Из чего делаем SOC? Анализ качества кода ПО SOC для ИБ и DevOps

Прилепский Александр, BDM

attack + killer

С какими проблемами сталкиваемся?

Вопросы

- Непрерывные изменения в коде
- Высокие темпы выпуска релизов
- Человеческие ошибки
- Бэкдоры

Специфика разработки in-house

- Несистемные требования по безопасности приложений
  Упор на функционал и нагрузку, игнорирование стандартов
- Специфическая модель угроз
  Классические модели угроз не актуальны
- Службы ИБ в разработке не участвуют
  Привлекаются только к расследованиям инцидентов
- Используются закрытые платформы
  SAP, Oracle, MS Dynamics, 1C, Lotus

Специфика сторонних приложений

- **Нет контроля при разработке**
  Контроль возможен только при приемке программного обеспечения
- **Большая предварительная работа**
  Все требования можно и нужно сформулировать до начала разработки
- **Периодический контроль обновлений**
  При смене версий контроль требований по безопасности нужно проводить заново

attack + killer

С чем сталкиваемся

1 **Непреднамеренные ошибки:**

- некорректная обработка входных данных
- недостатки авторизации

3 **Типичные недостатки программы:**

- OWASP, OWASP TOP 10
- Common Weakness Enumeration
- Language specific best practices

4 **Недостатки, зависящие от специфики приложения:**

- особенности реализации алгоритмов и систем защиты
- «это не баг, а фича»

2 Преднамеренные ошибки

5 Недостатки, зависящие от предметной области приложения

Рособрнадзор

Сценарий атаки:

Хакер получил доступ к БД ведомства с помощью SQL-инъекции.

В форму для проверки подлинности дипломов были введены некорректные данные. Это привело к SQL-инъекции, благодаря чему хакер смог получить доступ к БД и изучить ее структуру, а после с помощью скрипта на Python скачать данные.

Последствия:

- Прямой доступ к данным по дипломам, ПД 14 млн россиян

Magento:
Платежные данные

Сценарий атаки:

С помощью скриптов хакеры похищают данные банковских карт пользователей online-магазинов на Magento. Скрипты ищут на сайтах формы для введения платежных данных, похищают данные и передают на сервер.

Для маскировки хакеры используют скрипты Google Analytics и Angular. Код внедрен в легитимный JavaScript-файл.

Последствия:

28.02.2019 - поддельный скрипт содержался на **39 сайтах**, под управлением не только **Magento**, но и **WordPress, Joomla и Bitrix.**

GitHub: infected code

Сценарий атаки:

Обнаружена сеть учетных записей, распространявших вредоносные версии официальных библиотек и приложений для Windows, Mac и Linux.

Приложения содержали код, предназначенный для сохранения присутствия на зараженных системах и последующей загрузки вредоносного ПО, добавлявшего жертву в ботнет.

Последствия:

- 89 учетных записей
- 79 репозиториев
- > 300 приложений с бэкдорами

Фармацевтика

Описание:

