Cloud Thinking

Модуль 7, урок 1

В этом уроке мы поговорим о безопасности процесса разработки и поставки ПО. Также это называют термином DevSecOps или Supply Chain Security.

Почему это важно?

Чем больше мы пользуемся интернет-сервисами, чем больше платформ появляется, тем больше появляется периметров безопасности, уязвимых мест и так далее.

На каждом этапе разработки, создания и поставки ПО, а также его распространения и дальнейшего использования есть минимум одна (а на самом деле их целый спектр) уязвимость.

Личность/ социнжиниринг	Вредоносный код	Вредоносный код	Бэкдор/ malware	Тайпсквоттинг коллизии	Пакет внутри пайплайна	Бомба замедленного действия
Разработка	Коммит	Зависимости	Сборка	Загрузка	Использование	Продакшен

Каждая из этих уязвимостей при неправильном подходе (не закрываем, закрываем плохо или некорректно) способна привести не просто к нарушениям в процессе разработки ПО, но и нарушить ритм бизнеса.

По различным отчетам безопасности видно, что в последние годы эта тема занимает очень много умов, а бизнес несет существенные расходы именно на то, чтобы избежать или закрыть проблемы с безопасностью.

Выделим популярные угрозы:

- **На этапе разработки** — личность и социнжиниринг

В команде разработки есть человек, который может каким-то неправильным обращением с кодом, неверной стратегией развертывания или неудовлетворяющим процессом разработки заложить проблему в самом начале.

- **На уровне коммита** — вредоносный код

Вредоносный код может быть разных видов. Например, может срабатывать сразу или через некоторое время, как бомба замедленного действия, которую сложно обнаружить.

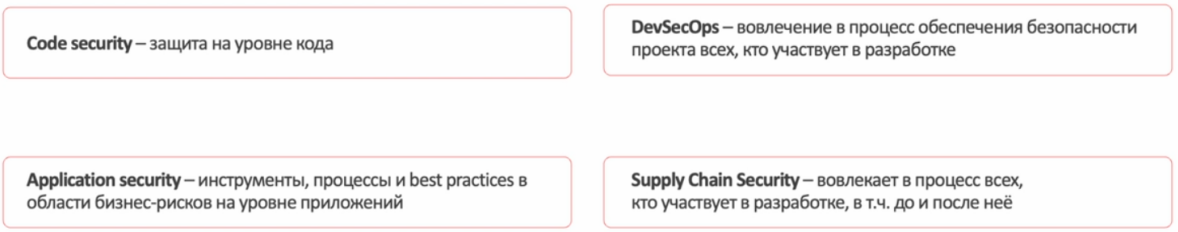
- **На уровне зависимостей** — вредоносный код

В последние годы управление зависимостями (все, что мы подключаем к программному коду) имеет критическое значение. В мире open source есть много полезных проектов и библиотек. Чтобы не заниматься изобретения велосипеда в процессе разработки, можно просто пойти на GitHub или взять готовую библиотеку, подключить ее и забыть о ней: работает и хорошо.
Однако это может быть очень опасным подходом. Зависимость — это тоже программный проект, и в нем тоже могут быть бомбы замедленного действия.
В некоторых случая зависимости подключаются не просто как файл, а через указание ссылкой, по которой она «подтягивается» на уровне сборки или развертывания. Эти зависимости могут подменяться другими библиотеками или изначально содержать собственные уязвимости.

- **На этапе сборки, загрузки и использования**

Тут возникают аналогичные угрозы, которые так или иначе связаны с тем, что мы делаем.

Уровни защиты



Microsoft выделяет несколько уровней безопасности:

1. **Уровень кода**

Все, что происходит еще до развертывания или использования, на уровне разработки конкретным разработчиком или конкретной командой. Здесь нам на помощь приходят различные статические и динамические анализы, а также инструменты, которые позволяют увидеть, что мы используем небезопасные практики, и отловить различные потенциально негативные эффекты — например, неправильное приведение типа и так далее.
Это тоже важный аспект защиты, поскольку неправильный процесс обработки информации может привести к эксплуатации уязвимостей злоумышленниками или к отказу системы в важный момент.

2. **Уровень приложения**

Он включает в себя не только инструменты, но и процессы, лучшие практики в области бизнес-рисков на уровне приложений. На данном этапе мы регламентируем разработку приложения, роли и зоны ответственности, процессы развертывания, используемые библиотеки и многое другое. На этом уровне мы сталкиваемся уже с организационными проблемами. Сейчас Code security и Application security достигли того уровня зрелости, когда мы можем смело говорить о том, что есть полноценный набор инструментов, позволяющий закрыть большинство популярных уязвимостей.

