Авторизация. Принципы разработки защищенной архитектуры

Самосадный Кирилл
https://t.me/kirsamosad
samosad@seclab.cs.msu.su

МГУ 2018

Авторизация

Общие принципы – как плохо

- Авторизация spaghetti-style а-ля Италиано
  - в SQL-запросах
  - собственные неунифицированные проверки в action’ах
  - == когда разработчик функции должен явно подумать и написать какой-то код
- Упаковка в JS-файлы кода вызова всех API-ендпойнтов
- Следствия
  - OWASP A5-Broken Access Control

DEMO: IDOR

Общие принципы – как хорошо

- Архитектурный подход
  - выбрать ролевую модель: ABAC (attribute-based access control)
  - внимание: ABAC более гибок чем классический RBAC
- Делать идентификаторы объектов непредсказуемыми
  - https://example.com/share/11f0ced2e24a/d7c5a9c86b
  - https://example.com/share/getSharedContent.do?sid=02adf8cb44577d631ce2d859

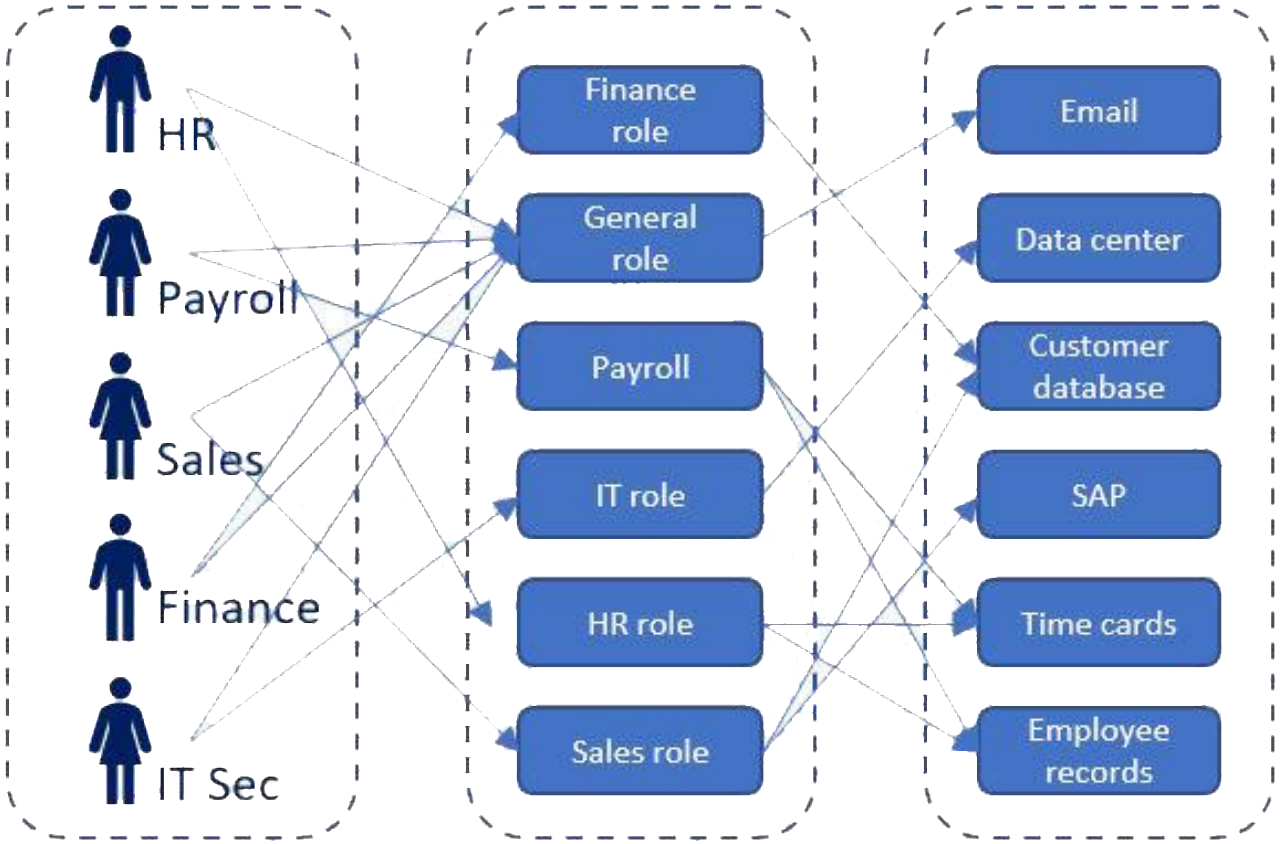
  vs

  - https://example.com/share/getSharedContent.do?sid=1425
  - https://example.com/share/getFile.do?owner=admin&file=/docs/annual/2016/Отчёт.docx

DEMO: Mass Assignment

Модель прав

Role Based Access Control



Открытый вопрос

- Бизнес-правило:
  - Пользователь может получить доступ только к своему счету
- Как выразить в RBAC?

Закрытый вопрос

- Бизнес-правило:
  - Пользователь может читать только те посты друзей, к которым для него друзья явно открыли доступ
- Как выразить в RBAC?

Attribute Based Access Control

- Давайте напишем бизнес-правило в виде кода
- Над контекстом:
  - Кто выполняет действие
  - Какое действие
  - Над каким объектом
  - В каком контексте
- RBAC – ABAC над одним параметром (роль)

Пример выражения правила

Бизнес-правило:

- *Старший менеджер из отдела закупок может подтверждать заказы на покупку, только если он не создатель заказа, заказ находится в его филиале, стоимость заказа больше 100 000 руб. и не превышается лимит утверждений заказов за день*

Состоит из условий:

- *Действие.Название = «Подтверждение»*
- *Субъект.Должность = «Старший менеджер»*
- *Субъект.Отдел = «Отдел закупок»*
- *Субъект.ЛимитУтверждений < Субъект.УтвержденоЗаДень*
- *Субъект != Ресурс.Создатель*
- *Субъект.Филиал = Ресурс.Филиал*
- *Ресурс.Тип = «Заказ на покупку»*
- *Ресурс.СтоимостьЗаказа > «100 000»*

ABAC пример

```php
<?php

use PhpAbac\Abac;

$abac = new Abac([
    'policy_rule_configuration.yml'
]);
$check = $abac->enforce('edit-group', $user, $group, [
    'dynamic-attributes' => [
        'group-owner' => $user->getId()
    ]
]);
```

