Контроль безопасности стороннего кода при разработке

Алексей Федулаев

DevSecOps Lead

$whoami

Алексей Федулаев

12+ лет в ИБ

Руководитель направления автоматизации безопасной разработки Wildberries

О чем сегодня поговорим?

- О зависимостях
- Какие опасности они в себе таят?
- От кого мы защищаемся?
- Как нужно защищаться?

Современную разработку сложно представить без зависимостей

Но зависимости – это тоже код и он содержит баги

Сканеры - не панацея тут

- На реакцию нужно какое-то время, эвристика не всегда помогает
- Некоторые производители сканеров не считают некоторые формы protestware чем-то плохим, соответственно, не блокируют их
- Некоторые сканеры ушли из российского сегмента рынка

Проблема гораздо шире

- Постоянно взламывают разработчиков различных пакетов, добавляя вредоносный код в их продукты
- Иногда сами разработчики добавляют вредоносный код
- В сети можно найти тысячи фейковых библиотек, можно только догадываться, что они содержат

Какие еще угрозы несут зависимости?

- Уязвимости
- Бэкдоры
- Стиллеры
- Майнеры
- Трояны
- Шифровальщики
- И др.

Что в итоге делать?

Запрет обновлений?

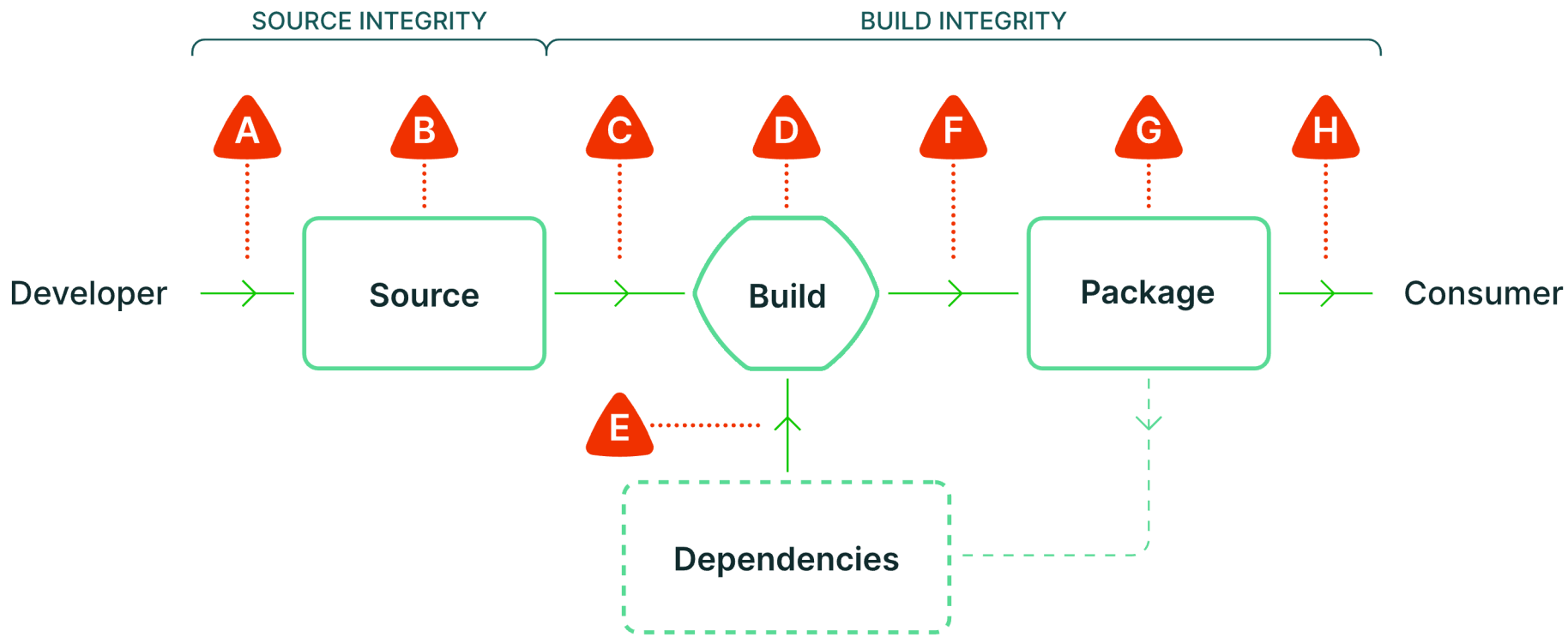
- Некоторые компании запретили обновляться после появления protestware

SLSA

Фреймворк для повышения уровня безопасности программного обеспечения и целостности цепочки поставок.

