Software Engineering Conference Russia

14-15 ноября, 2019. Санкт-Петербург

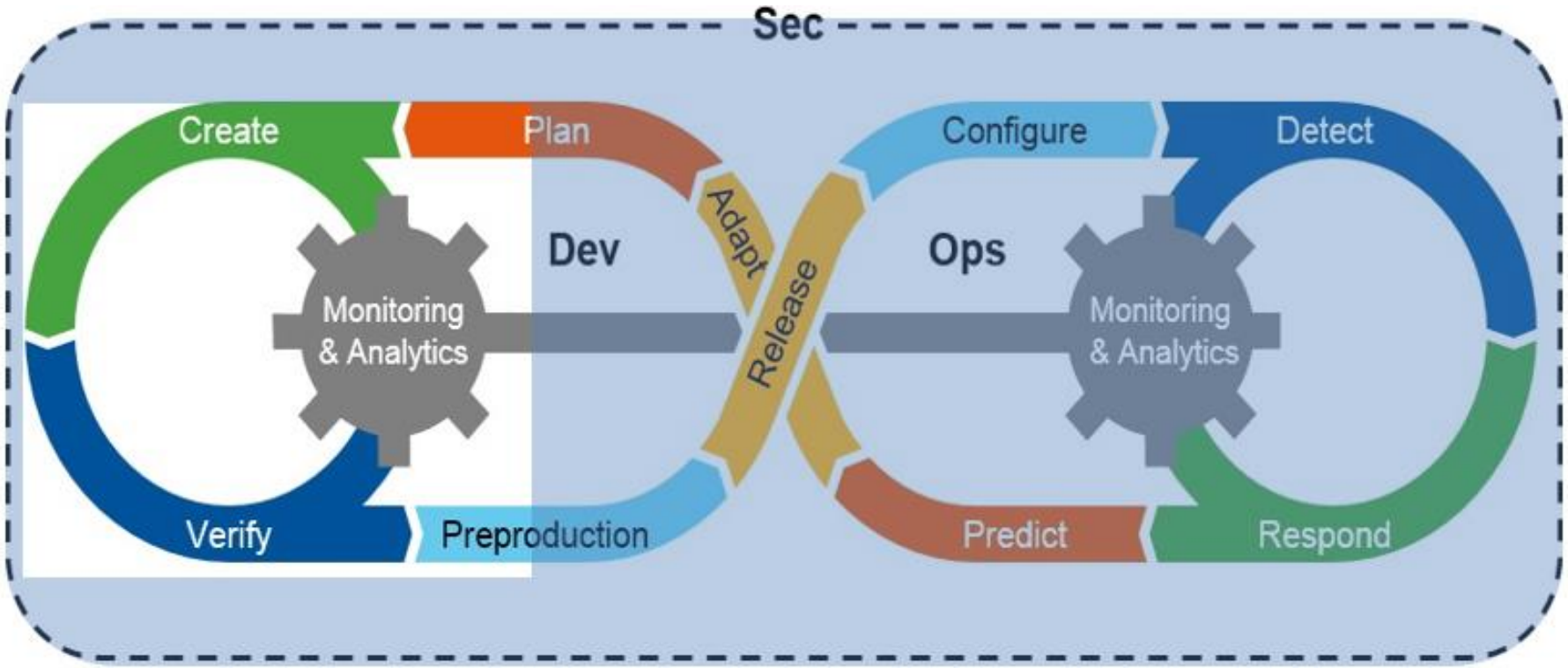
DevSecOps против восстания машин

Сергей Хренов

PVS-Studio

Зачем слушать этот доклад?

Чтобы больше зарабатывать денег!