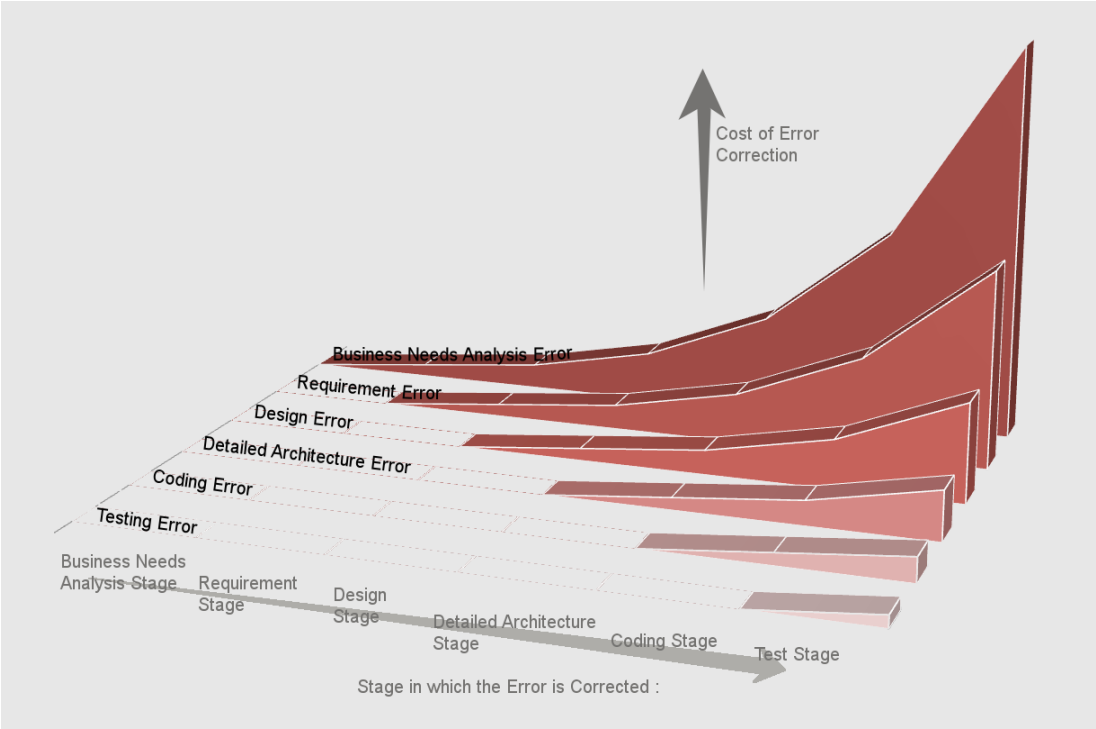
Безопасная архитектура

Основные принципы

- Хорошая архитектура === безопасная архитектура
- Saltzer and Schroeder (1975)
  - Economy of Mechanism
  - Fail-safe defaults
  - Complete mediation
  - Open design
  - Separation of privilege
  - Least privilege
  - Least common mechanism
  - Psychological acceptability

Экономичность механизмов

- Архитектура (механизмов безопасности) должна быть максимально простой и дешевой (но эффективной)
  - KISS – чем проще, тем тяжелее ошибиться
  - Brian W. Kernighan
    - Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.
- Исправить ошибки при проектировании архитектуры до написания кода можно быстро и дешево
- Для успешного аудита безопасности нужна простая архитектура



Защищенность по умолчанию

- Разрешать по умолчанию
  - Давать доступ, если явно не запрещено
  - В случае ошибки – разрешать
  - Удобнее для пользователя
  - Неправильная психологическая модель
- Запрещать по умолчанию
  - Давать доступ, если явно разрешено
  - В случае ошибки – запрещать
  - Увеличивает защищенность
  - Правильная модель с точки зрения безопасности

Безопасные настройки

- Конфигурация
  - Защищенность начальной конфигурации
  - Легко (пере)настроить
- Защищенность конфигурации
  - Нет стандартных (и простых) паролей
  - Нет стандартных (и тестовых) пользователей
  - Read only файлы, владелец – root
- Сообщения об ошибках
  - Для пользователя – в обобщенном виде
  - Подробности – в логах

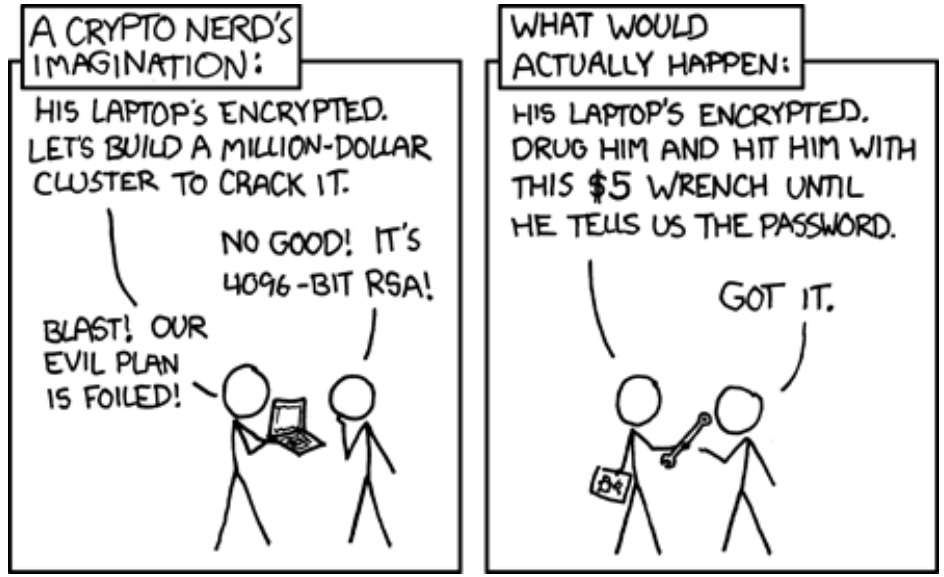
Oops! Something went wrong.
(but we still love you)

