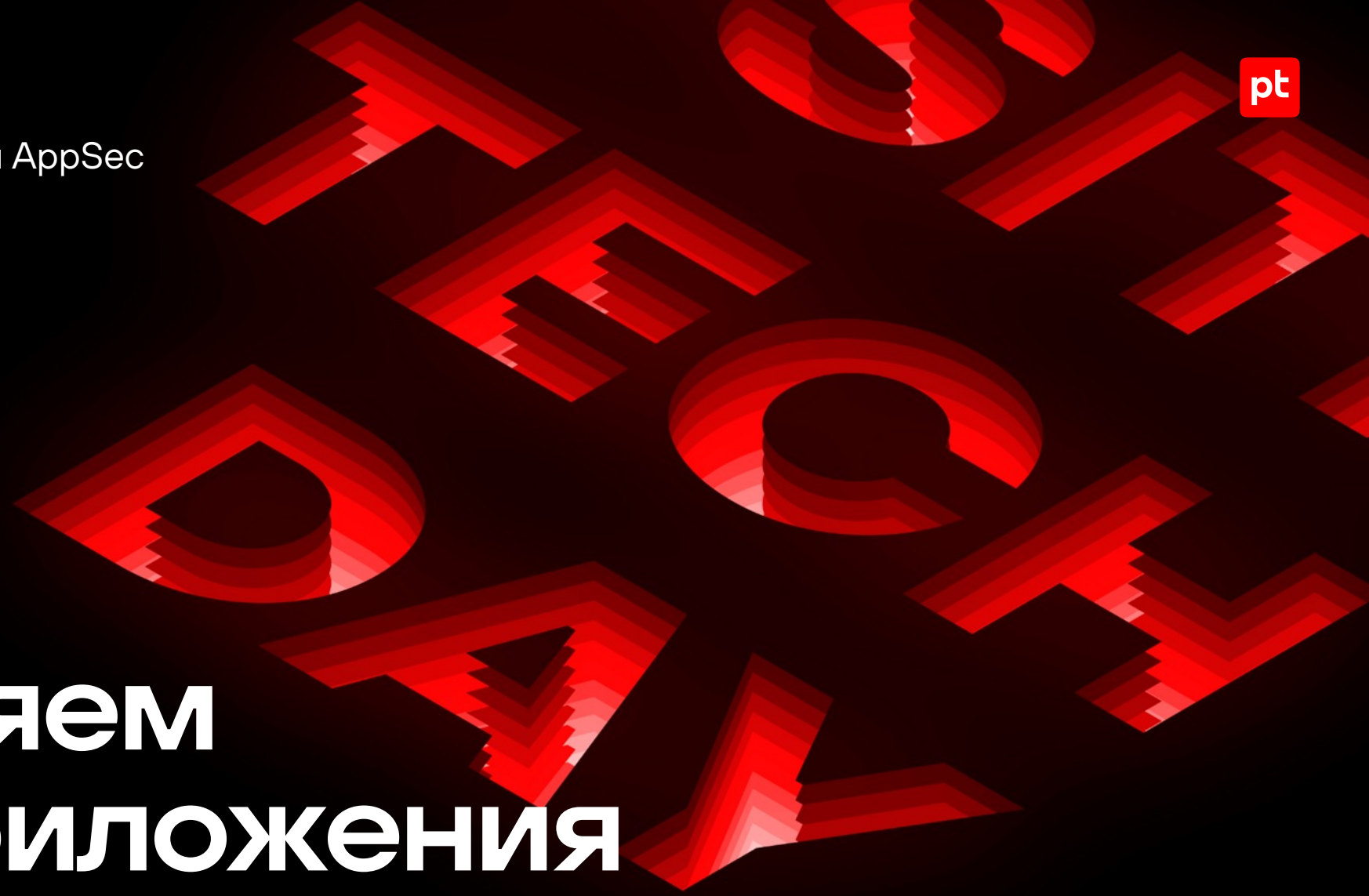
Алексей Жуков
Лидер продуктовой практики AppSec