DevOps = системный администратор + программист

QA = тестировщик + программист

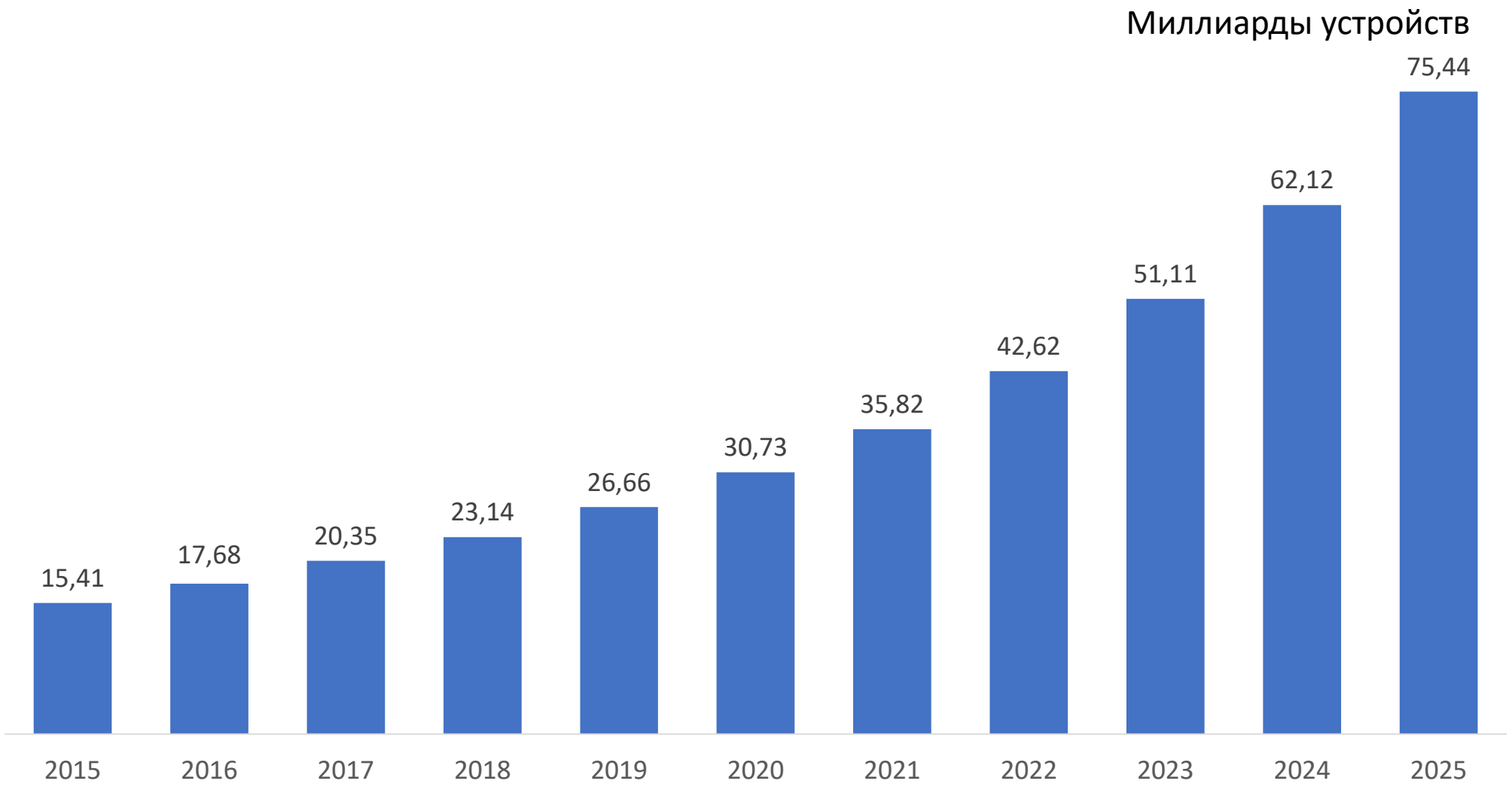
SecOps = security anything + программист