Всеобъемлющее посредничество

- Контроль доступа всегда (лучше лишний раз проверить)
  - Проверять каждое обращение к любому объекту
  - Аутентифицировать источник действия
  - См. ABAC
- Механизмы безопасности
  - Минималистичные
  - Централизованные
  - Непреодолимые
- Валидация всех входных данных

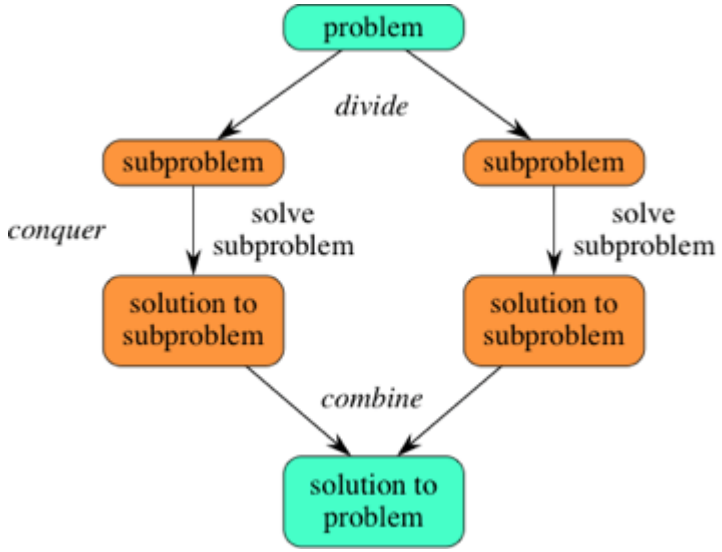
Открытая архитектура

- Архитектура не должна быть секретной
  - Механизмы безопасности не должны быть секретом
  - Периодические рецензии
  - Архитектуру трудно держать в секрете
  - Злоумышленник может понять, какие механизмы используются
- Защищенность основывается на секретности небольших токенов
  - Ключи
  - Пароли
- Данному принципу не нужно следовать напрямую
  - Держим механизмы в секрете, но не основываем на этом нашу защиту