https://slsa.dev

Threat model



A Submit unauthorized change

B Compromise source repo

C Build from modified source

D Compromise build process

E Use compromised dependency

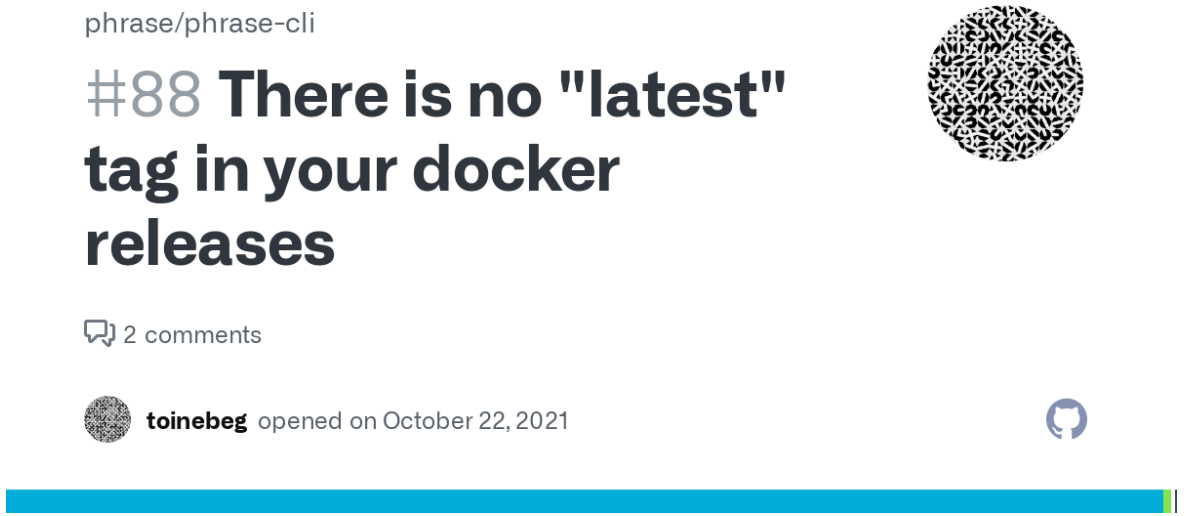
F Upload modified package

G Compromise package repo

H Use compromised package

Но кое-что в этом есть

- Запрет использования “latest”
- Тем самым в случае появления нежелательного кода в библиотеке мы автоматом не затащим его к себе

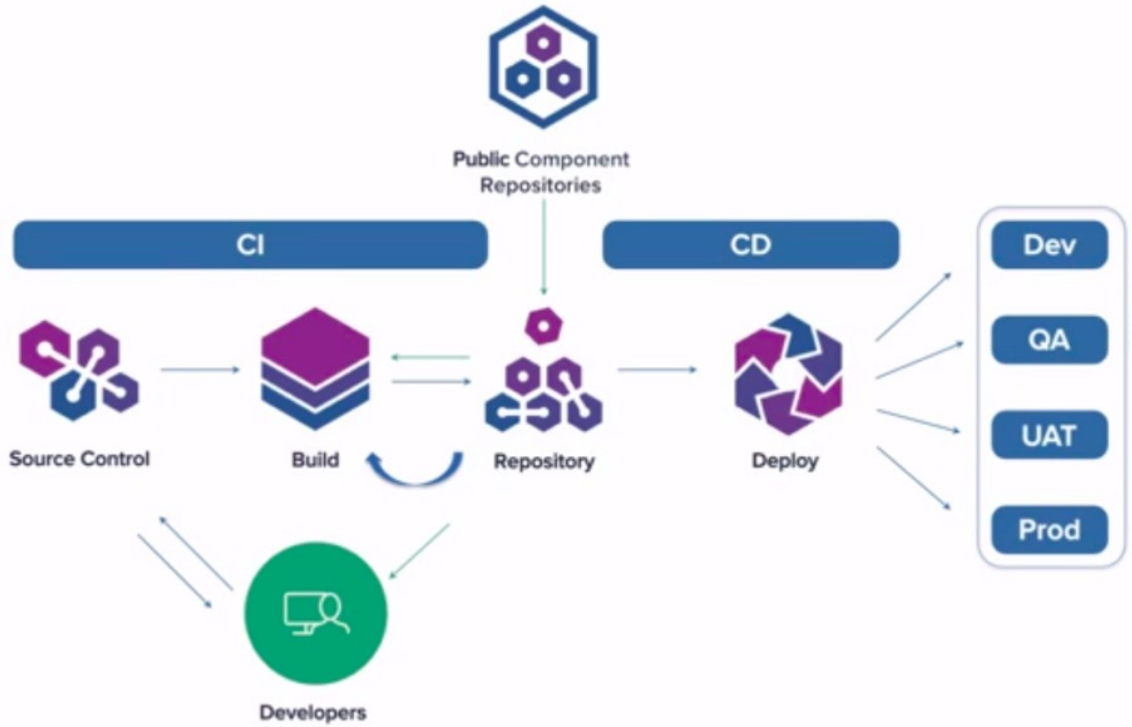


Закрепление версий пакетов

```
"dependencies": {
  "@ampproject/remapping": "2.2.0",
  "@angular-devkit/architect": "0.1402.10",
  "@angular-devkit/build-webpack":   "0.1402.10",
  "@angular-devkit/core": "14.2.10",
  "@babel/core": "7.18.10",
  "@babel/generator": "7.18.12",
  "@babel/helper-annotate-as-pure": "7.18.6",
```