Размер ИМЕЕТ значение

Ядро Linux 1.0.0 : **177 000** строк кода

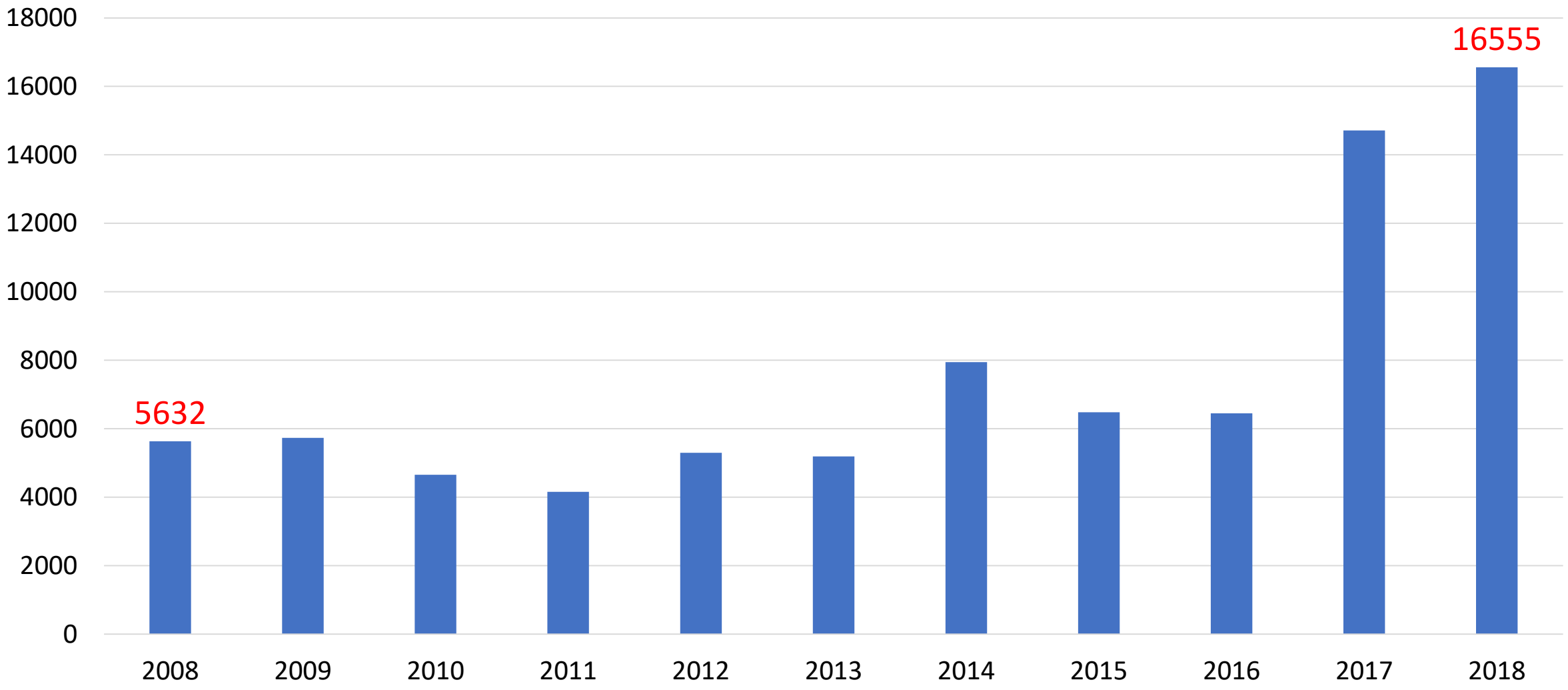
Ядро Linux 5.1-rc2 в **118** раз больше: **20 896 000** строк кода