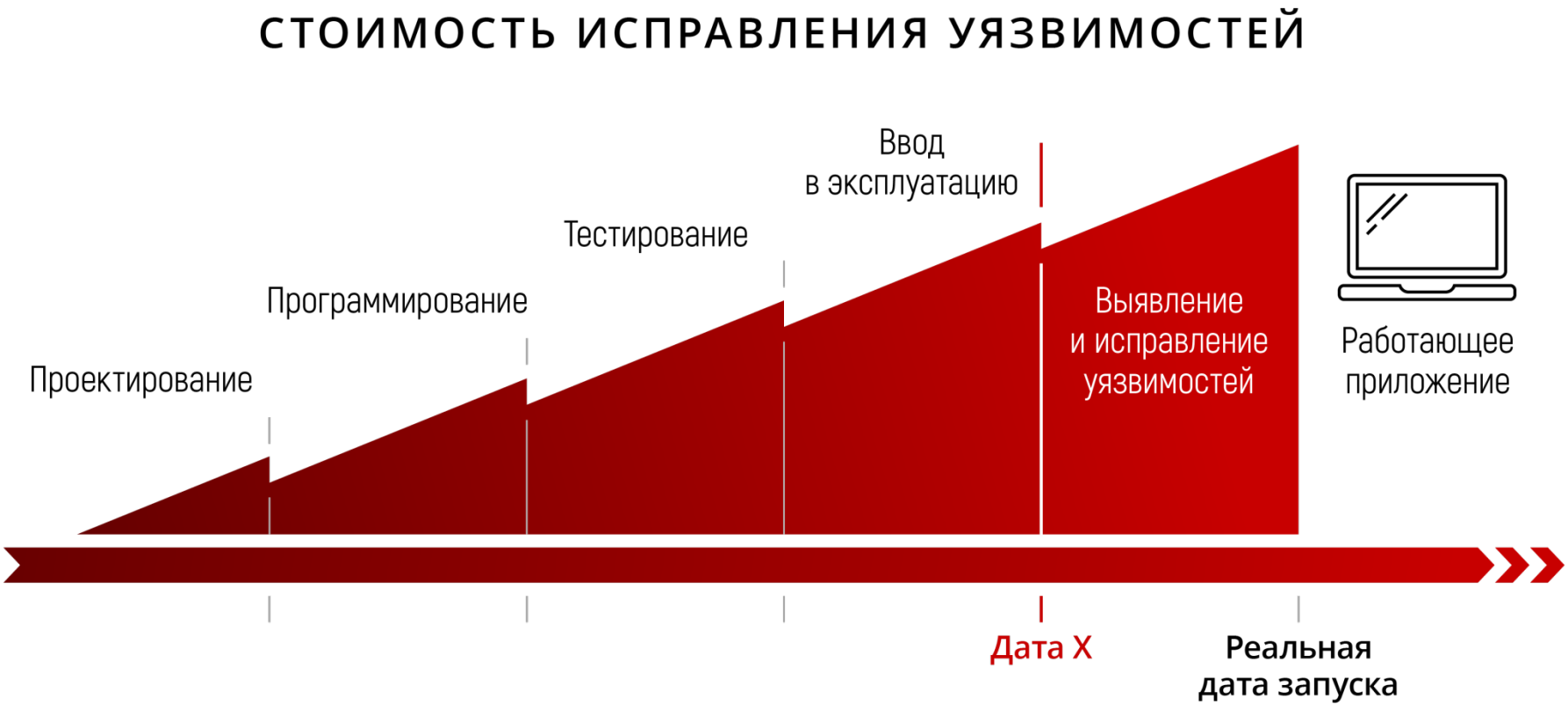
Аутсорсинговая компания в 2008 году подписала контракт с клиентом на разработку кода

Код предоставляется в обфусцированном виде, клиенту исходный код не передается, что находится “под капотом”, клиент не знает. Разработчики отказываются передавать код, ссылаясь на контракт

Последствия:

- Возможные уязвимости и закладки
- Клиент сидит “на игле”

Текущая ситуация



Жизненный цикл разработки

1. Сбор требований

2. Дизайн
   1. Уязвимая архитектура
   2. Устаревшие платформы

3. Разработка
   1. Непреднамеренные баги
   2. Backdoors
   3. Внешние средства разработки

4. Тестирование
   1. Tester – не хакер
   2. Это баг или фича?
   3. Как баг влияет на другие модули WebApp?