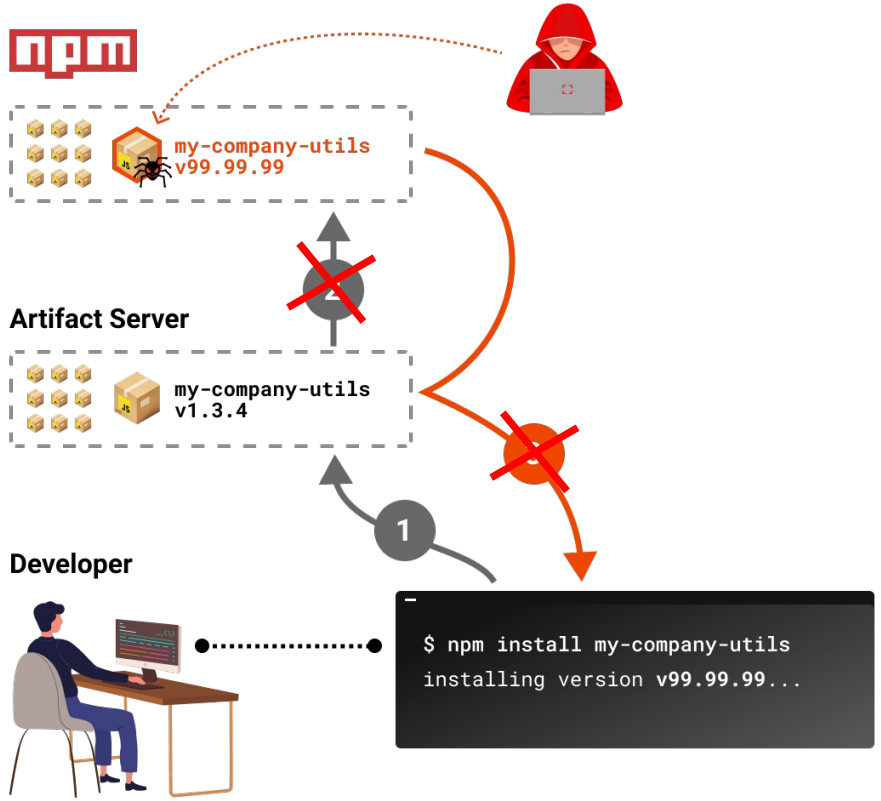
Создание локального хранилища артефактов

- Использование hosted-репозитория
- Вы контролируете все зависимости
- Все зависимости всегда доступны. Их невозможно удалить, заблокировать к ним доступ



Исключение доступа в глобальную сеть

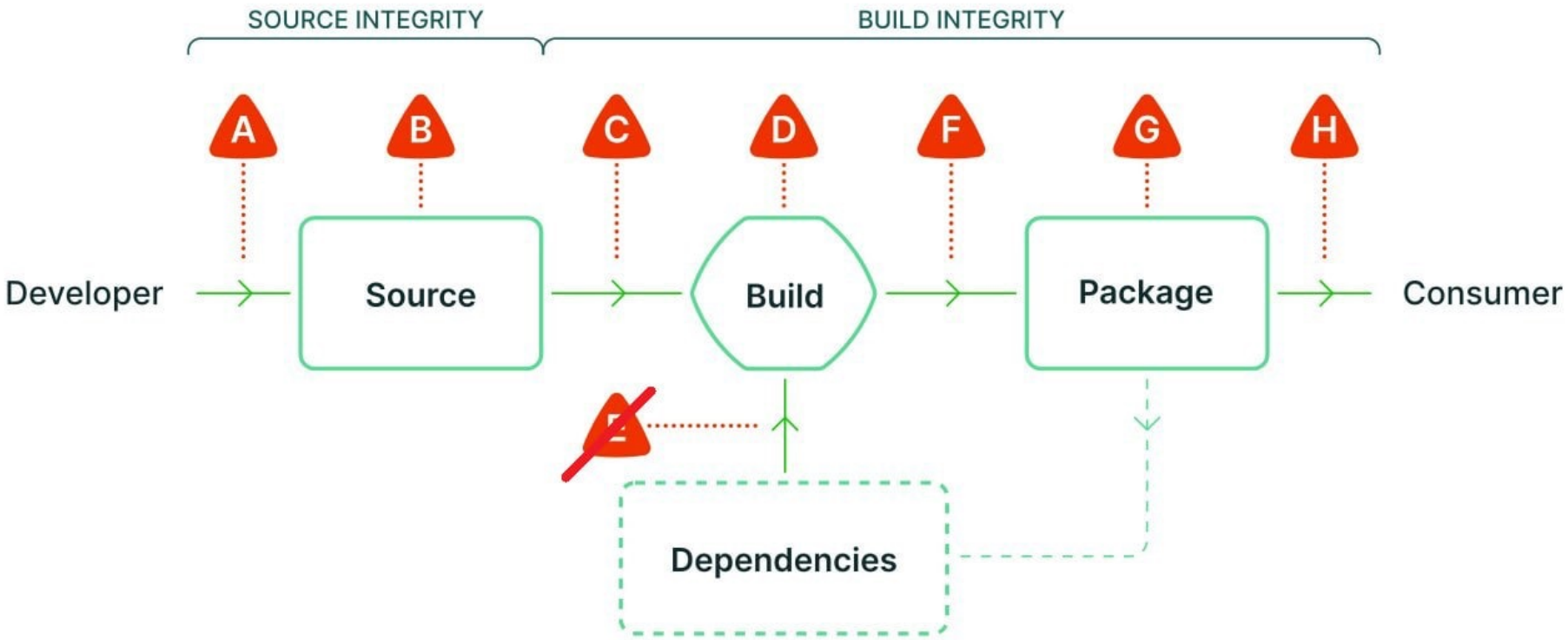
- Правильная конфигурация
- Запрет на уровне правил FW
- Манипуляция DNS
- Изоляция
- И др.