Проверяем код и приложения в тестовом контуре

Современная разработка

- Большие объемы кода
- Разнообразие языков и технологий
- Жесткий кадровый дефицит
- Повсеместное использование сторонних компонентов
- Высочайшие темпы для снижения time to market
- Общая недооценка темы ИБ применительно к R&D

AppSec — это, конечно, команда

BSIMM 14:
результаты обследования 130 компаний, разрабатывающих суммарно почти 100 тысяч приложений

270 000
Почти столько разработчиков

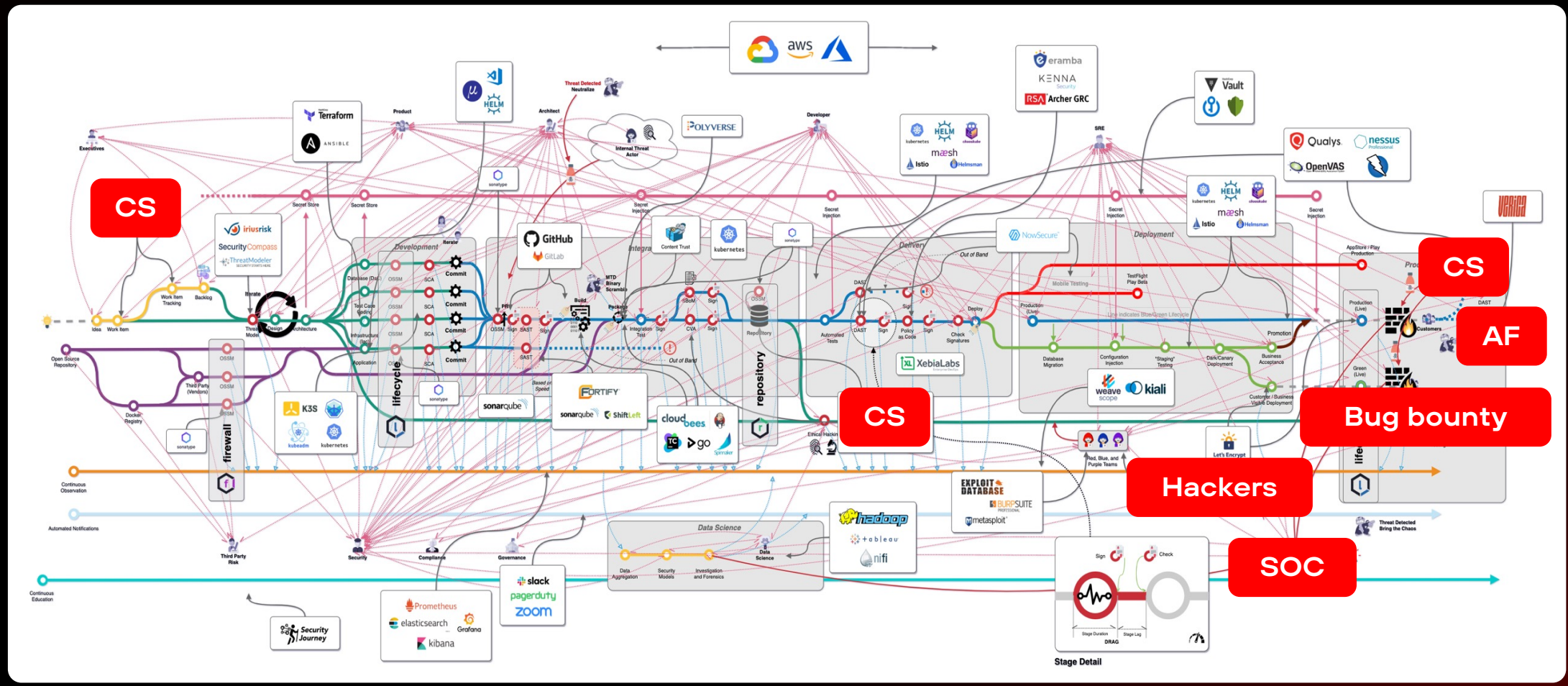
3572
Специалиста по ИБ

1/75
Соотношение ИБ и R&D

Но AppSec — это еще и инструменты

Все достаточно позитивно

Но кажется, что чего-то не хватает



Анализ кода приложения

Анализ кода — это просто

Приложение — это код, написанный разработчиком, плюс используемые им сторонние компоненты

