Стандарт верификации требований к безопасности приложений

Версия 5.0.0

Май 2025

Contents

Фронтиспис **7**
О стандарте . . . 7
Авторские права и лицензия . . . 7
Руководители проекта . . . 7
Рабочая группа . . . 7
Основные участники . . . 8
Другие участники и рецензенты . . . 8

Предисловие **8**
Введение . . . 8
Ключевые принципы версии 5.0 . . . 9
Взгляд в будущее . . . 9

Что такое ASVS? **10**
Область действия ASVS . . . 10
Приложение . . . 10
Безопасность . . . 11
Верификация . . . 11
Стандарт . . . 11
Требование . . . 11
Документированные решения по безопасности . . . 12
Уровни соответствия ASVS . . . 13
Оценка уровней соответствия . . . 14
Уровень 1 . . . 14
Уровень 2 . . . 15
Уровень 3 . . . 15
Какого уровня следует достичь . . . 15
Как применять этот стандарт . . . 15
Структура ASVS . . . 15
Стратегия выпусков . . . 16
Гибкость использования ASVS . . . 16
Как ссылаться на требования ASVS . . . 17
Создание ответвлений ASVS . . . 17
Сценарии применения ASVS . . . 18
Как подробное руководство по архитектуре безопасности . . . 18
Как специализированное руководство по безопасной разработке . . . 18
Как руководство по автоматизированному модульному и интеграционному тестированию . . . 18
Как курс по безопасной разработке . . . 19
Как руководство по приобретению безопасного программного обеспечения . . . 19

Применение ASVS на практике . . . 19

Оценка и сертификация **19**
Позиция OWASP в отношении сертификатов и знаков доверия ASVS . . . 19
Как проверять соответствие ASVS . . . 20
Отчет о проверке . . . 20
Область исследования . . . 20
Методы тестирования . . . 20

Изменения по сравнению с v4.x **22**
Введение . . . 22
Философия требований . . . 22
Целевая область применения . . . 22
Акцент на целях безопасности, а не на механизмах . . . 22
Документированные решения по безопасности . . . 23
Структурные изменения и новые главы . . . 23
Удаление прямых соответствий с другими стандартами . . . 24
Снижение связи с NIST Digital Identity Guidelines . . . 24
Отступление от Common Weakness Enumeration (CWE) . . . 24
Переосмысление определений уровней . . . 25
Снижение порога входа . . . 25
Иллюзия проверяемости . . . 25
Не только оценка рисков . . . 26

V1 Кодировка и нейтрализация **26**
Задачи контроля . . . 26
V1.1 Архитектура кодировки и нейтрализации . . . 26
V1.2 Предотвращение инъекций . . . 27
V1.3 Нейтрализация . . . 29
V1.4 Память, строки и неуправляемый код . . . 31
V1.5 Безопасная десериализация . . . 31
Ссылки . . . 32

V2 Валидация и бизнес-логика **33**
Задачи контроля . . . 33
V2.1 Документация валидации и бизнес-логики . . . 33
V2.2 Валидация ввода . . . 34
V2.3 Безопасность бизнес-логики . . . 35
V2.4 Анти-автоматизация . . . 36
Ссылки . . . 36

V3 Безопасность веб-интерфейса **37**
Задачи контроля . . . 37

V3.1 Документация по безопасности веб-интерфейса . . . 37
V3.2 Непреднамеренная интерпретация контента . . . 37
V3.3 Настройка файлов cookie . . . 38
V3.4 Заголовки механизма безопасности браузера . . . 39
V3.5 Браузерное разделение источников . . . 41
V3.6 Целостность внешних ресурсов . . . 43
V3.7 Другие соображения по безопасности браузера . . . 43
Ссылки . . . 44

V4 API и Веб-сервис **45**
Задачи контроля . . . 45
V4.1 Общая безопасность веб-сервисов . . . 45
V4.2 Валидация структуры HTTP-сообщений . . . 46
V4.3 GraphQL . . . 47
V4.4 WebSocket . . . 47
Ссылки . . . 48

V5 Работа с файлами **48**
Задачи контроля . . . 48
V5.1 Документация по работе с файлами . . . 48
V5.2 Загрузка файлов и содержимое . . . 49
V5.3 Хранение файлов . . . 50
V5.4 Скачивание файлов . . . 50
Ссылки . . . 51

V6 Аутентификация **51**
Задачи контроля . . . 51
V6.1 Документация по аутентификации . . . 52
V6.2 Безопасность паролей . . . 52
V6.3 Общие требования к безопасности аутентификации . . . 54
V6.4 Жизненный цикл и восстановление факторов аутентификации . . . 55
V6.5 Общие требования к многофакторной аутентификации . . . 56
V6.6 Внеполосные механизмы аутентификации . . . 57
V6.7 Криптографические механизмы аутентификации . . . 58
V6.8 Аутентификация с использованием провайдера идентификаций (IdP) . . . 59
Ссылки . . . 60

V7 Управление сессиями **61**
Задачи контроля . . . 61
V7.1 Документация по управлению сессиями . . . 61
V7.2 Базовая безопасность сессий . . . 62
V7.3 Тайм-ауты сессии . . . 62

V7.4 Завершение сессий . . . 63
V7.5 Защита от злоупотреблений сессией . . . 64
V7.6 Федеративная повторная аутентификация . . . 64
Ссылки . . . 65

V8 Авторизация **65**
Задачи контроля . . . 65
V8.1 Документация по авторизации . . . 65
V8.2 Архитектура авторизации . . . 66
V8.3 Авторизация на уровне операций . . . 67
V8.4 Прочие аспекты авторизации . . . 67
Ссылки . . . 68

V9 Автономные токены **68**
Задачи контроля . . . 68
V9.1 Источник токена и его целостность . . . 68
V9.2 Содержимое токена . . . 69
Ссылки . . . 70

V10 OAuth и OIDC **70**
Задачи контроля . . . 70
V10.1 Общие требования безопасности OAuth и OIDC . . . 72
V10.2 Клиент OAuth . . . 73
V10.3 Сервер ресурсов OAuth . . . 74
V10.4 Сервер авторизации OAuth . . . 75
V10.5 Клиент OIDC . . . 77
V10.6 Провайдер OpenID . . . 78
V10.7 Управление согласием . . . 79
Ссылки . . . 79

V11 Криптография **80**
Задачи контроля . . . 80
V11.1 Инвентаризация и документирование криптографии . . . 81
V11.2 Безопасная реализация криптографии . . . 82
V11.3 Алгоритмы шифрования . . . 83
V11.4 Хеширование и функции на основе хеширования . . . 84
V11.5 Случайные значения . . . 85
V11.6 Криптография с публичным ключом . . . 85
V11.7 Шифрование данных в процессе использования . . . 86
Ссылки . . . 86

V12 Безопасная коммуникация **87**
Задачи контроля . . . 87

V12.1 Общие рекомендации по безопасности TLS . . . 87
V12.2 HTTPS коммуникация с внешними сервисами . . . 88
V12.3 Общая безопасность межсервисного взаимодействия . . . 88
Ссылки . . . 89

V13 Конфигурация **90**
Задачи контроля . . . 90
V13.1 Документация конфигурации . . . 90
V13.2 Конфигурация бэкенд соединений . . . 91
V13.3 Управление секретами . . . 92
V13.4 Непреднамеренная утечка информации . . . 93
Ссылки . . . 94

V14 Защита информации **94**
Задачи контроля . . . 94
V14.1 Документация по защите информации . . . 95
V14.2 Базовые меры по защите информации . . . 95
V14.3 Защита данных на стороне клиента . . . 97
Ссылки . . . 97

V15 Безопасная разработка и архитектура **98**
Задачи контроля . . . 98
V15.1 Документация по безопасной разработке и архитектуре . . . 98
V15.2 Архитектура безопасности и зависимости . . . 99
V15.3 Безопасное программирование . . . 100
V15.4 Безопасная работа с параллельными процессами . . . 102
Ссылки . . . 102

V16 Журналирование событий безопасности и обработка ошибок **103**
Задачи контроля . . . 103
V16.1 Документация по ведению логов . . . 103
V16.2 Общее журналирование . . . 103
V16.3 Инциденты безопасности . . . 104
V16.4 Защита журналов . . . 105
V16.5 Обработка ошибок . . . 106
Ссылки . . . 107

V17 WebRTC **107**
Задачи контроля . . . 107
V17.1 TURN-сервер . . . 108
V17.2 Требования к медиасерверам . . . 109
V17.3 Сигнализация . . . 110
Ссылки . . . 111

Приложение A: Термины и сокращения **111**

Приложение B: Полезные ссылки **119**
Основные проекты OWASP . . . 119
Проект OWASP Cheat Sheet Series . . . 119
Проекты Mobile Security . . . 119
Проект OWASP Internet of Things . . . 119
Проект OWASP Serverless . . . 119
Другие . . . 119

Приложение C: Стандарты криптографии **120**
Инвентаризация и документирование криптографии . . . 120
Эквивалентная стойкость криптографических параметров . . . 121
Случайные значения . . . 121
Алгоритмы шифрования . . . 123
Режимы шифрования AES . . . 124
Шифрование ключей . . . 125
Аутентифицированное шифрование . . . 125
Хэш-функции . . . 126
Основные случаи использования хэш функций . . . 126
Хэш-функции для хранения паролей . . . 129
Функции формирования ключа (KDFs) . . . 130
Основные функции формирования ключа . . . 130
Функции формирования ключа на основе пароля . . . 130
Механизмы обмена ключами . . . 131
Схемы KEX . . . 131
Группы Диффи-Хеллмана . . . 132
Коды аутентификации сообщений (MAC) . . . 133
Цифровые подписи . . . 134
Стандарты постквантового шифрования . . . 134

Приложение D: Рекомендации **134**
Введение . . . 134
Рекомендуемые механизмы, входящие в область применения . . . 135
Принципы безопасности программного обеспечения . . . 135
Процессы обеспечения безопасности программного обеспечения . . . 136

Приложение E: Участники **137**

Фронтиспис

О стандарте

Стандарт верификации требований к безопасности приложений —это перечень требований к безопасности приложений (тестов), которыми могут пользоваться архитекторы, разработчики, тестировщики, специалисты по безопасности, разработчики инструментов и конечные пользователи для проектирования, разработки, тестирования и контроля безопасных приложений.

Авторские права и лицензия

Версия 5.0.0, май 2025

Figure 1: лицензия

Руководители проекта

Elar Lang	Josh C Grossman
Jim Manico	Daniel Cuthbert

Рабочая группа

Tobias Ahnoff	Ralph Andalis	Ryan Armstrong	Gabriel Corona
Meghan Jacquot	Shanni Prutchi	Iman Sharafaldin	Eden Yardeni

Основные участники

Sjoerd Langkemper	Isaac Lewis
Mark Carney	Sandro Gauci

Другие участники и рецензенты

Список остальных участников перечислен в Приложении E.

Если Вы —участник проекта, но Вашего имени нет в приведенном выше списке, пожалуйста, зарегистрируйте issue на GitHub для признания в будущих обновлениях.

Пятая версия стандарта верификации требований к безопасности приложений опирается на предыдущие версии ASVS, начиная с первой, вышедшей в 2008 году, и до четвертой, в 2019 году. Большая часть разделов оглавления и пунктов требований, до сих пор присутствующих в ASVS, изначально были написаны Эндрю ван дер Стоком, Майком Боберски, Джеффом Уильямсом и Дэйвом Уичерсом, но участников было гораздо больше - спасибо им всем. Полные списки всех, кто внес свой вклад в более ранние версии, есть в соответствующих версиях документа.

Предисловие

Добро пожаловать в Стандарт верификации требований к безопасности приложений (ASVS) версии 5.0

Введение

Изначально представленный в 2008 году совместными усилиями мирового сообщества, ASVS определяет исчерпывающий набор требований безопасности для проектирования, разработки и тестирования современных веб-приложений и сервисов.

После выхода версии ASVS 4.0 в 2019 году и ее незначительного обновления (v4.0.3) в 2021 году, версия 5.0 является значительным шагом вперед —она была модернизирована, чтобы учесть последние достижения в области безопасности программного обеспечения.

ASVS 5.0 стал результатом масштабного вклада руководителей проекта, членов рабочей группы и широкого сообщества OWASP, направленного на обновление и совершенствование этого важного стандарта.

Ключевые принципы версии 5.0

Эта масштабная редакция была разработана с учетом нескольких ключевых принципов:

- Уточнение границ и целей: Данная версия стандарта была разработана для более точного соответствия основным принципам, отраженным в его названии: Приложение (Application), Безопасность (Security), Проверка (Verification), Стандарт (Standard). Требования были переформулированы - теперь они направлены на предотвращение недостатков безопасности, а не на обязательное применение конкретных технических решений. Тексты требований являются самодостаточными и поясняют свою необходимость.
- Поддержка документирования решений по безопасности: ASVS 5.0 вводит требования к документированию ключевых решений по безопасности. Данный подход улучшает отслеживаемость решений и способствует контекстуально-ориентированным реализациям, предоставляя организациям возможность настраивать свою защиту в соответствии с индивидуальными требованиями и рисками.
- Обновленные уровни: Несмотря на сохранение трехуровневой модели, определения уровней были пересмотрены для упрощения внедрения ASVS. Уровень 1 предназначен для начального этапа внедрения стандарта и обеспечивает базовый уровень защиты. Уровень 2 представляет собой комплексный набор стандартных практик безопасности. Уровень 3 охватывает расширенные требования для обеспечения повышенной гарантии безопасности.
- Реструктуризированное и расширенное содержание: ASVS 5.0 включает приблизительно 350 требований, распределенных по 17 главам. Структура глав была переработана для большей ясности и удобства использования. Для облегчения перехода предоставлено двустороннее соответствие между версиями 4.0 и 5.0.

Взгляд в будущее

Так же, как и обеспечение безопасности приложения никогда не может быть полностью завершено, работа над ASVS также никогда не прекращается. Несмотря на то, что версия 5.0 является крупным релизом, разработка стандарта продолжается. Данный выпуск позволяет сообществу воспользоваться преимуществами накопленных улучшений и дополнений, а также закладывает основу для будущего развития. Это может включать в себя инициативы сообщества по созданию руководств по внедрению и проверке, основанных на базовом наборе требований.

ASVS 5.0 предназначен для того, чтобы служить надежной основой для разработки безопасного программного обеспечения. Сообществу предлагается применять этот стандарт, вносить в него свой вклад и развивать его, чтобы совместно двигать вперёд состояние безопасности приложений.

Что такое ASVS?

Стандарт верификации требований к безопасности приложений (ASVS) определяет требования безопасности для веб-приложений и сервисов, представляя собой ценный ресурс для всех, кто стремится проектировать, разрабатывать и поддерживать безопасные приложения или оценивать их уровень защищённости.

В этой главе описываются ключевые аспекты применения ASVS, включая его область действия, структуру уровней, основанных на приоритетах, и основные сценарии использования стандарта.

Область действия ASVS

Область действия ASVS определяется его названием: Приложение (Application), Безопасность (Security), Верификация (Verification) и Стандарт (Standard). Он устанавливает, какие требования включены в стандарт, а какие —исключены из него, с глобальной целью определения основополагающих принципов безопасности, которые должны быть достигнуты. Область действия также учитывает требования к документации, которые служат основой для требований к реализации.

Не существует такого понятия, как «область действия» для злоумышленника. Поэтому требования ASVS должны оцениваться в совокупности с рекомендациями по другим аспектам жизненного цикла приложения, включая процессы CI/CD, хостинг и операционную деятельность.

Приложение

ASVS определяет «Приложение» как разрабатываемый программный продукт, в который должны быть интегрированы механизмы безопасности. ASVS не регламентирует процессы жизненного цикла разработки и не указывает на методы сборки приложения в CI/CD. Его задача —описать требуемый уровень защищённости, который должен быть достигнут в конечном продукте.

Компоненты для обработки HTTP-трафика (WAF, балансировщики нагрузки, прокси) могут считаться частью приложения в контексте безопасности, поскольку некоторые механизмы безопасности напрямую зависят от них или могут быть реализованы с их помощью. Это касается требований по кешированию, rate limiting, а также фильтрации входящих/исходящих подключений по источнику и получателю.

В свою очередь, ASVS не включает требования, не относящиеся к приложению напрямую или находящиеся за пределами его зоны ответственности. Так, например, проблемы DNS обычно относятся в ведении отдельной команды или функции.

Аналогично, хотя в зону ответственности приложения входит обработка входящих данных и генерация исходящих данных, если внешний процесс взаимодействует с приложением или его данными, это считается выходящим за рамки ASVS. Например, резервное копирование приложения или его данных обычно выполняется внешними процессами и не контролируется самим приложением или его разработчиками.

Безопасность

Каждое требование должно иметь очевидное влияние на безопасность. Отсутствие требования должно привести к снижению уровня защищённости приложения, а реализация требования должна либо уменьшить вероятность возникновения риска безопасности, либо смягчить последствия.

Все прочие аспекты, такие как функциональные характеристики, стиль кода или требования политик, выходят за рамки стандарта.

Верификация

Требование должно быть верифицируемым, а его проверка должна приводить к однозначному решению: «не выполнено» или «выполнено».

Стандарт

ASVS представляет собой набор требований безопасности, которые необходимо реализовать для соответствия стандарту. Это означает, что требования ограничиваются определением целевого показателя безопасности, которого необходимо достичь. Прочая сопутствующая информация может быть надстроена на основе ASVS или связана с ним через сопоставления.

В частности, в рамках OWASP существует множество проектов, и ASVS намеренно избегает дублирования содержания других проектов. Например, у разработчиков может возникнуть вопрос: «как мне реализовать конкретное требование в моем технологическом стеке или окружении?» —на него должен отвечать проект Cheat Sheet Series. У тестировщиков может возникнуть вопрос: «как мне проверить это требование в данном окружении?» —на него должен отвечать проект Web Security Testing Guide.

Хотя ASVS предназначен не только для специалистов по безопасности, он ожидает, что читатель обладает техническими знаниями для понимания содержания или способен самостоятельно изучить конкретные концепции.

Требование

Слово «Требование» специально используется в ASVS, поскольку оно описывает обязательное условие, которое должно быть достигнуто для его выполнения. ASVS содержит только

требования (обязательные к выполнению) и не содержит рекомендаций.

Другими словами, рекомендации, независимо от того, являются ли они лишь одним из многих возможных вариантов для решения проблем или вопросов стиля кода, не удовлетворяют определению требования.

Требования ASVS определяют цели безопасности, не привязываясь к конкретным технологиям или методам реализации. Их назначение понятно из самого текста, и они не предписывают, как именно следует проводить проверку.

Документированные решения по безопасности

В области безопасности программного обеспечения заблаговременное планирование архитектуры безопасности и защитных механизмов позволяет добиться более последовательной и предсказуемой реализации в конечном продукте или функции.

Кроме того, некоторые требования сложны в реализации и зависят от специфики приложения. Например, система прав доступа, проверка вводимых данных и защита данных с разной степенью чувствительности.

Чтобы учесть это, вместо общих формулировок, таких как «все данные должны быть зашифрованы», или попыток охватить каждый возможный случай в требовании, были включены требования к документации. Они предписывают, чтобы подход и конфигурация, выбранные разработчиком приложения для механизмов управления, были задокументированы. Это позволяет провести оценку целесообразности подхода, а затем сравнить фактическую реализацию с документацией, чтобы оценить, соответствует ли реализация ожиданиям.

Задача этих требований —фиксировать решения, которые организация-разработчик приняла в отношении того, каким образом реализовать определенные требования безопасности.

Требования к документации всегда размещаются в первом разделе главы (хотя присутствуют не в каждой из них), и каждое из них всегда связано с соответствующим требованием к реализации, в рамках которого документированные решения должны быть воплощены на практике. Ключевая идея заключается в том, что проверка наличия документации и проверка фактической реализации —это два самостоятельных вида деятельности.

Существует две ключевые причины для включения этих требований. Первая причина заключается в том, что требование информационной безопасности часто предполагает принудительное применение правил, например, какие типы файлов разрешены для загрузки, какие бизнес-контроли должны быть реализованы, какие символы допустимы для конкретного поля. Эти правила уникальны для каждого приложения и поэтому ASVS не может предписать, какими они должны быть. Ни чек-лист, ни развернутое руководство в данном случае не будут полезны. Также, если эти решения не документированы, то невозможно проверить реализацию на соответствие требованиям.

Вторая причина заключается в необходимости обеспечить гибкость для разработчиков приложения в выборе подходов к решению отдельных вопросов безопасности. Например, в

прошлых версиях ASVS ограничения времени жизни сессии были строго фиксированы. На практике многие приложения, особенно ориентированные на прямое взаимодействие с потребителями, применяют более мягкие правила, отдавая предпочтение иным компенсирующим механизмам защиты. Таким образом, настоящие требования к документации прямо разрешают такую гибкость.

Очевидно, что предполагается, что подобные решения будут приниматься и документироваться не отдельными разработчиками, а организацией в целом. На организацию возлагается ответственность за принятие данных решений, доведение их до сведения разработчиков и обеспечение их соблюдения.

Предоставление разработчикам спецификаций и проектов новых функций является стандартной практикой в процессе разработки программного обеспечения. Аналогичным образом, от разработчиков ожидается использование общепринятых компонентов и механизмов пользовательского интерфейса, а не принятие самостоятельных решений каждый раз. Таким образом, распространение данного подхода на безопасность не должно восприниматься как нечто удивительное или спорное.

Кроме того, существует гибкость в выборе способа достижения данной цели. Решения в области безопасности могут быть документированы в виде явного документа, с которым разработчики обязаны знакомиться. Альтернативно, эти решения могут быть документированы и реализованы в рамках общей библиотеки, которую все разработчики обязаны использовать. Результат достигается в обоих случаях.

Уровни соответствия ASVS

Стандарт верификации требований к безопасности приложений определяет три уровня соответствия, каждый из которых усиливает требования предыдущего. Рекомендуемый подход для организаций —начинать с первого уровня для митигации наиболее критичных рисков безопасности с последующим переходом на более высокие уровни в зависимости от конкретных потребностей приложения и организации. В документации и требованиях используются сокращения L1, L2 и L3.

Каждый уровень соответствия определяет требования безопасности, которые необходимо выполнить для его достижения. Требования же вышестоящих уровней носят рекомендательный характер.

Во избежание дублирования требований или наличия требований, теряющих актуальность на более высоких уровнях, некоторые из них применяются к конкретному уровню, но предусматривают более строгие условия для последующих уровней.

Оценка уровней соответствия

Уровни соответствия определяются путём приоритизационной оценки каждого требования, основанной на опыте их внедрения и тестирования. Основное внимание уделяется сопоставлению снижения рисков с усилиями по реализации требования. Другим ключевым фактором является сохранение низкого порога вхождения.

Оценка снижения риска рассматривает, насколько требование уменьшает уровень риска безопасности в приложении. При этом принимаются во внимание классические факторы воздействия на Конфиденциальность, Целостность и Доступность, а также то, является ли данное требование основным рубежом обороны или же оно рассматривается как элемент эшелонированной защиты.

В результате тщательных обсуждений как самих критериев, так и решений по отнесению их к уровням соответствия, была выработана классификация, которая должна оставаться справедливой для подавляющего большинства случаев, при этом допуская, что она не может быть универсальной для всех ситуаций. Это означает, что в отдельных случаях организации могут пожелать отдать приоритет требованиям более высокого уровня ранее, основываясь на собственной оценке рисков.

Требования на каждом уровне можно охарактеризовать следующим образом:

Уровень 1

Данный уровень содержит минимальный набор требований для обеспечения безопасности приложения, и представляет собой критически важную отправную точку. На этот уровень приходится около 20% требований ASVS. Цель этого уровня —иметь как можно меньше требований для снижения порога вхождения.

Эти требования, как правило, являются критически важными или базовыми, представляя собой первый рубеж защиты от распространённых атак, для эксплуатации которых не требуется наличия других уязвимостей или предварительных условий.

Помимо требований первого рубежа защиты, некоторые требования имеют меньшее значение на более высоких уровнях, например, требования, связанные с паролями. Они более важны для первого уровня, поскольку, начиная с более высоких уровней становятся актуальными требования многофакторной аутентификации.

Первый уровень не обязательно поддаётся полноценному тестированию на проникновение внешним специалистом без доступа ко внутренней документации или коду (например, при тестировании по методу «чёрного ящика»), хотя меньшее количество требований должно упростить процесс верификации.

Уровень 2

Большинству приложений следует стремиться к достижению данного уровня безопасности. На Уровень 2 (L2) приходится около 50% требований ASVS, что означает, что приложению необходимо реализовать порядка 70% требований ASVS (все требования уровней L1 и L2) для соответствия этому уровню.

Эти требования, как правило, связаны либо с менее распространёнными атаками, либо с более сложными механизмами защиты от распространённых атак. Они могут по-прежнему представлять собой первый рубеж обороны, либо же для успешной атаки могут требоваться определённые предварительные условия.

Уровень 3

Данный уровень должен быть целью для приложений, стремящихся продемонстрировать наивысший уровень защищённости, и содержит оставшиеся ~30% требований для полного соответствия.

Требования в данном разделе, как правило, представляют собой либо механизмы глубокой эшелонированной обороны (defense-in-depth), либо иные полезные, но сложные в реализации средства контроля.

Какого уровня следует достичь

Уровни, основанные на приоритетах, призваны отражать степень зрелости системы безопасности организации и приложения. Вместо того, чтобы предписывать, какого уровня должно достигать приложение, ASVS предлагает организации проанализировать свои риски и самостоятельно определить целевой уровень, исходя из критичности приложения и, конечно же, ожиданий его пользователей.

Например, стартап на ранней стадии, собирающий лишь ограниченный объём конфиденциальных данных, может принять решение сфокусироваться на первом уровне на начальном этапе. В то же время банк не сможет объяснить клиентам выбор уровня ниже, чем Уровень 3, для своего онлайн банковского приложения.

Как применять этот стандарт

Структура ASVS

ASVS состоит в общей сложности из приблизительно 350 требований, которые разделены на 17 глав. Каждая глава, в свою очередь, делится на разделы.

Цель такого деления на главы и разделы —упростить выбор или исключение соответствующих глав и разделов в зависимости от релевантности для конкретного приложения. Например,

требования главы V3, касающиеся веб-интерфейсов, не будут актуальны в случае API для межмашинного взаимодействия. Если в системе не используется OAuth или WebRTC, то соответствующие главы также могут быть проигнорированы.

Стратегия выпусков

Выпуски ASVS следуют схеме “Major.Minor.Patch”, где номера версий указывают на характер изменений в выпуске. При мажорном выпуске изменяется первое число, при минорном — второе, а при патче —третье.

- Мажорный выпуск (Major release) —Полная реорганизация. Могут быть изменены практически все элементы, включая нумерацию требований. Для подтверждения соответствия потребуется проведение повторной оценки (например, переход с версии 4.0.3 на 5.0.0).
- Минорный выпуск (Minor release) —Требования могут добавляться или удаляться, но общая структура нумерации остаётся прежней. Для подтверждения соответствия потребуется повторная оценка, но она должна быть проще (например, переход с версии 5.0.0 на 5.1.0).
- Выпуск патча (Patch release) —Требования могут быть удалены (например, если они являются дубликатами или устарели) или смягчены. Приложение, соответствовавшее предыдущему выпуску, будет соответствовать и патч-выпуску (например, переход с версии 5.0.0 на 5.0.1).

Вышеизложенное относится только к самим требованиям. Изменения в пояснительных текстах и приложениях не являются критическими.

Гибкость использования ASVS

Ряд рассмотренных принципов (например, требования к документации и система уровней) предоставляет возможность применять ASVS гибко, с учётом специфики организации.

Кроме того, организациям настоятельно рекомендуется создавать специализированные ответвления (форки) стандарта, адаптированные для организации или домена, которые корректируют требования исходя из специфических характеристик и уровней риска их приложений. Однако важно сохранять прослеживаемость (traceability), чтобы соответствие требованию, например, 4.1.1, означало одно и то же во всех версиях.

В идеале, каждой организации следует создавать собственную адаптированную версию ASVS, исключая нерелевантные разделы (например, GraphQL, WebSockets, SOAP, если они не используются). Версия ASVS, адаптированная под организацию, или дополнение к ней также является хорошим способом предоставить специфичные для организации руководства по реализации, уточняя библиотеки или ресурсы, которые следует использовать для выполнения требований.

Как ссылаться на требования ASVS

Каждое требование имеет идентификатор в формате < >.< >.< > где каждый элемент является числом, например: 1.11.3.

- Значение < > соответствует главе, в которой находится это требование, например: все требования 1.#.# находятся в главе .
- Значение < > соответствует разделу в этой главе, где находится это требование, например: все требования 1.2.# находятся в разделе главы .
- Значение < > указывает на конкретное требование в главе и разделе, например: 1.2.5, которым в версии 5.0.0 этого стандарта является:

> Убедитесь, что приложение защищено от внедрения команд ОС и что вызовы операционной системы используют параметризованные запросы к ОС или используют контекстное кодирование вывода командной строки.

Идентификаторы требований могут меняться от версии к версии, поэтому желательно, чтобы другие документы, отчеты или инструменты использовали формат: v< >-< >.< >.< >, где: это метка версии ASVS. Например: v5.0.0-1.2.5 будет означать 5-е требование в разделе главы в версии 5.0.0. (Резюмируя, v< >-< _ >.)

Примечание. Буква v, предшествующая версии, должна быть в нижнем регистре.

Считается, что если идентификаторы указаны без элемента v< >, то они относятся к последней версии стандарта. По мере развития и изменения стандарта это станет проблематичным, поэтому авторы и разработчики должны включать элемент версии.

Списки требований ASVS доступны в форматах CSV, JSON и других, которые могут оказаться полезными для справки или использования в программах.

Создание ответвлений ASVS

Организации могут извлечь пользу из применения ASVS, выбрав один из трёх уровней или создав предметно-ориентированное ответвление (fork), которое адаптирует требования в соответствии с уровнем риска каждого приложения. Такой подход настоятельно рекомендуется при условии сохранения прослеживаемости (traceability), чтобы соответствие требованию, например, 4.1.1, означало одно и то же во всех версиях.

В идеале каждой организации следует создать собственную адаптированную версию ASVS, исключая нерелевантные разделы (например, GraphQL, WebSockets, SOAP, если они не используются). Создание ответвления следует начинать с использования ASVS Уровня 1 в качестве базового, с последующим переходом на Уровни 2 или 3 в зависимости от рисков, присущих приложению.