Индустрия уже давно работает над этими проблемами. Сейчас на сайтах вендоров, разрабатывающих решения безопасности, собраны лучшие практики, инструменты и рекомендации, которые можно имплементировать у себя.

На этом уровне подключаются сертифицированные эксперты безопасности и/или опытные разработчики, которые знают, как приложение ведет себя в разных сценариях.

3. DevSecOps

DevSecOps — это ответвление от парадигмы DevOps, одна из задач которой — убрать барьер между командами разработки и эксплуатации, то есть добиться максимальной синхронизации и регламентирования процесса разработки.
DevSecOps продолжает эту идею. Когда некая степень синхронизации между Dev и Ops достигнута, всплывает следующая проблема — безопасность. И Dev, и Ops должны выполнять рекомендации и работать в неком периметре безопасности. Как правило, обеспечение этого периметра в их задачи не входит и такого опыта тоже нет — требуется помощь со стороны.
DevSecOps подразумевает вовлечение в процесс обеспечения безопасности почти всех, кто участвует в разработке и использовании: специалистов по безопасности, техподдержку, а иногда даже ранних пользователей. Вся эта компания работает над обеспечением безопасности на уровнях приложения и кода и обогащением внешними рекомендациями, которые могут даже не иметь прямого отношения к команде разработки.

4. Supply Chain Security

С увеличением количества программных проектов и, как следствие, ростом количества уязвимостей было обнаружено, что большая часть угроз актуальна не только на этапе разработки или развертывания, а задолго до этого — на уровне планирования.

> **Пример.** Перед стартом разработки, в январе, мы запланировали использование в проекте неких библиотек. Разработка должна начаться в марте. В феврале стало известно об уязвимости, которую мы не отследили.

Supply Chain Security подразумевает вовлечение и анализ всех уровней, партнеров, игроков (даже тех, кто просто выложил библиотеку на GitHub). Сегодня вы узнаете, как эту проблему предлагают решать такие игроки отрасли, как Microsoft и GitHub.

Cloud Thinking

Области внимания DevSecOps и Supply Chain Security

- Физическая и виртуальная безопасность данных
- Идентификация и локализация данных
- Управление и «шеринг» данных
- Мошенничество и фрод
- (n)rd party

В область внимания DevSecOps и Supply Chain Security входят 5 важных направлений:

- **Физическая и виртуальная безопасность данных**

Как мы обеспечиваем безопасность? Какое шифрование используется? Как мы передаем данные? Какие ключи используются? Как эти ключи выпускаются? Какие сертификаты используются для передачи данных? Все эти вопросы входят в области внимания DevSecOps и Supply Chain Security.

- **Идентификация и локализация данных**

Как мы убеждаемся, что данные пришли от верного «отправителя», в корректном виде и будут далее переданы правильно? Это важный вопрос, особенно в условиях больших систем, поскольку даже незначительная ошибка может привести к тому, что данные одного клиента «пересекутся» с данными другого клиента.

- **Управление и «шеринг» данных**

Если у нас есть несколько проектов — например, мы потребляем API какой-то внешней системы, — как добиться того, чтобы с этого API приходили правильные данные? И как убедиться, что мы, отдавая данные на внешний API, передаем их в правильное место и в правильном виде? Этим вопросам также уделяется внимание DevSecOps и Supply Chain Security.

- **Мошенничество и фрод**

Атаки типа MITM, подмена компонентов или параметров, социальный инжиниринг. Если DevSecOps уделяет социнжинирингу меньше внимания, то Supply Chain Security — как раз про него.

- **(n)rd party**

Как мы контролируем качество кода, который разрабатывается в другой компании? Контактируем ли мы с этой компанией? Какие есть пути быстрого получения консультации и есть ли способы сообщить об ошибке и получить обратную связь?

Мы должны очень аккуратно использовать open-source-проекты, поскольку анализ некоторых из них показывает, что зависимости в виде заброшенных проектов могут быть важными компонентами системы и их надо постоянно контролировать и отслеживать. Для этого существуют ручные и автоматические инструменты, но тема нашей сегодняшней дискуссии — скорее регламентирование того, как это правильно делать. У нас должны быть четкие инструкции по использованию таких компонентов и решения проблем на случай, если что-то пойдет не так.