Код разработчика

1. Если соблюдать все **правила**, то код будет идеальным. И **безопасным**, разумеется
2. Нужно написать правила. Много правил. Чем **больше** правил, тем **безопаснее** будет код

Берем код, прогоняем через набор правил, формируем перечень фрагментов кода, нарушающих правила. Исправляем код

Сторонние компоненты

1. Информация про их уязвимости регулярно **появляется,** в том числе и в открытом виде
2. Для подключения библиотеки есть **стандартные** методы (Maven, Composer, NuGet, NPM, PyPI и т. д.)

Парсим файлы из второго пункта, прогоняем результаты через базу уязвимостей из первого. Меняем библиотеки на исправленные

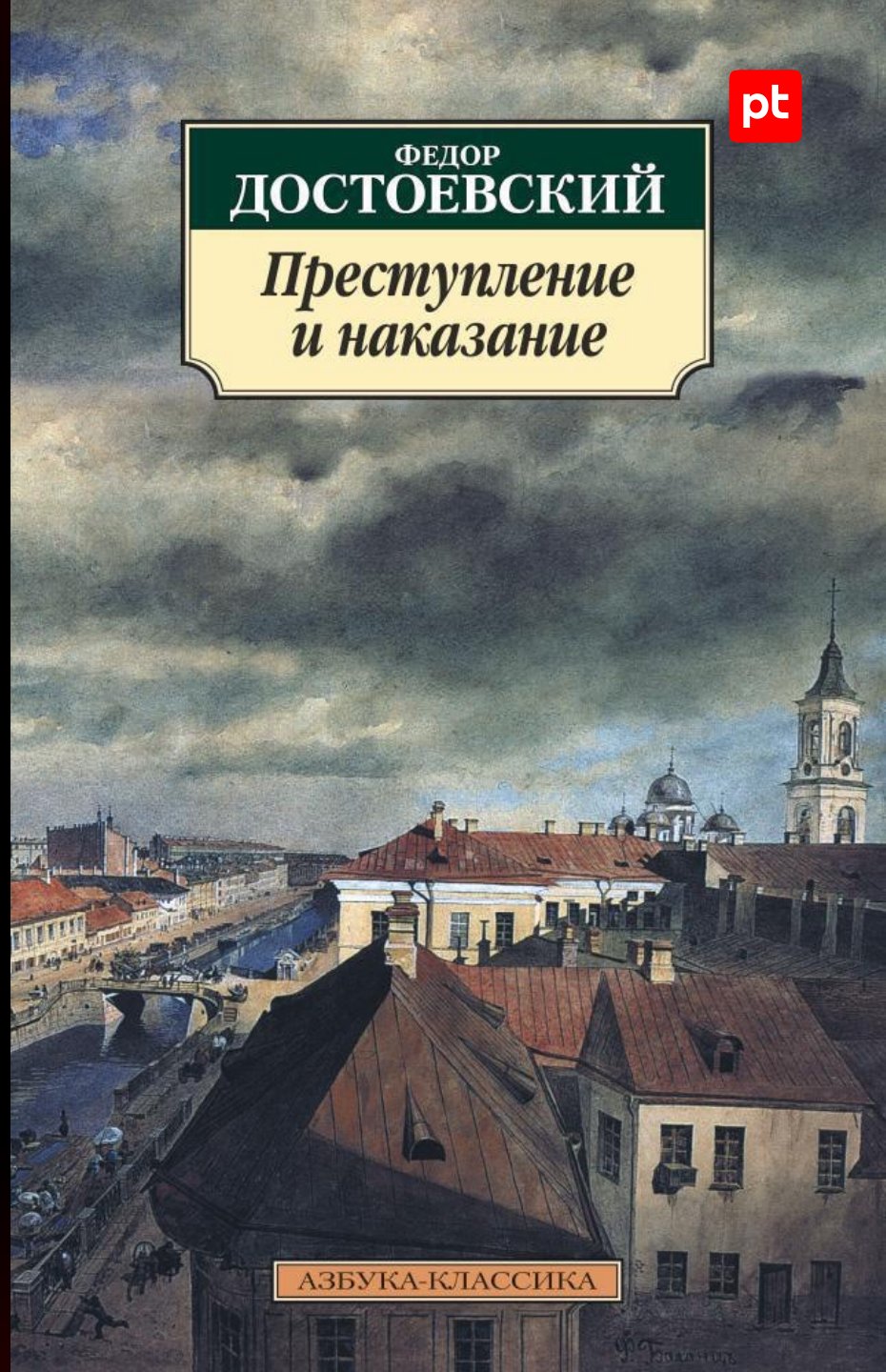
Жаль, но это так не работает

Код — это не просто текст

Код, как и книги, пишут люди, и он может быть устроен весьма сложно.

У «шаблонов», «правил» и «сигнатур» есть своя ниша, но нужны и другие методы анализа, выходящие за эти рамки.

Уязвимость — это не только подозрительная строка в коде, но и негативные **последствия** от непредусмотренного **варианта ее выполнения**



Ничего нет проще этой SQLi

1. $id = $_GET['id'];
2. $sql = "SELECT fname, lname FROM users WHERE id = '$id'";
3. $result = mysqli_query($link, $sql);

При значении id равном «' OR 1=1; -- » запрос к БД вернет все записи таблицы пользователей:

```
SELECT fname, lname FROM users WHERE id = '' OR 1=1; -- '";
```

Устраняем уязвимость

```
1. $id = $_GET['id'];
2. $id = preg_replace('/[^0-9]+/', '', $id);
3. $sql = "SELECT fname, lname FROM users WHERE id = '$id'";
4. $result = mysqli_query($link, $sql);
```

Оставляем только цифры в идентификаторе

А если так?

1. $id = $_GET['id'];
2. $id = preg_replace('/./', '.', $id);
3. $sql = "SELECT fname, lname FROM users WHERE id = '$id'";
4. $result = mysqli_query($link, $sql);

Просто меняем запятые на точки

Парадокс?

Код нужно выполнять

1. Иначе непонятно, как определить, что выполнение дойдет до уязвимой строки кода
2. И непонятно, как в противном случае узнать, нейтрализован ли вектор атаки