Сценарии применения ASVS

ASVS может быть использован для оценки безопасности приложения, и этот вопрос более глубоко исследуется в следующей главе. Тем не менее, был выявлен ряд других потенциальных вариантов использования ASVS (или его модифицированной версии).

Как подробное руководство по архитектуре безопасности

Одним из наиболее распространенных вариантов использования Стандарта верификации требований к безопасности приложений является его применение в качестве ресурса для архитекторов безопасности. В настоящее время существует ограниченное количество материалов о том, как построить безопасную архитектуру современного приложения. ASVS может быть использован для восполнения этих пробелов, позволяя архитекторам безопасности выбирать более эффективные меры для решения распространенных проблем, таких как шаблоны защиты и стратегии валидации входных данных. Требования, связанные с архитектурой и документацией, будут особенно полезны для этой цели.

Как специализированное руководство по безопасной разработке

ASVS может быть использован как основа для подготовки руководства по безопасной разработке в процессе разработки приложения, помогая разработчикам не упускать из виду аспекты безопасности при создании программного обеспечения. Хотя ASVS может служить основой, организациям следует подготовить свои собственные конкретные руководства, которые являются четкими и унифицированными, и в идеале их следует составлять на основе рекомендаций инженеров или архитекторов безопасности. В развитие этого подхода организациям настоятельно рекомендуется, по возможности, подготовить одобренные механизмы безопасности и библиотеки, на которые можно ссылаться в руководствах и которые разработчики могут использовать.

Как руководство по автоматизированному модульному и интеграционному тестированию

ASVS создавался таким, чтобы его можно было легко протестировать. Некоторые проверки носят технический характер, в то время как другие требования могут потребовать анализа документации или архитектуры. Благодаря модульным и интеграционным тестам, которые проверяют характерные случаи фаззинга и недокументированного использования, связанных с требованиями, поддающимися технической верификации, можно упростить проверку корректности работы этих контролей при каждой сборке. Например, для набора тестов для контроллера входа в систему можно разработать дополнительные тесты, проверяющие атрибут username на совпадение с наиболее распространенными именами пользователей по умолчанию, перебор учетных записей, LDAP- и SQL-инъекции, а также XSS. Аналогично, проверка атрибута password должна включать перебор самых

распространенных паролей, оценку длины пароля, вставку нулевого байта, удаление этого атрибута, XSS и многое другое.

Как курс по безопасной разработке

ASVS также можно использовать для определения характеристик защищенного программного обеспечения. Многие «курсы безопасной разработки» —это просто курсы по этичному хакингу с легким намеком на советы по безопасной разработке. Не факт, что это как-то поможет разработчикам писать более безопасный код. Вместо этого курсы по безопасной разработке должны делать упор на упреждающие меры из ASVS, а не на список того, чего делать не стоит (OWASP Top 10). Структура ASVS также предоставляет логичную основу для последовательного изучения различных тем, связанных с защитой приложения.

Как руководство по приобретению безопасного программного обеспечения

ASVS —это отличное руководство по приобретению безопасного программного обеспечения или заказных услуг по его разработке. Заказчику достаточно указать требование, что программное обеспечение, которое он хочет приобрести, должно быть разработано на уровне ASVS LX, и потребовать, чтобы Исполнитель доказал, что оно ему соответствует.

Применение ASVS на практике

Разные угрозы имеют разную мотивацию. В отдельных отраслях имеются уникальные информационные и технологические ресурсы или отраслевые регуляторные требования.

Организациям настоятельно рекомендуется внимательно изучить риски, присущие характеру их бизнеса, и на их основе, и с учетом бизнес-требований определить соответствующий уровень ASVS.

Оценка и сертификация

Позиция OWASP в отношении сертификатов и знаков доверия ASVS

OWASP, будучи независимой от поставщиков некоммерческой организацией, не сертифицирует разработчиков, аудиторов или программное обеспечение. Любые заверения, знаки доверия или сертификаты, претендующие на соответствие ASVS, официально не поддерживаются OWASP, поэтому организации должны проявлять осторожность в заявлении третьих сторон о сертификации по ASVS.

Организации могут предлагать услуги по предоставлению гарантий соответствия при условии, что они не претендуют на официальную сертификацию OWASP.

Как проверять соответствие ASVS

Стандарт намеренно не предписывает конкретные методы проверки соответствия на уровне руководства по тестированию. Тем не менее, важно выделить некоторые ключевые моменты.

Отчет о проверке

Традиционные отчёты о тестировании на проникновение фиксируют только неудачные проверки. Однако отчёт о проверке соответствия ASVS должен включать в себя область исследования, сводку всех проверяемых требований, а также требования, по которым были отмечены исключения, и руководство по устранению выявленных проблем. Некоторые требования могут быть неприменимы (например, управление сессиями в stateless API), и это должно быть явно указано в отчёте.

Область исследования

Организация, разрабатывающая приложение, как правило, реализует не все требования, поскольку некоторые из них могут быть нерелевантными или менее значимыми исходя из функциональности приложения. Аудитор должен чётко определить область проверки, включая то, какого уровня соответствия пытается достичь организация и какие требования были включены в оценку. Это должно быть представлено с точки зрения того, что было включено, а не того, что было исключено. Кроме того, аудитор должен предоставить заключение с обоснованием причин исключения требований, которые не были реализованы.

Это должно позволить заказчику отчёта о проверке понять контекст проведённого исследования и принять взвешенное решение о том, какой уровень доверия может быть предоставлен приложению.

Сертифицирующие организации могут выбирать свои методы тестирования, но должны раскрывать их в отчёте и в идеале эти методы должны быть воспроизводимыми. Для проверки различных аспектов (таких как валидация входных данных) в зависимости от приложения и конкретных требований могут использоваться разные методы, например, ручное тестирование на проникновение или анализ исходного кода.

Методы тестирования

Для верификации конкретных требований ASVS может потребоваться применение ряда различных методик. Помимо тестирования на проникновение (с использованием действительных учётных данных для обеспечения полного охвата приложения), проверка соответствия требованиям ASVS может потребовать доступа к документации, исходному коду, конфигурациям, а также к лицам, причастным к процессу разработки. Это особенно

актуально для верификации требований Уровней 2 и 3. Стандартной практикой является предоставление убедительных доказательств обнаруженных недостатков с подробным документированием, которое может включать записи тестировщика, скриншоты, скрипты и журналы тестирования. Простого запуска автоматизированного инструмента недостаточно для сертификации, поскольку каждое требование должно быть тщательно протестировано.

Использование средств автоматизации для верификации требований ASVS —это тема, которая неизменно вызывает значительный интерес. В связи с этим важно прояснить некоторые моменты, связанные с автоматизированным тестированием и тестированием по методу «чёрного ящика».

Роль инструментов автоматизированного тестирования безопасности Когда автоматизированные инструменты динамического и статического анализа безопасности приложений (DAST и SAST) корректно интегрированы в pipeline сборки, они могут выявлять некоторые проблемы безопасности, которые не должны существовать в принципе. Однако без тщательной конфигурации и настройки они не обеспечат необходимый охват, а большое количество ложных срабатываний помешает выявлению и устранению реальных проблем безопасности.

Хотя это может обеспечить покрытие некоторых более базовых и простых технических требований, относящихся к кодированию или нейтрализации выходных данных, крайне важно отметить, что эти инструменты неспособны в полной мере проверить многие из более сложных требований ASVS или тех, что связаны с бизнес-логикой и контролем доступа.

Что касается менее тривиальных требований, вероятно, что автоматизацию всё же можно задействовать, однако для этого потребуется написание специфичных для приложения проверок. Они могут быть аналогичны модульным и интеграционным тестам, которые организация, возможно, уже использует. Следовательно, может оказаться возможным использовать существующую инфраструктуру автоматизации тестирования для создания этих специализированных тестов под требования ASVS. Хотя для этого потребуются краткосрочные инвестиции, долгосрочные преимущества, связанные с возможностью постоянной верификации этих требований ASVS, будут значительными.

Подводя итог, автоматизированное тестирование != запуску стандартного инструмента «из коробки».

Роль тестирования на проникновение Несмотря на то, что Уровень 1 (L1) в версии 4.0 был оптимизирован для проведения тестирования по методу «чёрного ящика» (без доступа к документации и исходному коду), даже в то время стандарт четко давал понять, что это не является эффективной деятельностью, гарантирующей соответствие, и такой подход не следует активно применять.

Тестирование без доступа к необходимой дополнительной информации является неэффективным и малорезультативным, поскольку при этом упускается возможность проверки исходного

кода, выявления угроз и отсутствующих мер защиты, а также выполнения гораздо более тщательного тестирования в более короткие сроки.

Настоятельно рекомендуется проводить тестирование на проникновение, основанное на изучении документации и исходного кода (гибридный подход), в ходе которого тестировщики имеют полный доступ к разработчикам приложения и его документации, в отличие от традиционного тестирования на проникновение. Этот подход будет необходимым для верификации многих требований ASVS.

Изменения по сравнению с v4.x

Введение

Пользователям, знакомым со стандартом версии 4.x, может быть полезно ознакомиться с ключевыми изменениями, внесёнными в версию 5.0, включая обновления в содержании, области применения и основополагающей философии.

Из 286 требований версии 4.0.3 только 11 остались без изменений, в то время как 15 претерпели незначительные грамматические правки, не повлиявшие на их смысл. В общей сложности 109 требований (38%) более не являются отдельными требованиями в версии 5.0 –50 были просто удалены, 28 удалены как дубликаты и 31 объединено с другими требованиями. Остальные требования были так или иначе пересмотрены. Даже требования, не претерпевшие существенных изменений, имеют новые идентификаторы из-за переупорядочивания или реструктуризации.

Для облегчения перехода на версию 5.0 предоставляются документы сопоставления, которые помогают пользователям отследить, как требования из версии 4.x соотносятся с требованиями в версии 5.0. Эти сопоставления не привязаны к версиям выпусков и могут при необходимости обновляться или уточняться.

Философия требований

Целевая область применения

Версия 4.x включала требования, которые не соответствовали целевой области применения стандарта - они были удалены. Требования, которые не удовлетворяли критериям области применения для версии 5.0 или не поддавались верификации, также были исключены.

Акцент на целях безопасности, а не на механизмах

В версии 4.x многие требования были сфокусированы на конкретных механизмах, а не на лежащих в их основе целях безопасности. В версии 5.0 требования сконцентрированы

на целях безопасности, ссылаясь на конкретные механизмы только в тех случаях, когда они являются единственным практическим решением, или предоставляя их в качестве примеров или дополнительного руководства.

Такой подход допускает разные методы достижения цели, избегая излишних предписаний, которые могли бы ограничить гибкость компаний.

Кроме того, требования, касающиеся одной и той же проблемы безопасности, были объединены там, где это было необходимо.

Документированные решения по безопасности

Хотя концепция документированных решений по безопасности может показаться новой в версии 5.0, она является развитием более ранних требований, связанных с применением политик и моделированием угроз в версии 4.0. Ранее некоторые требования неявно предполагали проведение анализа для обоснования реализации средств контроля безопасности, такого как определение разрешенных сетевых подключений.

Чтобы обеспечить доступность необходимой информации для реализации и проверки, эти ожидания теперь явно определены как требования к документации, делая их понятными, практичными и проверяемыми.

Структурные изменения и новые главы

Несколько глав в версии 5.0 представляют полностью новый контент:

- OAuth и OIDC –Учитывая широкое распространение этих протоколов для делегирования доступа и единого входа, были добавлены специальные требования для решения разнообразных сценариев, с которыми могут столкнуться разработчики. Эта область может в конечном итоге развиться в самостоятельный стандарт, аналогично тому, как это произошло с требованиями для мобильных устройств и IoT в предыдущих версиях.
- WebRTC –Поскольку эта технология набирает популярность, ее уникальные проблемы и аспекты безопасности теперь рассматриваются в отдельном разделе.

Также были приложены усилия, чтобы обеспечить организацию глав и разделов вокруг логичных наборов связанных требований.

Эта реструктуризация привела к созданию дополнительных глав:

- Автономные токены (Self-contained Tokens) –Ранее относящиеся к управлению сессиями, автономные токены теперь признаны самостоятельным механизмом и базовым элементом для взаимодействия без сохранения состояния (такого как в OAuth и OIDC). Ввиду их уникальных аспектов безопасности, они рассматриваются в отдельной главе с некоторыми новыми требованиями, введенными в версии 5.x.

- Безопасность веб-интерфейсов (Web Frontend Security) –В связи с растущей сложностью браузерных приложений и распространением API-ориентированных архитектур, требования к безопасности фронтенда были выделены в отдельную главу.
- Безопасная разработка и архитектура (Secure Coding and Architecture) –Новые требования, касающиеся общих практик безопасности, которые не вписывались в существующие главы, были сгруппированы здесь.

Другие организационные изменения в версии 5.0 были внесены для уточнения целей. Например, требования к валидации входных данных были помещены рядом с бизнес-логикой, а не с кодировкой и обработкой, отражая их роль в обеспечении соблюдения бизнес-правил.

Бывшая глава V1 «Архитектура, проектирование и моделирование угроз» была удалена. Ее начальный раздел содержал требования, которые выходили за рамки области проверки, в то время как последующие разделы были перераспределены в соответствующие главы, с удалением дубликатов и уточнениями по мере необходимости.

Удаление прямых соответствий с другими стандартами

Прямые соответствия другим стандартам были удалены из основного тела стандарта. Цель состоит в подготовке соответствия с проектом OWASP Common Requirement Enumeration (CRE), который, в свою очередь, свяжет ASVS с рядом проектов OWASP и внешними стандартами.

Прямые соответствия CWE и NIST более не поддерживаются, как поясняется ниже.

Снижение связи с NIST Digital Identity Guidelines

Рекомендации NIST Digital Identity Guidelines (SP 800-63) долгое время служили ориентиром для средств контроля аутентификации и авторизации. В версии 4.x определенные главы были тесно связаны со структурой и терминологией NIST.

Хотя эти рекомендации остаются важным ориентиром, строгое соответствие создавало проблемы, включая менее распространенную терминологию, дублирование схожих требований и неполные соответствия. Версия 5.0 отходит от этого подхода для повышения ясности и релевантности.

Отступление от Common Weakness Enumeration (CWE)

Common Weakness Enumeration (CWE) предоставляет полезную систематику уязвимостей программного обеспечения. Однако такие проблемы, как широкие классы CWE (category-only CWEs), трудности в сопоставлении требований с отдельным CWE, а также наличие неточных соответствий в версии 4.x, привели к решению отказаться от прямых сопоставлений с CWE в версии 5.0.

Переосмысление определений уровней

Версия 4.x описывала уровни как L1 («Минимальный»), L2 («Стандартный») и L3 («Продвинутый»), с подразумеванием, что все приложения, обрабатывающие конфиденциальные данные, должны соответствовать как минимум L2.

Версия 5.0 решает несколько проблем этого подхода, которые описаны в следующих параграфах.

С практической точки зрения, в то время как версия 4.x использовала галочки для обозначения уровней, версия 5.x использует простое число во всех форматах стандарта, включая markdown, PDF, DOCX, CSV, JSON и XML. Для обратной совместимости также генерируются устаревшие версии выводов CSV, JSON и XML, которые все еще используют галочки.

Снижение порога входа

Обратная связь показала, что большое количество требований Уровня 1 (~120) в сочетании с его обозначением как «минимального» уровня, которого недостаточно для большинства приложений, препятствовало внедрению. Цель версии 5.0 —уменьшить этот барьер за счет определения требований Уровня 1 как первого эшелона защиты, благодаря чему они становятся более четкими и их количество сокращается. Чтобы продемонстрировать это численно, в v4.0.3 было 128 требований L1 из общего числа в 278 требований, что составляет 46%. В 5.0.0 насчитывается 70 требований L1 из общего числа в 345 требований, что составляет 20%.

Иллюзия проверяемости

Ключевым фактором при выборе средств контроля для Уровня 1 в версии 4.x была их пригодность для оценки с помощью внешнего тестирования на проникновение по методу «черного ящика». Однако этот подход не был полностью согласован с целью Уровня 1 как минимального набора средств обеспечения безопасности. Некоторые пользователи утверждали, что Уровня 1 недостаточно для защиты приложений, в то время как другие считали его слишком сложным для тестирования.

Опора на проверяемость как на критерий является относительной и, в ряде случаев, вводящей в заблуждение. Тот факт, что требование является проверяемым, не гарантирует, что его можно проверить автоматизированным или простым способом. Более того, наиболее легко проверяемые требования не всегда являются теми, которые оказывают наибольшее влияние на безопасность или которые проще всего реализовать.

Таким образом, в версии 5.0 решения об уровнях принимались в первую очередь с целью снижения рисков, а также с учетом необходимых усилий по реализации.

Не только оценка рисков

Использование предписывающих уровней, основанных на оценке риска, которые устанавливают определенный уровень для определенных приложений, оказалось слишком жестким. На практике расстановка приоритетов и внедрение средств безопасности зависят от множества факторов, включая как снижение рисков, так и усилия, необходимые для реализации.

Следовательно, организациям следует стремиться к достижению того уровня, который они считают необходимым, исходя из своей зрелости и ценностей, которые они хотят демонстрировать своим клиентам.

V1 Кодировка и нейтрализация

Задачи контроля

В этой главе рассматриваются наиболее распространенные уязвимости безопасности веб-приложений, связанные с небезопасной обработкой недоверенных (пользовательских) данных. Такие уязвимости могут привести к различным техническим уязвимостям, где такие данные интерпретируются в соответствии с правилами синтаксиса соответствующего интерпретатора.

Для современных веб-приложений всегда лучше использовать более архитектурно безопасные API, такие как параметризованные запросы, автоматическое экранирование или шаблонные фреймворки. В ином случае, самостоятельное тщательное выполнение кодирования выходных данных, экранирование или нейтрализация становятся критически важными для безопасности приложения.

Валидация входных данных служит механизмом многослойной/глубокоэшелонированной (defense-in-depth) защиты от неожидаемого или опасного контента. Однако, поскольку ее основная цель заключается в обеспечении соответствия входящих данных функциональным и бизнес-ожиданиям, требования, связанные с этим, можно найти в главе «Валидация и бизнес-логика».

V1.1 Архитектура кодировки и нейтрализации

В разделах ниже приведены требования для синтаксиса или интерпретатора, предназначенные для безопасной обработки небезопасного содержимого и предотвращения уязвимостей безопасности. Требования в этом разделе охватывают порядок, в котором должна происходить эта обработка, и где она должна происходить. Они также направлены на то, чтобы гарантировать, что при каждом сохранении данных они остаются в своем исходном состоянии и не хранятся в закодированной или экранированной форме (например, кодировка HTML), чтобы предотвратить проблемы с двойным кодированием.

#	Описание	Уровень
1.1.1	Убедитесь, что входные данные декодируются или деэкранируются в каноническую форму (в стандартное представление) только один раз. Декодирование происходит только тогда, когда ожидаются закодированные данные в этой форме, и что это делается перед дальнейшей обработкой входных данных, например, это не выполняется после проверки или нейтрализации.	2
1.1.2	Убедитесь, что приложение выполняет кодирование и экранирование выходных данных либо в качестве последнего шага перед использованием интерпретатором, для которого оно предназначено, либо самим интерпретатором.	2

V1.2 Предотвращение инъекций

Кодирование или экранирование выходных данных, выполняемое близко или смежно с потенциально опасным контекстом, критично для безопасности любого приложения. Обычно кодирование и экранирование выходных данных не сохраняется, а вместо этого используется для обеспечения безопасности выходных данных для немедленного использования в соответствующем интерпретаторе. Попытка выполнить это на более ранних этапах может привести к искажению содержимого или неэффективно отобразить кодирование или экранирование.

Во многих случаях библиотеки программного обеспечения включают более или полностью безопасные функции, которые выполняют это автоматически, хотя необходимо убедиться, что они корректны для текущего контекста.

#	Описание	Уровень
1.2.1	Убедитесь, что выходная кодировка для ответа HTTP, документа HTML или документа XML соответствует требуемому контексту, например, кодировка соответствующих символов для элементов HTML, атрибутов HTML, комментариев HTML, CSS или полей заголовка HTTP делается таким способом, чтобы избежать изменения структуры сообщения или документа.	1

#	Описание	Уровень
1.2.2	Убедитесь, что при динамическом построении URL-адресов недоверенные данные кодируются в соответствии с их контекстом (например, кодирование URL-адресов или кодирование base64url для query или path параметров или пути). Убедитесь, что разрешены только безопасные протоколы URL-адресов (например, запрещен javascript: или data:).	1
1.2.3	Убедитесь, что при динамическом построении содержимого JavaScript (включая JSON) используется кодирование или экранирование выходных данных, чтобы избежать изменения структуры сообщения или документа (чтобы избежать JavaScript и JSON-инъекций).	1
1.2.4	Убедитесь, что выборка данных (Redis, Memcached, чтение из файлов, обращение по API) или запросы к базе данных (например, SQL, HQL, NoSQL, Cypher) используют параметризованные запросы, ORM, Entity Frameworks или иным образом защищены от SQL-инъекций и других атак. Это также актуально при написании хранимых процедур.	1
1.2.5	Убедитесь, что приложение защищено от внедрения команд ОС и что вызовы операционной системы используют параметризованные запросы к ОС или используют контекстное кодирование вывода командной строки.	1
1.2.6	Убедитесь, что приложение защищено от LDAP-инъекций, или что реализованы специальные меры безопасности для их предотвращения.	2
1.2.7	Убедитесь, что приложение защищено от атак с использованием XPath-инъекций с помощью параметризации запросов или предварительно скомпилированных запросов.	2
1.2.8	Убедитесь, что процессоры LaTeX настроены безопасно (например, не используется флаг «–shell-escape») и используется разрешенный список команд для предотвращения LaTeX-инъекций.	2
1.2.9	Убедитесь, что приложение экранирует специальные символы в регулярных выражениях (обычно с помощью обратной косой черты), чтобы предотвратить их ошибочную интерпретацию как метасимволов.	2

#	Описание	Уровень
1.2.10	Убедитесь, что приложение защищено от внедрения формул CSV (CSV Injection). Приложение должно следовать правилам экранирования, определенным в RFC 4180, разделы 2.6 и 2.7, при экспорте содержимого CSV. Кроме того, при экспорте в CSV или другие форматы электронных таблиц (такие как XLS, XLSX или ODF) специальные символы (включая '=', '+', '-', '@', '\t'(табуляция), и '\0'(нулевой символ)) должны экранироваться одинарной кавычкой, если они появляются в качестве первого символа в значении поля.	3

Примечание: Использование параметризованных запросов или экранирование SQL не всегда достаточно. Части запроса, такие как имена таблиц и столбцов (включая имена столбцов «ORDER BY»), не могут быть экранированы. Включение экранированных пользовательских данных в эти поля приводит к сбоям запросов или SQL-инъекции.

V1.3 Нейтрализация

Идеальной защитой от использования недоверенных данных в небезопасном контексте является использование контекстно-зависимого кодирования или экранирования, которое сохраняет то же семантическое значение небезопасного контента, но делает его безопасным для использования в этом конкретном контексте, как обсуждалось более подробно в предыдущем разделе.

Если это невозможно, необходима нейтрализация, удаляющая потенциально опасные символы или данные. В некоторых случаях это может изменить семантическое значение ввода, но по соображениям безопасности альтернативы может не быть.

#	Описание	Уровень
1.3.1	Убедитесь, что все ненадежные входные данные HTML из редакторов WYSIWYG или аналогичных программ нейтрализуются с помощью известных и безопасных библиотек или функций фреймворка для санитизации HTML.	1
1.3.2	Убедитесь, что приложение избегает использования eval() или других динамических функций выполнения кода, таких как Spring Expression Language (SpEL). Если нет альтернативы, любой пользовательский ввод должен быть нейтрализован перед выполнением.	1

#	Описание	Уровень
1.3.3	Убедитесь, что данные, передаваемые в потенциально опасный контекст, предварительно нейтрализованы, чтобы обеспечить соблюдение мер безопасности, таких как разрешение только тех символов, которые безопасны для данного контекста, и обрезка слишком длинного ввода.	2
1.3.4	Убедитесь, что предоставленный пользователем файл .SVG проверен или нейтрализован, чтобы он содержал только теги и атрибуты (например, относящиеся к рисованию), которые безопасны для приложения, например, не содержат скриптов и foreignObject.	2
1.3.5	Убедитесь, что приложение нейтралует или отключает предоставленные пользователем скриптовые данные или выражения шаблонных языков, такие как таблицы стилей Markdown, CSS или XSL, BBCode или аналогичные.	2
1.3.6	Убедитесь, что приложение защищено от атак с подделкой запросов на стороне сервера (SSRF), проверяя ненадежные данные по разрешенному списку протоколов, доменов, путей и портов, а также нейтрализуя потенциально опасные символы перед использованием данных для вызова другой службы.	2
1.3.7	Убедитесь, что приложение защищено от атак с внедрением шаблонов (template injection), не позволяя создавать шаблоны на основе недоверенных входных данных. Если нет альтернативы, любые недоверенные входные данные, включаемые динамически во время создания шаблона, должны быть нейтрализованы или тщательно провалидированы.	2
1.3.8	Убедитесь, что приложение надлежащим образом нейтрализует недоверенные входные данные перед использованием в запросах Java Naming and Directory Interface (JNDI) и что JNDI настроен безопасно для предотвращения атак с использованием JNDI-инъекций.	2
1.3.9	Убедитесь, что приложение нейтрализует данные перед отправкой в Memcache, чтобы предотвратить атаки методом инъекций.	2
1.3.10	Убедитесь, что строки форматирования, которые могут быть использованы неожидаемым или вредоносным образом, нейтрализуются перед дальнейшим использованием.	2
1.3.11	Убедитесь, что приложение нейтрализует вводимые пользователем данные перед передачей в почтовые системы для защиты от SMTP или IMAP-инъекций.	2

#	Описание	Уровень
1.3.12	Убедитесь, что регулярные выражения не содержат элементов, вызывающих экспоненциальный возврат, и убедитесь в нейтрализации недоверенных входных данных для предотвращения атак ReDoS или Runaway Regex.	3

V1.4 Память, строки и неуправляемый код

Следующие требования касаются рисков, связанных с небезопасным использованием памяти, которые обычно применяются, когда приложение использует системный язык или неуправляемый код.

В некоторых случаях этого можно добиться, установив флаги компилятора, которые включают защиту от переполнения буфера и предупреждения, включая рандомизацию стека и предотвращение выполнения данных, и которые прерывают сборку, если обнаружены небезопасные операции с указателем, памятью, строкой формата, целым числом или строкой.

#	Описание	Уровень
1.4.1	Убедитесь, что для обнаружения или предотвращения ситуаций переполнения стека, буфера или кучи приложение учитывает размер доступной области памяти при строковых операциях, копировании в память, в арифметике указателей и т.п.	2
1.4.2	Убедитесь, что для предотвращения целочисленного переполнения применяется контроль знака, диапазона и типа допустимых входных данных.	2
1.4.3	Убедитесь, что динамически выделенная память и ресурсы освобождены, а ссылки или указатели на освобожденную память удалены или установлены в значение null, чтобы предотвратить появление висячих указателей и уязвимостей, связанных с использованием памяти после освобождения.	2

V1.5 Безопасная десериализация

Преобразование данных из сохраненного или переданного представления в фактические объекты приложения (десериализация) исторически было причиной различных уязвимостей внедрения кода. Важно выполнять этот процесс осторожно и безопасно, чтобы избежать подобных проблем.

В частности, некоторые методы десериализации были определены в документации по языку программирования или фреймворку как небезопасные и не могут безопасно применяться с недоверенными данными. Для каждого используемого механизма следует проводить тщательную проверку.

#	Описание	Уровень
1.5.1	Убедитесь, что приложение настраивает XML-парсеры на использование строгой конфигурации и что небезопасные функции, такие как разрешение внешних сущностей, отключены для предотвращения атак XML eXternal Entity (XXE).	1
1.5.2	Убедитесь, что десериализация недоверенных данных обеспечивает безопасную обработку ввода, например, используя список разрешенных типов объектов или ограничивая типы объектов, определяемые клиентом, для предотвращения атак десериализации. Механизмы десериализации, которые явно определены как небезопасные, не должны использоваться с недоверенным вводом.	2
1.5.3	Убедитесь, что различные парсеры, используемые в приложении для одного и того же типа данных (например, парсеры JSON, XML, URL), выполняют парсинг согласованным образом и используют один и тот же механизм кодирования символов, чтобы избежать таких проблем, как уязвимости взаимодействия JSON (JSON Interoperability) или различное поведение анализа URI или файлов, которые могут использоваться в атаках удаленного включения файлов (RFI) или подделки запросов на стороне сервера (SSRF).	3

Ссылки

Для дополнительной информации см. также:

- OWASP LDAP Injection Prevention Cheat Sheet
- OWASP Cross Site Scripting Prevention Cheat Sheet
- OWASP DOM Based Cross Site Scripting Prevention Cheat Sheet
- OWASP XML External Entity Prevention Cheat Sheet
- OWASP Web Security Testing Guide: Client-Side Testing
- OWASP Java Encoding Project
- DOMPurify - Client-side HTML Sanitization Library
- RFC4180 - Common Format and MIME Type for Comma-Separated Values (CSV) Files

Для более подробной информации о проблемах сериализации и парсинга см. также:

- OWASP Deserialization Cheat Sheet

- An Exploration of JSON Interoperability Vulnerabilities
- Orange Tsai - A New Era of SSRF Exploiting URL Parser In Trending Programming Languages

V2 Валидация и бизнес-логика

Задачи контроля

Цель этой главы —убедиться, что проверенное приложение соответствует следующим высокоуровневым целям:

- Входные данные, полученные приложением, соответствуют бизнес- или функциональным ожиданиям.
- Поток бизнес-логики последователен, выполняется по порядку и не может быть обойден.
- Бизнес-логика содержит ограничения и механизмы для обнаружения и предотвращения автоматизированных атак, таких как постоянные небольшие переводы средств или добавление миллиона друзей по одному за раз.
- Критически важные потоки бизнес-логики учитывают случаи злоупотреблений и действий злоумышленников и имеют защиту от подмены, подделки, раскрытия информации и атак с повышением привилегий.

V2.1 Документация валидации и бизнес-логики

Документация по валидации и бизнес-логике должна четко определять ограничения бизнес-логики, правила валидации и контекстную согласованность объединенных данных, чтобы было понятно, что необходимо реализовать в приложении.