Запрет бесконтрольного пуша зависимостей

- Добавление зависимостей в репозиторий только доверенными лицами
- Проверка зависимостей при добавлении (происхождение зависимости, количество загрузок, наличие CVE, Issues, контроль целостности зависимости)
- Тестирование зависимостей (использование в песочнице, отслеживание syscalls и т.д.)
- Проверка соответствия зависимостей в окружении
- Хардкор: дизассемблер и декомпилятор

Threat model



A Submit unauthorized change

B Compromise source repo

C Build from modified source

D Compromise build process

E Use compromised dependency

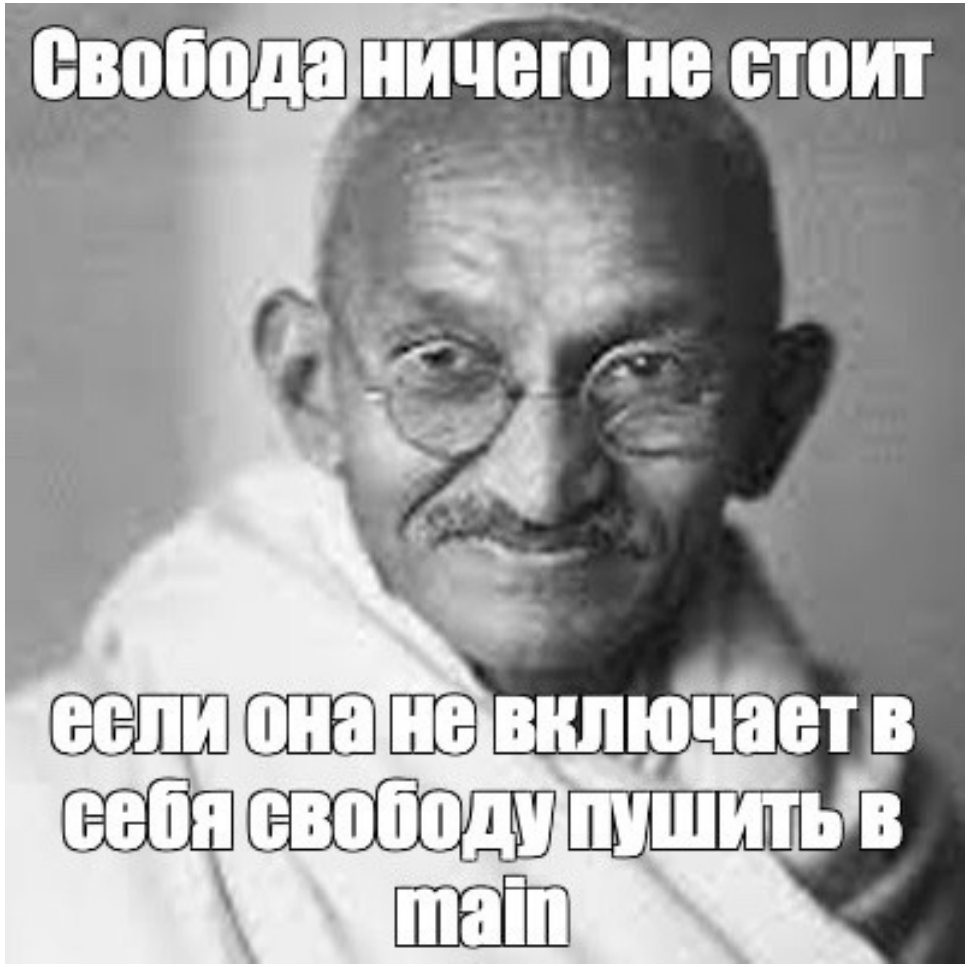
F Upload modified package

G Compromise package repo

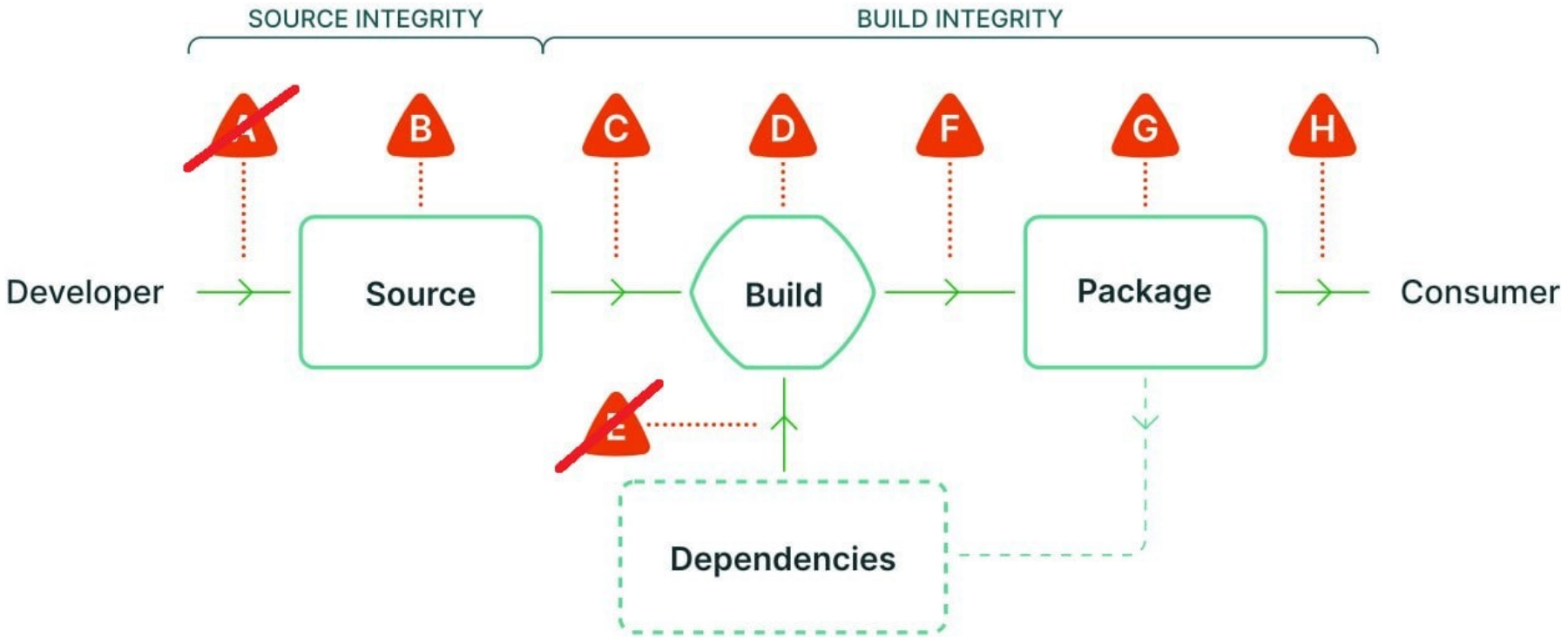
H Use compromised package

Контроль версий пакетов

- .lock-файлы в репозитории
- Запрет пуша в main, только через Merge request
- Review изменяемых .lock-файлов
- Проверка зависимостей
- Запрет влития без апрува
- Апрув несколькими лицами



Threat model



A Submit unauthorized change

B Compromise source repo

C Build from modified source

D Compromise build process

E Use compromised dependency

F Upload modified package

G Compromise package repo

H Use compromised package

Использование подписи

- Подпись коммитов
- Подпись зависимостей
- Проверка аутентичности артефактов при сборке приложения
- Подпись самого приложения
- Проверка артефактов при сборке образа и подпись образа

Не только лишь в сборке

- Контроль аутентичности образов при развертывании
- Запрет использования неподписанных образов политикой