> “Supply chain security is a multi-disciplinary problem, and requires close collaboration and execution between the business, customer support and IT organizations, which has its own challenges. The companies that get this right start with IT and a secure multi-enterprise business network, then build upward with carefully governed and secured access to analytics and visibility capabilities and, from there, continuously monitor every layer for anomalous behavior.”
>
> **Marshall Lamb, CTO, IBM Sterling**

По мнению Маршалла Лэмба, CTO в IBM Sterling, Supply Chain Security является мультидисциплинарной проблемой и нуждается в том, чтобы в нашей ИТ-организации, состоящей из департаментов разработки, операций, тестирования и других, существовало плотное взаимодействие. Часть этой задачи закрывается инструментами, но ее львиная доля — это процессы.

Посмотрим, с помощью каких инструментов это реализуется.

Автоматизация и интеграция – ключи к успешным DevSecOps и Supply Chain Security

Есть два ключевых направления, в которых мы должны крайне аккуратно подходить в реализации нашей стратегии DevSecOps и Supply Chain Security, — автоматизация и интеграция.

Автоматизация

Автоматизация действий, развертывания, автоматическая генерация кода — все это должно быть четко регламентировано. Мы должны точно знать, откуда к нам все приходит, постоянно мониторить доступность этих ресурсов, так как распространенной проблемой является падение CDN или источник, откуда мы «подтягиваем» зависимости. У нас должна быть инструкция и ее инструментальная реализация на случай, если это произойдет.

Например, переключение на другой дизайн, инстанс сервера или любые другие меры, которые укладываются в адекватную стратегию бизнеса.

Интеграция

Интеграция бывает двух типов:

- Интеграция с внешними партнерами (исходящая)

Например, когда нам нужно сканировать контейнеры при развертывании в Production, мы можем использовать не только свое средство или кастомный код, но и партнерское решение. И мы должны быть в курсе, что с ним происходит: каков road map, как оно развивается, какие планы у компании-разработчика на это решение.

- Интеграция с нашей системой

Также стоит анализировать, как мы позволяем интегрироваться с нашей системой внешним сущностям, например клиентам. Некоторые клиенты аккуратно используют ваш API, а некоторые выходят за регламент/лимиты, тем самым провоцируя риск выхода из строя вашей системы. Здесь мы приходим как к регламентам, так и к инструментальным средствам. И если по умолчанию у нас есть API, мы можем использовать набор инструментов

под названием API Management и на уровне API Management, согласно утвержденным регламентам, можем делать настройки логического уровня и определять, как этот API будет использован внешними клиентами.

Обратимся к опыту индустрии.
GitHub, являясь давним, как они сами заявляют, домом для разработчиков, имеет собственную зрелую стратегию обеспечения DevSecOps, которая называется «сдвиг справа влево».

GitHub разработал богатый инструментарий для автоматического анализа частично тех угроз, которые мы рассмотрели выше, частично тех, с которыми сталкивается их платформа, и частично даже инструментарий, который может дать нам аналитику, полезную для разработки регламентов, о которых мы говорили ранее.

DevSecOps с GitHub Advance Security

Выделяется три ключевых направления:

- **Dependency Updates**

Мониторинг использования open source и обнаружение уязвимостей в вашей кодовой базе.