5. Анализ, новые требования
   1. Старые грабли

Жизненный цикл разработки: место SAST

Куда движемся сейчас

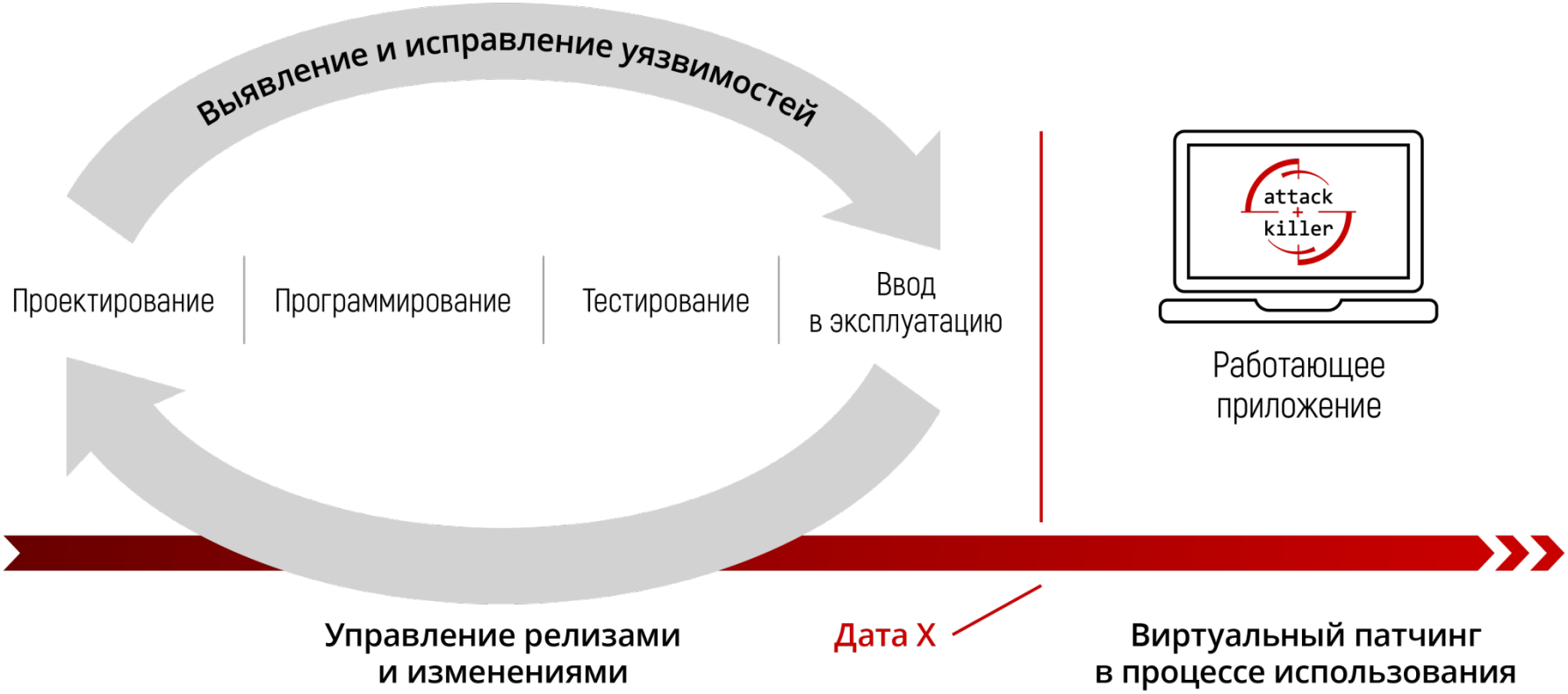
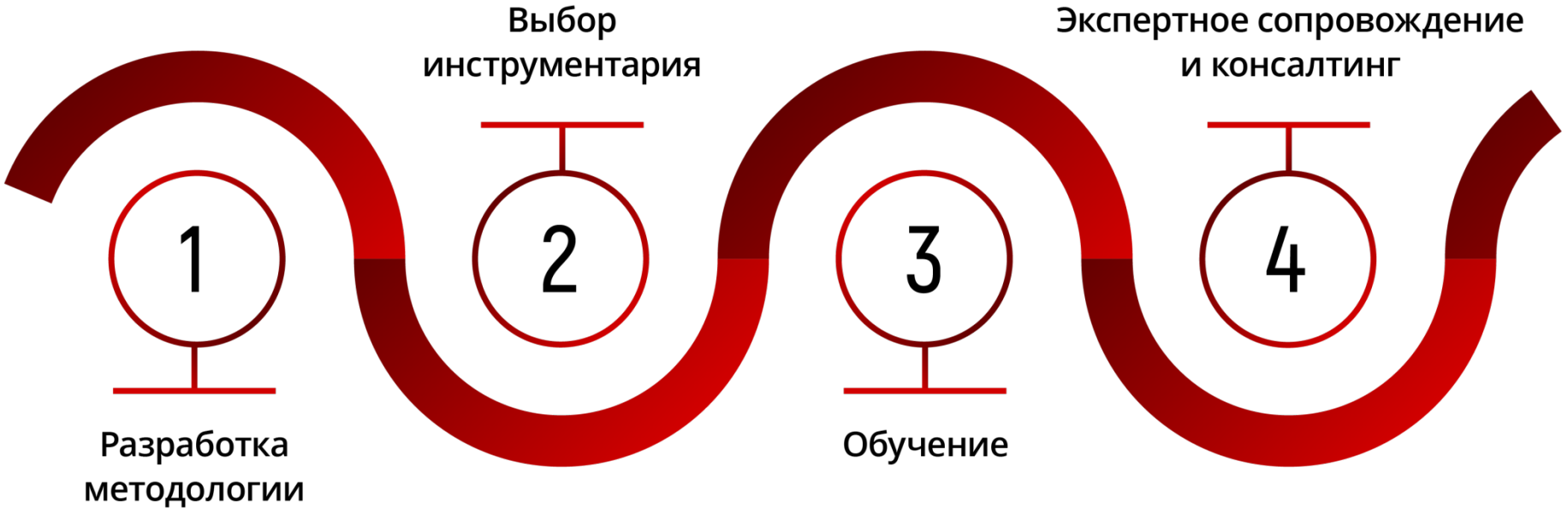


Схема работы SAST

С чего начать?