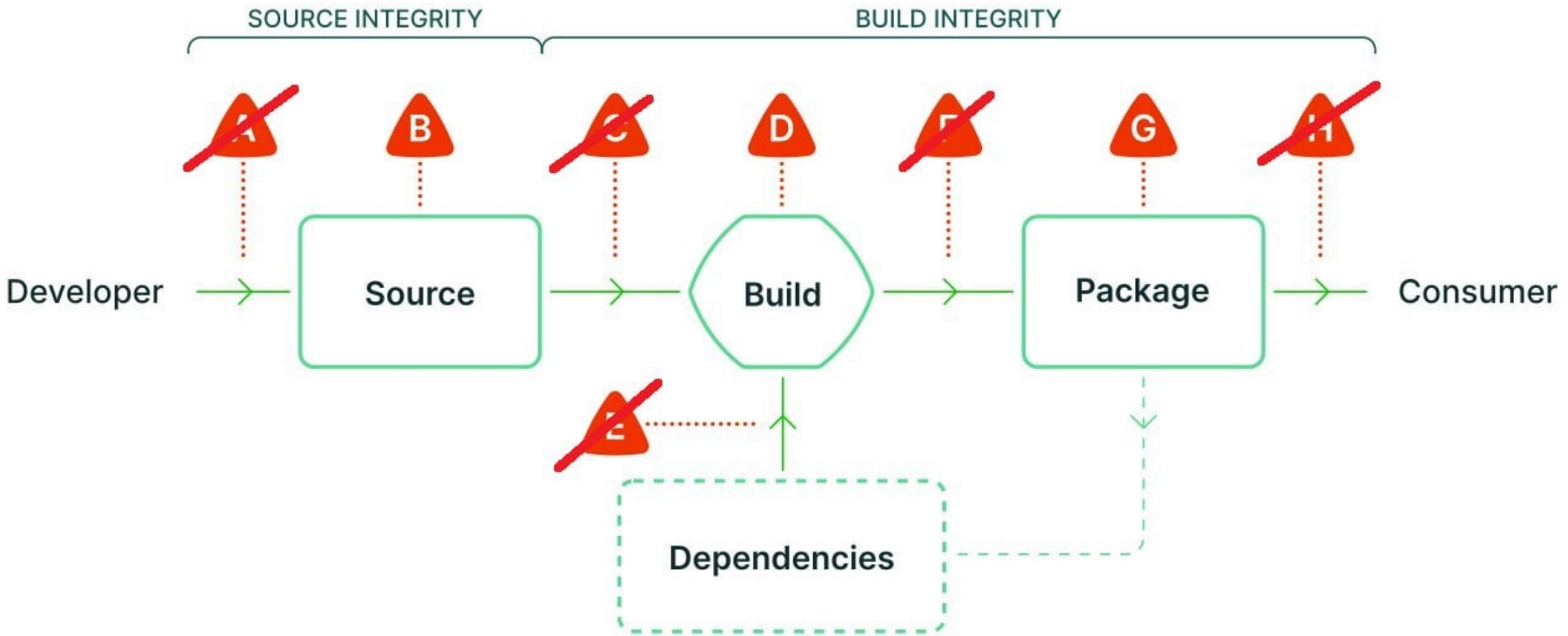
Open Source Security Foundation

https://openssf.org

Не забудьте про ключи

- Не храните закрытые ключи в репозитории и в переменных репозитория
- Используйте специализированные решения типа vault