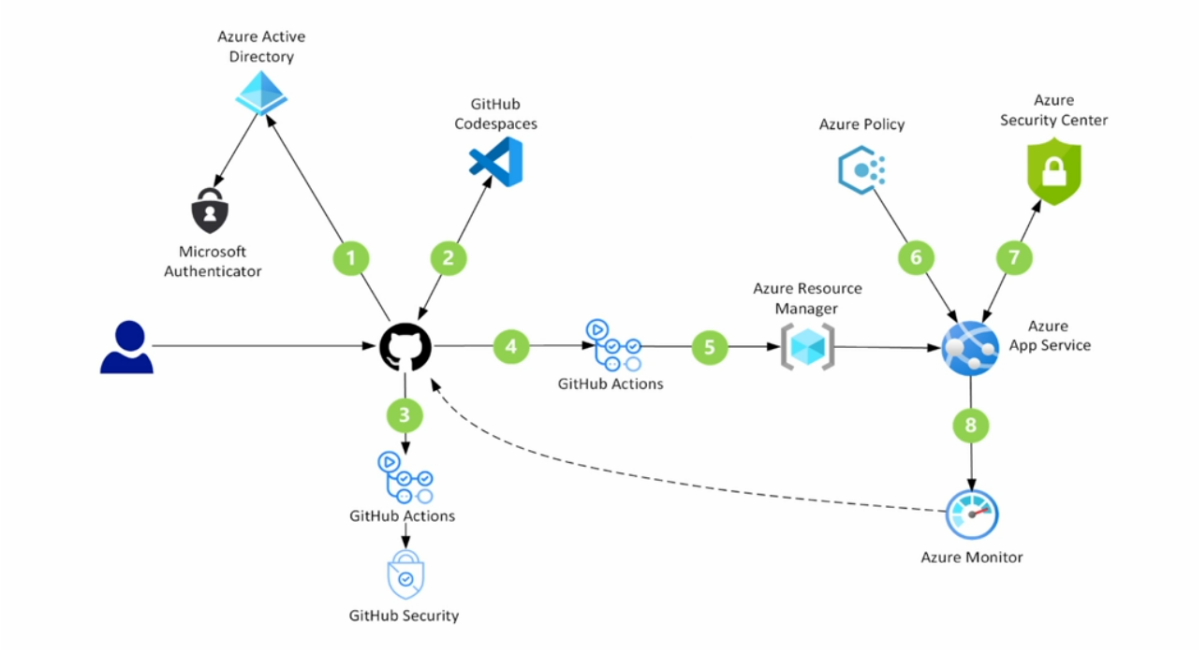
- **Secret scanning**

Обнаружение паролей, строк подключений и другой «секретной» информации в ваших репозиториях. Может показаться, что эта проблема надуманна, но, если смотреть на отчеты, разработчики очень часто оставляют критические credentials в коде, в репозиториях. Самый простой способ избежать этого — автоматизация обнаружения секретов. У GitHub есть инструменты, которые на разных стадиях могут отслеживать, что передается в коде. Также в рамках DevSecOps можно использовать внешние инструменты вроде Azure Key Vault для хранения секретов, credentials и на уровне кода забирать эти данные оттуда.

- **Code scanning**

Как правило, это статический анализ кодовой базы, например с помощью технологий и правил Code QL. Code QL — ноу-хау GitHub. Это автоматический движок для анализа на основе искусственного интеллекта, который применяется к коду, выложенному на GitHub. Разумеется, можно обойтись и без автоматического статического анализа — например, применять его только по факту или настраивать это в пайплайне разработки.

Обратите внимание на референсную схему пайплайна разработки, развертывания и дальнейшего мониторинга того, что мы развернули в облако Azure. В этой схеме применяются инструменты GitHub.



На схеме изображен богатый набор сервисов. Пройдемся по наиболее интересным.

Разработчик пишет код и делает коммит в GitHub. Перед коммитом он аутентифицируется и авторизуется с помощью каталога Azure Active Directory (каталог на основе ролей и разрешений с богатой моделью авторизации). Azure AD можно использовать как каталог в облаке, так и в различных гибридных режимах.

Разработчик получает токен, разрешение на развертывание кода, после чего инициируется выполнение сервиса GitHub Actions. Вы можете использовать этот инструмент, чтобы соединять несколько шагов пайплайна, который происходит даже где-то в другом месте, но наиболее удобно использовать его, конечно, для GitHub. GitHub Actions — это задача, которую вы конфигурируете, последовательность действий, которая что-то выполняет. Это может быть анализ, сканирование секретов или любая другая последовательность действий. Вы настроили скрипт в GitHub Actions, и он попеременно выполняет заданные действия.

В процессе и после выполнения вы можете изучить, что GitHub Actions отдал в качестве логов, ошибок, и проанализировать каждый этап выполнения.

