Тема: Анализ уязвимостей Регулярные аудиты безопасности Применение лучших практик защиты информации

Группа: 22П-2
Студент: Калмацкий Арсен Николаевич

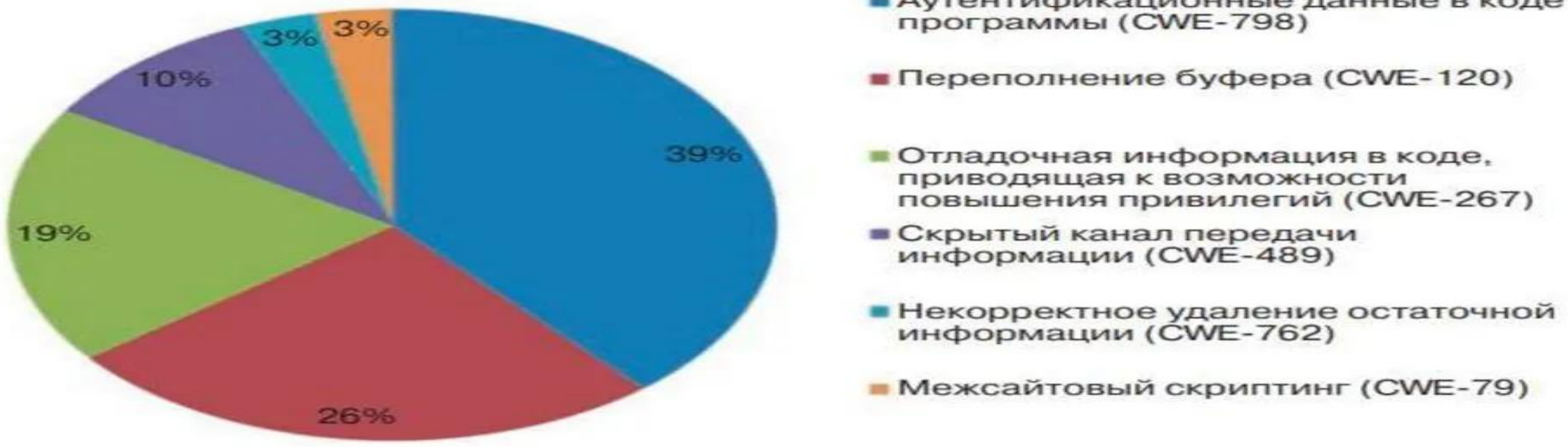
ВВЕДЕНИЕ

Что такое уязвимость?

- Недостаток в коде - может быть использован злоумышленником
- Слабое место в системе - через которую может пройти враг.
- Точка входа для атак - слабое место в системе безопасности, которое требует немедленного внимания.

СТАТИСТИКА ПО ТИПАМ УЯЗВИМОСТЕЙ

- Распределение уязвимостей (CWE)



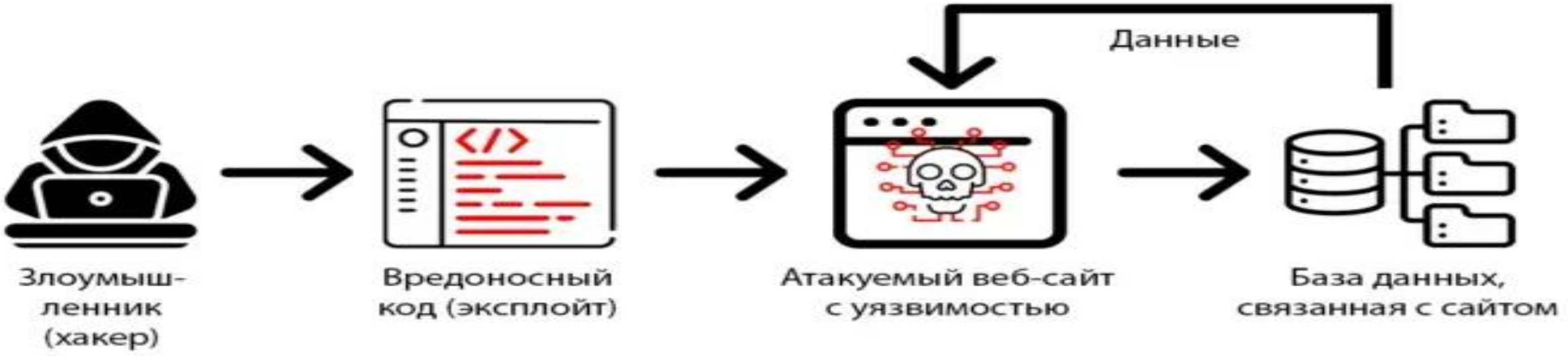
ТОР-3 УЯЗВИМОСТИ

- CWE-798 (39%) - Жесткие учетные данные в коде. Когда разработчик пишет пароль, API-ключ или токен прямо в исходном коде. Это как оставить ключ от замка в открытом месте.
- CWE-120 (26%) - Переполнение буфера. Это классическая уязвимость, когда программа не проверяет размер данных и они переполняют выделенную память, позволяя злоумышленнику выполнить произвольный код.
- CWE-267 (19%) - Повышение привилегий. Когда обычный пользователь получает права администратора благодаря ошибке в коде.

SQL-ИНЪЕКЦИЯ

- Механизм атаки

Хакер находит поле ввода в веб-приложении
Вводит вредоносный SQL-код вместо обычных данных
Приложение выполняет этот код в базе данных
Получается доступ к конфиденциальным данным
Например, вместо имени пользователя хакер вводит ' OR 1=1 -- и база данных выдает всех пользователей.