#	Описание	Уровень
2.1.1	Убедитесь, что в документации приложения определены правила проверки входных данных на соответствие ожидаемой структуре. Это могут быть распространённые форматы данных, такие как номера кредитных карт, адреса электронной почты, номера телефонов, или внутренний формат данных.	1
2.1.2	Убедитесь, что в документации приложения указано, как проверять логическую и контекстную согласованность объединенных данных, например, проверять соответствие района и почтового индекса.	2
2.1.3	Убедитесь, что ожидания относительно ограничений и проверок бизнес-логики задокументированы как для уровня отдельного пользователя, так и для глобального уровня всего приложения.	2

V2.2 Валидация ввода

Эффективные средства контроля валидации входных данных обеспечивают соответствие бизнес- или функциональных ожиданий приложения относительно типа данных. Такой контроль валидации обеспечивает высокое качество данных и уменьшает поверхность атаки. Однако такой контроль не устраняет и не заменяет необходимость использования корректного кодирования, параметризации или санитизации данных при использовании данных в другом компоненте или для их представления для вывода.

В данном контексте «входные данные» могут поступать из самых разных источников, включая поля HTML-форм, запросы REST, параметры URL, поля заголовков HTTP, файлы cookie, файлы на диске, базы данных и внешние API.

Механизм бизнес-логики может проверять, что конкретное входное значение представляет собой число меньше 100. Функционал будет проверять и ожидать, что число ниже определенного порогового значения, поскольку это число определяет количество повторений конкретного цикла, а большое значение может привести к чрезмерной нагрузке и потенциальному отказу в обслуживании.

Хотя валидация по схеме явно не предписана, она может быть наиболее эффективным механизмом для полного охвата валидацией HTTP API или других интерфейсов, использующих JSON или XML.

Обратите внимание на следующие моменты при валидации по схеме:

- «Опубликованная версия» спецификации валидации по JSON-схеме считается готовой к использованию, но, грубо говоря, «нестабильной». При использовании валидации по JSON-схеме убедитесь в отсутствии расхождений с приведенными ниже требованиями.
- Любые используемые библиотеки валидации по JSON-схеме также следует отслеживать и при необходимости обновлять после формализации стандарта.
- Валидацию DTD не следует использовать, а оценку DTD фреймворка следует отключить, чтобы избежать проблем с атаками XXE на DTD.

#	Описание	Уровень
2.2.1	Убедитесь, что входные данные провалидированы для обеспечения соответствия бизнес- или функциональным ожиданиям. Валидация должна осуществляться либо по списку допустимых значений, шаблонов и диапазонов, либо путем сравнения входных данных с ожидаемой структурой и логическими ограничениями в соответствии с предопределенными правилами. Для L1 валидация может быть сосредоточена на входных данных, используемых для принятия конкретных бизнес-решений или решений, связанных с безопасностью. Для L2 и выше это должно применяться ко всем входным данным.	1
2.2.2	Убедитесь, что приложение разработано с принудительной валидацией входных данных на уровне доверенного сервиса. Хотя проверка на стороне клиента повышает удобство использования и должна поощряться, на неё не следует полагаться как на средство контроля безопасности.	1
2.2.3	Убедитесь, что приложение обеспечивает обоснованность комбинаций данных в соответствии с предопределенными правилам. Например: возраст должен соответствовать дате рождения; страна и почтовый индекс; количество товара в заказе и на складе.	2

V2.3 Безопасность бизнес-логики

В этом разделе рассматриваются ключевые требования, гарантирующие, что приложение обеспечит корректное выполнение процессов бизнес-логики и не будет уязвимо для атак, использующих логику и поток работы приложения.

#	Описание	Уровень
2.3.1	Убедитесь, что приложение обрабатывает потоки бизнес-логики для одного и того же пользователя в ожидаемом последовательном порядке шагов и без пропуска шагов.	1
2.3.2	Убедитесь, что ограничения бизнес-логики реализованы в соответствии с документацией приложения, чтобы избежать эксплуатации её недостатков.	2

#	Описание	Уровень
2.3.3	Убедитесь, что транзакции используются на уровне бизнес-логики таким образом, что операция бизнес-логики либо выполняется полностью, либо откатывается к предыдущему корректному состоянию.	2
2.3.4	Убедитесь, что на уровне бизнес-логики используются механизмы блокировки, чтобы гарантировать невозможность двойного бронирования ограниченных по количеству ресурсов (например, мест в театре или слотов доставки) путем манипулирования логикой приложения.	2
2.3.5	Убедитесь, что для важных потоков бизнес-логики требуется многопользовательское (multi-user) одобрение, чтобы предотвратить несанкционированные или случайные действия. Потоки бизнес-логики могут включать, помимо прочего, крупные денежные переводы, утверждение контрактов, доступ к секретной информации или обход ограничений безопасности на производстве.	3

V2.4 Анти-автоматизация

В этом разделе представлены меры по борьбе с автоматизацией, гарантирующие необходимость взаимодействия, аналогичного человеческому, и предотвращают чрезмерные автоматизированные запросы.

#	Описание	Уровень
2.4.1	Убедитесь, что используются средства противодействия автоматизации для защиты от чрезмерных вызовов функций приложения, которые могут привести к утечке данных, созданию ненужных данных, исчерпанию квот, нарушению ограничений скорости, отказу в обслуживании или чрезмерному использованию дорогостоящих ресурсов.	2
2.4.2	Убедитесь, что потоки бизнес-логики требуют реалистичного человеческого времени, предотвращая чрезмерно быструю отправку транзакций.	3

Ссылки

Для дополнительной информации см. также:

- OWASP Web Security Testing Guide: Input Validation Testing
- OWASP Web Security Testing Guide: Business Logic Testing
- Анти-автоматизация может быть достигнута разными путями, включая применение OWASP Automated Threats to Web Applications
- OWASP Input Validation Cheat Sheet
- JSON Schema

V3 Безопасность веб-интерфейса

Задачи контроля

Эта категория фокусируется на требованиях, направленных на защиту от атак, воспроизводимых через веб-интерфейс. Эти требования не применяются к решениям «машина-машина».

V3.1 Документация по безопасности веб-интерфейса

В этом разделе описываются функции безопасности браузера, которые должны быть описаны в документации приложения.

#	Описание	Уровень
3.1.1	Убедитесь, что в документации приложения описаны ожидаемые функции безопасности, которые должны поддерживать браузеры, использующие приложение (такие как HTTPS, HTTP Strict Transport Security (HSTS), Content Security Policy (CSP) и другие соответствующие механизмы безопасности HTTP). В документации также должно быть обязательно определено, как приложение должно себя вести, если некоторые из этих функций недоступны (например, предупреждать пользователя или блокировать доступ).	3

V3.2 Непреднамеренная интерпретация контента

Отображение контента или функциональности в неправильном контексте может привести к выполнению или отображению вредоносного контента.

#	Описание	Уровень
3.2.1	Убедитесь, что реализованы меры безопасности, предотвращающие отображение контента или функций браузерами в HTTP-ответах в некорректном контексте (например, запрос API, загруженный пользователем файл или другой ресурс запрашивается напрямую). Возможные способы контроля включают: не обрабатывать содержимое, если поля заголовка HTTP-запроса (например, Sec-Fetch-*) не указывают на корректность контекста; использование заголовка и значения Content-Security-Policy: sandbox directive; или использование заголовка и значения Content-Disposition: attachment.	1
3.2.2	Убедитесь, что данные, предназначенные для отображения в виде текста, а не визуализации в виде HTML, обрабатывается с использованием безопасных функций визуализации (таких как createTextNode или textContent), чтобы предотвратить непреднамеренное выполнение содержимого, такого как HTML или JavaScript.	1
3.2.3	Убедитесь, что приложение избегает затирания DOM при использовании JavaScript на стороне клиента, применяя явные объявления переменных, выполняя строгую проверку типов, избегая сохранения глобальных переменных в объекте документа и реализуя изоляцию пространства имен.	3

V3.3 Настройка файлов cookie

В этом разделе изложены требования к безопасной настройке чувствительных cookie, позволяющие обеспечить более высокий уровень уверенности в том, что они были созданы самим приложением, а также предотвратить утечку или ненадлежащее изменение их содержимого.

#	Описание	Уровень
3.3.1	Убедитесь, что у файлов cookie установлен атрибут «Secure», и если префикс «__Host-» не используется в названии cookie, необходимо использовать префикс «__Secure-».	1

#	Описание	Уровень
3.3.2	Убедитесь, что значение атрибута SameSite каждой cookie установлено в соответствии с назначением cookie, чтобы ограничить подверженность атакам на восстановление пользовательского интерфейса и атакам с подделкой запросов на уровне браузера, обычно называемым подделкой межсайтовых запросов (CSRF).	2
3.3.3	Убедитесь, что cookie имеют префикс «__Host-» в названии, если только они явно не предназначены для совместного использования с другими хостами.	2
3.3.4	Убедитесь, что если значение cookie не должно быть доступно для клиентских скриптов (например, сессионный токен), для cookie должен быть установлен атрибут «HttpOnly», а само значение должно передаваться клиенту только через поле заголовка «Set-Cookie».	2
3.3.5	Убедитесь, что при записи cookie приложением общая длина имени и значения cookie не превышает 4096 байт. Слишком большие cookie не будут сохраняться браузером и, следовательно, не будут отправляться с запросами, что помешает пользователю использовать функции приложения, зависящие от этого cookie.	3

V3.4 Заголовки механизма безопасности браузера

В этом разделе описывается, какие заголовки безопасности следует устанавливать в HTTP-ответах, чтобы включить функции безопасности браузера и ограничения при обработке ответов от приложения.

#	Описание	Уровень
3.4.1	Убедитесь, что поле заголовка Strict-Transport-Security включено во все ответы для реализации политики HTTP Strict Transport Security (HSTS). Необходимо указать максимальное время не менее 1 года, а для L2 и выше политика должна применяться также ко всем поддоменам.	1
3.4.2	Убедитесь, что поле заголовка Cross-Origin Resource Sharing (CORS) «Access-Control-Allow-Origin» имеет фиксированное значение, заданное приложением, или, если используется заголовок Origin, оно проверяется по списку доверенных источников. При необходимости использования «Access-Control-Allow-Origin: *» убедитесь, что ответ не содержит конфиденциальной информации.	1

#	Описание	Уровень
3.4.3	Убедитесь, что HTTP-ответы содержат поле заголовка Content-Security-Policy, которое определяет директивы, гарантирующие загрузку и выполнение браузером только доверенного контента или ресурсов, чтобы ограничить выполнение вредоносного JavaScript. Как минимум, необходимо использовать глобальную политику, включающую директивы object-src «none» и base-uri «none», а также определяющую список разрешенных адресов или использующую одноразовые коды или хэши. Для приложений L3 необходимо определить политику использования ‘nonce’или hash-конструкций для каждого ответа.	2
3.4.4	Убедитесь, что все HTTP-ответы содержат поле заголовка «X-Content-Type-Options: nosniff». Это предписывает браузерам не использовать анализ контента и определение типа MIME для данного ответа, а также требовать, чтобы значение поля заголовка Content-Type в ответе соответствовало целевому ресурсу. Например, ответ на запрос стиля принимается только в том случае, если Content-Type в ответе — «text/css». Это также позволяет браузеру использовать функцию Cross-Origin Read Blocking (CORB).	2
3.4.5	Убедитесь, что приложение устанавливает политику реферера для предотвращения утечки технически конфиденциальных данных сторонним сервисам через поле заголовка HTTP-запроса «Referer». Это можно сделать с помощью поля заголовка HTTP-ответа «Referrer-Policy» или через атрибуты HTML-элемента. Конфиденциальные данные могут включать путь и данные запроса в URL-адресе, а для внутренних непубличных приложений также имя хоста.	2
3.4.6	Убедитесь, что веб-приложение использует директиву frame-ancestors поля заголовка Content-Security-Policy для каждого HTTP-ответа, чтобы гарантировать невозможность его внедрения по умолчанию и внедрение определённых ресурсов разрешено только при необходимости. Обратите внимание, что поле заголовка X-Frame-Options, хотя и поддерживается браузерами, устарело и не может быть использовано в качестве основы.	2
3.4.7	Убедитесь, что в поле заголовка Content-Security-Policy указано место, куда следует сообщать о нарушениях.	3

#	Описание	Уровень
3.4.8	Убедитесь, что все HTTP-ответы, инициирующие рендеринг документа (например, ответы с типом содержимого text/html), включают поле заголовка Cross-Origin-Opener-Policy с директивой same-origin или same-origin-allow-popups (при необходимости). Это предотвращает атаки, злоупотребляющие общим доступом к объектам Window, такие как Tabnabbing и Frame Counting.	3

V3.5 Браузерное разделение источников

Принимая запрос к чувствительному функционалу на стороне сервера, приложение должно убедиться, что запрос инициирован самим приложением или доверенной стороной и не был подделан злоумышленником.

К чувствительному функционалу в данном контексте могут относиться принятие форм POST для аутентифицированных и неаутентифицированных пользователей (например, запрос аутентификации), операции по изменению состояния или ресурсоемкие функции (например, экспорт данных).

Ключевыми средствами защиты здесь являются политики безопасности браузера, такие как Same Origin Policy для JavaScript и логика SameSite для cookie. Другим распространённым средством защиты является механизм предварительной проверки CORS. Этот механизм критически важен для конечных точек, спроектированных и предназначенных, чтобы вызываться из другого источника, но он также может быть полезным механизмом предотвращения подделки запросов для конечных точек, которые не предназначены для вызова из другого источника.

#	Описание	Уровень
3.5.1	Убедитесь, что, если приложение не использует механизм предварительной проверки CORS для предотвращения запрещённых кросс-доменных запросов на использование чувствительного функционала, эти запросы валидируются на предмет их происхождения от самого приложения. Это можно сделать, используя и проверяя специальные токены для защиты от подделки или требуя дополнительные поля HTTP-заголовка, не входящие в список «простых заголовков» (CORS-safelisted request-header). Это необходимо для защиты от атак с подделкой запросов через браузер, широко известных как межсайтовая подделка запросов (CSRF).	1

#	Описание	Уровень
3.5.2	Убедитесь, что если приложение использует механизм предварительной проверки CORS для предотвращения неразрешённого использования чувствительного функционала между источниками, вызов этого функционала невозможен с помощью запроса, который не запускает предварительную проверку CORS. Для этого может потребоваться проверка значений полей заголовка запроса «Origin» и «Content-Type» или использование дополнительного поля заголовка, не входящего в безопасный список CORS.	1
3.5.3	Убедитесь, что HTTP-запросы к чувствительному функционалу используют соответствующие HTTP-методы, такие как POST, PUT, PATCH или DELETE, а не методы, определённые спецификацией HTTP как «безопасные», такие как HEAD, OPTIONS или GET. В качестве альтернативы можно использовать строгую валидацию полей заголовка запроса Sec-Fetch-*, чтобы убедиться, что запрос не исходит из недопустимого кросс-доменного вызова, навигационного запроса или загрузки ресурса (например, источника изображения), где это не ожидается.	1
3.5.4	Убедитесь, что отдельные приложения размещены на разных именах хостов, чтобы использовать ограничения, предусмотренные политикой same-origin, включая то, как документы или скрипты, загруженные одним источником, могут взаимодействовать с ресурсами из другого источника, а также ограничения на cookie на основе имени хоста.	2
3.5.5	Убедитесь, что сообщения, полученные интерфейсом postMessage, отвергаются, если источник сообщения не является доверенным или если синтаксис сообщения недействителен.	2
3.5.6	Убедитесь, что функциональность JSONP не включена нигде в приложении, чтобы избежать атак с использованием межсайтового включения скриптов (XSSI).	3
3.5.7	Убедитесь, что данные, требующие авторизации, не включены в ответы ресурсов скриптов, таких как файлы JavaScript, чтобы предотвратить атаки с использованием межсайтового включения скриптов (XSSI).	3

#	Описание	Уровень
3.5.8	Убедитесь, что аутентифицированные ресурсы (такие как изображения, видео, скрипты и другие документы) могут быть загружены или внедрены от имени пользователя только по назначению. Этого можно добиться путём строгой валидации полей заголовка HTTP-запроса Sec-Fetch-*, чтобы убедиться, что запрос не исходит из недопустимого кросс-доменного вызова, или путём установки ограничительного поля заголовка HTTP-ответа Cross-Origin-Resource-Policy, чтобы браузер блокировал возвращаемый контент.	3

V3.6 Целостность внешних ресурсов

В этом разделе приведены рекомендации по безопасному размещению контента на сторонних сайтах.

#	Описание	Уровень
3.6.1	Убедитесь, что клиентские ресурсы, такие как библиотеки JavaScript, CSS или веб-шрифты, размещаются только на внешнем сервере (например, в сети доставки контента (Content Delivery Network)), если ресурс статический и имеет версии, а для проверки целостности ресурса используется механизм проверки целостности подресурсов (SRI). Если это невозможно, для каждого ресурса должно быть задокументировано решение по безопасности, обосновывающее это.	3

V3.7 Другие соображения по безопасности браузера

В этом разделе рассматриваются различные другие элементы управления безопасностью и современные функции безопасности браузера, необходимые для обеспечения безопасности браузера на стороне клиента.

#	Описание	Уровень
3.7.1	Убедитесь, что приложение использует только те клиентские технологии, которые по-прежнему поддерживаются и считаются безопасными. Примерами технологий, не соответствующих этому требованию, являются плагины NSAPI, Flash, Shockwave, ActiveX, Silverlight, NACL или клиентские Java-апплеты.	2
3.7.2	Убедитесь, что приложение будет автоматически перенаправлять пользователя на другое имя хоста или домен (который не контролируется приложением), только если этот хост или домен указаны в списке разрешенных.	2
3.7.3	Убедитесь, что приложение отображает уведомление, что пользователь перенаправляется на URL-адрес, находящийся вне контроля приложения, с возможностью отмены навигации.	3
3.7.4	Убедитесь, что домен верхнего уровня приложения (например, site.tld) добавлен в публичный список предварительной загрузки (preload list) для HTTP Strict Transport Security (HSTS). Это гарантирует, что использование TLS для приложения будет встроено непосредственно в основные браузеры, а не будет полагаться только на поле заголовка ответа Strict-Transport-Security.	3
3.7.5	Убедитесь, что приложение ведет себя так, как описано в документации (например, предупреждает пользователя или блокирует доступ), если браузер, используемый для доступа к приложению, не поддерживает ожидаемые функции безопасности.	3

Ссылки

Для дополнительной информации см. также:

- Set-Cookie __Host- prefix details
- OWASP Content Security Policy Cheat Sheet
- OWASP Secure Headers Project
- OWASP Cross-Site Request Forgery Prevention Cheat Sheet
- HSTS Browser Preload List submission form
- OWASP DOM Clobbering Prevention Cheat Sheet

V4 API и Веб-сервис

Задачи контроля

Существует несколько моментов, применимых только к приложениям, предоставляющим API для использования веб-браузерами или другими клиентами (обычно с использованием JSON, XML или GraphQL). В этой главе рассматриваются соответствующие настройки и механизмы безопасности, которые следует применять.

Обратите внимание, что вопросы аутентификации, управления сессиями и проверки входных данных, рассмотренные в других главах, также относятся к API. Поэтому эту главу нельзя рассматривать изолированно или тестировать отдельно.

V4.1 Общая безопасность веб-сервисов

В этом разделе рассматриваются общие вопросы безопасности веб-сервисов и базовые практики гигиены веб-сервисов.

#	Описание	Уровень
4.1.1	Убедитесь, что каждый HTTP-ответ с телом сообщения содержит заголовок Content-Type, который соответствует фактическому содержимому ответа, таким как «text/», «/+xml» и «/xml», включая параметр charset для указания безопасной кодировки символов (например, UTF-8, ISO-8859-1) согласно IANA Media Types.	1
4.1.2	Убедитесь, что только конечные точки, предназначенные для взаимодействия с пользователем через веб-браузер, автоматически перенаправляют HTTP на HTTPS, в то время как другие сервисы или конечные точки не реализуют прозрачные редиректы. Это необходимо, чтобы избежать ситуации, когда клиент ошибочно отправляет незашифрованные HTTP-запросы а из-за того, что запрос по HTTP автоматически перенаправляется на HTTPS другим сервисом без обработки на входе, утечка чувствительных данных остаётся незамеченной.	2
4.1.3	Убедитесь, что любой HTTP-заголовок, используемый приложением и устанавливаемый промежуточным уровнем, таким как балансировщик нагрузки, веб-прокси или backend-for-frontend сервис, не может быть переопределён конечным пользователем. Примеры заголовков: X-Real-IP, X-Forwarded-*, X-User-ID.	2

#	Описание	Уровень
4.1.4	Убедитесь, что разрешены только HTTP-методы, которые явно поддерживаются приложением или его API (включая OPTIONS для preflight-запросов), а неиспользуемые методы заблокированы.	3
4.1.5	Убедитесь, что для особо чувствительных запросов или транзакций, проходящих через несколько систем, используются цифровые подписи на уровне сообщений, дополнительно к транспортным средствам защиты.	3

V4.2 Валидация структуры HTTP-сообщений

Этот раздел объясняет, как должна проверяться структура и заголовки HTTP-сообщений, чтобы предотвратить атаки типа request smuggling, response splitting, header injection и отказ в обслуживании из-за слишком длинных HTTP-сообщений.

Эти требования актуальны для обработки и генерации HTTP-сообщений в целом, но особенно важны при конвертации сообщений между разными версиями HTTP.

#	Описание	Уровень
4.2.1	Убедитесь, что все компоненты приложения (включая балансировщики нагрузки, файрволы и серверы приложений) определяют границы входящих HTTP-сообщений с использованием механизма, соответствующего версии HTTP, чтобы предотвратить HTTP request smuggling. В HTTP/1.x, если присутствует заголовок Transfer-Encoding, заголовок Content-Length должен игнорироваться согласно RFC 2616. При использовании HTTP/2 или HTTP/3, если присутствует Content-Length, веб-сервер должен убедиться, что он совпадает с длиной DATA-фреймов.	2
4.2.2	Убедитесь, что при генерации HTTP-сообщений заголовок Content-Length не конфликтует с длиной содержимого, определяемой рамками протокола HTTP, чтобы предотвратить HTTP Request Smuggling.	3
4.2.3	Убедитесь, что приложение не отправляет и не принимает HTTP/2 и HTTP/3 сообщения с заголовками, специфичными для соединения, такими как Transfer-Encoding, чтобы предотвратить HTTP Response Splitting и HTTP Header Injection.	3

#	Описание	Уровень
4.2.4	Убедитесь, что приложение принимает HTTP/2 и HTTP/3 запросы только в случае, если поля заголовков и их значения не содержат последовательностей CR (\r), LF (\n) или CRLF (\r\n), чтобы предотвратить HTTP Header Injection.	3
4.2.5	Убедитесь, что если приложение (бекенд или фронтенд) формирует и отправляет запросы, оно использует валидацию, очистку или другие механизмы, чтобы не создавать URI (например, для API-вызовов) или HTTP-заголовки (например, Authorization или Cookie) слишком большой длины, чтобы их не принимал приёмник. Это может привести к отказу в обслуживании, например, при отправке слишком длинного запроса (длинный cookie-заголовок), из-за чего сервер постоянно возвращает ошибку.	3

V4.3 GraphQL

GraphQL становится всё более популярным способом создания клиентов с большим объёмом данных, которые не жёстко связаны с разными бэкенд-сервисами. В этом разделе рассмотрены вопросы безопасности для GraphQL.

#	Описание	Уровень
4.3.1	Убедитесь, что используется белый список запросов, ограничение глубины вложенности, ограничение количества или анализ стоимости запроса, чтобы предотвратить DoS-атаки на уровне GraphQL или слоя данных из-за дорогостоящих вложенных запросов.	2
4.3.2	Убедитесь, что introspection-запросы GraphQL отключены в продакшен-среде, если GraphQL API не предназначен для использования сторонними клиентами.	2

V4.4 WebSocket

WebSocket —это протокол связи, обеспечивающий двунаправленную одновременную коммуникацию по одному TCP-соединению. Он был стандартизирован IETF в RFC 6455 в 2011 году и отличается от HTTP, хотя и работает поверх портов HTTP 443 и 80.

В этом разделе приведены основные требования по безопасности, чтобы предотвратить атаки, связанные с безопасностью коммуникаций и управлением сессиями, которые эксплуатируют этот канал реального времени.

#	Описание	Уровень
4.4.1	Убедитесь, что для всех WebSocket-соединений используется WebSocket поверх TLS (WSS).	1
4.4.2	Убедитесь, что при первоначальном HTTP WebSocket handshake заголовок Origin проверяется на соответствие списку разрешённых источников для приложения.	2
4.4.3	Убедитесь, что если стандартное управление сессиями приложения не может быть использовано, применяются специальные токены, соответствующие требованиям безопасности управления сессиями.	2
4.4.4	Убедитесь, что специальные токены управления сессиями WebSocket изначально получаются или валидируются через ранее аутентифицированную HTTPS-сессию при переходе от HTTPS к WebSocket.	2

Ссылки

Для дополнительной информации см. также:

- OWASP REST Security Cheat Sheet
- Ресурсы по авторизации GraphQL от graphql.org и Apollo.
- OWASP Web Security Testing Guide: GraphQL Testing
- OWASP Web Security Testing Guide: Testing WebSockets

V5 Работа с файлами

Задачи контроля

Использование файлов может нести различные риски для приложения, включая отказ в обслуживании (DoS), несанкционированный доступ и переполнение хранилища. В этой главе приведены требования, для предотвращения этих рисков.

V5.1 Документация по работе с файлами

Этот раздел содержит требование документировать ожидаемые характеристики файлов, принимаемых приложением, как необходимое условие для разработки и проверки соответствующих мер безопасности.

#	Описание	Уровень
5.1.1	Убедитесь, что документация определяет допустимые типы файлов, ожидаемые расширения и максимальный размер (включая распакованный размер) для каждой функции загрузки. Также убедитесь, что в документации указано, как файлы обезвреживаются для конечных пользователей (например, как приложение ведёт себя при обнаружении вредоносного файла).	2

V5.2 Загрузка файлов и содержимое

Функциональность загрузки файлов является основным источником недоверенных файлов. Этот раздел описывает требования, обеспечивающие безопасность приложения от вреда, связанного с наличием, объёмом или содержимым этих файлов.

#	Описание	Уровень
5.2.1	Убедитесь, что приложение принимает только те файлы, которые может обработать без потери производительности или отказа в обслуживании.	1
5.2.2	Убедитесь, что при загрузке файла (включая файлы внутри архивов, таких как ZIP) приложение проверяет, соответствует ли расширение ожидаемому, и валидирует, соответствует ли содержимое типу, указанному в расширении. Это включает проверку начальных ‘magic bytes’, перезапись изображений, использование специализированных библиотек. Для уровня 1 —достаточно проверок для файлов, критичных с точки зрения бизнеса или безопасности. Для уровней 2 и выше —для всех файлов.	1
5.2.3	Убедитесь, что приложение проверяет архивы (например, zip, gz, docx, odt) на максимально допустимый распакованный размер и максимальное количество файлов до распаковки.	2
5.2.4	Убедитесь, что для каждого пользователя установлены квоты на общий размер и количество файлов, чтобы один пользователь не мог переполнить хранилище.	3
5.2.5	Убедитесь, что приложение не позволяет загружать архивы с символьными ссылками (symlinks), если это не требуется по бизнес-логике. Если требуется —разрешён только строго определённый список допустимых символьных ссылок.	3

#	Описание	Уровень
5.2.6	Убедитесь, что приложение отклоняет изображения с числом пикселей выше допустимого, чтобы предотвратить атаки ‘pixel flood’.	3

V5.3 Хранение файлов

Этот раздел содержит требования для предотвращения исполнения файлов после загрузки, обнаружения опасного содержимого и предотвращения использования недоверенных данных для управления путями хранения файлов.

#	Описание	Уровень
5.3.1	Убедитесь, что загруженные или сгенерированные из недоверенного ввода файлы, находящиеся в публичной директории, не исполняются как серверный код при обращении к ним по HTTP.	1
5.3.2	Убедитесь, что при создании путей к файлам приложение использует внутренние или доверенные данные, а не пользовательские имена файлов. Если использование пользовательских имён или метаданных необходимо —требуется строгая проверка и очистка. Это защита от path traversal, LFI, RFI и SSRF.	1
5.3.3	Убедитесь, что при обработке файлов на сервере (например, при распаковке) игнорируются пути, заданные пользователем, во избежание уязвимостей типа ‘zip slip’.	3

V5.4 Скачивание файлов

Этот раздел содержит требования для снижения рисков при выдаче файлов на загрузку, включая защиту от path traversal и инъекций, а также проверку содержимого.

#	Описание	Уровень
5.4.1	Убедитесь, что приложение валидирует или игнорирует имена файлов, переданных пользователем (включая в JSON, JSONP или URL-параметре), и указывает имя файла в заголовке Content-Disposition при ответе.	2

#	Описание	Уровень
5.4.2	Убедитесь, что имена файлов, передаваемые (например, в HTTP-заголовках или вложениях в email), кодируются или очищаются (например, по RFC 6266), чтобы сохранить структуру документа и предотвратить инъекции.	2
5.4.3	Убедитесь, что файлы, полученные из недоверенных источников, сканируются антивирусом, чтобы не распространять известное вредоносное содержимое.	2

Ссылки

Для дополнительной информации см. также:

- OWASP File Upload Cheat Sheet
- Example of using symlinks for arbitrary file read
- Explanation of “Magic Bytes”from Wikipedia

V6 Аутентификация

Задачи контроля

Аутентификация —это процесс установления или подтверждения подлинности личности или устройства. Он включает в себя проверку заявленных данных, устойчивость к подмене личности и защиту от перехвата или восстановления паролей.

Документ NIST SP 800-63 является современным и основанным на фактических данных стандартом, особенно актуальным для государственных структур США и организаций, взаимодействующих с ними.

Хотя многие требования этой главы основаны на разделе NIST SP 800-63B («Руководство по цифровой идентификации —Управление аутентификацией и жизненным циклом»), акцент сделан на распространённых угрозах и часто эксплуатируемых слабостях. Эта глава не охватывает весь стандарт. Для полной реализации NIST SP 800-63 необходимо обратиться к оригиналу.

В тексте используется более понятная терминология, отличающаяся от формулировок NIST.

Многие современные приложения адаптируют этапы аутентификации в зависимости от уровня риска. Эти механизмы рассматриваются в главе «Авторизация», поскольку они также касаются авторизационных решений.