Автоматизированное и комплексное управление безопасностью веб-ресурса



Коротко об Attack Killer

Мы знаем требования бизнеса к ИБ

Превентивная защита

Защита web-ресурсов от внешних угроз

Комплексный подход

Мы знаем требования регуляторов к бизнесу

Основаны в 2015

Лидерство:

- Первый СНГ разработчик системы комплексной защиты веб-инфраструктуры

Проекты по всему миру:

- От гос. сектора, до энергетической отрасли

Синергия технологий:

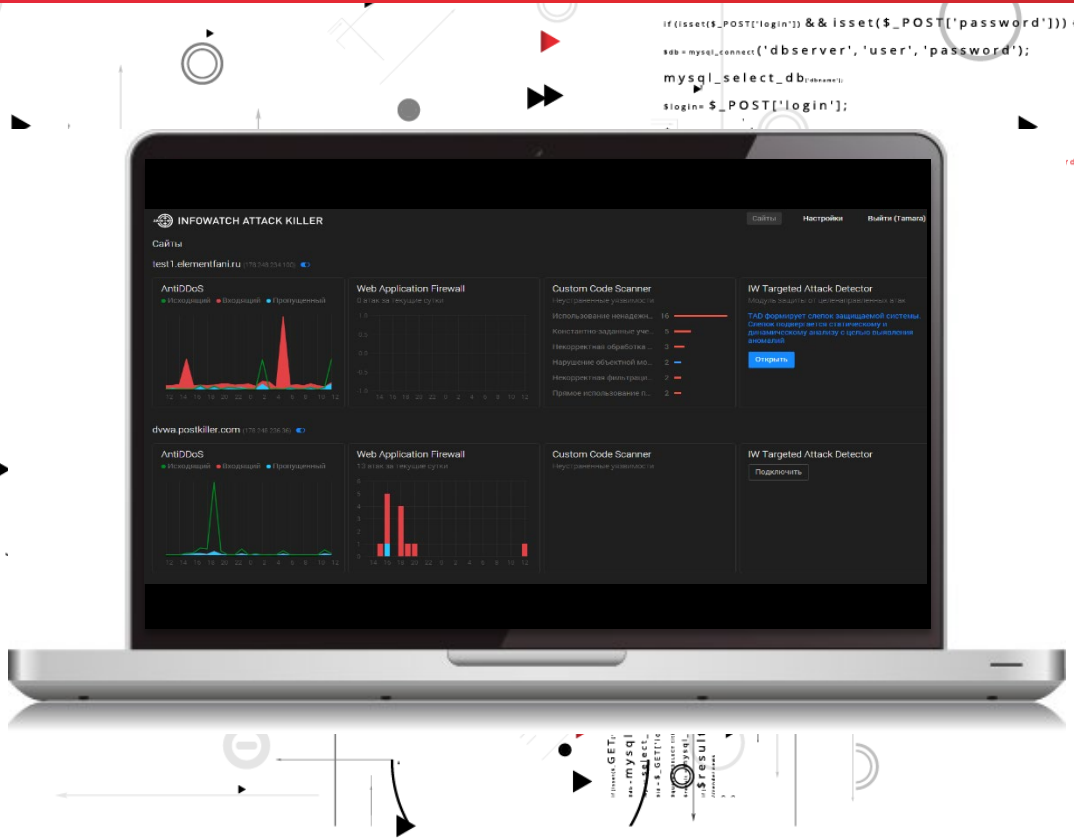
- От ведущих разработчиков ИБ

Текущая ситуация

- Простой и быстрый инструмент
- Обнаруживает уязвимости в исходном коде
- Поддерживает самые известные языки: 1C, Java, Java Script, PHP, Python, C#, ABAP, T-SQL and etc.
- API для создания собственных паттернов
- Интеграция со средами управления разработкой и баг-трекинговыми системами

Куда двигаемся дальше

- Необходимо разделить ролевую модель по направлениям
- Внутренние трекинговые системы в решении
- Возможность самим добавлять поиск уязвимостей с учетом предметной области



Спасибо за внимание!