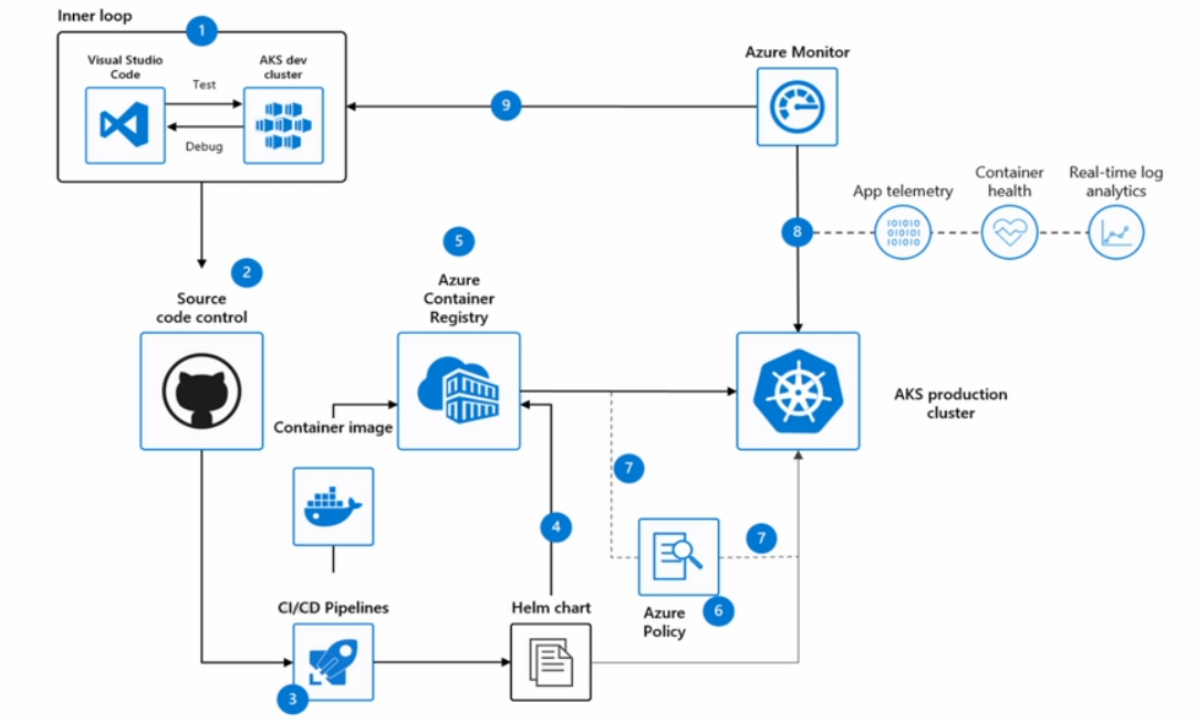
После того как код успешно развернулся и были пройдены все проверки безопасности, запускается еще один GitHub Action, который собирает наш код (возможно, дополнительно проверяет его) и разворачивает это все на виртуальную машину, Kubernetes, в Azure WebApps или любое другое место. В данном случае имеются как автоматические скрипты для развертывания, автоматическая интеграция для разных сервисов, так и возможность кастомного развертывания — например, через какой-нибудь веб-деплой.

GitHub Actions отрабатывает, наше приложение, пройдя некоторое количество автоматических проверок, попадает на хостинг. Это может быть как тестовый, так и продакшен-хостинг или любые другие модели. Далее к этому хостингу подключаются внешние сервисы, настроенные согласно нашим регламентам. А именно — один из самых интересных и полезных сервисов на платформе Azure — Azure Policy. Это движок политик. Вы настраиваете политику — например, что данный код должен удовлетворять определенным политикам или, если это виртуальные машины, можно даже настроить, что на этих ВМ должно быть выбранное окружение.

Если ресурсы не удовлетворяют политикам, запускается mitigation-задача — установка софта, редеплой или любые другие задачи по избеганию того, что не соответствует нашим политикам. В Azure Policy можно без проблем настроить то, что может быть сложно настроить программным образом на каком-то уровне разработки или развертывания.

К сервису подключен Azure Monitor, который на уровне приложения, на уровне вычислительных ресурсов, на уровне логических приложений может сообщать нам, что данные ресурсы периодически переиспользуются. Иными словами, нагрузка такова, что теоретически может привести к тому, что наш сервис выпадет из рабочего режима.