V6.1 Документация по аутентификации

Этот раздел содержит требования к документации по аутентификации, которую необходимо вести для приложения. Это критически важно для того, чтобы правильно настроить и оценить соответствующие механизмы управления аутентификацией.

#	Описание	Уровень
6.1.1	Убедитесь, что в документации приложения описано, как используются механизмы, такие как ограничение частоты запросов (rate limiting), защита от автоматизации и адаптивные реакции, для защиты от атак, таких как подстановка учётных данных и перебор паролей. Документация должна ясно описывать конфигурацию этих механизмов и предотвращение злонамеренной блокировки аккаунтов.	1
6.1.2	Убедитесь, что задокументирован список контекстно-зависимых слов, запрещённых к использованию в паролях. В него могут входить производные от названий организаций, продуктов, идентификаторов систем, кодовых названий проектов, названий отделов или ролей и тому подобное.	2
6.1.3	Убедитесь, что если приложение поддерживает несколько способов аутентификации, то все они документированы вместе с описанием применяемых средств защиты и уровнем стойкости аутентификации, которые должны быть одинаково применены ко всем способам.	2

V6.2 Безопасность паролей

Пароли (в терминологии NIST SP 800-63 —«запоминаемые секреты») включают в себя пароли, кодовые фразы, PIN-коды, графические шаблоны разблокировки и, например, выбор нужного котёнка или другого элемента на изображении. Они относятся к категории «что-то, что вам известно» и часто используются как однофакторный механизм аутентификации.

Соответственно, в этом разделе перечислены требования, направленные на безопасное создание и обработку паролей. Большинство требований относятся к уровню L1, поскольку на этом уровне они наиболее важны. Начиная с уровня L2, требуется использование многофакторной аутентификации, в которой пароли могут быть одним из факторов.

Требования в данном разделе в основном соответствуют § 5.1.1.2 руководства NIST.

#	Описание	Уровень
6.2.1	Убедитесь, что пользовательские пароли содержат не менее 8 символов. Рекомендуется минимальная длина в 15 символов.	1
6.2.2	Убедитесь, что пользователи могут менять свои пароли.	1
6.2.3	Убедитесь, что для смены пароля требуется ввести как текущий, так и новый пароль.	1
6.2.4	Убедитесь, что при регистрации или смене пароля введённый пароль проверяется по списку как минимум из 3000 самых популярных паролей, соответствующих политике (например, минимальной длине).	1
6.2.5	Убедитесь, что допускаются пароли любой структуры, без ограничений на использование определённых символов. Не должно быть требований по обязательному наличию заглавных/строчных букв, цифр или специальных символов.	1
6.2.6	Убедитесь, что поля ввода пароля используют type=password для скрытия символов. Разрешено временное отображение всего пароля или последнего введённого символа.	1
6.2.7	Убедитесь, что разрешены вставка (paste), использование менеджеров паролей браузера и внешних менеджеров паролей.	1
6.2.8	Убедитесь, что пароль проверяется строго в том виде, в котором он был введён пользователем —без обрезки, изменения регистра и других модификаций.	1
6.2.9	Убедитесь, что система позволяет использовать пароли длиной не менее 64 символов.	2
6.2.10	Убедитесь, что пароль остаётся действительным, пока не будет скомпрометирован или не будет изменён пользователем. Приложение не должно требовать регулярной смены пароля.	2
6.2.11	Убедитесь, что используется документированный список контекстно-зависимых слов (например, названия компаний, проектов и т.д.) для предотвращения использования легко угадываемых паролей.	2
6.2.12	Убедитесь, что при регистрации или смене пароля проверяется, не входит ли он в список скомпрометированных (утёкших) паролей.	2

V6.3 Общие требования к безопасности аутентификации

В этом разделе изложены общие требования к безопасности механизмов аутентификации, а также различные ожидания для разных уровней защиты. Для приложений уровня L2 обязательно использование многофакторной аутентификации (MFA). Для приложений уровня L3 требуется аппаратная аутентификация, выполняемая в аттестованной и доверенной среде исполнения (TEE). Это может включать: ключи passkey, привязанные к устройству; аутентификаторы, соответствующие eIDAS LoA High; аутентификаторы с уровнем доверия NIST AAL3; или эквивалентные механизмы.

Это достаточно строгий подход к MFA, но он необходим для повышения уровня защиты пользователей. Любое смягчение этих требований должно сопровождаться документированным обоснованием и чётким планом по смягчению рисков с учётом рекомендаций и исследований NIST.

Обратите внимание: на момент публикации NIST SP 800-63 рассматривает email как недопустимый механизм аутентификации (archived copy).

Требования в этом разделе опираются на следующие части руководства NIST, including: § 4.2.1, § 4.3.1, § 5.2.2, и § 6.1.2.

#	Описание	Уровень
6.3.1	Убедитесь, что реализованы меры защиты от атак типа credential stuffing (массовая подстановка учётных данных) и brute-force (перебор паролей), согласно документации по безопасности приложения.	1
6.3.2	Убедитесь, что в приложении отсутствуют или отключены учётные записи по умолчанию (например, root, admin, sa).	1
6.3.3	Убедитесь, что для доступа к приложению используется либо механизм многофакторной аутентификации, либо комбинация однофакторных. Для уровня L3 один из факторов должен быть аппаратным, обеспечивающим защиту от фишинга, устойчивость к компрометации и подтверждение намерения аутентификации через явное действие пользователя (например, нажатие кнопки на FIDO-ключе или телефоне). Ослабление этих требований допускается только при наличии обоснования и компенсирующих мер.	2
6.3.4	Убедитесь, что при наличии нескольких способов аутентификации, они все задокументированы, и для них единообразно применяются меры защиты и уровни доверия.	2

#	Описание	Уровень
6.3.5	Убедитесь, что пользователи получают уведомления о подозрительных попытках аутентификации (удачных и неудачных). Это может включать попытки (например, подключения) с необычного устройства или места, частично успешную MFA, попытку после долгого отсутствия или успешную аутентификацию после серии неудачных.	3
6.3.6	Убедитесь, что email не используется ни как однофакторный, ни как многофакторный механизм аутентификации.	3
6.3.7	Убедитесь, что пользователи получают уведомление при изменении данных аутентификации, таких как сброс пароля или изменение логина/email.	3
6.3.8	Убедитесь, что из сообщений об ошибке, HTTP-кодов ответа или различий во времени ответа нельзя определить, существует ли пользователь с таким логином. Это правило также должно применяться к функциям регистрации и восстановления пароля.	3

V6.4 Жизненный цикл и восстановление факторов аутентификации

Факторы аутентификации могут включать пароли, программные токены (soft tokens), аппаратные токены (hardware tokens) и биометрические устройства. Безопасное управление жизненным циклом этих механизмов критически важно для безопасности приложения, и данный раздел включает соответствующие требования.

Требования в этом разделе в основном соответствуют разделам § 5.1.1.2 или § 6.1.2.3 руководства NIST.

#	Описание	Уровень
6.4.1	Убедитесь, что начальные пароли или коды активации, генерируемые системой, создаются с использованием надёжного генератора случайных чисел, соответствуют текущей политике паролей и истекают через короткий период времени либо после первого использования. Эти секреты не должны использоваться как постоянные пароли.	1
6.4.2	Убедитесь, что в системе отсутствуют подсказки к паролю или вопросы на знание («секретные вопросы»).	1

#	Описание	Уровень
6.4.3	Убедитесь, что реализован безопасный процесс сброса забытого пароля, который не обходит активные механизмы многофакторной аутентификации.	2
6.4.4	Убедитесь, что в случае утраты одного из факторов MFA проводится удостоверение личности (identity proofing) на том же уровне, что и при первичной регистрации.	2
6.4.5	Убедитесь, что инструкции по обновлению истекающих механизмов аутентификации отправляются заранее, с достаточным запасом времени для выполнения замены, при необходимости с настройкой автоматических напоминаний.	3
6.4.6	Убедитесь, что администраторы могут инициировать процесс сброса пароля, но не могут задать или выбрать новый пароль за пользователя. Это исключает ситуацию, при которой администратор знает пароль пользователя.	3

V6.5 Общие требования к многофакторной аутентификации

Этот раздел содержит общие рекомендации, применимые к различным методам многофакторной аутентификации (MFA).

К таким механизмам относятся:

- Lookup-секреты
- Временные одноразовые пароли (TOTP)
- Out-of-Band (внеполосные) механизмы

Lookup-секреты —заранее сгенерированные списки одноразовых кодов, аналогичные TAN-кодам (Transaction Authorization Numbers), кодам восстановления для соцсетей или таблицам с наборами случайных значений. Эти коды считаются фактором типа «что-то, что у вас есть», поскольку они не запоминаются и должны быть где-то сохранены.

Временные одноразовые пароли (TOTP) —аппаратные или программные токены, генерирующие постоянно меняющиеся псевдослучайные коды. Это также фактор типа «что-то, что у вас есть». MFA-варианты TOTP предполагают дополнительное подтверждение —например, PIN-код, биометрию, вставку USB-ключа, NFC или дополнительные значения (например, калькуляторы для подписания транзакций).

Out-of-Band (внеполосные) механизмы рассматриваются в следующем разделе.

Требования этого раздела в основном соответствуют § 5.1.2, § 5.1.3, § 5.1.4.2, § 5.1.5.2, § 5.2.1, и § 5.2.3 руководства NIST.

#	Описание	Уровень
6.5.1	Убедитесь, что lookup-секреты, внеполосные коды/запросы и TOTP-коды можно использовать только один раз.	2
6.5.2	Убедитесь, что при хранении lookup-секретов с энтропией менее 112 бит (например, 19 случайных буквенно-цифровых символов или 34 случайные цифры) они хэшируются с использованием разрешённого алгоритма хеширования паролей и случайной соли минимум 32 бита. Если у секрета ≥112 бит энтропии —допустимо использовать стандартный хэш.	2
6.5.3	Убедитесь, что lookup-секреты, коды внеполосной аутентификации и сиды для TOTP генерируются с использованием криптографически стойкого генератора случайных чисел (CSPRNG), чтобы избежать предсказуемых значений.	2
6.5.4	Убедитесь, что lookup-секреты и внеполосные коды аутентификации имеют минимум 20 бит энтропии (достаточно, например, 4 случайных алфавитно-цифровых символов или 6 случайных цифр).	2
6.5.5	Убедитесь, что все внеполосные запросы, коды и токены, а также TOTP-коды имеют ограниченный срок действия. Максимум для out-of-band —10 минут, для TOTP —30 секунд.	2
6.5.6	Убедитесь, что любой фактор аутентификации (включая физические устройства) может быть отозван в случае кражи или утери.	3
6.5.7	Убедитесь, что биометрические механизмы аутентификации используются только как вторичный фактор —в сочетании с «чем-то, что вы знаете» или «чем-то, что у вас есть».	3
6.5.8	Убедитесь, что TOTP-коды проверяются по источнику времени, полученному от доверенного сервиса, а не от клиента или недоверенного источника.	3

V6.6 Внеполосные механизмы аутентификации

Обычно внеполосная аутентификация предполагает, что сервер аутентификации взаимодействует с физическим устройством пользователя через безопасный вторичный канал. Например, путём отправки push-уведомления на мобильное устройство. Такой механизм считается фактором типа «что-то, что у вас есть».

Небезопасные внеполосные методы аутентификации, такие как электронная почта и VOIP, не допускаются. Аутентификация через PSTN и SMS в настоящее время считается ограниченной по версии NIST и должна постепенно заменяться на временные одноразовые

пароли (TOTP), криптографические механизмы или аналогичные средства. Руководство NIST SP 800-63B § 5.1.3.3 рекомендует принимать меры против таких угроз, как замена устройства, смена SIM-карты, перенос номера и прочие аномалии, если всё же приходится использовать телефонную или SMS-аутентификацию. Хотя этот раздел ASVS не требует строго выполнять эти рекомендации, их игнорирование для чувствительных приложений уровня L2 или L3 считается серьёзным сигналом риска.

Также стоит отметить, что NIST недавно выпустил рекомендации, в которых отговаривает от использования push-уведомлений. Хотя ASVS пока этого не запрещает, важно учитывать риск push-бомбинга (массовой отправки уведомлений для перегрузки пользователя).

#	Описание	Уровень
6.6.1	Убедитесь, что механизмы аутентификации через телефонную сеть общего пользования (PSTN) для доставки одноразовых паролей (OTP) по телефону или SMS используются только в случае предварительной валидации номера, при этом предлагаются альтернативные более безопасные методы (например, TOTP), и пользователю сообщаются риски. Для приложений уровня L3 использование телефона и SMS недопустимо.	2
6.6.2	Убедитесь, что внеполосные запросы, коды или токены привязаны к исходному запросу аутентификации, для которого они были сгенерированы, и не могут быть использованы повторно для других попыток.	2
6.6.3	Убедитесь, что внеполосные механизмы на основе кода защищены от атак методом подбора (brute force) путём применения ограничения частоты запросов (rate limiting). Также рекомендуется использовать коды с энтропией не менее 64 бит.	2
6.6.4	Убедитесь, что при использовании push-уведомлений для MFA применяется ограничение частоты (rate limiting), чтобы предотвратить атаки push-бомбинга. Также может помочь механизм сопоставления номеров (number matching).	3

V6.7 Криптографические механизмы аутентификации

Криптографические механизмы аутентификации включают, например, смарт-карты или ключи FIDO, при использовании которых пользователь должен подключить или сопрячь криптографическое устройство с компьютером для завершения аутентификации. Аутентификационный сервер отправляет случайную строку (nonce) на криптографическое устройство или программное обеспечение, и устройство или программа вычисляет ответ на основе надежно хранимого криптографического ключа. Требования в этом разделе

содержат рекомендации по реализации подобных механизмов. Рекомендации по выбору криптографических алгоритмов приведены в главе «Криптография».

Если для криптографической аутентификации используются общие или секретные ключи, они должны храниться с использованием тех же механизмов, что и другие системные секреты —как указано в разделе «Управление секретами» главы «Конфигурация».

Требования этого раздела в основном соответствуют пункту § 5.1.7.2 руководства NIST.

#	Описание	Уровень
6.7.1	Убедитесь, что сертификаты, используемые для проверки криптографических утверждений об аутентификации, хранятся таким образом, чтобы исключить возможность их модификации.	3
6.7.2	Убедитесь, что nonce (случайная строка), используемая в процессе вызова, имеет длину не менее 64 бит и является статистически уникальной или уникальной на протяжении всего срока службы криптографического устройства.	3

V6.8 Аутентификация с использованием провайдера идентификаций (IdP)

Провайдеры идентификаций (IdP) предоставляют пользователям федеративную идентификацию. У одного пользователя может быть несколько различных идентификаторов через разные IdP, например, корпоративная учётная запись через Azure AD, Okta, Ping Identity или Google, а также потребительская через Facebook, Twitter, Google или WeChat. Этот перечень не является рекламой данных сервисов, а лишь отражает реальность: у многих пользователей есть уже существующие идентификации. Организациям стоит учитывать возможность интеграции с такими внешними IdP в зависимости от уровня доверия к процедурам подтверждения личности у этих провайдеров. Например, государственная система вряд ли примет соцсети в качестве логина, так как там легко создать фальшивую учётную запись, а вот мобильной игре может быть полезна авторизация через соцсети для роста пользовательской базы.

Безопасное использование внешних IdP требует тщательной настройки и проверки, чтобы не допустить подделки личности или ложных утверждений (assertions). Данный раздел описывает требования, направленные на снижение этих рисков.

#	Описание	Уровень
6.8.1	Убедитесь, что при поддержке несколькими IdP, невозможно подделать личность пользователя через другой IdP (например, с тем же user ID). Стандартным способом защиты является регистрация пользователя с использованием сочетания ID самого IdP (в качестве пространства имён) и пользовательского ID внутри IdP.	2
6.8.2	Убедитесь, что наличие и целостность цифровой подписи в утверждениях аутентификации (например, в JWT или SAML) всегда проверяются, а неподписанные или недействительные утверждения отклоняются.	2
6.8.3	Убедитесь, что SAML-утверждения обрабатываются только один раз в течение периода их действия, чтобы исключить атаки воспроизведения (replay attacks).	2
6.8.4	Убедитесь, что если приложение использует внешний поставщик идентификации (IdP) и для определенных функций требуется особый уровень, метод или срок действия аутентификации, то приложение использует данные, возвращаемые IdP, для проверки этих условий. Например, при использовании OIDC можно проверить значения полей ID Token —acr, amr, auth_time (если они есть). Если IdP не предоставляет эти данные, приложение должно иметь документированный план действий по умолчанию, предполагающий использование минимально допустимой силы аутентификации (например, одноступенчатая —логин и пароль).	2

Ссылки

Для дополнительной информации см. также:

- NIST SP 800-63 - Digital Identity Guidelines
- NIST SP 800-63B - Authentication and Lifecycle Management
- NIST SP 800-63 FAQ
- OWASP Web Security Testing Guide: Testing for Authentication
- OWASP Password Storage Cheat Sheet
- OWASP Forgot Password Cheat Sheet
- OWASP Choosing and Using Security Questions Cheat Sheet
- CISA Guidance on “Number Matching”
- Details on the FIDO Alliance

V7 Управление сессиями

Задачи контроля

Механизмы управления сессиями позволяют приложениям связывать взаимодействия пользователя и устройства во времени, даже при использовании протоколов без состояния (например, HTTP). Современные приложения могут использовать несколько токенов сессии с разными характеристиками и назначением. Безопасная система управления сессиями —это система, предотвращающая получение, использование или иное злоупотребление сессией жертвы злоумышленником. Приложения, поддерживающие сессии, должны соответствовать следующим концептуальным требованиям:

- Сессии уникальны для каждого пользователя, их идентификаторы нельзя угадать, и ими нельзя «поделиться» с другими.
- Сессии становятся недействительными, когда они больше не нужны, т.е. при бездействии пользователя срок действия сессии истекает.

Многие требования в этой главе основаны на NIST SP 800-63 Digital Identity Guidelines, с фокусом на распространённые угрозы и уязвимости аутентификации.

Обратите внимание, что конкретные требования по реализации различных механизмов управления сессиями приведены в других разделах:

- Требования по безопасности HTTP cookies —в разделе «Безопасность веб-интерфеса».
- Требования по автономным токенам —в разделе «Автономные токены».

V7.1 Документация по управлению сессиями

Нет единого шаблона, подходящего для всех приложений. Поэтому необходимо проводить анализ рисков с документированием решений по безопасности, касающихся управления сессиями, до реализации и тестирования. Это обеспечивает адаптацию системы управления сессиями к конкретным требованиям приложения.

Независимо от того, используется ли механизм сессий с сохранением состояния (stateful) или без (stateless), анализ должен быть завершён и задокументирован, чтобы продемонстрировать соответствие всем требованиям безопасности. Также необходимо учесть взаимодействие с системами единого входа (SSO), если они используются.

#	Описание	Уровень
7.1.1	Убедитесь, что тайм-аут бездействия и абсолютное максимальное время жизни сессии задокументированы, соответствуют другим контрольным мерам и сопровождаются обоснованием любых отклонений от требований повторной аутентификации NIST SP 800-63B.	2
7.1.2	Убедитесь, что документация определяет допустимое количество одновременных (параллельных) сессий для одного аккаунта, а также поведение системы и действия при достижении этого лимита.	2
7.1.3	Убедитесь, что все системы, создающие и управляющие сессиями пользователей в рамках федеративной идентификации (например, SSO), задокументированы вместе с механизмами координации сроков действия сессий, условий завершения и случаев, требующих повторной аутентификации.	2

V7.2 Базовая безопасность сессий

Требования этого раздела обеспечивают безопасное создание и валидацию сессионных токенов.

#	Описание	Уровень
7.2.1	Убедитесь, что все проверки сессионных токенов выполняются на доверенной серверной стороне.	1
7.2.2	Убедитесь, что приложение использует либо автономные токены, либо ссылочные токены, которые генерируются динамически, не используя статические API-секреты или ключи.	1
7.2.3	Убедитесь, что ссылочные токены являются уникальными, создаются с использованием криптографически безопасного генератора псевдослучайных чисел (CSPRNG) и обладают энтропией не менее 128 бит.	1
7.2.4	Убедитесь, что при аутентификации пользователя (включая повторную) создается новый токен сессии, а текущий —аннулируется.	1

V7.3 Тайм-ауты сессии

Механизмы тайм-аутов сессии уменьшают временное окно для угонов и других атак. Тайм-ауты должны соответствовать документированным решениям по безопасности.

#	Описание	Уровень
7.3.1	Убедитесь, что есть тайм-аут бездействия, после которого требуется повторная аутентификация, в соответствии с анализом рисков и документированными решениями.	2
7.3.2	Убедитесь, что установлено абсолютное максимальное время жизни сессии, по истечении которого требуется повторная аутентификация, в соответствии с анализом рисков и документированными решениями.	2

V7.4 Завершение сессий

Завершение сессии может происходить как со стороны самого приложения, так и через поставщика SSO, если управление сессиями делегировано ему. Некоторые требования этого раздела могут быть реализованы на стороне поставщика SSO.

Завершение сессии должно требовать повторной аутентификации и быть эффективным во всех компонентах: приложении, федеративной аутентификации и других доверяющих сторонах.

Сессии с сохранением состояния обычно аннулируются на серверной стороне. self-contained токены нужно отзывать или блокировать, иначе они могут оставаться действительными до истечения срока их действия.

#	Описание	Уровень
7.4.1	Убедитесь, что после завершения сессии (logout, истечение срока) дальнейшее её использование невозможно. Для ссылочных токенов или сессий с сохранением состояния это означает аннулирование на сервере. Для self-contained токенов необходимо реализовать механизм отзыва, например, список отозванных токенов, проверку времени выпуска или ротацию ключей подписи для каждого пользователя.	1
7.4.2	Убедитесь, что при удалении или отключении пользователя все активные сессии немедленно завершаются.	1
7.4.3	Убедитесь, что после изменения или удаления факторов аутентификации (например, смена пароля или MFA) приложение предлагает завершить все другие активные сессии.	2
7.4.4	Убедитесь, что на всех страницах, требующих аутентификации, имеется простой и видимый способ выхода.	2

#	Описание	Уровень
7.4.5	Убедитесь, что администраторы могут завершать активные сессии конкретного пользователя или всех пользователей.	2

V7.5 Защита от злоупотреблений сессией

Этот раздел направлен на снижение рисков при активных сессиях, включая атаки с использованием действующей сессии жертвы.

При рассмотрении требований этого раздела нужно учитывать рекомендации для конкретных уровней, приведенные в разделе «Аутентификация»

#	Описание	Уровень
7.5.1	Убедитесь, что приложение требует полной повторной аутентификации перед изменением чувствительных атрибутов аккаунта, которые могут влиять на аутентификацию (например, email, номер телефона, настройки MFA или данных восстановления доступа).	2
7.5.2	Убедитесь, что пользователь может просматривать и (после повторной аутентификации хотя бы по одному фактору) завершать активные сессии.	2
7.5.3	Убедитесь, что перед выполнением критически важных операций приложение требует повторной аутентификации (по крайней мере, одного фактора) или дополнительной проверки.	3

V7.6 Федеративная повторная аутентификация

Требования для сторон, реализующих логику Relying Party (RP) и Identity Provider (IdP), основаны на NIST SP 800-63C.

#	Описание	Уровень
7.6.1	Убедитесь, что срок действия и завершение сессий между RP и IdP соответствуют документации и включают повторную аутентификацию, если достигнут максимальный допустимый интервал между событиями аутентификации.	2

#	Описание	Уровень
7.6.2	Убедитесь, что создание сессии требует согласия пользователя или явного действия, предотвращающего несанкционированное создание сессий без участия пользователя.	2

Ссылки

Для дополнительной информации см. также:

- OWASP Web Security Testing Guide: Session Management Testing
- OWASP Session Management Cheat Sheet

V8 Авторизация

Задачи контроля

Механизмы авторизации обеспечивают доступ только разрешённым субъектам (пользователям, сервисам и другим клиентам). Для соблюдения принципа наименьших привилегий (POLP) проверяемые приложения должны соответствовать следующим концептуальным требованиям:

- Документировать правила авторизации, включая факторы принятия решений и контекст окружения.
- Субъекты должны иметь доступ только к тем ресурсам, на которые у них есть разрешения.

V8.1 Документация по авторизации

Полная документация по авторизации необходима для обеспечения согласованного применения решений в области безопасности, возможности аудита и соответствия политикам организации. Это снижает риск несанкционированного доступа, делая требования безопасности понятными и выполнимыми для разработчиков, администраторов и тестировщиков.

#	Описание	Уровень
8.1.1	Убедитесь, что документация по авторизации определяет правила ограничения доступа на уровне функций и на уровне данных на основе прав субъекта и атрибутов ресурса.	1

#	Описание	Уровень
8.1.2	Убедитесь, что документация по авторизации определяет правила ограничения доступа на уровне полей (как на чтение, так и на запись) на основе прав субъекта и атрибутов ресурса. Эти правила могут зависеть от других атрибутов объекта данных, например, статуса.	2
8.1.3	Убедитесь, что документация определяет, какие атрибуты окружения и контекста (например, время суток, IP-адрес, местоположение, устройство) используются приложением для принятия решений в области аутентификации и авторизации.	3
8.1.4	Убедитесь, что документация по аутентификации и авторизации определяет, как используются факторы окружения и контекста в принятии решений, наряду с правилами доступа к функциям, данным и полям. Документация должна содержать оцениваемые атрибуты, пороговые значения риска, действия (разрешить, отказать, повысить уровень аутентификации и др.).	3

V8.2 Архитектура авторизации

Реализация детализированных механизмов авторизации на уровне функций, данных и полей обеспечивает субъектам наличие прав доступа только к тем компонентам, которые были им явно предоставлены.

#	Описание	Уровень
8.2.1	Убедитесь, что приложение обеспечивает доступ на уровне функций (function-level) только субъектам с явными разрешениями.	1
8.2.2	Убедитесь, что приложение обеспечивает доступ на уровне данных (data-specific) только тем субъектам, которым явно предоставлены права к определённым элементам данных, чтобы предотвратить IDOR (Insecure Direct Object Reference) и BOLA-уязвимости (Broken Object Level Authorization).	1
8.2.3	Убедитесь, что приложение обеспечивает доступ на уровне полей (field-level) только тем субъектам, которым явно предоставлены права к определённым полям, чтобы предотвратить BOPLA (Broken Object Property Level Authorization).	2

#	Описание	Уровень
8.2.4	Убедитесь, что реализованы адаптивные меры безопасности, основанные на атрибутах окружения и контекста субъекта (время, местоположение, IP, устройство), которые используются для принятия решений об аутентификации и авторизации, как это определено в документации приложения. Эти меры должны применяться как при попытке субъекта начать новую сессию, так и во время существующей сессии.	3

V8.3 Авторизация на уровне операций

Немедленное применение изменений авторизации на соответствующем уровне архитектуры критически важно для предотвращения несанкционированных действий, особенно в динамически меняющихся системах.

#	Описание	Уровень
8.3.1	Убедитесь, что правила авторизации реализуются на доверенном сервисном уровне и не зависят от элементов, которые может контролировать клиент (например, JavaScript на стороне клиента).	1
8.3.2	Убедитесь, что изменения значений, влияющих на авторизацию, применяются немедленно. Если это невозможно (например, при использовании автономных токенов), должны быть реализованы компенсирующие меры: оповещения о попытке несанкционированного действия и откат изменений. Обратите внимание, что данный подход не предотвращает утечку информации.	3
8.3.3	Убедитесь, что доступ к объекту осуществляется на основе прав субъекта, инициировавшего запрос, а не прав промежуточного сервиса, действующего от его имени. Например, если пользователь обращается к веб-сервису с автономным токеном для аутентификации, а сервис затем запрашивает данные у другого сервиса, второй сервис должен использовать токен пользователя, а не токен первого сервиса для принятия решения о доступе.	3

V8.4 Прочие аспекты авторизации

Дополнительные меры контроля, особенно в административных интерфейсах и многопользовательских (мультитенантных) системах, помогают предотвратить несанкционированный доступ.

#	Описание	Уровень
8.4.1	Убедитесь, что в мультитенантных приложениях реализованы механизмы, предотвращающие действия потребителя в отношении арендаторов, к которым у него нет доступа.	2
8.4.2	Убедитесь, что доступ к административным интерфейсам защищён несколькими уровнями контроля, включая непрерывную проверку личности, контроль защищённости устройства и анализ контекстных рисков. Недостаточно опираться только на IP-адрес или доверенные устройства.	3

Ссылки

Для дополнительной информации см. также:

- OWASP Web Security Testing Guide: Authorization
- OWASP Authorization Cheat Sheet

V9 Автономные токены

Задачи контроля

В оригинальном RFC 6749 OAuth 2.0 от 2012 года упоминается концепция автономного токена (self-contained token). Она относится к токенам, которые содержат данные или утверждения (claims), на основе которых получатель будет принимать решения по безопасности. Автономный токен следует отличать от простого токена, содержащего только идентификатор, который принимающая сторона использует для локального поиска данных.

Наиболее распространёнными примерами автономных токенов являются JSON Web Tokens (JWTs) и SAML Assertions. Но автономные токены получили широкое распространение даже за пределами OAuth и OIDC. Безопасность этой концепции зависит от возможности проверить целостность токена и убедиться, что он действителен для конкретного контекста. Этот процесс сопряжен с множеством подводных камней, и в этой главе подробно описываются механизмы, помогающие приложениям обходить их.

V9.1 Источник токена и его целостность

В данном разделе изложены требования, гарантирующие, что токен был выпущен доверенной стороной и не был модифицирован.

#	Описание	Уровень
9.1.1	Убедитесь, что автономные токены проверяются на целостность с помощью цифровой подписи или MAC, прежде чем доверять их содержимому.	1
9.1.2	Проверяйте, что автономные токены создаются и проверяются исключительно с помощью алгоритмов из списка допустимых в данном контексте. Список допустимых алгоритмов должен включать только разрешенные алгоритмы (желательно либо только симметричные, либо только асимметричные) и ни в коем случае не содержать алгоритм None. Если необходимо поддерживать оба типа алгоритмов, потребуются дополнительные средства контроля для предотвращения путаницы с ключами (key confusion).	1
9.1.3	Убедитесь, что ключи, используемые для проверки автономных токенов, получены из доверенных, предварительно настроенных источников. Это предотвращает возможность указания злоумышленниками ненадёжных источников и ключей. Для JWT и других JWS структур заголовки ‘jku’, ‘x5u’и ‘jwk’должны проверяться по белому списку.	1

V9.2 Содержимое токена

Прежде чем принимать решения по безопасности на основе данных из токена, важно проверить два момента: что срок действия токена актуален на момент предъявления, и что токен предназначен для использования принимающим сервисом в тех целях, для которых он был передан. Это предотвращает небезопасное перекрестное использование токена между различными сервисами или использование разного типа токенов от одного и того же эмитента.