Прогнозируемые темпы роста IoT



Источник: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

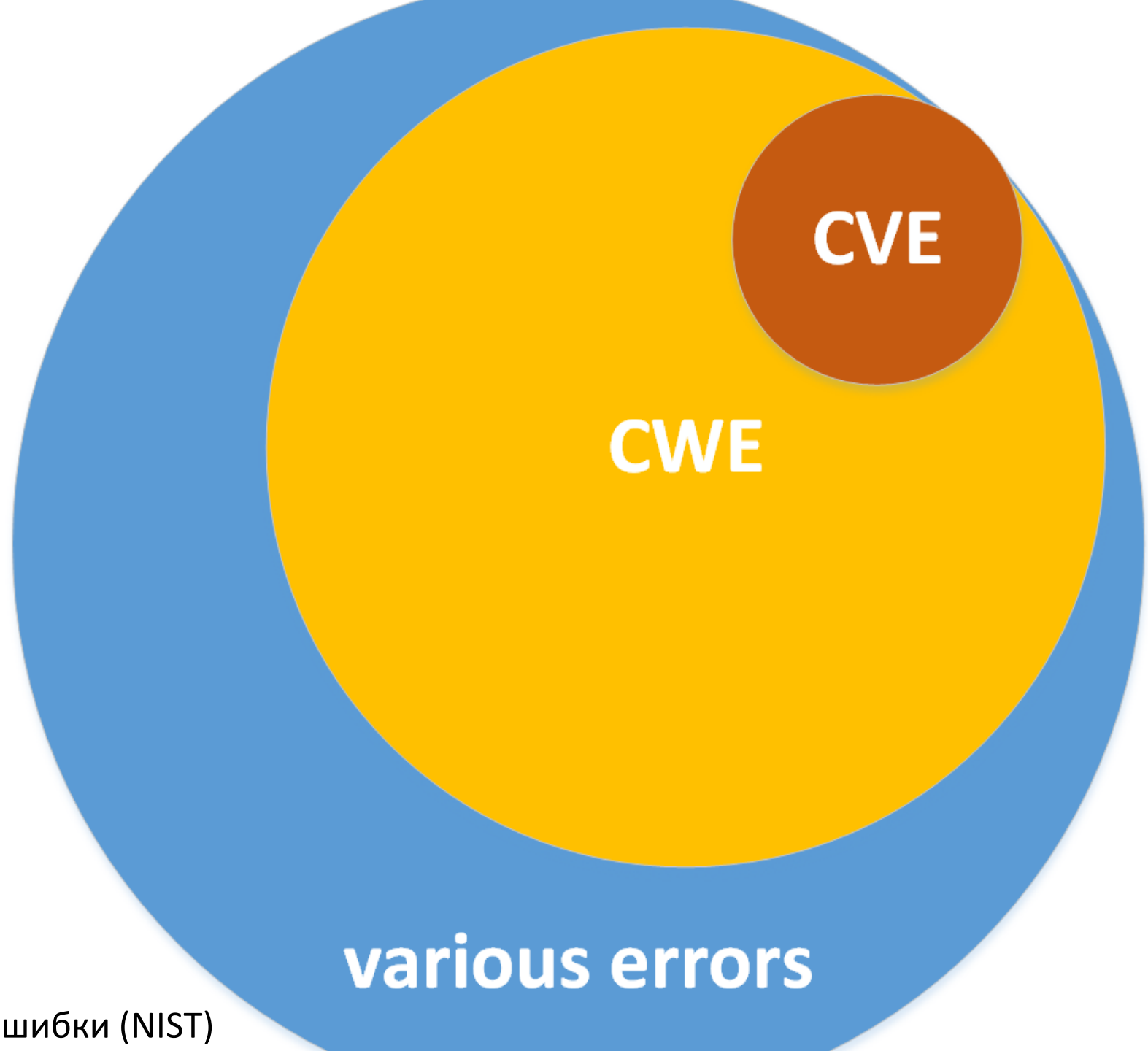
Число выявленных уязвимостей



Источник: https://www.cvedetails.com

CWE - Common Weakness Enumeration

CVE - Common Vulnerabilities and Exposures



Около 64% уязвимостей – программные ошибки (NIST)

“Хватит графиков! Покажите уже код”


```
static void SHA1Final(unsigned char digest[20],
                      SHA1_CTX *context)
{
  u32 i;
  unsigned char finalcount[8];
  ....
  memset(context->count, 0, 8);
  memset(finalcount, 0, 8);
}
```


```
static void SHA1Final(unsigned char digest[20],
                      SHA1_CTX *context)
{
  u32 i;
  unsigned char finalcount[8];
  ....
  memset(context->count, 0, 8);
  memset(finalcount, 0, 8);
}
```

V597 [CWE-14] The compiler could delete the 'memset' function call, which is used to flush 'finalcount' buffer. The memset_s() function should be used to erase the private data. wifi_generate_pin.c 185


```
void
isf_wsc_context_del (WSCContextISF *wsc_ctx)
{
   ....
   WSCContextISF* old_focused = _focused_ic;
   _focused_ic = context_scim;
   _focused_ic = old_focused;
   ....
}
```


```
void
isf_wsc_context_del (WSCContextISF *wsc_ctx)
{
   ....
   WSCContextISF* old_focused = _focused_ic;
   _focused_ic = context_scim;
   _focused_ic = old_focused;
   ....
}
```

V519 [CWE-563] The '_focused_ic' variable is assigned values twice successively. Perhaps this is a mistake. Check lines: 1260, 1261. wayland_panel_agent_module.cpp 1261

Хочется больше примеров из Tizen?