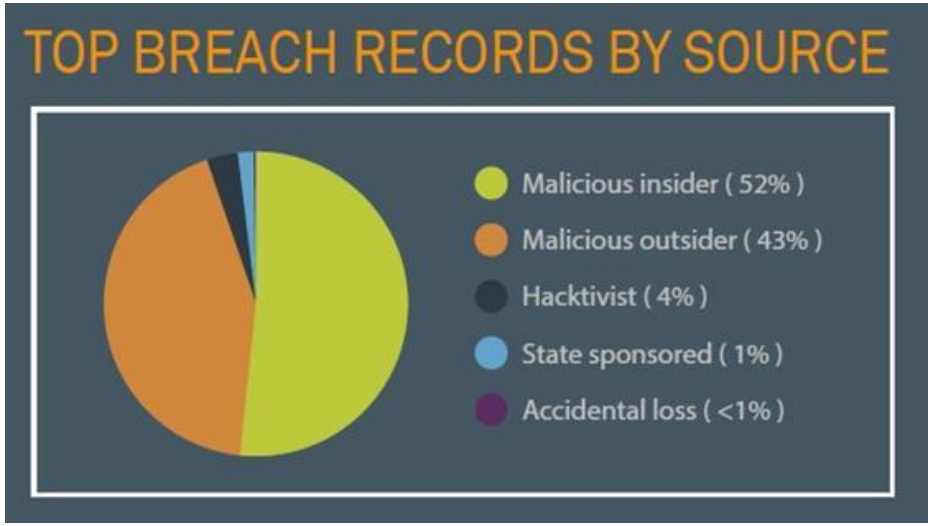
Разделение привилегий

- Доступ зависит от двух и более выполненных условий
  - Запуск МБР, открытие ячейки в банке требует двух человек
  - Условия не зависят друг от друга
- Обособление (уменьшение поверхности атаки)
  - Разбиение системы на независимые части (микросервисы)
  - Минимальные привилегии у каждой части
  - Ни в коем случае не использовать модель 'все или ничего' (веб-сервер из-под root)
- Песочницы (примеры)
  - Механизмы языка программирования (JavaScript в браузере)
  - На уровне операционной системы, системных вызовов…



Минимальные привилегии

- Необходимо выдавать те и только те привилегии, которые необходимы и достаточны для функционирования данной части системы
- Минимальные необходимые для выполнения задачи права
  - Минимизируют возможный ущерб
  - Минимизируют привилегированное взаимодействие
- Минимизация времени доступности привилегии
- Минимизация количества модулей, обладающими привилегиями – песочница наоборот

