целостности снапшотов оказалось неожиданно высоким

Нагрузка на кластер неожиданно высока

Решилось это введением ограничений на количество параллельных проверок и их частоту

Фантомное потребление дисков

При остановке контейнеров утекало дисковое пространство под containerd

Mount point	Inodes usage	Space usage
/var/lib/containerd	10,0 %	91,1 %

Фантомное потребление дисков

При остановке контейнера получалась ситуация, аналогичная unlink файла с более чем одной ссылкой, только в нашем случае вместо hardlink были точки монтирования для bind-mount, которые продолжали удерживать данные внутри mount namespace контейнеров.

Фантомное потребление дисков

Решили эту проблему написанием небольшой утилиты, которая при остановке контейнера входит в его неймспейс и размонтирует за ним все остатки.

Посмотреть можно на GitHub.



GitHub

Выводы

- Контроль целостности это **must have** а не nice to have
- Защита нужна на всём пути: сборка → runtime
- Чистый образ — только половина дела
- Runtime остаётся главной зоной риска
- Безопасность контейнера — это постоянный мониторинг его целостности

Контроль целостности трёх «К» в Kubernetes: как не доставить в прод вредоносный код



Habr

БЕКОН'26

КОНФЕРЕНЦИЯ ПО БЕЗОПАСНОСТИ КОНТЕЙНЕРОВ И КОНТЕЙНЕРНЫХ СРЕД

Deckhouse

ФЛАНТ

Спасибо за внимание!



@deckhouse_news

github.com/deckhouse

deckhouse.ru

bekon.luntry.ru