Проверка Tizen OS: часть 1.
27000 ошибок в операционной системе Tizen

Проверка Tizen OS: часть 2.
Поговорим о микрооптимизациях на примере кода Tizen

Проверка Tizen OS: часть 3.
Продолжаем изучать Tizen: C# компоненты оказались высокого качества

“Ошибки, уязвимости…

С этим можно как-то бороться?”

Один из вариантов борьбы с уязвимостями в IoT

Два подхода к поиску уязвимостей

- Можно (и нужно) искать потенциальные уязвимости, которые могут привести к реальным проблемам при написании нового кода

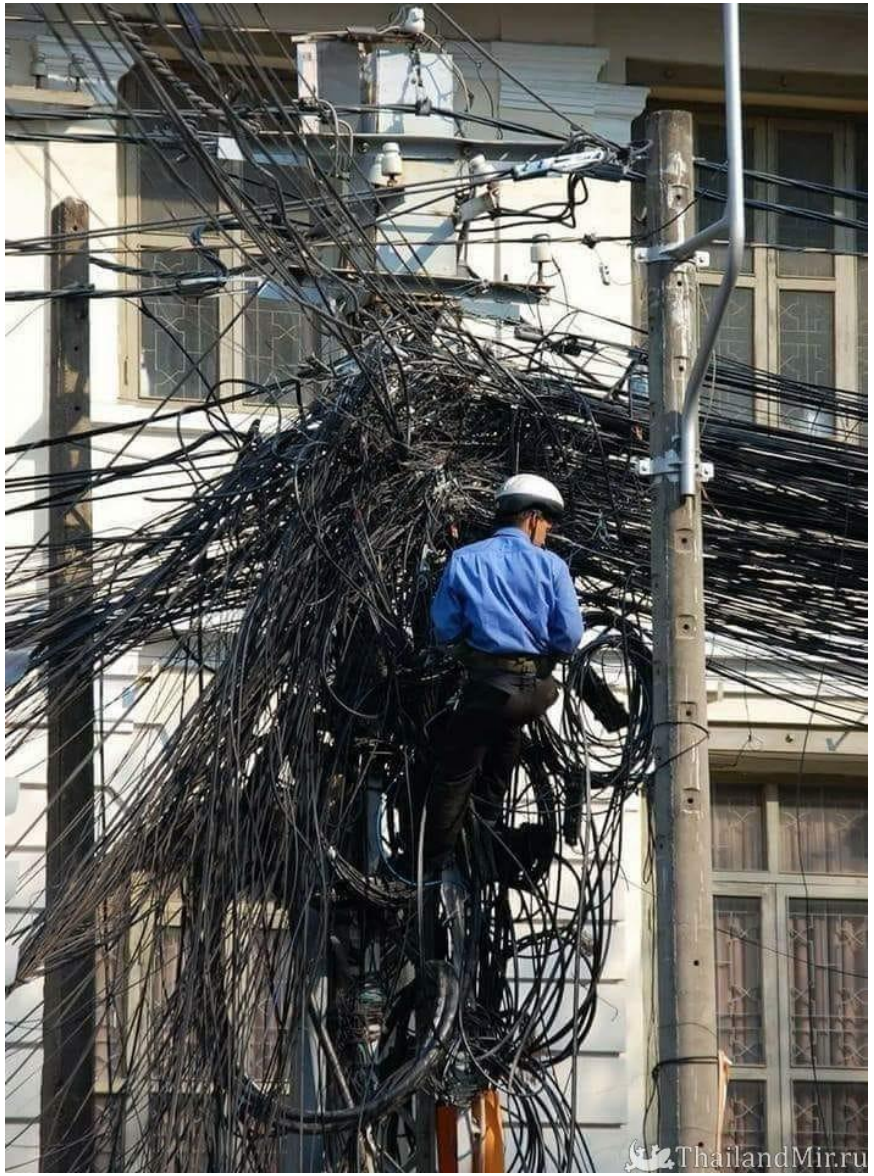
- Можно искать по базе существующих известных уязвимостей в своем имеющемся коде (с помощью автоматизированных средств)

Как обнаружить уязвимости

- Code review
- Юнит-тесты
- Динамический анализ (DAST)
- Статический анализ (SAST)
- Стандарты кодирования

Code Review

- Плюсы:
  - Обмен опытом
  - Правки “по горячим следам”
- Минусы:
  - Дорого и долго
  - Быстро устаешь от просмотра кода



Динамический анализ кода

- Отладчики
- Профилировщики
- Санитайзеры (AddressSanitizer, ThreadSanitizer, ...)