Threat model



A Submit unauthorized change

B Compromise source repo

C Build from modified source

D Compromise build process

E Use compromised dependency

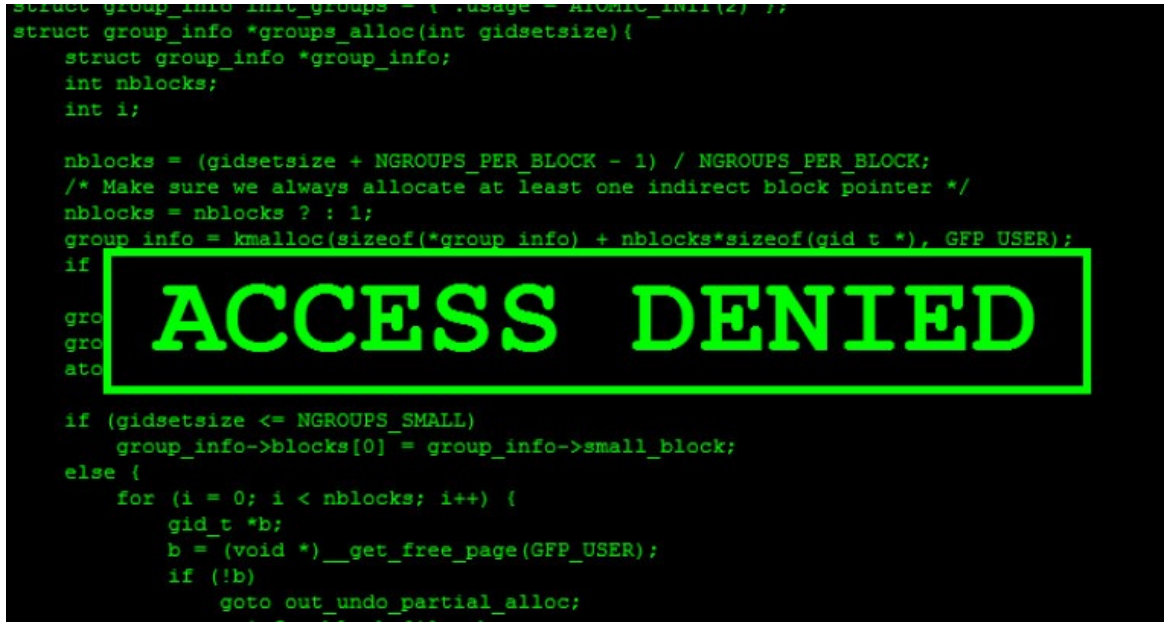
F Upload modified package

G Compromise package repo

H Use compromised package

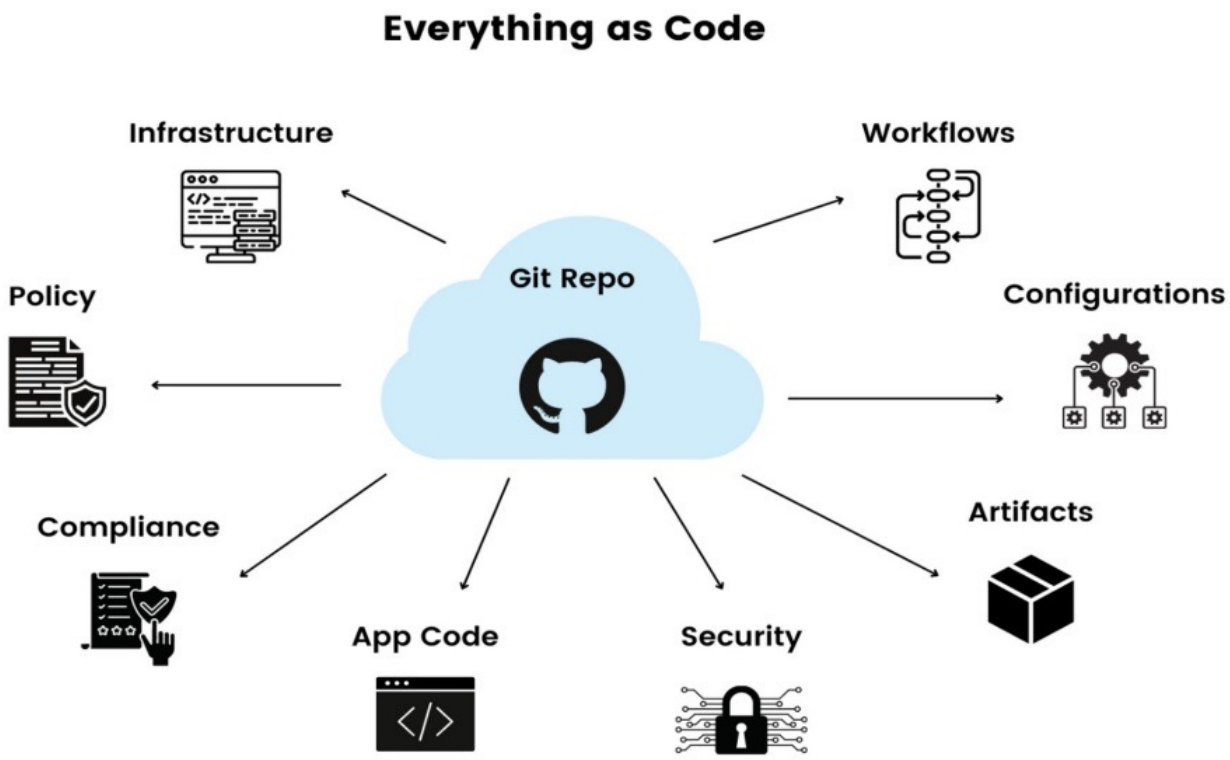
Но как тогда командам разработки исследовать новые зависимости?

- Создание hosted- и cache-репозиториев