ПРИМЕРЫ КОДА - SQL-ИНЪЕКЦИИ И ЗАЩИТА

❌ УЯЗВИМЫЙ КОД	✅ ЗАЩИЩЕННЫЙ КОД
«PHP» $user = $_GET['username']; $query = "SELECT * FROM users WHERE username = '$user'"; «Python» username = request.args.get('username') query = f"SELECT * FROM users WHERE username = '{username}'" cursor.execute(query) Java String user = request.getParameter("username"); String query = "SELECT * FROM users WHERE username = '" + user + "'"; Statement stmt = conn.createStatement(); ResultSet rs = stmt.executeQuery(query);	«PHP» $user = $_GET['username']; $query = $conn->prepare("SELECT * FROM users WHERE username = ?"); $query->execute([$user]); «Python» username = request.args.get('username') query = "SELECT * FROM users WHERE username = %s" cursor.execute(query, (username,)) Java String user = request.getParameter("username"); String query = "SELECT * FROM users WHERE username = ?"; PreparedStatement pstmt = conn.prepareStatement(query); pstmt.setString(1, user); ResultSet rs = pstmt.executeQuery();

СЛАЙД 7: ЗАЩИТА ОТ SQL-ИНЪЕКЦИЙ

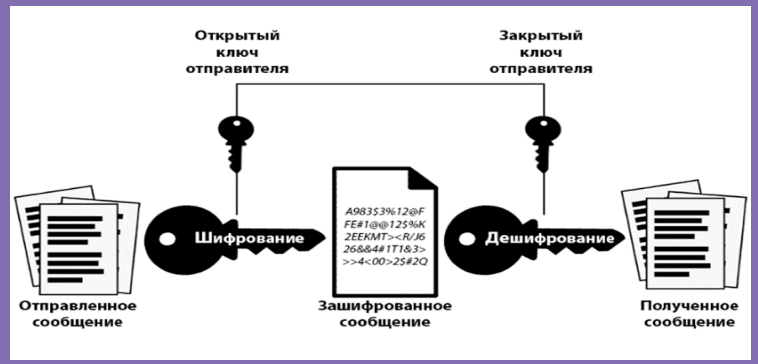
Методы защиты:

- Параметризованные запросы - это самый эффективный способ. Вместо вставки пользовательских данных прямо в SQL, мы используем заполнители, и база данных понимает, что это данные, а не команды. Валидация входных данных
- Экранирование спецсимволов - преобразуем опасные символы так, чтобы они не выполнялись как код.
- ORM-фреймворки - используем готовые библиотеки, которые автоматически защищают от инъекций.

БЕЗОПАСНАЯ АУТЕНТИФИКАЦИЯ

Лучшие практики:

1. Хеширование паролей (SHA-256)
2. Криптографическая соль - это случайные данные, которые добавляются к паролю перед хешированием.
3. Требования к сложности - пароль должен быть минимум 12 символов с заглавными, строчными буквами, цифрами, спецсимволами и подтверждение пароля.
4. Защита от брутфорса - блокировка аккаунта после 5 неудачных попыток входа или неверном вводе капчи заблокировать дальнейшие запросы.



РЕГУЛЯРНЫЕ АУДИТЫ

План аудита безопасности:

1. Статический анализ кода - специальные инструменты читают исходный код и ищут потенциальные проблемы, не запуская его.
2. Динамическое тестирование - запускаем приложение и пытаемся его взломать, вводя опасные данные.
3. Пентесты - профессиональные хакеры с разрешением пытаются взломать систему, чтобы найти реальные уязвимости.
4. Логирование и мониторинг - отслеживаем все подозрительные действия в системе в реальном времени.

СОВРЕМЕННЫЕ АУДИТЫ БЕЗОПАСНОСТИ

Используемые сейчас подходы и стандарты:

SAST (Static Application Security Testing)

- Анализ исходного кода без запуска приложения;
- Обнаружение уязвимостей на ранних стадиях разработки.

DAST (Dynamic Application Security Testing)

- Тестирование запущенного приложения как «черный ящик»;
- Имитация атак реального злоумышленника.

IAST (Interactive Application Security Testing)

- Комбинация SAST и DAST;
- Агент встроен в приложение для более точного анализа.

API Security Testing

- Специализированное тестирование REST и GraphQL API;
- Проверка аутентификации и авторизации.

Cloud Security Audits

- Аудит облачной инфраструктуры (AWS, Azure, GCP);
- Проверка конфигураций и доступов.

Supply Chain Security

- Проверка безопасности зависимостей и библиотек;
- Анализ уязвимостей в third-party компонентах.

Bug Bounty Programs

- Привлечение независимых специалистов по безопасности;
- Поиск уязвимостей в боевых условиях.

AI-Powered Security Analysis

- Машинное обучение для обнаружения аномалий;
- Предиктивный анализ потенциальных угроз.

ИНСТРУМЕНТЫ И ТЕХНОЛОГИИ

Для анализа уязвимостей:

1. OWASP ZAP - бесплатный сканер безопасности веб-приложений.
2. SQLMap - специализированный инструмент для поиска и эксплуатации SQL-инъекций.
3. Acunetix - комплексный сканер уязвимостей веб-приложений.
4. SonarQube - анализирует качество кода и находит уязвимости на ранних стадиях разработки.

ЗАКЛЮЧЕНИЕ

Ключевые моменты:

1. Безопасность – процесс а не продукт. Это постоянная работа, которая должна быть встроена в процесс разработки.
2. Регулярные проверки обязательны - не ждите, пока вас взломают. Практично ищите уязвимости в коде.
3. Обучение команды критично - лучший инструмент защиты - это образованные разработчики, которые знают, как писать безопасный код.
4. Применяйте best practices сегодня - не откладывайте. Начните внедрять лучшие практики прямо сейчас.


Спасибо
за
внимание!