- Нашли ошибку? Можно её сразу исправлять!

(нет ложных срабатываний)

Но...

- Отладить и протестировать под embedded не всегда просто
- Санитайзеры и профилировщики медленные
- Часто нужны специальные входные данные

Static Application Security Testing (SAST)

- Статический анализ, нацеленный на поиск и предотвращение уязвимостей
- Уязвимости - те же самые обыкновенные ошибки (по данным NIST, более 60%)
- Инструменты SAST помогают предотвращать уязвимости и обеспечивают поддержку стандартов безопасной разработки: CWE, MISRA, SEI CERT и т.п.

Static Application Security Testing (SAST)

- Преимущества:
  - Раннее обнаружение проблем
  - Покрытие всего кода
  - Хорош в поиске опечаток и ошибок типа copy-paste

- Недостатки:
  - Ложные срабатывания
  - Неизвестна точная критичность ошибки
  - Слабое диагностирование утечек памяти и параллельных ошибок

Стандарты кодирования

- Common Weakness Enumeration
- SEI CERT Coding Standards
- MISRA C, MISRA C++

CWE

- **CWE™** is a community-developed list of common software security weaknesses
- https://cwe.mitre.org
- Список из более чем **800** потенциальных уязвимостей, которые могут стать реальными. А могут и не стать

CWE: примеры потенциальных уязвимостей

- CWE-570\571: Expression is Always False\True
- CWE-467: Use of sizeof() on a Pointer Type
- CWE-476: NULL Pointer Dereference
- CWE-14: Compiler Removal of Code to Clear Buffers
- CWE-369: Divide By Zero
- CWE-20: Improper Input Validation

CWE-570: Expression is Always False


```
bool operator <(const TSegment& other) const {
  if (m_start < other.m_start)
    return true;
  if (m_start == other.m_start)
    return m_len < m_len;
  return false;
}
```

V501 There are identical sub-expressions to the left and to the right of the '<' operator: m_len < m_len segmentor.h 65

CWE-697: Insufficient Comparison


```
static void __page_focus_changed_cb(void *data)
{
  int i = 0;
  int *focus_unit = (int *)data;
  if (focus_unit == NULL || focus_unit < 0) {
    _E("focus page is wrong");
    return;
  }
  ....
}
```

V503 This is a nonsensical comparison: pointer < 0. apps_view_circle_indicator.c 193

CVE

- **CVE®** is a list of publicly known cybersecurity vulnerabilities
- https://cve.mitre.org/
- Список из более чем **114 000** реальных уязвимостей, найденных в приложениях

MISRA C/C++

- Motor Industry Software Reliability Association
- Закрытый стандарт кодирования. Доступен только за деньги
- Уменьшает вероятность допущения ошибки для ответственных встраиваемых систем
- Анализаторы, проверяющие соответствие MISRA, не ищут ошибки
- MISRA C 2012 содержит **143** правила
- MISRA C++ 2008 содержит **228** правил