Конкретные требования для OAuth и OIDC рассматриваются в соответствующей главе.

#	Описание	Уровень
9.2.1	Убедитесь, что при указанном сроке действия токен и его содержимое принимаются только в том случае, если срок действия токена не истёк. Например, для JWT нужно проверить поля ‘nbf’и ‘exp’.	1

#	Описание	Уровень
9.2.2	Убедитесь, что сервис, принимающий токен, проверяет его тип и соответствие назначению перед обработкой содержимого токена. Например, для принятия решений об авторизации могут быть приняты только Access Token, а для подтверждения аутентификации пользователя —только ID Token.	2
9.2.3	Убедитесь, что сервис принимает только токены, предназначенные для использования с этим сервисом (audience). Для JWT это можно сделать, проверив наличие утверждения «aud» по белому списку, определенному в сервисе.	2
9.2.4	Убедитесь, что, если эмитент токенов использует один и тот же закрытый ключ для выпуска токенов различным сервисам (audiences), выпущенные токены содержат явное указание целевых сервисов. Это предотвратит повторное использования токенов в не предназначенных для них сервисах. Если идентификатор сервиса (audience identifier) назначается динамически, эмитент токенов должен проверить audiences для исключения подмены получателей (audience impersonation).	2

Ссылки

Для дополнительной информации см. также:

- OWASP JSON Web Token Cheat Sheet for Java Cheat Sheet (но имеет полезные общие рекомендации)

V10 OAuth и OIDC

Задачи контроля

OAuth2 (в этой главе именуется просто OAuth) —это общепринятый отраслевой стандарт делегированной авторизации. Например, с помощью OAuth клиентское приложение может получить доступ к API (ресурсам сервера) от имени пользователя, если тот дал на это разрешение.

Сама по себе технология OAuth не предназначена для аутентификации пользователей. Фреймворк OpenID Connect (OIDC) расширяет OAuth, добавляя слой идентификации пользователя. OIDC поддерживает такие функции, как стандартизированная информация о пользователе, единый вход (SSO) и управление сессиями. Поскольку OIDC является

надстройкой над OAuth, все требования, предъявляемые к OAuth, также применимы к OIDC.

В OAuth определены следующие роли:

- OAuth-клиент —приложение, пытающееся получить доступ к серверным ресурсам (например, через вызов API с использованием access token). Чаще всего это серверное приложение.
  - Конфиденциальный клиент (confidential client) —клиент, способный сохранять в тайне свои учетные данные (например, client_secret) для аутентификации на авторизационном сервере.
  - Публичный клиент (public client) —клиент, неспособный хранить учетные данные в секрете. Вместо аутентификации (например, с использованием пары client_id и client_secret) он лишь идентифицирует себя через client_id.
- OAuth-ресурсный сервер (Resource Server, RS) —сервер API, предоставляющий ресурсы клиентам OAuth.
- OAuth-авторизационный сервер (Authorization Server, AS) —сервер, выдающий клиентам access token'ы. Эти токены позволяют клиенту получать доступ к ресурсам RS —либо от имени пользователя, либо от имени самого клиента. Часто AS —это отдельное приложение, но при необходимости может быть интегрирован в RS.
- Владелец ресурса (Resource Owner, RO) —конечный пользователь, разрешающий OAuth-клиенту ограниченный доступ к своим данным на RS. Пользователь предоставляет это разрешение через взаимодействие с AS.

В OIDC определены следующие роли:

- Relying Party (RP) —клиентское приложение, запрашивающее аутентификацию пользователя через OpenID-провайдера. В терминах OAuth оно также является OAuth-клиентом.
- OpenID-провайдер (OP) —авторизационный сервер OAuth, способный аутентифицировать пользователя и предоставлять RP claims в рамках OIDC. OP может быть IdP (провайдером идентификации), но в федеративных сценариях OP и IdP могут быть разными приложениями.

Изначально OAuth и OIDC были разработаны для сторонних приложений. Однако в настоящее время они часто используются и в first-party сценариях. Например, для аутентификации и управления сессиями. В таких случаях протоколы добавляют некоторую сложность, которая может привести к новым проблемам с безопасностью.

OAuth и OIDC могут использоваться в различных типах приложений, однако в OWASP ASVS акцент делается на веб-приложения и API.

Поскольку OAuth и OIDC работают поверх веб-технологий, на них распространяются общие требования из других глав. Этот раздел не следует рассматривать изолированно.

В этой главе рассматриваются лучшие современные практики OAuth2 и OIDC в соответствии со спецификациями, представленными на https://oauth.net/2/ и https://openid.net/developers/specs/. Поскольку RFC часто обновляются, при применении требований, изложенных в этой главе, следует сверяться с их последними версиями. Подробнее см. в разделе «Ссылки».

С учетом сложности темы крайне важно использовать проверенные стандартные решения (например, авторизационные серверы от крупных вендоров) и применять рекомендованные параметры безопасности.

Терминология в этой главе соответствует RFC OAuth и спецификациям OIDC. Однако OIDC-термины используются только в OIDC-специфичных требованиях. В остальных случаях применяется терминология OAuth.

В этом разделе под «токеном» понимается:

- Access Token используется только ресурсным сервером (RS). Он может быть ссылочным (reference token), проверяемым через introspection, или автономным (self-contained), проверяемым через криптографически подписанный материал.
- Refresh Token используется только авторизационным сервером, выдавшим токен.
- OIDC ID Token используется только клиентским приложением, инициировавшим процесс авторизации.

Уровни риска для некоторых требований зависят от того, является ли клиент конфиденциальным или публичным. Поскольку сильная клиентская аутентификация снижает многие риски, в случае конфиденциальных клиентов на уровне L1 возможны послабления.

V10.1 Общие требования безопасности OAuth и OIDC

В этом разделе изложены универсальные архитектурные требования, применимые ко всем приложениям, использующим OAuth или OIDC.

#	Описание	Уровень
10.1.1	Убедитесь, что токены передаются только тем компонентам, которым они действительно необходимы. Например, при использовании схемы backend-for-frontend для браузерных JavaScript-приложений, access и refresh токены должны быть доступны только на backend.	2

#	Описание	Уровень
10.1.2	Убедитесь, что клиент принимает значения от авторизационного сервера (например, authorization code или ID Token) только если они получены в результате авторизационного потока, инициированного в той же сессии пользовательского агента и в рамках той же транзакции. Для этого клиентские секреты, такие как proof key для code exchange (PKCE) —‘code_verifier’, ‘state’или OIDC ‘nonce’—должны быть непредсказуемыми, специфичными для транзакции и надежно привязанными как к клиенту, так и к сессии пользовательского агента, в которой началась транзакция.	2

V10.2 Клиент OAuth

Данные требования описывают обязанности приложений-клиентов OAuth. Клиентом может быть, например, backend веб-сервера (часто выступающий как Backend For Frontend, BFF), интеграция backend-сервиса или frontend-приложение одностраничного типа (SPA, то есть браузерное приложение).

В общем случае backend-клиенты считаются конфиденциальными (confidential clients), а frontend —публичными (public clients). Однако нативные приложения, работающие на устройстве конечного пользователя, могут рассматриваться как конфиденциальные при использовании динамической регистрации OAuth клиента.

#	Описание	Уровень
10.2.1	Убедитесь, что если используется механизм authorization code flow, OAuth клиент защищён от CSRF, вызывающих запросы токенов либо за счёт использования механизма proof key for code exchange (PKCE), либо за счёт проверки параметра ‘state’, который был отправлен в запросе авторизации.	2
10.2.2	Убедитесь, что если OAuth клиент может взаимодействовать с более чем одним сервером авторизации, у него есть защита от атак смешивания (mix-up attacks). Например, клиент должен требовать, чтобы сервер авторизации возвращал параметр ‘iss’и проверять его в ответах на авторизацию и токены.	2
10.2.3	Убедитесь, что OAuth клиент запрашивает только необходимые области доступа (scopes) или другие параметры авторизации в запросах к серверу авторизации.	3

V10.3 Сервер ресурсов OAuth

В контексте ASVS и данного раздела сервер ресурсов —это API. Для обеспечения безопасного доступа сервер ресурсов должен:

- Валидировать access token в соответствии с форматом токена и спецификациями протокола, например, валидация JWT или интроспекция OAuth токена.
- Если токен действителен, принимать решения по авторизации на основе информации из токена и предоставленных разрешений. Например, сервер ресурсов должен проверить, что клиент (действующий от имени владельца ресурса) имеет право доступа к запрошенному ресурсу.

Таким образом, требования здесь специфичны для OAuth или OIDC и должны выполняться после валидации токена и перед принятием решения об авторизации на основе информации из токена.

#	Описание	Уровень
10.3.1	Убедитесь, что сервер ресурсов принимает только те access токены, которые предназначены для использования с этим сервисом (свойство audience). Эта принадлежность сервису может быть включена в структурированный access токен (например, в поле ‘aud’в JWT) или проверяться через endpoint интроспекции токена.	2
10.3.2	Убедитесь, что сервер ресурсов принимает решения по авторизации на основе claims из access токена, определяющих делегированную авторизацию. Если присутствуют claims, такие как ‘sub’, ‘scope’и ‘authorization_details’, они должны участвовать в принятии решения.	2
10.3.3	Убедитесь, что если для принятия решения по контролю доступа необходимо однозначно идентифицировать пользователя из access токена (JWT или ответа интроспекции), сервер ресурсов использует claims, которые не могут быть переприсвоены другим пользователям. Обычно это комбинация claims ‘iss’и ‘sub’.	2
10.3.4	Убедитесь, что если сервер ресурсов требует определённой силы аутентификации, методов или свежести аутентификации, он проверяет, что предъявленный access токен удовлетворяет этим требованиям. Например, при наличии, могут использоваться claims OIDC ‘acr’, ‘amr’и ‘auth_time’.	2

#	Описание	Уровень
10.3.5	Убедитесь, что сервер ресурсов предотвращает использование украденных access токенов или их повторное использование (replay) неавторизованными лицами, требуя access токены с подтверждением отправителя —например, Mutual TLS для OAuth 2 или OAuth 2 Demonstration of Proof of Possession (DPoP).	3

V10.4 Сервер авторизации OAuth

Эти требования описывают обязанности серверов авторизации OAuth, включая OpenID провайдеров.

Для аутентификации клиента разрешён метод ‘self_signed_tls_client_auth’при выполнении предварительных требований, указанных в разделе 2.2 стандарта RFC 8705.

#	Описание	Уровень
10.4.1	Убедитесь, что сервер авторизации проверяет URI перенаправления (redirect URI), сверяя его с заранее зарегистрированным списком разрешённых URI для данного клиента (allowlist), используя точное сравнение строк.	1
10.4.2	Убедитесь, что если сервер авторизации возвращает authorization code в ответе авторизации, этот код может быть использован только один раз для запроса токена. При втором валидном запросе с уже использованным кодом сервер должен отклонить запрос и отозвать любые токены, связанные с этим кодом.	1
10.4.3	Убедитесь, что authorization code имеет короткое время жизни. Максимальная продолжительность —до 10 минут для приложений L1 и L2 и до 1 минуты для приложений L3.	1
10.4.4	Убедитесь, что сервер авторизации для конкретного клиента разрешает использование только тех типов авторизации, которые необходимы этому клиенту. Обратите внимание, что тип авторизации Implicit flow и Resource Owner Password Credentials flow больше не должны использоваться.	1

#	Описание	Уровень
10.4.5	Убедитесь, что сервер авторизации предотвращает атаки повторного использования refresh токенов для публичных клиентов, предпочтительно используя sender-constrained refresh токены, такие как DPoP или сертификат-связанные токены с mutual TLS (mTLS). Для L1 и L2 допускается ротация refresh токенов с обязательной их инвалидацией после использования. Если используется ротация, сервер должен отозвать все refresh токены при попытке использовать уже инвалидированный токен.	1
10.4.6	Убедитесь, что при использовании типа авторизации «Authorization Code» сервер предотвращает атаки перехвата authorization code, требуя Proof Key for Code Exchange (PKCE). В запросах авторизации сервер должен требовать валидное значение ‘code_challenge’и не принимать ‘code_challenge_method’со значением ‘plain’. В запросах токена обязательно валидировать параметр ‘code_verifier’.	2
10.4.7	Убедитесь, что если сервер авторизации поддерживает динамическую регистрацию клиентов без аутентификации, он минимизирует риски вредоносных клиентов. Сервер должен валидировать метаданные клиента (например, зарегистрированные URI), обеспечивать согласие пользователя и предупреждать пользователя перед обработкой запроса авторизации от недоверенного клиента.	2
10.4.8	Убедитесь, что refresh токены имеют абсолютный срок действия, включая случаи, когда применяется скользящее время истечения срока.	2
10.4.9	Убедитесь, что пользователи могут отзывать refresh токены и reference access токены через пользовательский интерфейс сервера авторизации для снижения риска от вредоносных клиентов или кражи токенов.	2
10.4.10	Убедитесь, что конфиденциальный клиент аутентифицируется при взаимодействии с сервером авторизации через backchannel, например при запросах токена, pushed authorization requests (PAR) и запросах на отзыв токенов.	2
10.4.11	Убедитесь, что конфигурация сервера авторизации назначает OAuth клиенту только необходимые области доступа (scopes).	2
10.4.12	Убедитесь, что для конкретного клиента сервер авторизации разрешает только те значения ‘response_mode’, которые необходимы этому клиенту. Например, сервер должен валидировать это значение против ожидаемых или использовать pushed authorization request (PAR) или JWT-secured Authorization Request (JAR).	3

#	Описание	Уровень
10.4.13	Убедитесь, что грант типа ‘code’всегда используется вместе с pushed authorization requests (PAR).	3
10.4.14	Убедитесь, что сервер авторизации выдаёт только sender-constrained (Proof-of-Possession) access токены, либо с сертификат-связанными токенами через mutual TLS (mTLS), либо DPoP-bound токены (Demonstration of Proof of Possession).	3
10.4.15	Убедитесь, что для серверного клиента (не выполняющегося на устройстве конечного пользователя) сервер авторизации проверяет, что значение параметра ‘authorization_details’исходит из клиентского backend и что пользователь не изменял его. Например, это достигается использованием pushed authorization request (PAR) или JWT-secured Authorization Request (JAR).	3
10.4.16	Убедитесь, что клиент является конфиденциальным и сервер авторизации требует использования сильных методов аутентификации клиента (основанных на криптографии с открытым ключом и устойчивых к атакам повторного воспроизведения), таких как mutual TLS (‘tls_client_auth’, ‘self_signed_tls_client_auth’) или private key JWT (‘private_key_jwt’).	3

V10.5 Клиент OIDC

Поскольку OIDC клиент (relying party) выступает в роли OAuth клиента, на него также распространяются требования из раздела «OAuth Client».

Обратите внимание, что в разделе «Аутентификация с использованием провайдера идентификаций (IdP)» главы «Аутентификация» содержатся дополнительные общие требования.

#	Описание	Уровень
10.5.1	Убедитесь, что клиент (relying party) защищается от повторного использования ID Token (replay-атак). Например, это достигается проверкой того, что значение ‘nonce’в ID Token совпадает со значением ‘nonce’, отправленным в запросе аутентификации провайдеру OpenID (authorization server).	2

#	Описание	Уровень
10.5.2	Убедитесь, что клиент однозначно идентифицирует пользователя по утверждениям (claims) ID Token, обычно это ‘sub’, который не может быть переназначен другому пользователю (в пределах конкретного провайдера идентификации).	2
10.5.3	Убедитесь, что клиент отклоняет попытки злоумышленного сервера авторизации выдавать себя за другой сервер авторизации через метаданные сервера авторизации. Клиент должен отклонить метаданные, если URL издателя (issuer) в метаданных не совпадает точно с преднастроенным URL издателя, ожидаемым клиентом.	2
10.5.4	Убедитесь, что клиент проверяет, что ID Token предназначен для него (проверка audience), сравнивая значение ‘aud’в токене с ‘client_id’ клиента.	2
10.5.5	Убедитесь, что при использовании OIDC back-channel logout клиент снижает риск отказа в обслуживании, вызванный принудительным выходом из системы и путаницей между разными типами JWT (cross-JWT confusion) в процессе логаута. Клиент должен проверить, что токен выхода корректно типизирован со значением ‘logout+jwt’, содержит claim ‘event’с правильным именем участника и не содержит ‘nonce’. Также рекомендуется установить короткий срок действия токена (например, 2 минуты).	2

V10.6 Провайдер OpenID

Поскольку провайдеры OpenID выступают в роли OAuth серверов авторизации, на них также распространяются требования из раздела «Сервер авторизации OAuth».

Обратите внимание, что при использовании потока ID Token (а не code flow) access токены не выдаются, и многие требования к OAuth AS в этом случае неприменимы.

#	Описание	Уровень
10.6.1	Убедитесь, что провайдер OpenID допускает только значения ‘code’, ‘ciba’, ‘id_token’или ‘id_token code’для параметра response mode. При этом предпочтение отдаётся ‘code’по сравнению с ‘id_token code’ (гибридный поток OIDC), а значение ‘token’(любой Implicit flow) не должно использоваться.	2

#	Описание	Уровень
10.6.2	Убедитесь, что провайдер OpenID предотвращает отказ в обслуживании (DoS) через принудительный выход (logout), запрашивая явное подтверждение от конечного пользователя или, если применимо, проверяя параметры в запросе выхода (инициированном relying party), например параметр ‘id_token_hint’.	2

V10.7 Управление согласием

Эти требования касаются проверки согласия пользователя авторизационным сервером. Без правильной проверки согласия пользователя злоумышленник может получить разрешения от имени пользователя с помощью подделки или социальной инженерии.

#	Описание	Уровень
10.7.1	Убедитесь, что авторизационный сервер обеспечивает получение согласия пользователя для каждого запроса авторизации. Если личность клиента не может быть подтверждена, сервер авторизации всегда должен явно запрашивать согласие пользователя.	2
10.7.2	Убедитесь, что при запросе согласия пользователя авторизационный сервер предоставляет достаточную и понятную информацию о том, на что даётся согласие. При необходимости это должно включать характер запрашиваемых разрешений (обычно основанных на scope, ресурсном сервере, деталях авторизации Rich Authorization Requests (RAR)), идентификацию авторизованного приложения и срок действия этих разрешений.	2
10.7.3	Убедитесь, что пользователь может просматривать, изменять и отзывать согласия, которые он предоставил, через авторизационный сервер.	2

Ссылки

Для дополнительной информации об OAuth см.:

- oauth.net
- OWASP OAuth 2.0 Protocol Cheat Sheet

Для требований, связанных с OAuth, в ASVS используются следующие опубликованные и находящиеся в статусе черновика RFC:

- RFC6749 The OAuth 2.0 Authorization Framework
- RFC6750 The OAuth 2.0 Authorization Framework: Bearer Token Usage
- RFC6819 OAuth 2.0 Threat Model and Security Considerations
- RFC7636 Proof Key for Code Exchange by OAuth Public Clients
- RFC7591 OAuth 2.0 Dynamic Client Registration Protocol
- RFC8628 OAuth 2.0 Device Authorization Grant
- RFC8707 Resource Indicators for OAuth 2.0
- RFC9068 JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens
- RFC9126 OAuth 2.0 Pushed Authorization Requests
- RFC9207 OAuth 2.0 Authorization Server Issuer Identification
- RFC9396 OAuth 2.0 Rich Authorization Requests
- RFC9449 OAuth 2.0 Demonstrating Proof of Possession (DPoP)
- RFC9700 Best Current Practice for OAuth 2.0 Security
- draft OAuth 2.0 for Browser-Based Applications
- draft The OAuth 2.1 Authorization Framework

Для получения дополнительной информации об OpenID Connect см.:

- OpenID Connect Core 1.0
- FAPI 2.0 Security Profile

V11 Криптография

Задачи контроля

Цель этой главы - определить наилучшие методы общего использования криптографии, а также привить фундаментальное понимание ее принципов и стимулировать переход к более устойчивым и современным подходам. В ней рассмотрены следующие пункты:

- Внедрение надежных криптографических систем, которые являются отказоустойчивыми, адаптируются к меняющимся угрозам и ориентированы на будущее.
- Использование криптографических механизмов, которые одновременно являются безопасными и соответствуют лучшим практикам индустрии.
- Поддержка безопасной системы управления криптографическими ключами с надлежащим контролем доступа и аудитом.
- Регулярная оценка криптографической отрасли для определения рисков и соответствующей адаптации алгоритмов.
- Обнаружение и управление криптографическими сценариями использования на протяжении всего жизненного цикла приложения с целью обеспечения учёта и защиты всех криптографических ресурсов.

В дополнение к обозначенным общим принципам и лучшим практикам данный документ также представляет более углубленную техническую информацию о требованиях в

Приложении C - Стандарты криптографии. Он содержит алгоритмы и конфигурации, которые считаются «утвержденными» в соответствии с требованиями, изложенными в данном разделе.

Требования для решения отдельных криптографических задач, таких как управление секретами или обеспечение безопасного соединения будут рассмотрены в разных частях стандарта.

V11.1 Инвентаризация и документирование криптографии

Приложения должны проектироваться с использованием надёжной криптографической архитектуры для защиты данных в соответствии с их классификацией. Шифровать всё подряд —это расточительно, а не шифровать ничего —юридическая халатность. Необходимо найти баланс, который обычно достигается на этапе архитектурного или высокоуровневого проектирования, в ходе дизайн-спринтов или архитектурных исследований. Проектирование криптографии «на ходу» или доработка её после разработки приложения неизбежно приведёт к гораздо большим затратам на обеспечение безопасности, чем если бы она была заложена с самого начала.

Важно регулярно обнаруживать, вести учёт и оценивать все криптографические активы. В приложении приведена дополнительная информация о том, как это делать.

Также критически важно подготовить криптографические системы к будущему появлению квантовых компьютеров. Постквантовая криптография (Post-Quantum Cryptography, PQC) —это криптографические алгоритмы, разработанные так, чтобы оставаться надёжными против атак квантовых компьютеров, которые, как ожидается, смогут взламывать широко используемые алгоритмы, такие как RSA и эллиптические кривые (ECC).

В приложении также содержатся текущие рекомендации по проверенным PQC-примитивам и стандартам.

#	Описание	Уровень
11.1.1	Убедитесь, что задокументирована политика управления криптографическими ключами и описан их жизненный цикл в соответствии со стандартом управления ключей, таким как NIST SP 800-57. Убедитесь, что ключи известны только тем сущностям, которым они должны быть доступны (например, ключи не передавались более чем двум участникам для общих секретов и более чем одному участнику для приватных ключей).	2

#	Описание	Уровень
11.1.2	Убедитесь, что проводится инвентаризация криптографических средств, которая непрерывно поддерживается, регулярно обновляется и включает все криптографические ключи, алгоритмы и сертификаты, используемые в приложении. Также должна быть задокументирована информация о том, где ключи могут и не могут использоваться в системе, а также какие типы данных могут и не могут быть защищены с помощью этих ключей.	2
11.1.3	Убедитесь, что используются механизмы обнаружения криптографии для идентификации всех случаев применения криптографии в системе , включая операции шифрования, хеширования и цифровой подписи.	3
11.1.4	Убедитесь, что ведётся и поддерживается инвентаризация криптографических средств. В неё должен входить документированный план, описывающий переход на новые криптографические стандарты, такие как постквантовая криптография, чтобы своевременно реагировать на будущие угрозы.	3

V11.2 Безопасная реализация криптографии

Данный раздел содержит требования для определения (выбора), реализации и дальнейшего управления основными криптографическими алгоритмами в приложении. Основная цель - убедиться в том, что используются исключительно надежные, утвержденные криптографические примитивы в соответствии с текущими стандартами (н-р, NIST, ISO/IEC) и лучшими практиками разработки. Организации должны убедиться, что каждый криптографический компонент выбран с учетом рецензируемых исследований и практического тестирования безопасности.

#	Описание	Уровень
11.2.1	Убедитесь, что для криптографических операций используются утвержденные в индустрии реализации (включая библиотеки и аппаратные реализации).	2

#	Описание	Уровень
11.2.2	Убедитесь, что архитектура приложения поддерживает быструю реконфигурацию, усовершенствование или смену в любой момент случайных чисел, аутентифицированного шифрования, MAC, алгоритмов хеширования, длин ключей, количества раундов, шифров и режимов шифрования для защиты от криптографических атак. Также должна поддерживаться возможность смены ключей, паролей и повторного шифрования данных. Это позволит беспрепятственно перейти на постквантовую криптографию (PQC), как только станут широко доступны высоконадёжные реализации одобренных схем или стандартов PQC.	2
11.2.3	Убедитесь, что все криптографические примитивы используют как минимум 128-бит стойкости с учетом алгоритма, размера ключа и конфигурации. Например, 256-битный ECC ключ обеспечивает примерно 128 бит безопасности, тогда как для достижения того же уровня безопасности в RSA требуется ключ длиной 3072 бита.	2
11.2.4	Убедитесь, что все криптографические операции выполняются за константное время, без использования «short-circuit» операций в сравнениях, вычислениях, возвратах функций для избежания утечек информации.	3
11.2.5	Убедитесь, что все криптографические модули обеспечивают безопасное завершение работы при ошибках, а обработка ошибок реализована таким образом, чтобы не создавать уязвимостей, например, таких как атаки Padding Oracle.	3

V11.3 Алгоритмы шифрования

Алгоритмы аутентифицированного шифрования на основе AES и ChaCha20 составляют основу современной криптографической практики.

#	Описание	Уровень
11.3.1	Убедитесь, что не используются небезопасные режимы блочного шифрования (н-р, ECB) и слабые схемы дополнения (padding) (н-р, PKCS#1 v1.5).	1
11.3.2	Убедитесь, что используются только утверждённые шифры и режимы, такие как AES с GCM.	1

#	Описание	Уровень
11.3.3	Убедитесь, что зашифрованные данные защищены от несанкционированного изменения, предпочтительно с использованием утверждённого метода аутентифицированного шифрования или путём комбинирования утверждённого метода шифрования с утверждённым алгоритмом MAC.	2
11.3.4	Убедитесь, что случайные числа (nonce), векторы инициализации (IV) и другие одноразовые числа не используются более одного раза для одной и той же пары ключа шифрования и элемента данных. Метод их генерации должен соответствовать требованиям используемого алгоритма.	3
11.3.5	Убедитесь, что любая комбинация алгоритма шифрования и алгоритма MAC работает в режиме encrypt-then-MAC (EtM).	3

V11.4 Хеширование и функции на основе хеширования

Криптографические хеш-функции применяются во множестве криптографических протоколов, таких как цифровые подписи, HMAC, функции генерации ключа (KDF), генерация случайных битов и хранение паролей. Безопасность криптографической системы напрямую зависит от стойкости используемых хеш-функций. В этом разделе изложены требования к использованию надёжных хеш-функций в криптографических операциях.

Для рекомендаций по хранению паролей, вместе с Приложениием C - Стандарты криптографии, рекомендуем использовать OWASP Password Storage Cheat Sheet

#	Описание	Уровень
11.4.1	Убедитесь, что только утвержденные хэш-функции используются для стандартных сценариев использования в криптографии, включая цифровую подпись, HMAC, KDF, генерацию случайных чисел. Запрещено использовать неустойчивые хэш-функции, (н-р, MD5) для любого использования в криптографии.	1
11.4.2	Убедитесь, что пароли хранятся с использованием утверждённой, вычислительно затратной функции генерации ключа (также известной как «функция хеширования паролей»), с параметрами, настроенными в соответствии с актуальными рекомендациями. Настройки должны обеспечивать баланс между безопасностью и производительностью, чтобы сделать атаки перебором достаточно сложными для требуемого уровня безопасности.	2

#	Описание	Уровень
11.4.3	Убедитесь, что хэш-функции, используемые в формировании цифровой подписи, как часть аутентификации и целостности данных, устойчивы к коллизиям и имеют достаточную длину.	2
11.4.4	Убедитесь, что приложение использует утвержденную функцию генерации ключа с использованием параметров расширения ключа (key stretching) при генерации ключа из пароля. Параметры должны быть сбалансированы с точки зрения безопасности, производительности и устойчивости к атакам полного перебора при компрометации результирующего криптографического ключа.	2

V11.5 Случайные значения

Криптографически стойкая генерация псевдослучайных чисел (Cryptographically secure Pseudo-random Number Generation (CSPRNG)) чрезвычайно сложна для правильной реализации. Как правило, хорошие источники энтропии в системе быстро истощаются при чрезмерном использовании, тогда как источники с меньшей степенью случайности могут приводить к предсказуемым ключам и секретам.

#	Описание	Уровень
11.5.1	Убедитесь, что все случайные числа и строки, которые должны быть непредсказуемы, должны быть сгенерированы с помощью CSPRNG и иметь хотя бы 128 бит энтропии. Учтите, что UUID не удовлетворяют данному условию.	2
11.5.2	Убедитесь, что механизм генерации случайных чисел является надежным даже при высокой нагрузке.	3

V11.6 Криптография с публичным ключом

Криптография с публичным ключом используется там, где невозможно или нежелательно распределять ключ между несколькими сторонами.

Также существует потребность в утвержденных методах обмена ключами, таких как Diffie-Hellman (DH) и Elliptic Curve Diffie-Hellman (ECDH), чтобы убедиться, что криптосистема остается безопасной к современным угрозам. Раздел «Безопасная коммуникация» содержит требования для TLS, так что требования данной секции применимы в ситуациях, где криптография с открытым ключом используется отдельно от TLS.

#	Описание	Уровень
11.6.1	Убедитесь, что только утвержденные криптографические алгоритмы и режимы операций используются для генерации ключа и инициализации случайного значения для его генерации (seeding), а также для генерации цифровой подписи и верификации. Алгоритмы генерации ключей не должны генерировать небезопасные ключи уязвимые к известным атакам, например, ключи RSA уязвимы к факторизации Ферма.	2
11.6.2	Убедитесь, что для обмена ключами используются утверждённые криптографические алгоритмы (например, Диффи-Хеллман) с особым вниманием к обеспечению использования безопасных параметров. Это предотвратит атаки на процесс установления ключа, которые могут привести к атакам «человек посередине» или криптографическим взломам.	3

V11.7 Шифрование данных в процессе использования

Защита данных во время их обработки имеет первостепенное значение. Рекомендуется использовать такие методы, как полное шифрование памяти, шифрование данных при передаче, а также обеспечение как можно более быстрого шифрования данных после их использования.