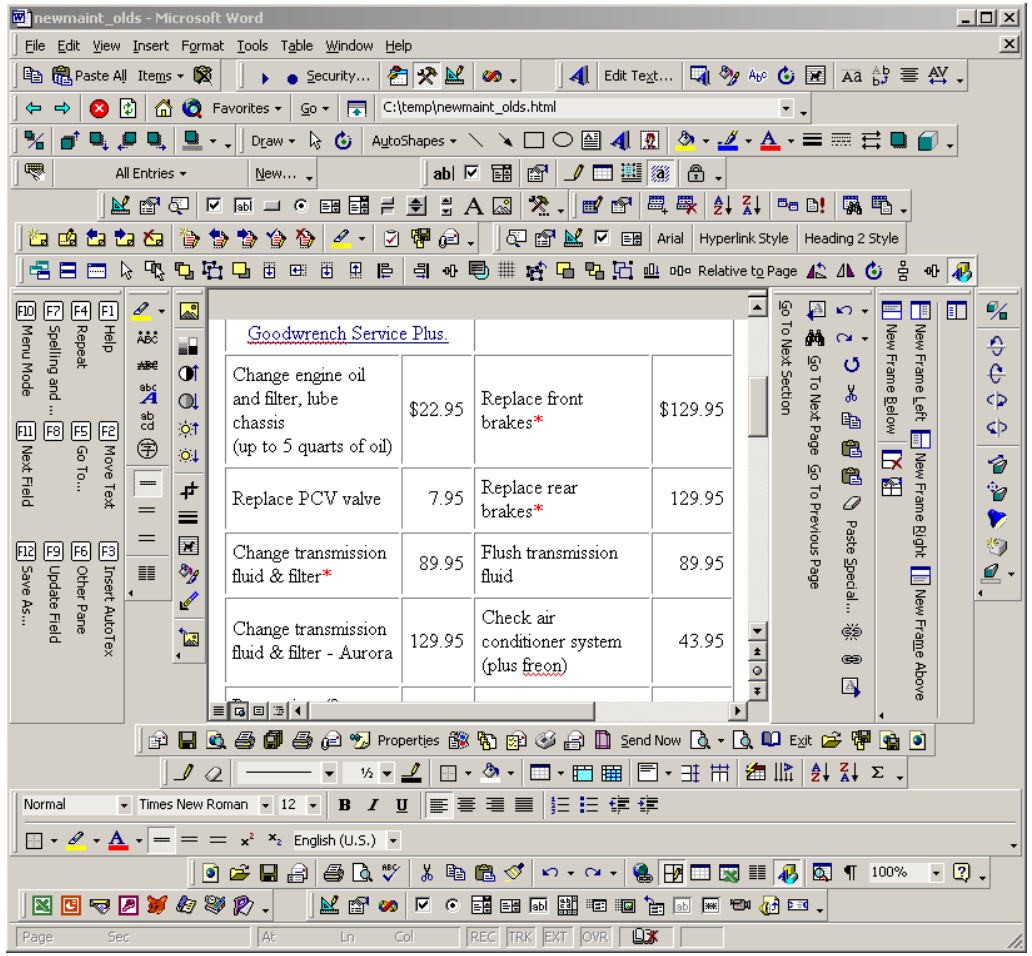


Наименее общие механизмы

- Минимизация доступной функциональности механизма
  - Не путать с централизованной валидацией
  - Уменьшает типы доступных взаимодействий с функциональностью
  - eval(s) vs ast.literal_eval(s) vs int(s)
- Пример:
  - Для чтения локальных файлов можно использовать механизмы, поддерживающие открытие сетевых файлов
    - PHP - fopen
  - HTTP 302
    Location: file:///etc/passwd

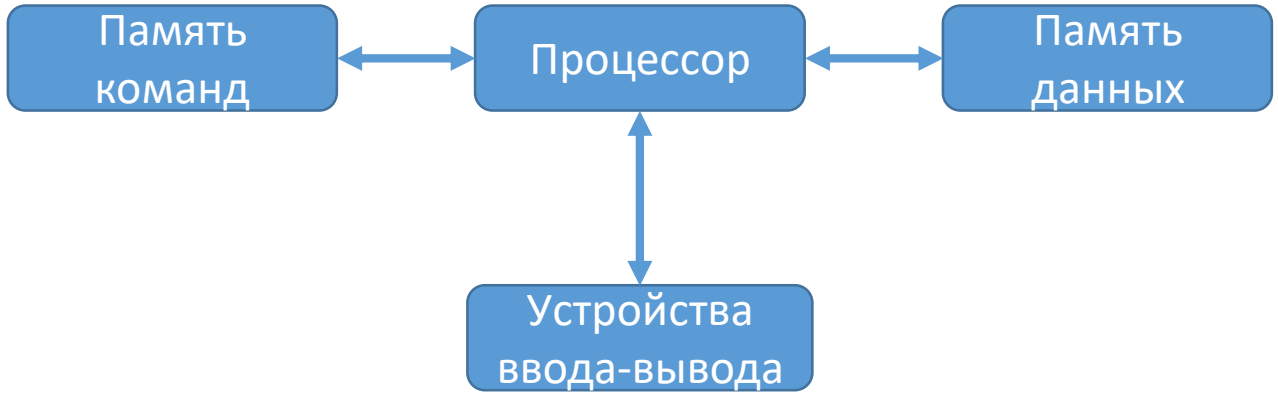
Приятие пользователем

- Понятные интерфейсы для пользователя
  - Легко постоянно использовать
  - Легко правильно использовать
  - Выполняют ту функцию, которую ожидает пользователь
- Аутентификация:
  - Пароль и парольные политики
  - Инфраструктура открытых ключей
    - Нужно поддерживать



Бонус: золотой принцип

- Нужно разделять команды и данные
  - Большое количество уязвимостей является следствием нарушения этого принципа
  - Гарвардская архитектура
- Проблемы:
  - Исполняемый код в файлах данных
    - JavaScript в HTML
    - Макросы в Word
  - Мобильные платформы
    - Код выполняется локально в недоверенном окружении



Модифицирование существующей функциональности

- Как добавить защищенность в существующее приложение?
- Обертки:
  - Перемещаем существующие входные точки в новое место
  - Заменяем их на код, выполняющийся до и после существующей функциональности
    - Валидация входных данных
    - Логирование
    - Вызов исходной функциональности
    - Санитизация результата

Литература

- RBAC vs ABAC
  - https://habr.com/company/custis/blog/248649/
- PHP ABAC
  - https://packagist.org/packages/kilix/php-abac
- Список реализаций различных моделей для разных языков программирования
  - https://github.com/casbin/awesome-auth
- Jerome H. Saltzer, Michael D. Schroeder. The Protection of Information in Computer Systems