Примеры рекомендаций из MISRA

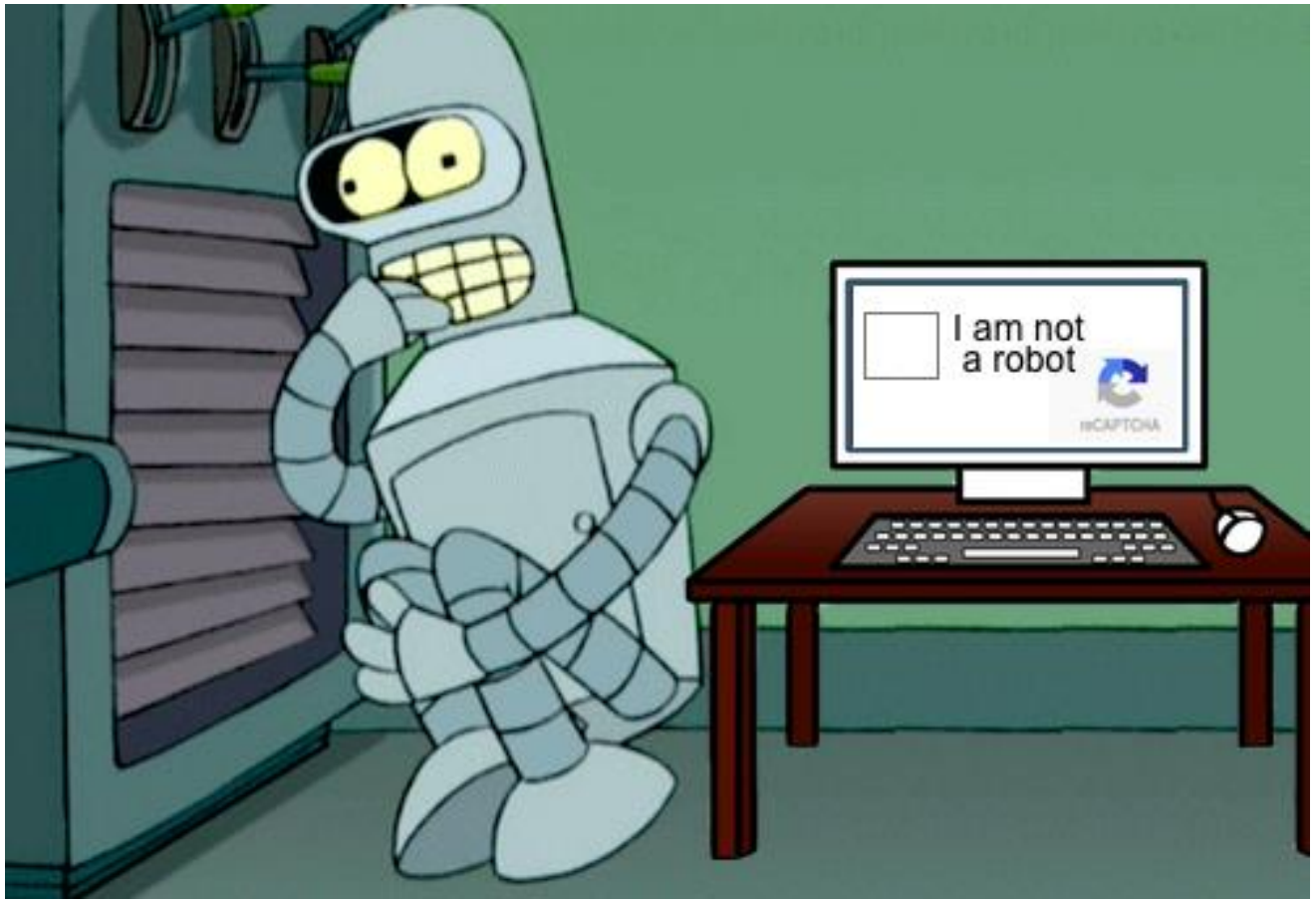
- Каждый 'switch' должен содержать 'default'
- Возвращаемое значение non-void функций должно быть использовано
- Функция должна иметь одну точку выхода
- Указатели должны иметь не более двух уровней вложенности
- Тела циклов и условных выражений должны быть заключены в {}
- Каждый 'if ... else if' должен иметь завершающий 'else'
- Не использовать unions

SEI CERT

- Стандарт кодирования
- Разрабатывается координационным центром CERT (CERT Coordination Center, CERT/CC)
- Предназначен для языков C, C++, Java, Perl
- Очень похож на CWE

Заблуждения о SAST

- Дорого
- Не для новичков
- Сложно внедрять на большом проекте
- Панацея от всех бед



Внедряем и используем SAST правильно

- Выбор подходящего анализатора, настройка под особенности проекта
- Создание suppress-базы, инкрементальный анализ
- Регулярное использование на CI-серверах и рабочих местах разработчиков

Снижаем потери

Уязвимость

Прямые и косвенные потери: эксплуатация злоумышленниками, bug bounty, репутация

Исправление

Выпуск обновления

Уязвимость

Обнаружение с помощью SAST, исправление

Выводы

- Не допускайте попадания проблем с безопасностью в конечный продукт
- Инструменты SAST – лишь один из способов поиска уязвимостей
- Используйте все доступные методологии контроля качества кода
- Не экономьте на разработке в ущерб безопасности, тогда сможете избежать трансформации затрат в расплату

Контакты

Сергей Хренов

C# разработчик, PVS-Studio

khrenov@viva64.com

www.viva64.com