#	Описание	Уровень
11.7.1	Убедитесь, что используется полное шифрование памяти, которое защищает конфиденциальные данные во время их использования, предотвращая доступ к ним неавторизованных пользователей или процессов.	3
11.7.2	Убедитесь, что минимизация данных обеспечивает раскрытие минимально необходимого объёма данных во время обработки, а также что данные шифруются сразу после использования или как можно скорее.	3

Ссылки

Для дополнительной информации см. также:

- OWASP Web Security Testing Guide: Testing for Weak Cryptography
- OWASP Cryptographic Storage Cheat Sheet

- FIPS 140-3
- NIST SP 800-57

V12 Безопасная коммуникация

Задачи контроля

В этой главе изложены требования, связанные с механизмами защиты данных при передаче, как между клиентом конечного пользователя и серверной частью, так и между компонентами внутренней сервисной архитектуры.

Основные концепции главы:

- Шифрование внешних коммуникаций, а в идеале —и внутри системы.
- Настройку механизмов шифрования в соответствии с актуальными рекомендациями, включая предпочтительные алгоритмы и шифры.
- Использование подписанных сертификатов для защиты от перехвата данных третьими лицами.

Кроме того, помимо общих принципов и лучших практик, ASVS предоставляет более глубокую техническую информацию о криптографической стойкости в Приложении C - Стандарты криптографии.

V12.1 Общие рекомендации по безопасности TLS

В этом разделе приводятся первоначальные рекомендации по обеспечению безопасности TLS-соединений. Для постоянного контроля конфигурации TLS следует использовать актуальные инструменты.

Хотя использование универсальных (wildcard) TLS-сертификатов само по себе не является небезопасным, компрометация такого сертификата, который используется во всех управляемых окружениях (например, в production, staging, development, test), может привести к нарушению безопасности приложений, использующих этот сертификат. Поэтому при возможности рекомендуется применять надёжную защиту, управление и использовать отдельные TLS-сертификаты для разных сред.

#	Описание	Уровень
12.1.1	Убедитесь, что включены только актуальные и рекомендуемые версии протокола TLS, такие как TLS 1.2 и TLS 1.3. Наиболее свежая версия TLS должна быть выбрана как предпочтительная.	1

#	Описание	Уровень
12.1.2	Убедитесь, что включены рекомендуемые шифронаборы (cipher suites), предпочтение также стоит отдать самому сильному шифронабору. L3 приложения должны поддерживать только те наборы шифров (cipher suites), которые обеспечивают прямую (forward) секретность. Пояснение: Прямая секретность (Forward Secrecy, FS) —это криптографическая особенность, при которой сессионные ключи уникальны для каждой сессии и не зависят от долгосрочных ключей. Это гарантирует, что даже при компрометации долгосрочного ключа злоумышленник не сможет расшифровать ранее перехваченные зашифрованные сессии.	2
12.1.3	Убедитесь, что приложение проверяет доверенность клиентских сертификатов mTLS, прежде чем использовать идентификацию по сертификату для аутентификации или авторизации.	2
12.1.4	Убедитесь, что включена и настроена правильная проверка отзыва сертификатов, такая как Online Certificate Status Protocol (OCSP) Stapling.	3
12.1.5	Убедитесь, что в настройках TLS приложения включена функция Encrypted Client Hello (ECH) для предотвращения раскрытия конфиденциальных метаданных, таких как Server Name Indication (SNI), во время процесса TLS-рукопожатия.	3

V12.2 HTTPS коммуникация с внешними сервисами

Убедитесь, что весь HTTP-трафик к внешним сервисам, которые предоставляет приложение, передавался в зашифрованном виде с использованием публично доверенных сертификатов.

#	Описание	Уровень
12.2.1	Убедитесь, что для всех соединений между клиентом и внешними HTTP-сервисами используется TLS и не происходит отката к небезопасным или незашифрованным протоколам.	1
12.2.2	Убедитесь, что внешние сервисы используют публично доверенные TLS-сертификаты.	1

V12.3 Общая безопасность межсервисного взаимодействия

Все взаимодействия между серверами (включая внутренние и внешние подключения) должны быть защищены, предпочтительно с применением TLS-шифрования. Соединения с

другими системами также должны быть защищены, желательно с использованием TLS.

#	Описание	Уровень
12.3.1	Убедитесь, что для всех входящих и исходящих соединений приложения используется зашифрованный протокол, например TLS, включая системы мониторинга, инструменты управления, удалённый доступ и SSH, промежуточное ПО, базы данных, мейнфреймы, партнёрские системы и внешние API. Сервер не должен допускать откат к небезопасным или незашифрованным протоколам.	2
12.3.2	Убедитесь, что TLS-клиент проверяет сертификат сервера до установления защищенного соединения с TLS-сервером.	2
12.3.3	Убедитесь, что все внутренние HTTP-взаимодействия между компонентами приложения используют TLS или другой подходящий механизм транспортного шифрования, и не происходит откат к небезопасным или незашифрованным коммуникациям.	2
12.3.4	Убедитесь, что TLS-соединения между внутренними сервисами используют доверенные сертификаты. Если используются внутренние или самоподписанные сертификаты, потребляющий сервис должен быть настроен на доверие только к конкретным внутренним центрам сертификации (CA) и конкретным самоподписанным сертификатам.	2
12.3.5	Убедитесь, что сервисы, взаимодействующие внутри системы (intra-service communications), используют строгую аутентификацию для проверки каждого конечного узла. Для проверки подлинности взаимодействующих сторон должны применяться методы строгой аутентификации, такие как аутентификация TLS-клиента, с использованием инфраструктуры открытых ключей и механизмов, устойчивых к атакам повтора (replay attacks). Для микросервисных архитектур рекомендуется использование service mesh для упрощения управления сертификатами и повышения безопасности.	3

Ссылки

Для дополнительной информации см. также:

- OWASP - Transport Layer Security Cheat Sheet
- Mozilla’s Server Side TLS configuration guide
- Mozilla’s tool to generate known good TLS configurations.
- O-Saft - OWASP Project to validate TLS configuration

V13 Конфигурация

Задачи контроля

Стандартная конфигурация приложения должна быть безопасной для использования в Интернете.

В этой главе приводятся рекомендации по различным настройкам, необходимым для достижения этого, включая те, которые применяются на этапах разработки, сборки и развертывания.

Рассматриваемые темы включают предотвращение утечек данных, безопасное управление коммуникациями между компонентами и защиту секретов.

V13.1 Документация конфигурации

Этот раздел описывает требования к документации, связанные с тем, как приложение взаимодействует с внутренними и внешними сервисами, а также методы предотвращения потери доступности из-за недоступности сервисов. Также рассматриваются вопросы, касающиеся документации секретных данных.

#	Описание	Уровень
13.1.1	Убедитесь, что все необходимые доступы приложения задокументированы. Сюда должны быть включены как внешние сервисы, от которых зависит приложение, так и сценарии, когда конечный пользователь может задавать адреса для внешнего подключения.	2
13.1.2	Убедитесь, что для каждого сервиса, используемого приложением, в документации определено максимальное количество одновременных соединений (например, лимиты пула соединений) и описано поведение приложения при достижении этого лимита, включая любые механизмы отказоустойчивости или восстановления, чтобы предотвратить ситуации отказа в обслуживании (DoS).	3

#	Описание	Уровень
13.1.3	Убедитесь, что в документации приложения определены стратегии управления ресурсами для каждой внешней системы или сервиса, которые оно использует (например, базы данных, файловые дескрипторы, потоки, HTTP-соединения). Стратегии должны включать процедуры освобождения ресурсов, настройки таймаутов, обработку сбоев, а также, если реализована логика повторных попыток, указание лимитов, задержек и алгоритмов back-off. Для синхронных HTTP-запросов должны быть заданы короткие таймауты, а повторные запросы должны быть отключены, либо строго ограничены по числу попыток, чтобы предотвратить каскадные задержки и истощение ресурсов.	3
13.1.4	Убедитесь, что в документации приложения определены секреты, критически важные для безопасности приложения, а также установлен график их ротации в соответствии с моделью угроз организации и её бизнес-требованиями.	3

V13.2 Конфигурация бэкенд соединений

Приложения взаимодействуют с множеством сервисов, включая API, базы данных и другие компоненты. Эти сервисы могут считаться внутренними для приложения, но не включёнными в стандартные механизмы контроля доступа приложения, либо быть полностью внешними. В любом случае необходимо настроить приложение для безопасного взаимодействия с этими компонентами и, при необходимости, защитить соответствующую конфигурацию.

Важно: Глава «Безопасная коммуникация» содержит рекомендации по шифрованию данных при передаче.

#	Описание	Уровень
13.2.1	Убедитесь, что коммуникации между компонентами бэкенда приложения (API, промежуточное ПО и уровни данных,), которые не поддерживают стандартный механизм пользовательских сессий, проходят с аутентификацией. Аутентификация должна использовать индивидуальные сервисные аккаунты, краткосрочные токены или аутентификацию на основе сертификатов, а не постоянные учётные данные, такие как пароли, API-ключи или общие аккаунты с привилегированным доступом.	2

#	Описание	Уровень
13.2.2	Убедитесь, что коммуникации между компонентами бэкенда приложения, включая локальные или системные службы, API, промежуточное ПО и уровни данных, выполняются с использованием аккаунтов, которым назначены минимально необходимые привилегии.	2
13.2.3	Убедитесь, что если для аутентификации в сервисе используются учётные данные, то используемые потребителем учётные данные не являются стандартными (например, root/root или admin/admin).	2
13.2.4	Убедитесь, что используется белый список (allowlist) для определения внешних ресурсов или систем, с которыми приложению разрешено взаимодействовать (например, для исходящих запросов, загрузки данных или доступа к файлам). Этот белый список может быть реализован на уровне приложения, веб-сервера, межсетевого экрана или комбинации различных уровней.	2
13.2.5	Убедитесь, что веб- или прикладной сервер настроен с белым списком ресурсов или систем, к которым сервер может отправлять запросы или из которых может загружать данные и файлы.	2
13.2.6	Убедитесь, что при подключении приложения к другим сервисам соблюдается задокументированная конфигурация для каждого соединения, включая максимальное количество параллельных соединений, поведение при достижении их максимума, таймауты и стратегии повторных попыток подключения.	3

V13.3 Управление секретами

Управление секретами —это важная задача конфигурации, необходимая для обеспечения защиты данных, используемых в приложении. Конкретные требования к криптографии изложены в главе «Криптография», а данный раздел сосредоточен на аспектах управления и обработки секретов.

#	Описание	Уровень
13.3.1	Убедитесь, что для безопасного создания, хранения, контроля доступа и уничтожения секретов бэкенда используется решение для управления секретами, например, хранилище ключей (key vault). Секретами могут быть пароли, ключевой материал, интеграции с базами данных и сторонними системами, ключи и seed-значения для токенов с ограниченным временем действия, другие внутренние секреты и API-ключи. Секреты не должны включаться в исходный код приложения или артефакты сборки. Для приложений уровня L3 необходимо использовать аппаратное решение, например, аппаратный модуль безопасности (HSM).	2
13.3.2	Убедитесь, что доступ к секретным ресурсам осуществляется в соответствии с принципом минимально необходимых привилегий (least privilege).	2
13.3.3	Убедитесь, что все криптографические операции выполняются с использованием изолированного модуля безопасности (например, хранилища ключей или аппаратного модуля безопасности), чтобы надежно управлять ключевым материалом и защищать его от утечки за пределы модуля безопасности.	3
13.3.4	Убедитесь, что секреты настроены на срок действия и ротацию в соответствии с документацией приложения.	3

V13.4 Непреднамеренная утечка информации

Конфигурации в промышленной среде должны быть усилены, чтобы избежать раскрытия избыточных данных. Многие из подобных проблем редко оцениваются как значительные риски, но часто используются в связке с другими уязвимостями. Если таких проблем нет по умолчанию, это повышает уровень защиты приложения.

Например, скрытие версии серверных компонентов не отменяет необходимости обновлять все компоненты, а отключение отображения содержимого папок не исключает необходимости использовать механизмы авторизации или хранить файлы вне публичных каталогов, но всё это повышает уровень безопасности.

#	Описание	Уровень
13.4.1	Убедитесь, что приложение развернуто либо без метаданных систем контроля версий (включая папки .git или .svn), либо так, чтобы эти папки были недоступны как извне, так и для самого приложения.	1

#	Описание	Уровень
13.4.2	Убедитесь, что режимы отладки отключены для всех компонентов в промышленной среде, чтобы предотвратить раскрытие функций отладки и утечку информации.	2
13.4.3	Убедитесь, что веб-серверы не показывают клиентам списки содержимого каталогов, если это явно не предусмотрено.	2
13.4.4	Убедитесь, что метод HTTP TRACE не поддерживается в продуктовой среде, чтобы избежать потенциальной утечки информации.	2
13.4.5	Убедитесь, что документация (например, по внутренним API) и мониторинговые эндпоинты не доступны, если это явно не предусмотрено.	2
13.4.6	Убедитесь, что приложение не раскрывает подробную информацию о версиях компонентов бэкенда.	3
13.4.7	Убедитесь, что веб-слой настроен на обслуживание только файлов с определёнными расширениями, чтобы предотвратить непреднамеренную утечку информации, конфигураций и исходного кода.	3

Ссылки

Для дополнительной информации см. также:

- OWASP Web Security Testing Guide: Configuration and Deployment Management Testing

V14 Защита информации

Задачи контроля

Приложения не могут предусмотреть все сценарии использования и поведение пользователей, поэтому они должны реализовывать механизмы, ограничивающие несанкционированный доступ к конфиденциальным данным на устройствах клиентов.

В эту главу входят требования, связанные с определением перечня данных, подлежащих защите, методы их защиты, а также конкретные механизмы реализации и ошибки, которых следует избегать.

Ещё один важный аспект защиты данных —предотвращение массового извлечения, изменения или чрезмерного использования данных. Требования для каждой системы, вероятно, будут сильно различаться, поэтому определение «аномального» поведения должно

основываться на модели угроз и бизнес-рисках. С точки зрения ASVS, обнаружение таких проблем рассматривается в главе «Журналирование событий безопасности и обработка ошибок», а установка ограничений —в главе «Валидация и бизнес-логика».

V14.1 Документация по защите информации

Ключевым условием для защиты данных является классификация данных по уровню конфиденциальности. Данные следует классифицировать по нескольким уровням конфиденциальности, где для каждого уровня требуются разные меры защиты.

Существует множество законов и нормативных актов в области защиты персональных данных, которые определяют то, как приложения должны подходить к хранению, использованию и передаче конфиденциальных данных. Этот раздел не пытается дублировать такие законы и нормы, а сосредоточен на ключевых технических аспектах защиты чувствительных данных. При работе с данными следует учитывать требования местного законодательства и нормативных актов, а при необходимости консультироваться с профильным специалистом по защите данных или юристом.

#	Описание	Уровень
14.1.1	Убедитесь, что все конфиденциальные данные, создаваемые и обрабатываемые приложением, идентифицированы и классифицированы по уровням защиты. Включая закодированные данные, которые легко декодируются, например, строки Base64 или незашифрованные данные в полезной нагрузке JWT. Уровни защиты должны учитывать требования по защите данных и конфиденциальности, которым должно соответствовать приложение.	2
14.1.2	Убедитесь, что для каждого уровня защиты конфиденциальных данных требования к защите задокументированы. Они должны включать (но не ограничиваться) требования по шифрованию, проверке целостности, срокам хранения, способам логирования данных, контролю доступа к чувствительным данным в логах, шифрованию на уровне базы данных, применению технологий повышения конфиденциальности, а также другим требованиям по конфиденциальности.	2

V14.2 Базовые меры по защите информации

Этот раздел содержит различные практические требования, связанные с защитой данных. Большинство из них направлены на непреднамеренную утечку данных, но также есть общее

требование —реализовывать меры защиты в зависимости от необходимого уровня защиты для каждого типа данных.

#	Описание	Уровень
14.2.1	Убедитесь, что конфиденциальные данные передаются на сервер только в теле HTTP-запроса или в заголовках. URL и строка запроса не должны содержать конфиденциальную информацию, такую как API-ключ или сессионный токен.	1
14.2.2	Убедитесь, что приложение предотвращает кэширование конфиденциальных данных в серверных компонентах, таких как балансировщики нагрузки и кэши приложений, либо гарантирует их надежное удаление после использования.	2
14.2.3	Убедитесь, что заранее определённые конфиденциальные данные не отправляются ненадёжным сторонам (например, системам трекинга пользователей), чтобы предотвратить нежелательный сбор данных вне контроля приложения.	2
14.2.4	Убедитесь, что для конфиденциальных данных реализованы меры по шифрованию, проверке целостности, срокам хранения, способам логирования, контролю доступа к данным в логах, а также применяются технологии повышения конфиденциальности в соответствии с документацией для соответствующего уровня защиты данных.	2
14.2.5	Убедитесь, что механизмы кэширования настроены так, чтобы кэшировать только ответы с ожидаемым типом содержимого для данного ресурса и не содержать конфиденциального динамического контента. При обращении к несуществующему файлу, веб-сервер должен возвращать ответ 404 или 302, а не подменять на другой существующий файл, что позволит предотвратить атаки типа Web Cache Deception.	3
14.2.6	Убедитесь, что приложение возвращает только минимально необходимый объем конфиденциальных данных для выполнения своей функциональности. Например, показывает только часть цифр номера кредитной карты, а не полный номер. Если требуется отображение полных данных, они должны быть замаскированы в пользовательском интерфейсе, пока пользователь специально не запросил их просмотр.	3

#	Описание	Уровень
14.2.7	Убедитесь, что конфиденциальная информация подлежит классификации по срокам хранения, чтобы устаревшие или ненужные данные удалялись автоматически, по расписанию или по мере необходимости.	3
14.2.8	Убедитесь, что конфиденциальная информация удаляется из метаданных файлов, загружаемых пользователями, если пользователь явно не согласился на ее хранение.	3

V14.3 Защита данных на стороне клиента

В этом разделе содержатся требования, направленные на предотвращение утечки данных на стороне клиента или user agent приложения.

#	Описание	Уровень
14.3.1	Убедитесь, что аутентифицированные данные очищаются из клиентского хранилища, например, из DOM браузера, после завершения работы клиента или сессии. Заголовок HTTP-ответа Clear-Site-Data может помочь в этом, но клиентская сторона также должна уметь очищать данные, если соединение с сервером недоступно при завершении сессии.	1
14.3.2	Убедитесь, что приложение устанавливает HTTP-заголовки, достаточные для предотвращения кэширования (например, Cache-Control: no-store), чтобы конфиденциальные данные не сохранялись в браузерах.	2
14.3.3	Убедитесь, что данные, хранящиеся в браузерном хранилище (например, localStorage, sessionStorage, IndexedDB или cookie), не содержат конфиденциальной информации, за исключением сессионных токенов.	2

Ссылки

Для дополнительной информации см. также:

- Consider using the Security Headers website to check security and anti-caching header fields
- Documentation about anti-caching headers by Mozilla
- OWASP Secure Headers project

- OWASP Privacy Risks Project
- OWASP User Privacy Protection Cheat Sheet
- Australian Privacy Principle 11 - Security of personal information
- European Union General Data Protection Regulation (GDPR) overview
- European Union Data Protection Supervisor - Internet Privacy Engineering Network
- Information on the “Clear-Site-Data”header
- White paper on Web Cache Deception

V15 Безопасная разработка и архитектура

Задачи контроля

Многие требования ASVS относятся либо к конкретной области безопасности, такой как аутентификация или авторизация, либо к определённому типу функциональности приложения, например, к логированию или работе с файлами.

В этой главе представлены общие требования безопасности, которые следует учитывать при проектировании и разработке приложений. Эти требования касаются не только чистоты архитектуры и качества кода, но и конкретных практик проектирования и программирования, необходимых для обеспечения безопасности приложений.

V15.1 Документация по безопасной разработке и архитектуре

Многие требования по созданию безопасной и защищенной архитектуры зависят от четкого документирования решений, принятых в отношении реализации конкретных механизмов безопасности и компонентов, используемых в приложении.

В этом разделе излагаются требования к документации, включая определение компонентов, которые считаются содержащими «опасную функциональность» или «рискованными компонентами».

Компонент с «опасной функциональностью» может быть разработан внутри компании или представлять собой внешнюю зависимость, который выполняет такие операции, как десериализация ненадежных данных, парсинг необработанных файлов или двоичных данных, динамическое выполнение кода или непосредственное управление памятью. Уязвимости в таких операциях создают высокий риск компрометации приложения и его инфраструктуры.

«Компонент с повышенным риском» —это сторонняя библиотека (т.е. не разработанная внутри компании) с отсутствующими или неэффективно реализованными средствами контроля безопасности в отношении процессов разработки или функциональности. Примерами служат компоненты, которые не поддерживаются, находятся на этапе завершения жизненного цикла или ранее содержали серьезные уязвимости.

В этом разделе также подчеркивается важность определения соответствующих сроков устранения уязвимостей в сторонних компонентах.

#	Описание	Уровень
15.1.1	Убедитесь, что в документации по приложению определены сроки устранения уязвимостей на основе оценки рисков для внешних зависимостей с уязвимостями и для обновления библиотек в целом, чтобы минимизировать риск, связанный с этими компонентами.	1
15.1.2	Убедитесь, что ведется реестр всех используемых сторонних библиотек (например, Software Bill of Materials, SBOM), с обязательной проверкой, что компоненты получены из предопределенных, доверенных и регулярно поддерживаемых репозиториев.	2
15.1.3	Убедитесь, что в документации приложения указан функционал, требующий значительных временных или вычислительных ресурсов. Необходимо указать, как предотвратить потерю доступности из-за чрезмерного использования этого функционала и как избежать ситуаций, когда формирование ответа занимает больше времени, чем установленный системой таймаут. Возможные меры защиты могут включать асинхронную обработку, использование очередей и ограничение количества параллельных процессов для каждого пользователя и приложения.	2
15.1.4	Убедитесь, что в документации приложения указаны внешние библиотеки, которые считаются «компонентами с повышенным риском».	3
15.1.5	Убедитесь, что в документации приложения выделены те части приложения, в которых используются «функционал с повышенным риском».	3

V15.2 Архитектура безопасности и зависимости

В этом разделе содержатся требования по работе с уязвимыми, устаревшими и небезопасными зависимостями и компонентами посредством управления зависимостями.

Он также включает использование методов архитектурного уровня, таких как «песочница», инкапсуляция, контейнеризация и сетевая изоляция, для снижения влияния использования «операций с повышенным риском» и «компонентов с повышенным риском» (как определено в предыдущем разделе) и предотвращения потери доступности из-за чрезмерного использования ресурсоемких функций.

#	Описание	Уровень
15.2.1	Убедитесь, что приложение содержит только те компоненты, которые не нарушили задокументированные сроки обновления и исправления.	1
15.2.2	Убедитесь, что в приложении реализованы меры защиты от потери доступности из-за функциональности, требующей много времени или ресурсов, на основе задокументированных решений и стратегий безопасности для этого.	2
15.2.3	Убедитесь, что производственная среда включает только те функции, которые необходимы для работы приложения, и не предоставляет лишние функции, такие как тестовый код, примеры кода и инструменты разработки.	2
15.2.4	Убедитесь, что сторонние компоненты и все их транзитивные зависимости загружаются только из ожидаемых репозиториев (внутренних или внешних) и отсутствует риск атаки через подмену зависимостей (dependency confusion attack)	3
15.2.5	Убедитесь, что приложение реализует дополнительные меры защиты для тех частей приложения, которые документированы как содержащие «функционал с повышенным риском» или использующие сторонние библиотеки, считающиеся «функционалом с повышенным риском». Это могут быть такие методы, как «песочница», инкапсуляция, контейнеризация или изоляция на сетевом уровне, чтобы задержать и предотвратить продвижение (pivoting) злоумышленников, скомпрометировавших одну часть приложения, к другим частям приложения.	3

V15.3 Безопасное программирование

В этом разделе рассматриваются типы уязвимостей, включая type juggling, prototype pollution и другие, возникающие из-за использования небезопасных паттернов программирования на конкретном языке. Некоторые уязвимости актуальны не для всех языков, тогда как другие требуют исправлений, зависящих от конкретного языка программирования, или связаны с особенностями обработки функций (например, HTTP-параметров) в конкретном языке или фреймворке. Также рассматривается риск установки обновлений приложения без криптографической проверки.

В данном разделе также рассматриваются риски, связанные с использованием объектов для представления данных и их передачей через внешние API. В этом случае приложение должно гарантировать, что поля данных, которые не должны быть доступны для записи, не будут изменены пользовательским вводом (mass assignment), и что API избирательно определяет,

какие поля данных возвращаются. Если доступ к полю зависит от прав пользователя, это следует рассматривать в контексте требования к контролю доступа на уровне полей в главе «Авторизация».

#	Описание	Уровень
15.3.1	Убедитесь, что приложение возвращает только необходимое подмножество полей из объекта данных. Например, оно не должно возвращать весь объект данных, поскольку некоторые отдельные поля должны быть недоступны пользователям.	1
15.3.2	Убедитесь, что при вызовах внешних URL-адресов бэкэнд приложения настроен так, чтобы не выполнять перенаправления, если только это не предусмотрено функционалом.	2
15.3.3	Убедитесь, что в приложении реализованы защитные механизмы против атак типа массового присвоения (mass assignment) путем строгого ограничения разрешенных полей для каждого контроллера и действия, исключая возможность добавления или изменения значений полей, которые не должны быть доступны для данной конкретной операции.	2
15.3.4	Убедитесь, что все прокси и промежуточное ПО корректно передают исходный IP-адрес пользователя, используя доверенные поля данных, которые не могут быть изменены конечным пользователем, а приложение и веб-сервер используют это правильное значение для ведения логирования и принятия решений по безопасности, таких как ограничение запросов. При этом следует учитывать, что даже исходный IP-адрес может быть ненадежным из-за динамических IP-адресов, VPN или корпоративных файрволов.	2
15.3.5	Убедитесь, что приложение явно проверяет соответствие переменных ожидаемому типу данных и использует строгие операции сравнения. Это необходимо для предотвращения уязвимостей, связанных с type juggling и type confusion, вызванных тем, что код приложения делает предположения о типе переменной.	2
15.3.6	Убедитесь, что JavaScript-код написан таким образом, чтобы предотвратить prototype pollution, например, используя Set() или Map() вместо обычных литералов объекта.	2
15.3.7	Убедитесь, что приложение защищено от атак типа HTTP Parameter Pollution (HPP), особенно если используемый фреймворк не различает источники параметров запроса (строка запроса, тело запроса, куки или заголовки).	2

V15.4 Безопасная работа с параллельными процессами

Проблемы безопасности работы параллельных процессов, такие как состояния гонки (race conditions), уязвимости типа «проблем синхронизации проверки и использования ресурсов» (TOCTOU), взаимные блокировки (deadlocks), постоянно повторяющиеся блокировки (livelocks), нехватка ресурсов у потоков (thread starvation) и некорректная синхронизация (improper synchronization), могут приводить к неожиданному поведению системы и создавать угрозы безопасности. В этом разделе рассматриваются различные методы и стратегии, помогающие снизить эти риски.

#	Описание	Уровень
15.4.1	Убедитесь, что к общим объектам в многопоточном коде (например, кэшам, файлам или объектам в памяти, используемым несколькими потоками) используется безопасный многопоточный доступ и механизмов синхронизации, таких как блокировки или семафоры, чтобы избежать состояний гонки и повреждения данных.	3
15.4.2	Убедитесь, что проверки состояния ресурса, например его существования или разрешений, а также зависящие от него действия выполняются как одна атомарная операция, чтобы предотвратить условия гонки времени проверки и времени использования (TOCTOU). Например, проверка существования файла перед его открытием или проверка прав доступа пользователя перед его предоставлением.	3
15.4.3	Убедитесь, что блокировки применяются согласованно, чтобы предотвратить взаимные блокировки потоков (deadlocks) или их бесконечные повторы, а также что логика блокировки инкапсулирована в коде, отвечающем за управление ресурсом. Это гарантирует, что блокировки не могут быть случайно или намеренно изменены внешними классами или кодом.	3
15.4.4	Убедитесь, что политики распределения ресурсов предотвращают нехватку ресурсов у потоков, обеспечивая справедливый доступ к ресурсам, например, используя пулы потоков, позволяя потокам с более низким приоритетом выполняться в течение разумного периода времени.	3

Ссылки

Для дополнительной информации см. также:

- OWASP Prototype Pollution Prevention Cheat Sheet
- OWASP Mass Assignment Prevention Cheat Sheet

- OWASP CycloneDX Bill of Materials Specification
- OWASP Web Security Testing Guide: Testing for HTTP Parameter Pollution

V16 Журналирование событий безопасности и обработка ошибок

Задачи контроля

Журналы безопасности отличаются от журналов ошибок или производительности и служат для записи событий, связанных с безопасностью, таких как решения по аутентификации, контроль доступа или попытки обхода механизмов защиты (например, проверка входных данных или бизнес-логики). Их основная задача —обеспечить обнаружение, реагирование и расследование инцидентов за счёт предоставления структурированных данных с высокой информативной ценностью для аналитических систем, таких как SIEM.

Журналы не должны содержать конфиденциальные персональные данные, если это не предусмотрено законом, а все залогированные сведения должны защищаться как актив высокой важности. Процесс журналирования не должен нарушать конфиденциальность или безопасность системы. Кроме того, приложения должны отрабатывать ошибки безопасным образом, исключая необоснованное раскрытие информации или disruption (нарушение работы).

Подробные инструкции по внедрению см. в OWASP Cheat Sheets в разделе ссылок.

V16.1 Документация по ведению логов

Этот раздел обеспечивает четкий и полный учет журналирования во всем стеке приложения. Это необходимо для эффективного мониторинга безопасности, реагирования на инциденты и соответствия требованиям.

#	Описание	Уровень
16.1.1	Убедитесь в существовании реестра, в котором документируется ведение журнала на каждом уровне технологического стека приложения, включая какие события регистрируются, форматы логов, где хранятся эти журналы, как они используются, как контролируется доступ к ним и как долго хранятся журналы.	2

V16.2 Общее журналирование

В этом разделе устанавливаются требования, обеспечивающие единообразную структуру журналов безопасности и наличие метаданных. Цель —обеспечить машиночитаемость логов и возможность их анализа в распределённых системах и инструментах мониторинга.