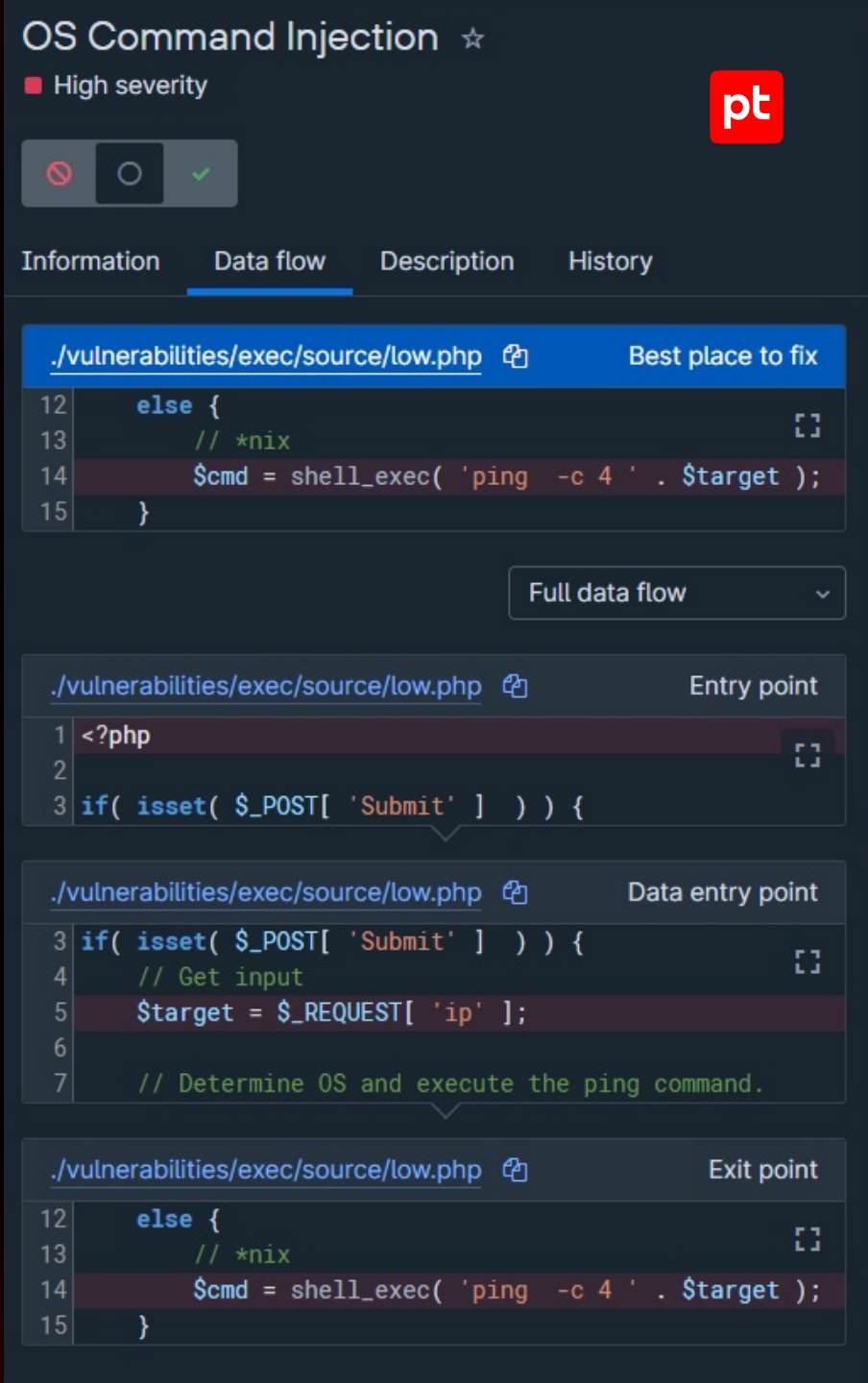
Код нельзя выполнять

1. Выполнение кода может зависеть от входных данных, непонятно, как перебрать их все
2. Выполнение кода может быть длительной задачей, мы не можем себе позволить проводить анализ слишком долго

PT Application Inspector

SAST:

- Точность и наглядность получаемых результатов
- Снижение трудозатрат на обработку срабатываний
- Возможность встраивания в конвейер сборки CI/CD
- Бесплатные плагины для IDE от JetBrains и VS Code
- Открытый API для интеграции



С SCA тоже непросто

Сам факт использования компонента с известной уязвимостью еще ни о чем не говорит.