- Разделение среды и системы сборки для dev и production
- Использование только локального репозитория артефактов для prod
- Изолированные сборки
- Генерируемая среда сборки для каждой job



Исключение воздействия на процесс сборки

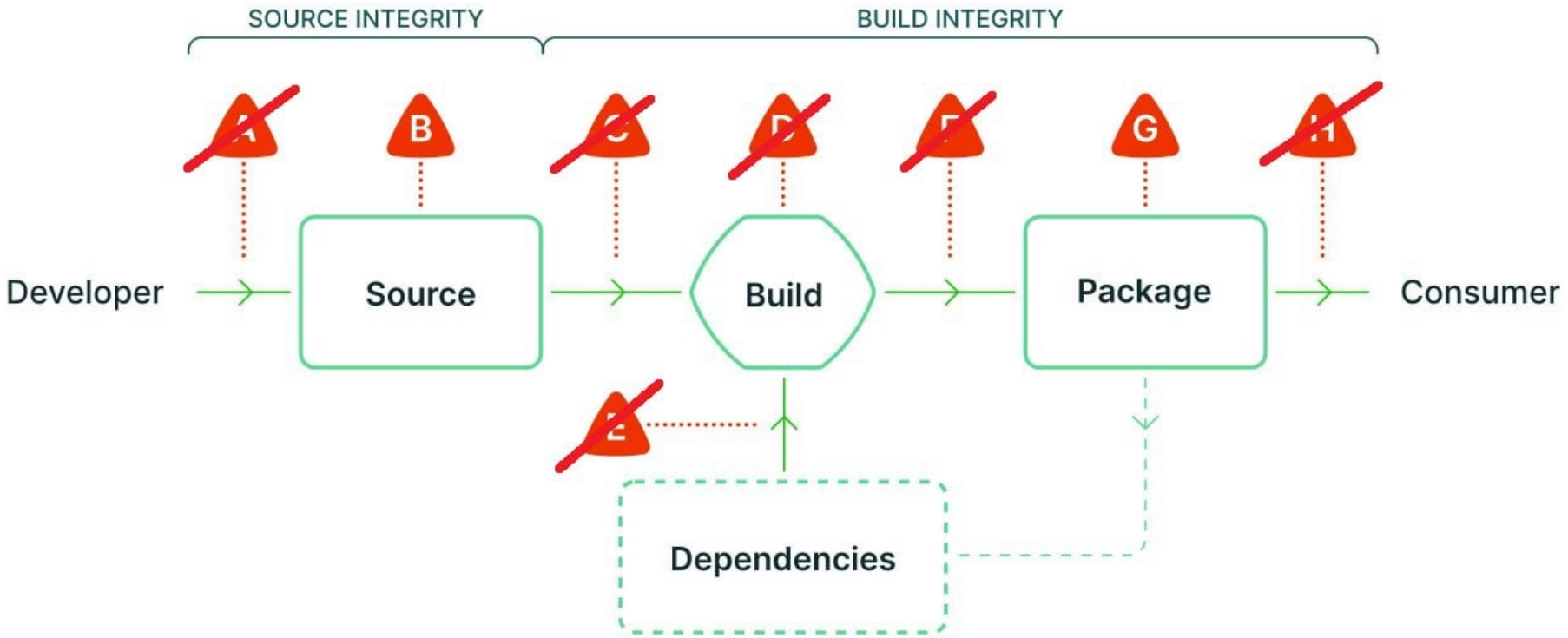
- Everything as Code
- Легкий контроль с помощью системы контроля версий
- GitOps – единственный источник правды и эталонная модель для верификации