Естественно, события безопасности часто связаны с конфиденциальными данными. Если такие данные регистрируются без должного внимания, сами журналы становятся секретными и, следовательно, подпадают под действие требований шифрования (прим. автора - законодательства), более строгих политик хранения и могут быть раскрыты в ходе аудита.

Крайне важно регистрировать только необходимые данные и обращаться с данными журналов с той же осторожностью, что и с другими конфиденциальными активами.

Приведенные ниже требования устанавливают основополагающие требования к регистрации метаданных, синхронизации, форматированию и контролю.

#	Описание	Уровень
16.2.1	Убедитесь, что каждая запись журнала содержит необходимые метаданные (например, когда, где, кто, что), которые позволят провести подробное исследование временной шкалы, когда произошло событие.	2
16.2.2	Убедитесь, что все источники времени компонентов системы журналирования синхронизированы, а временные метки в метаданных событий безопасности используют UTC или содержат явное указание смещения часового пояса. Рекомендуется применять UTC для обеспечения согласованности в распределённых системах и предотвращения ошибок при переходе на летнее/зимнее время.	2
16.2.3	Убедитесь, что приложение записывает и передаёт логи исключительно в те файлы и сервисы, которые указаны в реестре журналирования.	2
16.2.4	Убедитесь, что логи могут быть прочитаны и коррелированы используемой системой обработки журналов, предпочтительно с использованием общего стандартного формата логирования.	2
16.2.5	Убедитесь, что приложение применяет соответствующие меры защиты при логировании конфиденциальных данных в зависимости от уровня их чувствительности: например, полностью запрещает запись в логи таких данных, как учетные данные или платежные реквизиты, а для других данных (например, сессионных токенов) использует хеширование или частичное/полное маскирование.	2

V16.3 Инциденты безопасности

В этом разделе определяются требования к журналированию событий безопасности в приложении. Фиксация этих событий критически важна для выявления подозрительной

активности, проведения расследований и выполнения нормативных требований.

В этом разделе описываются типы регистрируемых событий, но он не претендует на исчерпывающую информацию. Каждое приложение имеет уникальные факторы риска и эксплуатационный контекст.

Обратите внимание, что, хотя ASVS включает регистрацию событий безопасности в свою область применения, функции оповещения и корреляции (например, правила SIEM или инфраструктура мониторинга) не входят в сферу применения и обрабатываются операционными системами и системами мониторинга.

#	Описание	Уровень
16.3.1	Убедитесь, что все операции аутентификации регистрируются, включая успешные и неудачные попытки. Также следует фиксировать дополнительные метаданные, такие как тип аутентификации или использованные факторы проверки.	2
16.3.2	Убедитесь, что ведется журнал неудачных попыток авторизации. Для уровня L3 это должно включать в себя журналирование всех решений об авторизации, включая журналирование доступа к конфиденциальным данным (без регистрации самих конфиденциальных данных).	2
16.3.3	Убедитесь, что приложение регистрирует все события безопасности, определенные в документации, а также фиксирует попытки обхода механизмов защиты, включая проверку входных данных, бизнес-логики и системы защиты от автоматизированных атак.	2
16.3.4	Убедитесь, что приложение регистрирует в журналах неожиданные ошибки и сбои механизмов безопасности, включая, например, ошибки TLS-соединения на стороне сервера.	2

V16.4 Защита журналов

Журналы представляют собой ценные источники forensic-данных и должны быть надёжно защищены. Если логи можно легко изменить или удалить, они теряют целостность и становятся ненадёжными для расследования инцидентов или судебных разбирательств. Кроме того, журналы могут раскрывать внутреннюю логику работы приложения или содержать конфиденциальные метаданные, что делает их привлекательной целью для злоумышленников.

В данном разделе определены требования, обеспечивающие защиту журналов от несанкционированного доступа, изменений и раскрытия и безопасную передачу и хранение в изолированных защищённых системах.

#	Описание	Уровень
16.4.1	Убедитесь, что все компоненты системы логирования корректно кодируют данные, чтобы предотвратить атаки типа внедрения в журналы (log injection).	2
16.4.2	Убедитесь, что логи защищены от неавторизованного доступа и не могут быть модифицированы.	2
16.4.3	Убедитесь, что логи безопасно передаются в логически изолированную систему для анализа, обнаружения угроз, генерации оповещений и эскалации инцидентов. Это гарантирует сохранность журналов даже в случае компрометации основного приложения.	2

V16.5 Обработка ошибок

Данный раздел определяет требования к безопасному и корректному завершению работы приложений в случае сбоев, исключающему раскрытие конфиденциальной внутренней информации.

#	Описание	Уровень
16.5.1	Убедитесь, что при возникновении непредвиденной или связанной с безопасностью ошибки пользователю возвращается только общее сообщение, гарантируя отсутствие раскрытия конфиденциальных внутренних данных системы, таких как трассировки стека, запросы, секретные ключи и токены.	2
16.5.2	Убедитесь, что приложение продолжает работать в безопасном режиме при недоступности внешних ресурсов, используя такие механизмы, как автоматические выключатели (circuit breakers) или плавное снижение функциональности (graceful degradation).	2
16.5.3	Убедитесь, что приложение корректно и безопасно завершает работу при возникновении исключений, предотвращая опасные сценарии по типу «fail-open», когда, например, транзакция ошибочно выполняется, несмотря на ошибки в логике валидации.	2

#	Описание	Уровень
16.5.4	Убедитесь, что определён обработчик ошибок «крайние меры», который будет перехватывать все необработанные ошибки. Это необходимо как для предотвращения потери информации об ошибках, которая должна быть записана в файлы журналов, так и для того, чтобы ошибка не остановила весь процесс приложения, что привело бы к потере доступности.	3

Примечание: Некоторые языки программирования (включая Swift, Go и многие функциональные языки в силу их парадигмы) не поддерживают исключения или обработчики событий «крайней меры». В таких случаях архитекторам и разработчикам следует использовать подходы, соответствующие паттернам, языковым особенностям или возможностям фреймворка, чтобы гарантировать безопасную обработку исключительных, непредвиденных или связанных с безопасностью событий.

Ссылки

Для дополнительной информации см. также:

- OWASP Web Security Testing Guide: Testing for Error Handling
- OWASP Authentication Cheat Sheet section about error messages
- OWASP Logging Cheat Sheet
- OWASP Application Logging Vocabulary Cheat Sheet

V17 WebRTC

Задачи контроля

Web Real-Time Communication (WebRTC) —технология, обеспечивающая передачу голоса, видео и данных в реальном времени в современных приложениях. По мере роста WebRTC обеспечение безопасности инфраструктуры WebRTC становится критически важным. В данном разделе представлены требования по безопасности для разработчиков, хостинг-провайдеров и интеграторов решений на основе WebRTC.

Рынок WebRTC можно разделить на три основных сегмента:

1. Разработчики продуктов —поставщики, создающие и поставляющие продукты и решения WebRTC как с собственным, так и с открытым исходным кодом. Они специализируются на разработке надежных и безопасных технологий WebRTC, которые могут использоваться другими.

2. Платформы коммуникаций как услуга (CPaaS) —провайдеры, предлагающие API, SDK и инфраструктуру для реализации WebRTC-функциональности. Они могут использовать продукты первой категории или разрабатывать собственное ПО WebRTC для предоставления этих услуг.
3. Поставщики услуг —организации, внедряющие WebRTC-решения (на базе готовых продуктов или собственных разработок) для создания приложений для онлайн-конференций, здравоохранения, электронного обучения и других областей, где критически важна коммуникация в режиме реального времени.

Изложенные здесь требования безопасности в первую очередь ориентированы на разработчиков продуктов, CPaaS-провайдеров, поставщиков услуг, которые:

- Используют open-source решения для создания своих WebRTC-приложений.
- Применяют коммерческие WebRTC-продукты как часть своей инфраструктуры.
- Разрабатывают собственные WebRTC-решения или интегрируют различные компоненты в единый сервис.

Важно отметить, что эти требования безопасности не распространяются на разработчиков, использующих исключительно SDK и API, предоставляемые поставщиками CPaaS. Для таких разработчиков CPaaS-провайдеры обычно несут ответственность за обеспечение базовой безопасности своей платформы, и универсальные стандарты безопасности (например, ASVS) могут не полностью покрывать их потребности.

V17.1 TURN-сервер

В этом разделе определяются требования безопасности для систем, использующих собственные серверы TURN (Traversal Using Relays around NAT). Серверы TURN помогают ретранслировать медиаданные в сетевых средах с ограничениями, но могут представлять опасность при неправильной настройке. Эти меры контроля направлены на безопасную фильтрацию адресов и защиту от исчерпания ресурсов.

#	Описание	Уровень
17.1.1	Убедитесь, что сервис TURN (Traversal Using Relays around NAT) разрешает доступ только к IP-адресам, не зарезервированным для специальных целей (например, внутренние сети, широковещательная рассылка, loopback-адреса). Данное требование применяется как к IPv4, так и к IPv6 адресам.	2
17.1.2	Убедитесь, что сервис TURN (Traversal Using Relays around NAT) защищен от исчерпания ресурсов при попытках легитимных пользователей открыть чрезмерное количество портов на сервере.	3

V17.2 Требования к медиасерверам

Данные требования применяются только к системам, использующим собственные WebRTC-медиасерверы, такие как, SFU (Selective Forwarding Units), MCU (Multipoint Control Units), серверы записи, шлюзовые серверы. Медиасерверы обрабатывают и распределяют медиапотоки, поэтому их безопасность критически важна для защиты взаимодействия между участниками. Обеспечение безопасности медиапотоков в WebRTC-приложениях является приоритетной задачей для предотвращения перехвата данных, манипуляций с трафиком, атак типа «отказ в обслуживании». Эти угрозы могут нарушить конфиденциальность пользователей и качество связи.

В частности, необходимо реализовать защиту от флуд-атак, включая: ограничение частоты запросов (rate limiting), проверку временных меток, использование синхронизированных часов для контроля временных интервалов и управление буферами для предотвращения переполнения и поддержания правильной временной синхронизации. Если пакеты для конкретного медиасеанса прибывают слишком быстро, избыточные пакеты должны отбрасываться. Также важно защитить систему от некорректных пакетов путем валидации входных данных, безопасной обработки целочисленных переполнений, предотвращения переполнения буферов и применения других надежных методов обработки ошибок.

Системы, использующие исключительно peer-to-peer медиакоммуникацию между веб-браузерами без промежуточных медиасерверов, не подпадают под эти специфические требования к медиабезопасности.

Данный раздел относится к использованию Datagram Transport Layer Security (DTLS) в контексте WebRTC. Требования к документированной политике управления криптографическими ключами приведены в главе «Криптография». Информацию об одобренных криптографических методах можно найти в Приложении C - Криптография ASVS или в таких документах, как NIST SP 800-52 Rev. 2 или BSI TR-02102-2 (версия 2025-01).

#	Описание	Уровень
17.2.1	Убедитесь, что ключ сертификата Datagram Transport Layer Security (DTLS) управляется и защищается в соответствии с документированной политикой управления криптографическими ключами.	2
17.2.2	Убедитесь, что медиасервер настроен на использование только одобренных наборов шифров Datagram Transport Layer Security (DTLS) и защищенного профиля безопасности для расширения DTLS, применяемого при установке ключей для Secure Real-time Transport Protocol (DTLS-SRTP).	2

#	Описание	Уровень
17.2.3	Убедитесь, что медиасервер проверяет аутентификацию Secure Real-time Transport Protocol (SRTP) для предотвращения атак с внедрением Real-time Transport Protocol (RTP), которые могут привести к отказу в обслуживании (Denial of Service) или вставке поддельного аудио-/видеоконтента в медиапотоки.	2
17.2.4	Убедитесь, что медиасервер способен продолжать обработку входящего медиатрафика при получении некорректных пакетов Secure Real-time Transport Protocol (SRTP).	2
17.2.5	Убедитесь, что медиасервер способен продолжать обработку входящего медиатрафика даже при флуде (массовой атаке) пакетами Secure Real-time Transport Protocol (SRTP) от легитимных пользователей.	3
17.2.6	Убедитесь, что медиасервер не подвержен уязвимости «Гонка ClientHello» (ClientHello Race Condition) в Datagram Transport Layer Security (DTLS), проверив наличие известных уязвимостей в публичных источниках или выполнив тестирование на воспроизведение условий гонки.	3
17.2.7	Убедитесь, что механизмы аудио- и видеозаписи, связанные с медиасервером, способны продолжать обработку входящего медиатрафика даже при флуде (массовой атаке) пакетов Secure Real-time Transport Protocol (SRTP) от легитимных пользователей.	3
17.2.8	Убедитесь, что сертификат Datagram Transport Layer Security (DTLS) проверяется на соответствие атрибуту отпечатка (fingerprint) в Session Description Protocol (SDP), с немедленным прекращением медиапотока в случае несоответствия —это гарантирует подлинность передаваемых данных.	3

V17.3 Сигнализация

Данный раздел устанавливает требования для систем, использующих собственные серверы сигнализации WebRTC. Сигнализация координирует peer-to-peer (P2P) взаимодействие и должна быть устойчива к атакам, способным нарушить установление сеансов или управление соединениями.

Для обеспечения безопасной сигнализации системы должны корректно обрабатывать некорректные входные данные и оставаться доступными под нагрузкой.

#	Описание	Уровень
17.3.1	Убедитесь, что сигнальный сервер способен продолжать обработку легитимных входящих сообщений сигнализации даже в условиях флуд-атаки. Для этого необходимо реализовать механизм ограничения частоты запросов (rate limiting) на уровне сигнализации.	2
17.3.2	Убедитесь, что сигнальный сервер способен продолжать обработку легитимных сообщений сигнализации даже при получении некорректных сообщений, которые могут вызвать отказ в обслуживании (Denial of Service, DoS). Для этого необходимо реализовать: проверку входных данных (input validation), безопасную обработку целочисленных переполнений (integer overflows), защиту от переполнения буферов (buffer overflows) и другие надежные методы обработки ошибок.	2

Ссылки

Для дополнительной информации см. также:

- DoS-атака WebRTC DTLS ClientHello лучше всего документирована в записи блога Enable Security для специалистов по безопасности и соответствующем техническом документе для разработчиков WebRTC
- RFC 3550 - RTP: A Transport Protocol for Real-Time Applications
- RFC 3711 - The Secure Real-time Transport Protocol (SRTP)
- RFC 5764 - Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP))
- RFC 8825 - Overview: Real-Time Protocols for Browser-Based Applications
- RFC 8826 - Security Considerations for WebRTC
- RFC 8827 - WebRTC Security Architecture
- DTLS-SRTP Protection Profiles

Приложение A: Термины и сокращения

- **Absolute Maximum Session Lifetime** (абсолютный максимальный срок действия сессии) —также известный как “Overall Timeout”в NIST, это максимальное время, в течение которого сеанс может оставаться активным после аутентификации, независимо от действий пользователя. Это составная часть механизма завершения сессии.
- **Allowlist** —список разрешённых данных или операций, например список символов, разрешённых форматно-логическим контролем.

- **Anti-forgery token** (токен защиты от подделки) —механизм, при котором один или несколько токенов передаются в запросе и проверяются сервером приложения, чтобы гарантировать, что запрос поступил из ожидаемой конечной точки.
- **Application Security** (безопасность приложений) —это анализ компонентов, составляющих прикладной уровень эталонной модели взаимодействия открытых систем (модели OSI), и в меньшей степени –инфраструктуры, например, операционных систем или сетей.
- **Application Security Verification** (верификация требований к безопасности приложений) —технический анализ приложения по стандарту OWASP ASVS.
- **Application Security Verification Report** —отчёт по тестированию конкретного приложения, в котором аудитор документирует общие результаты и даёт сопроводительные пояснения.
- **Authentication** (аутентификация) —проверка учетных данных, предъявленных пользователем приложения.
- **Automated Verification** (автоматизированная проверка) –поиск сигнатур уязвимостей с помощью автоматизированных инструментов динамического и/или статического анализа безопасности кода.
- **Black box testing** (метод «чёрного ящика») —метод тестирования программного обеспечения, который проверяет функциональность приложения, не заглядывая в его внутреннюю структуру и операции.
- **Common Weakness Enumeration** (CWE, список общих недостатков) –разработанный сообществом список распространенных недостатков безопасности программного обеспечения. Он служит общим языком, мерилом для средств обеспечения безопасности программного обеспечения, а также основой для выявления недостатков, их устранения и предотвращения.
- **Component** (компонент) –автономный блок кода со связанными дисковыми и сетевыми интерфейсами, который взаимодействует с другими компонентами.
- **Credential Service Provider** (CSP) —также называемый поставщиком идентификационных данных (IdP). Источник пользовательских данных, который может использоваться другими приложениями для аутентификации.
- **Cross-Site Script Inclusion** (XSSI) —разновидность атаки межсайтового скриптинга (XSS), при которой веб-приложение загружает вредоносный код из внешнего источника и включает его в состав собственного содержимого.
- **Cross-Site Scripting** (XSS, межсайтовый скриптинг) —уязвимость в системе безопасности, обычно обнаруживаемая в web-приложениях, позволяющая внедрять скрипты в контент на стороне клиента.
- **Cryptographic module** (криптографический модуль) —аппаратное, программное и/или микропрограммное обеспечение, реализующее криптографические алгоритмы и/или генерирующее криптографические ключи.
- **Cryptographically secure pseudo-random number generator** (CSPRNG, криптографически стойкий генератор псевдослучайных чисел) —генератор псевдослучайных чисел со свойствами, делающими его пригодным для использования в криптографии, также

называется криптографическим генератором случайных чисел (cryptographic random number generator, CRNG).

- **Datagram Transport Layer Security** (DTLS) —криптографический протокол, обеспечивающий безопасность связи по сетевому соединению. Он основан на протоколе TLS, но адаптирован для защиты ориентированных на датаграммы протоколов (обычно поверх UDP). Версия DTLS 1.3 стандартизирована в RFC 9147.
- **Datagram Transport Layer Security Extension to Establish Keys for the Secure Real-time Transport Protocol** (DTLS-SRTP) —механизм использования DTLS-рукопожатия для формирования ключей сеанса SRTP. Стандартизирован в RFC 5764.
- **Design Verification** (верификация архитектуры) —технический анализ архитектуры безопасности приложения.
- **Dynamic Application Security Testing** (DAST, динамический анализ приложения) — технологии, предназначенные для обнаружения условий, указывающих на уязвимость системы безопасности в приложении, производящийся во время его выполнения.
- **Dynamic Verification** (динамическая верификация) —использование автоматизированных инструментов, использующих сигнатуры уязвимостей для поиска проблем во время выполнения приложения.
- **Fast IDentity Online** (FIDO) —набор стандартов аутентификации, которые позволяют использовать различные методы аутентификации, включая биометрические данные, модули доверенной платформы (TPM), USB-токены и т.д.
- **Hardware Security Module** (HSM, аппаратный модуль безопасности) —аппаратный компонент, способный хранить криптографические ключи и другие секреты в защищённом виде.
- **Hibernate Query Language** (HQL) —язык запросов, внешне похожий на SQL, используемый библиотекой Hibernate ORM.
- **HTTP Strict Transport Security** (HSTS) —политика, которая предписывает браузеру подключаться к домену, возвращающему заголовок, только через TLS и при наличии действительного сертификата. Активируется с помощью поля заголовка ответа Strict-Transport-Security.
- **HyperText Transfer Protocol** (HTTP) —протокол приложений для распределенных, гипермедиа (гипертекст+мультимедиа)-систем и совместной работы. Основа для передачи данных в World Wide Web.
- **HyperText Transfer Protocol over SSL/TLS** (HTTPS) —Способ защиты HTTP-соединения путём его шифрования с использованием TLS.
- **Identity Provider** (IdP) —также называется провайдером услуг аутентификации (CSP) в ссылках NIST. Сущность, которая предоставляет источник аутентификации для других приложений.
- **Inactivity Timeout** (таймаут бездействия) —промежуток времени, в течение которого сессия может оставаться активной при отсутствии взаимодействия пользователя с приложением. Это составная часть механизма завершения сессии.
- **Input Validation** (форматно-логический контроль) —нормализация и проверка

корректности недоверенных входных данных.

- **JSON Web Token** (JWT) –RFC 7519 определяет стандарт для объекта данных JSON, состоящего из раздела заголовка (header), который объясняет, как проверять объект, раздела тела (body), содержащего набор утверждений (claims), и раздела подписи (signature), содержащей цифровую подпись для проверки содержимого раздела тела. Это тип автономного (self-contained) токена.
- **Local File Inclusion** (LFI) –атака, использующая уязвимые процедуры включения файлов в приложении, что приводит к подключению локальных файлов, уже присутствующих на сервере.
- **Malicious Code** (вредоносный код) —код, попавший в приложение во время его разработки без ведома владельца, который обходит имеющуюся политику безопасности приложения. Не путать с вредоносным ПО, таким как вирусы или черви.
- **Malware** (вредоносное ПО) —исполняемый код, который попадает в приложение во время выполнения без ведома пользователя или администратора приложения.
- **Message authentication code** (MAC) —криптографическая контрольная сумма данных, вычисляемая алгоритмом генерации MAC, которая используется для обеспечения целостности и подлинности.
- **Multi-factor authentication** (MFA, многофакторная аутентификация) —аутентификация, использующая два или более различных факторов.
- **Mutual TLS** (mTLS) –См. TLS client authentication.
- **Object-relational Mapping** (ORM) –система, позволяющая приложению делать запросы к реляционной/табличной базе данных, используя объектную модель, совместимую с приложением.
- **One-time Password** (OTP, одноразовый пароль) —уникальный пароль, сгенерированный для одноразового использования.
- **Open Worldwide Application Security Project** (OWASP) –бесплатно распространяемый проект и открытое международное сообщество, занимающееся повышением безопасности прикладного программного обеспечения. Наша миссия —сделать безопасность приложений «видимой», чтобы люди и организации могли принимать обоснованные решения о рисках безопасности приложений. См.: https://www.owasp.org/.
- **Password-Based Key Derivation Function 2** (PBKDF2) –специальный односторонний алгоритм для формирования стойкого криптографического ключа на основе введённого текста (например, пароля) и дополнительного случайного значения —соли, которая затрудняет взлом пароля в режиме offline, если вместо исходного пароля хранится результирующее значение.
- **Public Key Infrastructure** (PKI, инфраструктура открытых ключей) —механизм, который связывает открытые ключи с соответствующими идентификаторами сущностей. Привязка устанавливается посредством процесса регистрации и выдачи сертификатов в центре сертификации (CA) и с его помощью.
- **Public Switched Telephone Network** (PSTN, коммутируемая телефонная сеть общего пользования, ТСОП, ТфОП) —традиционная телефонная сеть, соединяющая как

стационарные, так и мобильные телефоны.

- **Real-time Transport Protocol** (RTP) и **Real-time Transport Control Protocol** (RTCP) — два протокола, используемые совместно для передачи мультимедийных потоков. Применяются в стеке WebRTC. Стандартизированы в RFC 3550.
- **Reference Token** (ссылочный токен) —тип токена, который служит указателем или идентификатором для состояния или метаданных, хранящихся на сервере, также известен как случайный или непрозрачный токен. В отличие от автономных (self-contained) токенов, сслылочный токен не содержат собственной информации, а зависят от контекста сервера. Ссылочный токен либо является идентификатором сессии, либо содержит его.
- **Relying Party** (RP, проверяющая сторона) —как правило, приложение, которое полагается на то, что пользователь прошёл аутентификацию у отдельного поставщика аутентификации. Приложение проверяет токен или набор подписанных утверждений, предоставляемых этим поставщиком, чтобы убедиться, что пользователь является тем, за кого себя выдает.
- **Remote File Inclusion** (RFI, подключение удалённых файлов) —атака, эксплуатирующая уязвимые процедуры включения файлов в приложении, что приводит к подключению удалённых файлов.
- **Scalable Vector Graphics** (SVG, масштабируемая векторная графика) —язык разметки на основе XML для описания двумерной векторной графики.
- **Secure Real-time Transport Protocol** (SRTP) и **Secure Real-time Transport Control Protocol** (SRTCP) –расширение протоколов RTP и RTCP, предоставляющее функции шифрования сообщений, проверки подлинности и обеспечения целостности. Стандартизирован в RFC 3711.
- **Security Architecture** (архитектура безопасности) —абстракция дизайна приложения, которая определяет и описывает, где и как применяются меры безопасности, а также местоположение и категории информации.
- **Security Assertion Markup Language** (SAML) —открытый стандарт для единого входа (SSO), основанный на передаче подписанных утверждений (обычно в формате XML) между поставщиком идентификации и проверяющей стороной.
- **Security Configuration** (конфигурация безопасности) —конфигурация среды выполнения приложения, которая влияет на то, как применяются меры безопасности.
- **Security Control** (мера обеспечения безопасности) —функция или компонент, который контролирует (например, меры контроля доступа) или приводит к действию по обеспечению безопасности (например, регистрирует запись в журнале аудита).
- **Security information and event management** (SIEM) —система для обнаружения угроз, обеспечения соответствия требованиям и управления инцидентами безопасности путём сбора и анализа связанных с безопасностью данных из различных источников ИТ-инфраструктуры организации.
- **Self-Contained Token** (автономный токен) —токен, который инкапсулирует один или несколько атрибутов, не зависящих от серверного состояния или другого внешнего

хранилища. Эти токены обеспечивают подлинность и целостность содержащихся в них атрибутов, позволяя осуществлять безопасный обмен информацией между системами «без сохранения состояния». Автономные токены обычно защищаются с помощью криптографических методов, таких как цифровые подписи или коды аутентификации сообщений (MAC), для обеспечения подлинности, целостности, а в некоторых случаях и конфиденциальности своих данных. Распространёнными примерами являются SAML Assertions и JWT.

- **Server-side Request Forgery** (SSRF, подделка запросов на стороне сервера) —атака, которая злоупотребляет функциональными возможностями сервера для чтения или обновления внутренних ресурсов путем предоставления или изменения URL-адреса, по которому код, запущенный на сервере, будет читать или отправлять данные.
- **Session Description Protocol** (SDP) —формат сообщений для настройки мультимедийного сеанса (используется, например, в WebRTC). Стандартизирован в RFC 4566.
- **Session Identifier** или **Session ID** (идентификатор сессии) —ключ, который идентифицирует сессию с сохранением состояния, хранящийся на серверной стороне. Передаётся клиенту и обратно с помощью ссылочного токена (reference token).
- **Session Token** (сессионный токен) —общий термин, используемый в данном стандарте для обозначения токена или значения, применяемого в сессиях без сохранения состояния (которые используют автономный токен) или в сессиях с сохранением состояния (которые используют ссылочный токен).
- **Session Traversal Utilities for NAT** (STUN) —протокол, используемый для преодоления ограничений NAT с целью установления одноранговой связи. Стандартизирован в RFC 3489.
- **Single-factor authenticator** —Механизм проверки аутентификации пользователя. Фактором аутентификации может быть что-то, что вы знаете (запоминаемые секреты, пароли, парольные фразы, PIN-коды), что-то, что вы представляете собой (биометрия, отпечатки пальцев, сканы лица), или что-то, чем вы владеете (OTP-токены, криптографическое устройство, такое как смарт-карта).
- **Single Sign-on Authentication** (SSO, аутентификация единого входа) —пользователь входит в одно приложение, одновременно получая возможность войти в несколько других, без необходимости повторной аутентификации. Например, когда вы входите в Google, вы автоматически входите в другие сервисы Google, такие как YouTube, Google Docs, Gmail и т.д.
- **Software bill of materials** (SBOM) —структурированный и исчерпывающий перечень всех компонентов, модулей, библиотек, фреймворков и других ресурсов, необходимых для сборки или компиляции программного приложения.
- **Software Composition Analysis** (SCA, композиционный анализ) —Совокупность технологий, предназначенных для анализа состава приложений, зависимостей, библиотек и пакетов на наличие уязвимостей безопасности в используемых версиях компонентов. Не следует путать с анализом исходного кода, который теперь обычно обозначается как SAST.

- **Software development lifecycle** (SDLC, жизненный цикл разработки программного обеспечения) –пошаговый процесс разработки ПО, начиная с первоначальных требований, и заканчивая развертыванием и сопровождением.
- **SQL Injection** (SQLi, SQL-инъекция) —метод внедрения кода, используемый для атаки приложений, управляемых данными, в котором вредоносные SQL-выражения вставляются в точку входа.
- **Stateful Session Mechanism** (механизм сессии с сохранением состояния) —механизм управления сессией, когда приложение сохраняет состояние сеанса на серверной стороне, который обычно соответствует токену сеанса, сгенерированному с использованием криптографически стойкого генератора псевдослучайных чисел (CSPRNG), и выданному конечному пользователю.
- **Stateless Session Mechanism** (механизм сессии без сохранения состояния) —механизм управления сессией, который использует автономный токен, который передаётся клиентам и содержит информацию о сеансе, не обязательно хранящуюся в службе, которая затем получает и проверяет токен. На практике службе потребуется доступ к некоторой информации о сеансе (например, к списку отозванных JWT), чтобы иметь возможность применять необходимые меры безопасности.
- **Static application security testing** (SAST, статический анализ безопасности приложений) —набор технологий, предназначенных для анализа исходного кода, байт-кода и бинарных файлов приложения с целью выявления условий на уровне кода или архитектуры, указывающих на уязвимости безопасности. Решения SAST анализируют приложение «изнутри наружу» в статическом состоянии (без выполнения).
- **Threat Modeling** (моделирование угроз) —метод, состоящий в совершенствовании архитектур безопасности для выявления нарушителей, границ доверия, мер безопасности, а также ключевых технических и бизнес-активов.
- **Time-of-check to time-of-use** (TOCTOU) —ситуация, когда приложение проверяет состояние ресурса перед его использованием, но состояние ресурса может быть изменено между проверкой и использованием. Это может сделать результаты проверки недействительными и привести к ситуации, когда приложение выполняет некорректные действия из-за несоответствия состояния.
- **Time based One-time Passwords** (TOTPs) —метод создания OTP, при котором текущее время является входным аргументом для алгоритма формирования пароля.
- **TLS client authentication**, также называемый **Mutual TLS** (mTLS) —в стандартном TLS-соединении клиент может использовать сертификат, предоставленный сервером, для проверки подлинности сервера. При использовании mTLS клиент также использует свой собственный закрытый ключ и сертификат, чтобы позволить серверу проверить подлинность клиента.
- **Transport Layer Security** (TLS, безопасность транспортного уровня) —криптографические протоколы, обеспечивающие безопасность канала связи по сетевому соединению.
- **Traversal Using Relays around NAT** (TURN) —расширение протокола STUN, использующее TURN-сервер в качестве ретранслятора, когда прямое одноранговое соединение не

может быть установлено. Стандартизировано в RFC 8656.