Чтобы доказать, что уязвимость существует, нужны дополнительные исследования

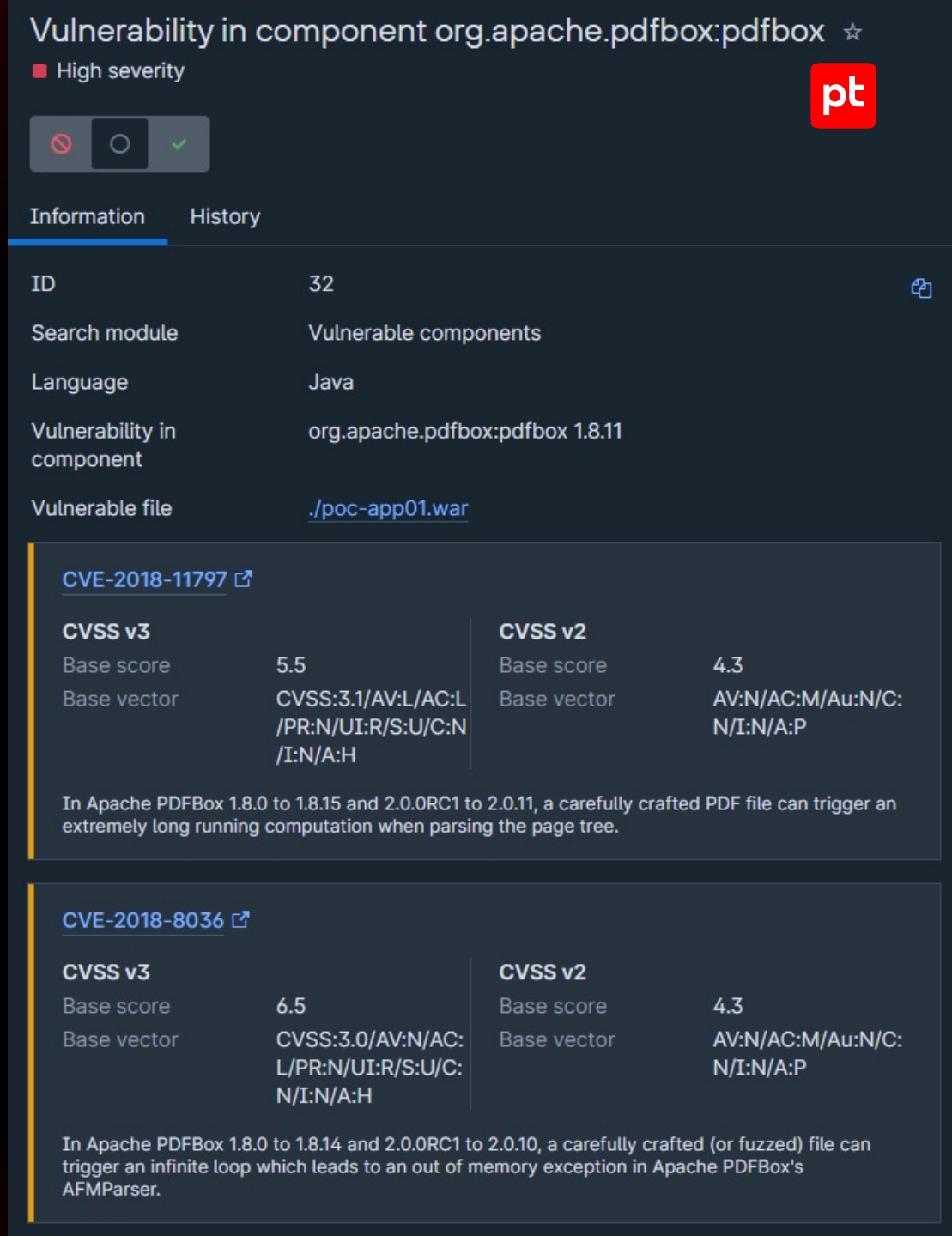
PT Application Inspector

SCA сейчас:

- Выявление компонентов, используемых приложением
- Определение перечня известных уязвимостей

В перспективе:

- Определение возможности эксплуатации уязвимости в конкретном приложении



Динамический анализ приложения

Анализ запущенного ПО

1' adn '59098'='59099

Request

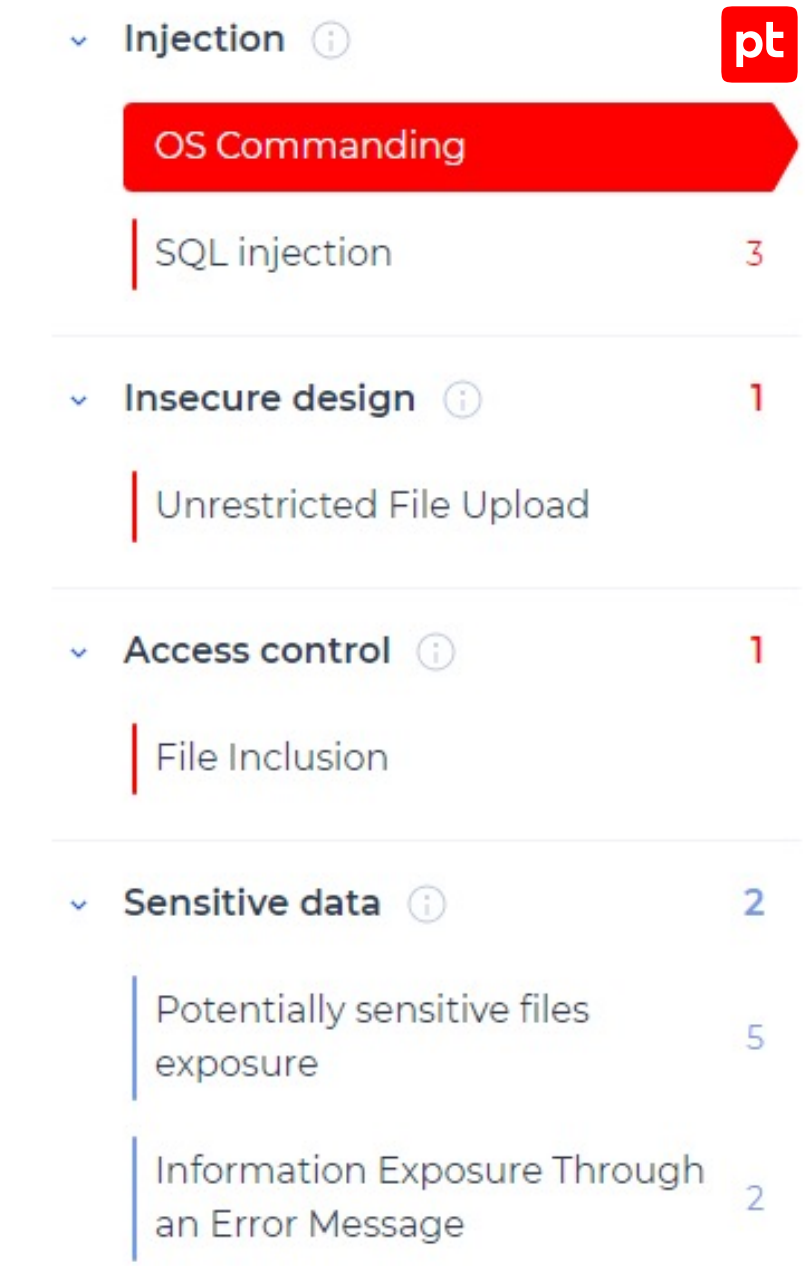
```
1  GET /vulnerabilities/sqli/?id=1'%20adn%20'59098'='59099&Submit= HTTP/1.1
2  Host: dvwa.ptdemo.local
3  Accept: */*
4  Accept-Encoding: deflate, gzip, br
5  Cookie: security=low; PHPSESSID=d288c76fa9ccfe6169b9099a60513a6c
6  cache-control: no-cache
7  pragma: no-cache
8  user-agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
9
10
```

Response

```
1  <br />
2  <b>Fatal error</b>:  Uncaught mysqli_sql_exception: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the ri
3  Stack trace:
4  #0 /var/www/html/vulnerabilities/sqli/source/low.php(11): mysqli_query(Object(mysqli), 'SELECT first_na...')
5  #1 /var/www/html/vulnerabilities/sqli/index.php(34): require_once('/var/www/html/v...')
6  #2 {main}
7    thrown in <b>/var/www/html/vulnerabilities/sqli/source/low.php</b> on line <b>11</b><br />
8
```