И контроль этого кода!

- Хранение кода инфраструктуры, скриптов сборки, конвейеров сборки и т.д. в репозитории/иях
- Контроль версий этого кода
- Ревью всех изменений, апрув при MergeRequest

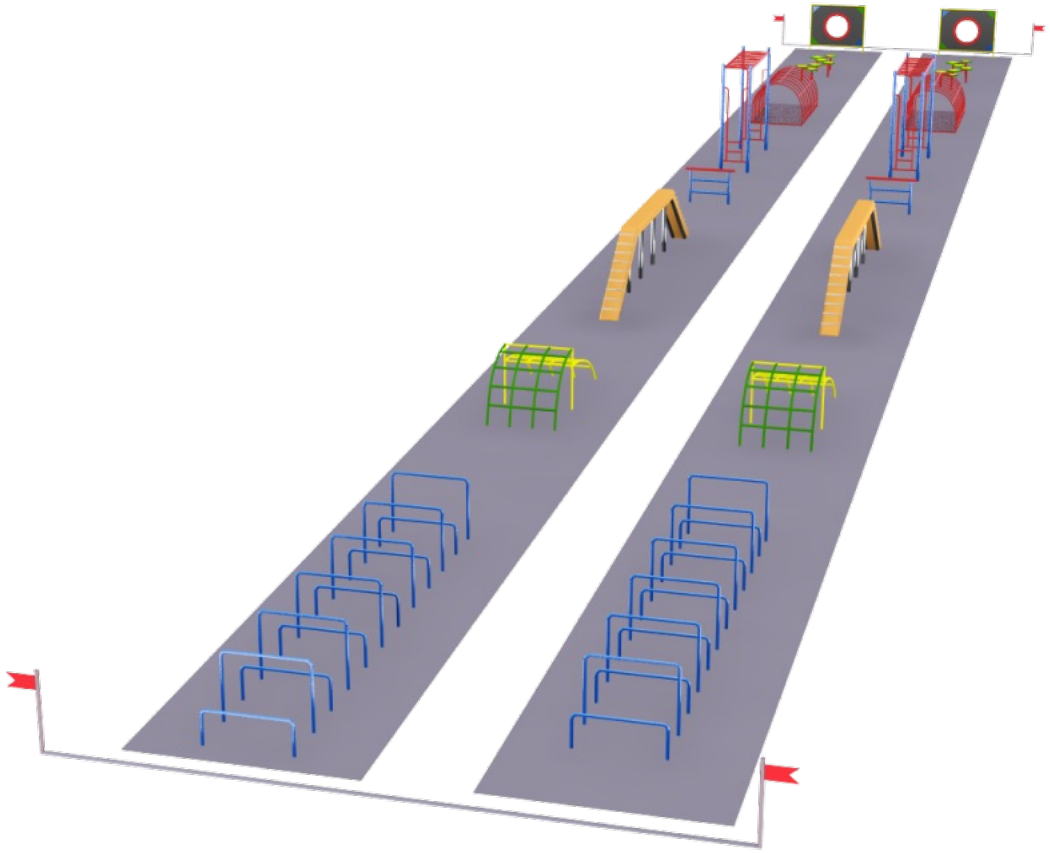
Threat model



A Submit unauthorized change
B Compromise source repo
C Build from modified source
D Compromise build process
E Use compromised dependency
F Upload modified package
G Compromise package repo
H Use compromised package

Эшелонированная безопасность

- Каждая из используемых практик легко обходится по отдельности
- Однако все вместе превращается в серьезную проблему для нарушителя



Выводы

- Зависимости содержат множество разных опасностей
- Даже «хорошая» зависимость может быть взломана
- Если вы не контролируете свои зависимости – их контролируют злоумышленники

Вопросы?

- Алексей Федулаев
- tg: @int0x80h