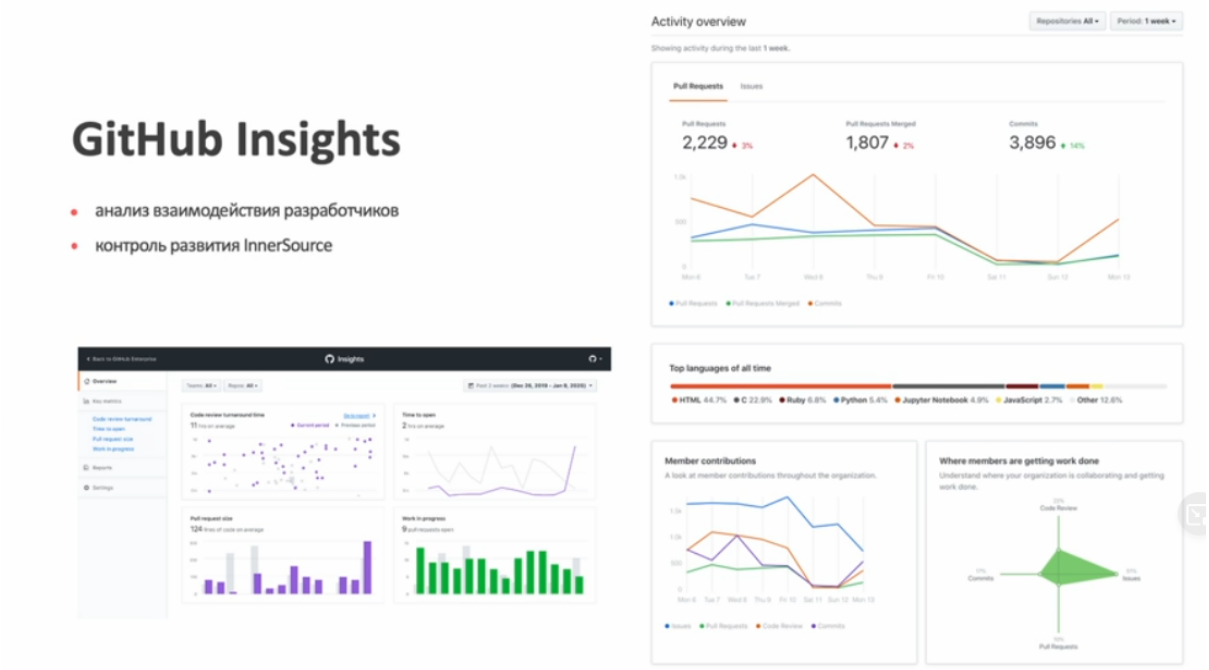
Давайте посмотрим на еще одну схему.



Она чем-то похожа на предыдущую, но осложнена за счет добавления Kubernetes. Как известно, когда мы начинаем использовать микросервисы, контейнеры и другие хорошие вещи, наша жизнь может как значительно упроститься, так и стать значительно тяжелее. В данном случае — за счет того, что ресурсы внутри Kubernetes постоянно меняют свое состояние (абстракция немного больше, чем на простом хостинге) и нам нужны инструменты, которые позволят обеспечивать безопасность на уровне K8s.

На этой схеме видно, что разработчик/Ops/специалист по безопасности может настроить это в том числе на логическом уровне. Парадигма DevSecOps в отношении Kubernetes привела к разработке большого количества документов и лучших практик как по обеспечению безопасности на уровне контейнера, кластера и нагрузок, так и, как следствие, мониторинга всех этих лучших практик и безопасности в реальном времени в автоматическом режиме. Все эти инструменты есть и на GitHub — они абсолютно официальны и вы можете использовать их в своей стратегии автоматизации пайплайна разработки и развертывания.

Также нельзя оставить без внимания GitHub Insights. До этого мы рассматривали прикладные инструменты. Однако индустрия и проблемы с безопасностью процесса разработки ПО показали, что нам нужны не просто прикладные инструменты, а инструменты, которые дадут аналитику о том, как работают разработчики, что они делают. И GitHub Insights позволяет анализировать взаимодействие разработчиков практически в реальном времени: какие коммиты произошли, что настроено, что работает, а что нет. GitHub Insights может стать как важным компонентом вашей стратегии безопасности, стратегии DevSecOps и Supply Chain Security, так и важным фактором развития InnerSource.



Помните, мы говорили, что DevSecOps был призван решить одну из серьезных проблем взаимодействия между командами Dev, Ops и Sec? Но как это сделать?

Концепция InnerSource подразумевает, что мы, инкрементальным образом анализируя происходящее (как разработчики работают, как используют инструменты, как взаимодействуют между собой), на каждом этапе принимаем какое-то решение, которое приводит нас к реализации задачи по объединению команд.

InnerSource получил большое внимание особенно в последние годы по той же самой причине: проекты усложняются, повышается необходимость работать вместе и как можно более плотно обеспечивать взаимодействие между всеми командами, чтобы сделать максимально безопасное и расширяемое к новым вызовам решение.