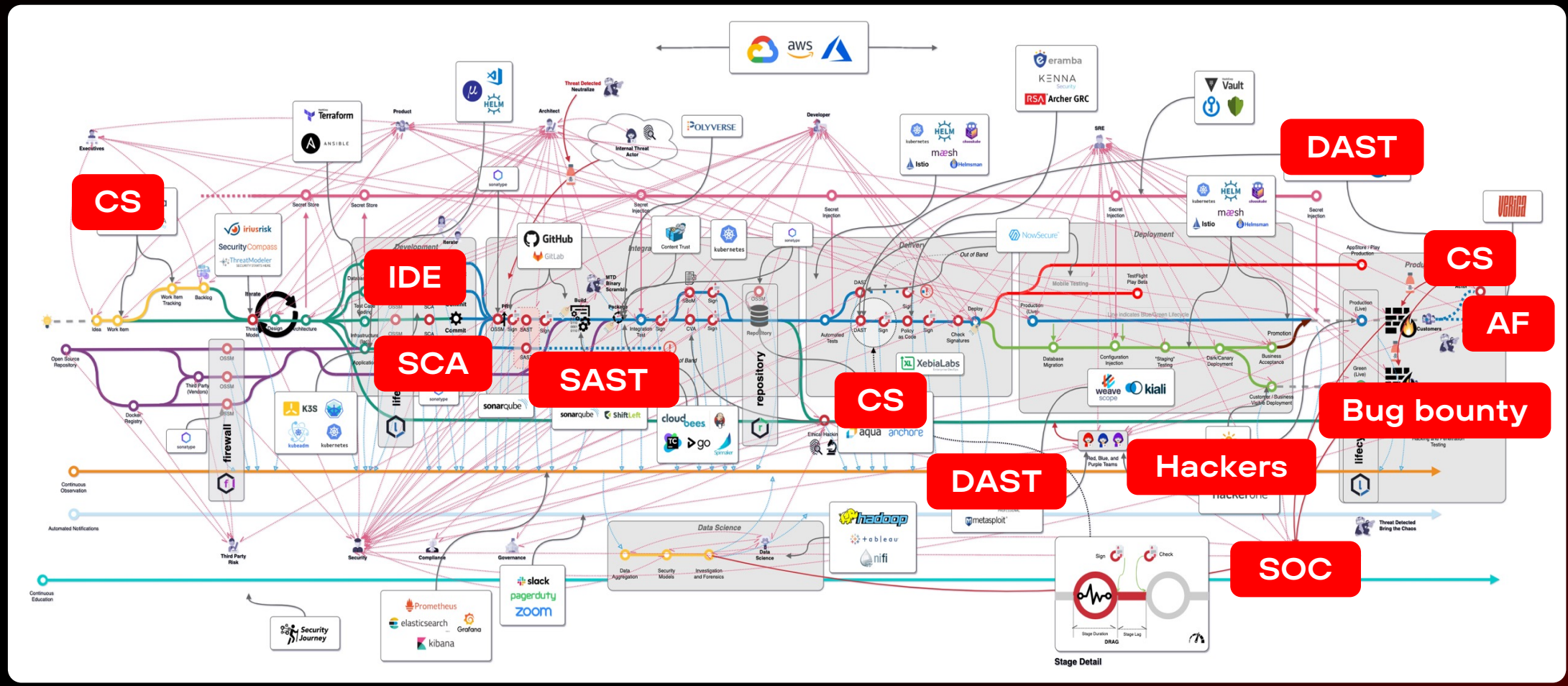
DAST в PT BlackBox

- Анализ веб-приложений и микросервисов без доступа к исходному коду
- Развитие технологий анализа с 2011 года
- Комбинация сигнатурного и эвристического методов анализа
- Акцент на качество результатов с минимумом ручных действий
- Тонкая настройка параметров сканирования
- Встраивание в конвейер сборки CI/CD



Так гораздо лучше

Комплексный подход к инструментарию безопасной разработки



t.me/positive_
technologies

IDE-плагин
для VS Code

t.me/POSIdev

habrahabr.ru/
company/pt

IDE-плагин
для JetBrains

Спасибо!