- **Trusted execution environment** (TEE) —изолированная среда выполнения, обеспечивающая безопасное исполнение приложений вне зависимости от состояния остальной системы.
- **Trusted Platform Module** (TPM, модуль доверенной платформы) —тип HSM, который обычно является компонентом другого оборудования, например, материнской платы, и выступает в роли «корня доверия» для этой системы.
- **Trusted Service Layer** (уровень доверенных служб) —любая доверенная точка принудительного контроля, такая как микросервис, бессерверный API, серверная часть, доверенный API на клиентском устройстве с безопасной загрузкой, партнёрские или внешние API и т.д. «Доверенный» означает, что нет опасений, что недоверенный пользователь сможет обойти или пропустить слой или элементы управления, реализованные на этом уровне.
- **Uniform Resource Identifier** (URI, унифицированный идентификатор ресурса) —это строка символов, используемая для идентификации ресурса, например, веб-страницы, почтового адреса, места.
- **Uniform Resource Locator** (URL, унифицированный адрес ресурса) —это строка, указывающая местоположение ресурса в Интернете.
- **Universally Unique Identifier** (UUID, универсальный уникальный идентификатор) — уникальный номер-идентификатор, используемый в качестве средства идентификации в программном обеспечении.
- **Verifier** (верификатор/аудитор) —лицо или группа людей, которые анализируют безопасность приложения на соответствие требованиям OWASP ASVS.
- **Web Real-Time Communication** (WebRTC) —стек протоколов и связанный с ним веб-API, используемые для передачи мультимедийных потоков в веб-приложениях, обычно в контексте телеконференций. Основан на SRTP, SRTCP, DTLS, SDP и STUN/TURN.
- **WebSocket over TLS** (WSS) —практика защиты WebSocket-соединения путём размещения протокола WebSocket поверх протокола TLS.
- **What You See Is What You Get** (WYSIWYG) –тип редактора гипермедиа, который показывает, как на самом деле будет выглядеть содержимое при отображении, а не код разметки, управляющий отображением.
- **X.509 Certificate** —электронный сертификат, соответствующий общепринятому международному стандарту инфраструктуры открытых ключей X.509 (PKI) для проверки принадлежности открытого ключа идентификатору пользователя, компьютера или сервиса, указанному в сертификате.
- **XML eXternal Entity** (XXE) —тип элемента XML, который может получать доступ к локальному или удаленному содержимому через объявленный идентификатор. Может привести к атакам инъекции.

Приложение B: Полезные ссылки

Следующие проекты OWASP, скорее всего, будут полезны для читателей/последователей этого стандарта:

Основные проекты OWASP

1. Проект OWASP Top 10: https://owasp.org/www-project-top-ten/
2. OWASP Web Security Testing Guide: https://owasp.org/www-project-web-security-testing-guide/
3. OWASP Proactive Controls: https://owasp.org/www-project-proactive-controls/
4. OWASP Software Assurance Maturity Model (SAMM): https://owasp.org/www-project-samm/
5. OWASP Secure Headers Project: https://owasp.org/www-project-secure-headers/

Проект OWASP Cheat Sheet Series

В этом проекте есть памятки, которые будут актуальны для многих требований ASVS.

Сопоставление памяток с требованиями ASVS можно найти здесь: https://cheatsheetseries.owasp.org/IndexASVS.html

Проекты Mobile Security

1. OWASP Mobile Security Project: https://owasp.org/www-project-mobile-security/
2. OWASP Mobile Top 10 Risks: https://owasp.org/www-project-mobile-top-10/
3. OWASP Mobile Security Testing Guide and Mobile Application Security Verification Standard: https://owasp.org/www-project-mobile-security-testing-guide/

Проект OWASP Internet of Things

1. OWASP Internet of Things Project: https://owasp.org/www-project-internet-of-things/

Проект OWASP Serverless

1. OWASP Serverless Project: https://owasp.org/www-project-serverless-top-10/

Другие

Следующие веб-сайты, скорее всего, также будут полезны для читателей/последователей этого стандарта:

1. SecLists Github: https://github.com/danielmiessler/SecLists
2. MITRE Common Weakness Enumeration: https://cwe.mitre.org/
3. PCI Security Standards Council: https://www.pcisecuritystandards.org/
4. PCI Data Security Standard (DSS) v3.2.1 Requirements and Security Assessment Procedures: https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2-1.pdf
5. PCI Software Security Framework - Secure Software Requirements and Assessment Procedures: https://www.pcisecuritystandards.org/documents/PCI-Secure-Software-Standard-v1_0.pdf
6. PCI Secure Software Lifecycle (Secure SLC) Requirements and Assessment Procedures: https://www.pcisecuritystandards.org/documents/PCI-Secure-SLC-Standard-v1_0.pdf
7. OWASP ASVS 4.0 Testing Guide https://github.com/BlazingWind/OWASP-ASVS-4.0-testing-guide

Приложение C: Стандарты криптографии

Глава «Криптография» выходит за рамки простого описания передовых практик. Она направлена на углубление понимания принципов криптографии и поощрение внедрения более надёжных и современных методов безопасности. В этом приложении представлена подробная техническая информация по каждому требованию, дополняющая общие стандарты, изложенные в главе «Криптография».

В этом приложении определяются уровни допустимости различных криптографических механизмов:

- Допустимые механизмы (A) могут использоваться в приложениях.
- Устаревшие механизмы (L) не должны использоваться в приложениях, но могут использоваться только для обеспечения совместимости с существующими устаревшими приложениями или кодом. Хотя использование таких механизмов в настоящее время само по себе не считается уязвимостью, их следует как можно скорее заменить более безопасными и перспективными механизмами.
- Запрещенные механизмы (D) не должны использоваться, поскольку в настоящее время они считаются ненадежными или не обеспечивают достаточной безопасности.

Этот список может быть переопределен в контексте конкретного приложения по разным причинам, включая:

- новые разработки в области криптографии;
- соблюдение нормативных требований.

Инвентаризация и документирование криптографии

В этом разделе представлена дополнительная информация для V11.1 «Инвентаризация и документирование криптографии».

Важно обеспечить регулярное обнаружение, инвентаризацию и оценку всех криптографических активов, таких как алгоритмы, ключи и сертификаты. Для уровня 3 это должно включать статическое и динамическое сканирование для обнаружения использования криптографии в приложении. Такие инструменты, как SAST и DAST, могут помочь в этом, но для более полного охвата могут потребоваться специализированные инструменты. Примеры бесплатных инструментов:

- CryptoMon - Network Cryptography Monitor - using eBPF, written in python
- Cryptobom Forge Tool: Generating Comprehensive CBOMs from CodeQL Outputs

Эквивалентная стойкость криптографических параметров

Относительные уровни безопасности различных криптографических систем приведены в следующей таблице (из NIST SP 800-57 Часть 1, стр.71):

Количество бит стойкости	Симметричные алгоритмы	Конечное поле	Целочисленная факторизация	Эллиптическая кривая
<= 80	2TDEA	L = 1024 N = 160	k = 1024	f = 160-223
112	3TDEA	L = 2048 N = 224	k = 2048	f = 224-255
128	AES-128	L = 3072 N = 256	k = 3072	f = 256-383
192	AES-192	L = 7680 N = 384	k = 7680	f = 384-511
256	AES-256	L = 15360 N = 512	k = 15360	f = 512+

Примеры применения:

- Криптография на конечных полях: DSA, FFDH, MQV
- Криптография на целочисленной факторизации: RSA
- Криптография на эллиптических кривых: ECDSA, EdDSA, ECDH, MQV

Примечание: в этом разделе предполагается, что квантового компьютера не существует; если бы такой компьютер существовал, оценки для последних трех столбцов стали бы недействительными.

Случайные значения

В этом разделе представлена дополнительная информация для V11.5 «Случайные значения».

Имя	Версия/Ссылка	Примечания	Статус
`/dev/random`	Linux 4.8+ (Окт 2016), также встречается в iOS, Android и других операционных системах POSIX на базе Linux. Основан на RFC7539	Использование ChaCha20. Найдено в iOS `SecRandomCopyBytes` и Android `SecureRandom` с правильными настройками, предоставленными для каждого.	A
`/dev/urandom`	Специальный файл ядра Linux для предоставления случайных данных	Обеспечивает высококачественный источник энтропии на основе аппаратной случайности.	A
`AES-CTR-DRBG`	NIST SP800-90A	Используется в распространенных реализациях, таких как Windows CNG API `BCryptGenRandom` при установке `BCRYPT_RNG_ALGORITHM`.	A
`HMAC-DRBG`	NIST SP800-90A		A
`Hash-DRBG`	NIST SP800-90A		A

Имя	Версия/Ссылка	Примечания	Статус
`getentropy()`	OpenBSD, доступный в Linux glibc 2.25+ и macOS 10.12+	Обеспечивает безопасный доступ к случайным байтам непосредственно из источника энтропии ядра с помощью простого и минималистичного API. Он более современный и позволяет избежать ошибок, связанных со старыми API.	A

Базовая хэш-функция, используемая с HMAC-DRBG или Hash-DRBG, должна быть допустима для такого применения.

Алгоритмы шифрования

В этом разделе представлена дополнительная информация для V11.3 «Алгоритмы шифрования».

Допустимые алгоритмы шифрования перечислены в порядке предпочтения.

Симметричные алгоритмы	Ссылка	Статус
AES-256	FIPS 197	A
Salsa20	Salsa 20 specification	A
XChaCha20	XChaCha20 Draft	A
XSalsa20	Extending the Salsa20 nonce	A
ChaCha20	RFC 8439	A
AES-192	FIPS 197	A
AES-128	FIPS 197	L
2TDEA		D
TDEA (3DES/3DEA)		D
IDEA		D
RC4		D

Симметричные алгоритмы	Ссылка	Статус
Blowfish		D
ARC4		D
DES		D

Режимы шифрования AES

Блочные шифры, такие как AES, могут использоваться в различных режимах работы. Многие режимы работы, такие как Electronic codebook (ECB), небезопасны и не должны использоваться. Режимы работы Galois/Counter Mode (GCM) и «Счётчик с кодом аутентификации сообщений цепочки блоков шифра» (CCM) обеспечивают аутентифицированное шифрование и должны использоваться в современных приложениях.

Допустимые режимы перечислены в порядке предпочтения.

Режим	Поддержка аутентификации	Ссылка	Статус	Ограничение
GCM	Yes	NIST SP 800-38D	A	
CCM	Yes	NIST SP 800-38C	A	
CBC	No	NIST SP 800-38A	L	
CCM-8	Yes		D	
ECB	No		D	
CFB	No		D	
OFB	No		D	
CTR	No		D	

Примечания:

- Все зашифрованные сообщения должны быть аутентифицированы. Для ЛЮБОГО использования режима CBC ОБЯЗАТЕЛЬНО должен быть соответствующий алгоритм хеширования MAC для проверки сообщения. Как правило, он ДОЛЖЕН применяться в методе «Шифрование-затем-хеширование» (но в TLS 1.2 вместо этого используется «Хеширование-затем-шифрование»). Если это невозможно гарантировать, то CBC НЕ ДОЛЖЕН использоваться. Единственной опцией, где разрешено шифрование без алгоритма MAC, является шифрование диска.
- При использовании CBC должно быть гарантировано, что проверка заполнения выполняется за константное время.

- При использовании CCM-8 тег MAC имеет только 64 бита стойкости. Это не соответствует требованию 6.2.9, которое требует не менее 128 бит стойкости.
- Шифрование диска считается выходящим за рамки ASVS. Поэтому в данном приложении не перечислены какие-либо утверждённые методы шифрования диска. В этом случае обычно допускается шифрование без аутентификации, и обычно используются режимы XTS, XEX и LRW.

Шифрование ключей

Шифрование ключа (key wrap) и соответствующее расшифрование ключа (key unwrap) —это метод защиты существующего ключа путём его инкапсуляции с использованием дополнительного механизма шифрования, чтобы исходный ключ не был явно раскрыт, например, во время передачи. Этот дополнительный ключ, используемый для защиты исходного ключа, называется ключом обёртывания (wrap key).

Эта операция может выполняться в случае необходимости защиты ключей в местах, которые считаются недоверенными, а также для передачи конфиденциальных ключей по ненадежным сетям или внутри приложений. Однако перед началом процедуры шифрования/расшифрования ключей необходимо оценить характер исходного ключа (например, идентичность и назначение). Это может повлиять на безопасность и соответствие требованиям как исходных (до передачи), так и целевых систем/приложений, включая требования по аудиту действий с ключами (например, подпись) и хранению ключей.

В частности, для шифрования ключей ОБЯЗАТЕЛЬНО должен использоваться алгоритм AES-256, соответствующий NIST SP 800-38F и с учётом перспективных мер защиты от квантовых угроз. Режимы шифрования с использованием AES применяются в следующем порядке предпочтения:

Шифрование ключа	Ссылка	Статус
KW	NIST SP 800-38F	A
KWP	NIST SP 800-38F	A

AES-192 и AES-128 МОГУТ использоваться, если того требует сценарий использования, но обоснование этого ДОЛЖНО быть задокументировано в реестре криптографии организации.

Аутентифицированное шифрование

За исключением шифрования диска, зашифрованные данные должны быть защищены от несанкционированного изменения с использованием какой-либо схемы аутентифицированного шифрования (AE), обычно с использованием схемы аутентифицированного шифрования с соответствующими данными (AEAD).

Предпочтительно, чтобы приложение использовало утверждённую схему AEAD. В качестве альтернативы, оно может комбинировать утверждённую схему шифрования и утверждённый алгоритм MAC с конструкцией Encrypt-then-MAC.

MAC-then-encrypt по-прежнему разрешен для совместимости со старыми приложениями. Он используется в TLS версии 1.2 со старыми наборами шифров.

Схема AEAD	Ссылка	Статус
AES-GCM	SP 800-38D	A
AES-CCM	SP 800-38C	A
ChaCha-Poly1305	RFC 7539	A
AEGIS-256	AEGIS: A Fast Authenticated Encryption Algorithm (v1.1)	A
AEGIS-128	AEGIS: A Fast Authenticated Encryption Algorithm (v1.1)	A
AEGIS-128L	AEGIS: A Fast Authenticated Encryption Algorithm (v1.1)	A
Encrypt-then-MAC		A
MAC-then-encrypt		L

Хэш-функции

В этом разделе представлена дополнительная информация для V11.4 «Хеширование и функции на основе хеширования».

Основные случаи использования хэш функций

В следующей таблице перечислены хэш-функции, допустимые для основных случаев использования криптографии, таких как цифровые подписи:

- Допустимые хеш-функции обеспечивают высокую устойчивость к коллизиям и подходят для приложений с высоким уровнем безопасности.

- Некоторые из этих алгоритмов обеспечивают высокую устойчивость к атакам при использовании с надлежащим управлением криптографическими ключами, поэтому они дополнительно допустимы для функций HMAC, KDF и RBG.
- Хэш-функции с длиной выходного значения менее 254 бит обладают недостаточной устойчивостью к коллизиям и не должны использоваться для цифровой подписи или других приложений, требующих устойчивости к коллизиям. В других случаях они могут использоваться ТОЛЬКО для обеспечения совместимости и верификации с устаревшими системами, но не должны использоваться в новых разработках.

Хэш-функция	Ссылка	Статус	Ограничения
SHA3-512	FIPS 202	A	
SHA-512	FIPS 180-4	A	
SHA3-384	FIPS 202	A	
SHA-384	FIPS 180-4	A	
SHA3-256	FIPS 202	A	
SHA-512/256	FIPS 180-4	A	
SHA-256	FIPS 180-4	A	
SHAKE256	FIPS 202	A	
BLAKE2s	BLAKE2: simpler, smaller, fast as MD5	A	
BLAKE2b	BLAKE2: simpler, smaller, fast as MD5	A	
BLAKE3	BLAKE3 one function, fast everywhere	A	
SHA-224	FIPS 180-4	L	Not suitable for HMAC, KDF, RBG, digital signatures

Хэш-функция	Ссылка	Статус	Ограничения
SHA-512/224	FIPS 180-4	L	Not suitable for HMAC, KDF, RBG, digital signatures
SHA3-224	FIPS 202	L	Not suitable for HMAC, KDF, RBG, digital signatures

Хэш-функция	Ссылка	Статус	Ограничения
SHA-1	RFC 3174 & RFC 6194	L	Not suitable for HMAC, KDF, RBG, digital signatures
CRC (any length)		D	
MD4	RFC 1320	D	
MD5	RFC 1321	D	

Хэш-функции для хранения паролей

Для безопасного хеширования паролей необходимо использовать специальные хеш-функции. Эти медленные алгоритмы хеширования снижают риск атак методом перебора и перебора по словарю, увеличивая вычислительную сложность взлома паролей.

KDF	Ссылка	Обязательные параметры	Статус
argon2id	RFC 9106	t = 1: m ≥ 47104 (46 MiB), p = 1t = 2: m ≥ 19456 (19 MiB), p = 1t ≥ 3: m ≥ 12288 (12 MiB), p = 1	A
scrypt	RFC 7914	p = 1: N ≥ 2^17 (128 MiB), r = 8p = 2: N ≥ 2^16 (64 MiB), r = 8p ≥ 3: N ≥ 2^15 (32 MiB), r = 8	A
bcrypt	A Future-Adaptable Password Scheme	cost ≥ 10	A
PBKDF2-HMAC-SHA-512	NIST SP 800-132, FIPS 180-4	iterations ≥ 210,000	A

KDF	Ссылка	Обязательные параметры	Статус
PBKDF2-HMAC-SHA-256	NIST SP 800-132, FIPS 180-4	iterations ≥ 600,000	A
PBKDF2-HMAC-SHA-1	NIST SP 800-132, FIPS 180-4	iterations ≥ 1,300,000	L

Для хранения паролей можно использовать допустимые функции формирования ключей на основе паролей.

Функции формирования ключа (KDFs)

Основные функции формирования ключа

KDF	Ссылка	Статус
HKDF	RFC 5869	A
TLS 1.2 PRF	RFC 5248	L
MD5-based KDFs	RFC 1321	D
SHA-1-based KDFs	RFC 3174 & RFC 6194	D

Функции формирования ключа на основе пароля

KDF	Ссылка	Обязательные параметры	Статус
argon2id	RFC 9106	t = 1: m ≥ 47104 (46 MiB), p = 1t = 2: m ≥ 19456 (19 MiB), p = 1t ≥ 3: m ≥ 12288 (12 MiB), p = 1	A
scrypt	RFC 7914	p = 1: N ≥ 2^17 (128 MiB), r = 8p = 2: N ≥ 2^16 (64 MiB), r = 8p ≥ 3: N ≥ 2^15 (32 MiB), r = 8	A

KDF	Ссылка	Обязательные параметры	Статус
PBKDF2-HMAC-SHA-512	NIST SP 800-132, FIPS 180-4	iterations ≥ 210,000	A
PBKDF2-HMAC-SHA-256	NIST SP 800-132, FIPS 180-4	iterations ≥ 600,000	A
PBKDF2-HMAC-SHA-1	NIST SP 800-132, FIPS 180-4	iterations ≥ 1,300,000	L

Механизмы обмена ключами

В этом разделе представлена дополнительная информация для V11.6 «Криптография с публичным ключом».

Схемы KEX

Для всех схем обмена ключами ДОЛЖНА быть обеспечена стойкость в 112 бит или выше, а их реализация ДОЛЖНА соответствовать выбору параметров, представленному в следующей таблице.

Схема	Параметры	Прямая секретность	Статус
Диффи-Хеллман на конечных полях (FFDH)	L >= 3072 & N >= 256	Yes	A
Диффи-Хеллман на эллиптических кривых (ECDH)	f >= 256-383	Yes	A
Передача зашифрованного ключа с помощью RSA-PKCS#1 v1.5		No	D

Где следующие параметры:

- k —длина ключа для RSA.
- L —длина открытого ключа, а N —длина закрытого ключа для криптографии на конечном поле.
- f —диапазон длин ключей для ECC.

Любая новая реализация НЕ ДОЛЖНА использовать схему, НЕ соответствующую NIST SP 800-56A & B и NIST SP 800-77. В частности, IKEv1 НЕ ДОЛЖЕН использоваться в промышленной среде.

Группы Диффи-Хеллмана

Следующие группы допустимы для реализации обмена ключами Диффи-Хеллмана. Уровни безопасности описаны в NIST SP 800-56A, Приложение D, и NIST SP 800-57 Часть 1, Ред. 5.

Группа	Статус
P-224, secp224r1	A
P-256, secp256r1	A
P-384, secp384r1	A
P-521, secp521r1	A
K-233, sect233k1	A
K-283, sect283k1	A
K-409, sect409k1	A
K-571, sect571k1	A
B-233, sect233r1	A
B-283, sect283r1	A
B-409, sect409r1	A
B-571, sect571r1	A
Curve448	A
Curve25519	A
MODP-2048	A
MODP-3072	A
MODP-4096	A
MODP-6144	A
MODP-8192	A
ffdhe2048	A

Группа	Статус
ffdhe3072	A
ffdhe4096	A
ffdhe6144	A
ffdhe8192	A

Коды аутентификации сообщений (MAC)

Коды аутентификации сообщений (MAC) —это криптографические конструкции, используемые для проверки целостности и подлинности сообщения. MAC принимает сообщение и секретный ключ в качестве входных данных и создаёт тег фиксированного размера (значение MAC). MAC широко используется в протоколах защищённой связи (например, TLS/SSL) для обеспечения подлинности и целостности сообщений, которыми обмениваются стороны.

Алгоритм MAC	Ссылка	Статус	Ограничения
HMAC-SHA-256	RFC 2104 & FIPS 198-1	A	
HMAC-SHA-384	RFC 2104 & FIPS 198-1	A	
HMAC-SHA-512	RFC 2104 & FIPS 198-1	A	
KMAC128	NIST SP 800-185	A	
KMAC256	NIST SP 800-185	A	
BLAKE3 (keyed_hash mode)	BLAKE3 one function, fast everywhere	A	
AES-CMAC	RFC 4493 & NIST SP 800-38B	A	
AES-GMAC	NIST SP 800-38D	A	
Poly1305-AES	The Poly1305-AES message-authentication code	A	
HMAC-SHA-1	RFC 2104 & FIPS 198-1	L	

Алгоритм MAC	Ссылка	Статус	Ограничения
HMAC-MD5	RFC 1321	D	

Цифровые подписи

Схемы подписи ДОЛЖНЫ использовать утвержденные размеры ключей и параметры согласно NIST SP 800-57 Часть 1.

Алгоритм подписи	Ссылка	Статус
EdDSA (Ed25519, Ed448)	RFC 8032	A
XEdDSA (Curve25519, Curve448)	XEdDSA	A
ECDSA (P-256, P-384, P-521)	FIPS 186-4	A
RSA-RSSA-PSS	RFC 8017	A
RSA-SSA-PKCS#1 v1.5	RFC 8017	D
DSA (any key size)	FIPS 186-4	D

Стандарты постквантового шифрования

Реализации постквантовой криптографии (PQC) должны соответствовать стандартам FIPS-203, FIPS-204 и FIPS-205. В настоящее время для этих стандартов доступно не так много примеров эталонных реализаций. Подробнее см. объявление NIST о первых трёх финализированных постквантовых стандартах шифрования (август 2024 г.).

Предлагаемый постквантовый гибридный метод согласования ключей TLS mlkem768x25519 поддерживается основными браузерами, такими как Firefox версии 132 и Chrome версии 131. Он может использоваться в средах тестирования криптографии или при наличии в библиотеках, допустимых отраслевыми или государственными органами.

Приложение D: Рекомендации

Введение

При подготовке версии 5.0 Стандарта верификации требований к безопасности приложений (ASVS) стало очевидно, что ряд существующих и вновь предложенных пунктов не следует

включать в требования версии 5.0. Это могло быть связано с тем, что они не входят в область действия ASVS согласно определению для версии 5.0, или же потому, что, хотя эти идеи и представлялись полезными, их нельзя было сделать обязательными.

Не желая полностью потерять все эти материалы, некоторые из них были включены в это приложение.

Рекомендуемые механизмы, входящие в область применения

Следующие пункты входят в область действия ASVS. Они не должны быть обязательными, но настоятельно рекомендуется рассматривать их как часть безопасного приложения.

- Необходимо предоставить индикатор надежности пароля, чтобы помочь пользователям установить более стойкий пароль.
- Создайте общедоступный файл security.txt в корневом каталоге или каталоге .well-known приложения, в котором будет чётко указана ссылка или адрес электронной почты, по которым пользователи смогут связаться с владельцами по вопросам безопасности.
- В дополнение к проверке на уровне доверенных служб следует применять проверку входных данных на стороне клиента, поскольку это даёт хорошую возможность обнаружить, когда кто-то обходит клиентские средства защиты при попытке атаки на приложение.
- Предотвращайте появление в поисковых системах случайно доступных и конфиденциальных страниц с помощью файла robots.txt, заголовка ответа X-Robots-Tag или метатега robots.html.
- При использовании GraphQL реализуйте логику авторизации на уровне бизнес-логики, а не на уровне GraphQL или резолвера, чтобы избежать необходимости обрабатывать авторизацию на каждом отдельном интерфейсе.

Ссылки:

- Дополнительная информация о security.txt, включая ссылку на RFC

Принципы безопасности программного обеспечения

Следующие пункты ранее присутствовали в ASVS, но они не являются требованиями. Скорее, это принципы, которые следует учитывать при реализации мер безопасности, соблюдение которых приведет к повышению надежности этих мер. К ним относятся:

- Средства управления безопасностью должны быть централизованными, простыми (экономия времени), проверяемо безопасными и допускающими повторное использование. Это позволит избежать дублирования, отсутствия или неэффективности средств управления.
- По возможности используйте ранее написанные и проверенные реализации средств управления безопасностью, а не полагайтесь на их реализацию с нуля.

- В идеале для доступа к защищённым данным и ресурсам следует использовать единый механизм управления доступом. Все запросы должны проходить через этот механизм, чтобы избежать копирования и вставки или небезопасных альтернативных путей.
- Управление доступом на основе атрибутов или функций —рекомендуемый шаблон, при котором код проверяет авторизацию пользователя для функции или элемента данных, а не только его роль. Разрешения по-прежнему должны предоставляться с использованием ролей.

Процессы обеспечения безопасности программного обеспечения

Ряд процессов обеспечения безопасности были исключены из ASVS 5.0, но всё ещё представляют собой хорошую идею. Проект OWASP SAMM может стать хорошим источником информации о том, как эффективно реализовать эти процессы. Ранее в ASVS присутствовали следующие элементы:

- Убедитесь в использовании безопасного жизненного цикла разработки программного обеспечения, который обеспечивает безопасность на всех этапах разработки.
- Убедитесь в применении моделирования угроз для каждого изменения дизайна или планирования спринта, чтобы выявлять угрозы, планировать контрмеры, способствовать надлежащему реагированию на риски и управлять тестированием безопасности.
- Убедитесь, что все пользовательские истории и функции содержат функциональные ограничения безопасности. Например, “Как пользователь, я должен иметь возможность просматривать и редактировать свой профиль, но не должен иметь возможности просматривать или редактировать чужой профиль”.
- Убедитесь в наличии чек-листа для безопасной разработки, требований, политик и стандартов безопасности, а также в их доступности для разработчиков и тестировщиков.
- Убедитесь, что существует непрерывный процесс, гарантирующий отсутствие в исходном коде приложения бэкдоров, вредоносного кода (такого как атаки типа «нарезка салями», обход логики или «логические бомбы») и недокументированных или скрытых функций (например, «пасхальных яиц», небезопасных инструментов отладки). Выполнение требований этого раздела невозможно без полного доступа к исходному коду, включая сторонние библиотеки, и поэтому, вероятно, подходит только для приложений, требующих высочайшего уровня безопасности.
- Убедитесь, что в средах развертывания реализованы механизмы обнаружения и реагирования на отклонения конфигурации. Это может включать использование неизменяемой инфраструктуры, автоматическое повторное развертывание с безопасной базовой конфигурацией или инструменты обнаружения отклонений, сравнивающие текущее состояние с утверждёнными конфигурациями.
- Убедитесь, что усиление защиты конфигурации выполняется для всех сторонних продуктов, библиотек, фреймворков и служб в соответствии с их индивидуальными рекомендациями.

Ссылки:

- OWASP Threat Modeling Cheat Sheet
- OWASP Threat modeling
- OWASP Software Assurance Maturity Model Project
- Microsoft SDL

Приложение E: Участники

Мы выражаем благодарность следующим людям, которые комментировали или открывали пул-реквесты после выпуска ASVS 4.0.0.

Если вам известно о каких-либо ошибках или вы хотите, чтобы ваше имя отображалось иначе, пожалуйста, сообщите нам об этом.

Johan Sydseter (sydseter)	luis servin (lfservin)	Oleksii Dovydkov (oleksiidov)	IZUKA Masahiro (maizuka)
James Sulinski (jsulinski)	Eli Saad (ThunderSon)	kkshitish9	Andrew van der Stock (vanderaj)
Rick M (kingthorin)	Bankde Eakasit (Bankde)	Michael Gargiullo (mgargiullo)	Raphael Dunant (Racater)
Cesar Kohl (cesarkohl)	inaz0	Joerg Bruenner (JoergBruenner)	David Deatherage (securitydave)
John Carroll (yosignals)	Jim Fenton (jimfenton)	Matteo Pace (M4tteoP)	Sebastien gioria (SPoint42)
Steven van der Baan (vdbaan)	Jeremy Bonghwan Choi (jeremychoi)	craig-shony	Riccardo Sirigu (ricsirigu)
Tomasz Wrobel (tw2as)	Alena Dubeshko (belalena)	Rafael Green (RafaelGreen1)	mjang-cobalt
clallier94	Kevin W. Wall (kwwall)	Jordan Sherman (jsherm-fwdsec / deleterepo)	Ingo Rauner (ingo-rauner)
Dirk Wetter (drwetter)	Moshe Zioni (moshe-apiiro)	Patrick Dwyer (coderpatros)	David Clarke (davidclarke-au)
Takaharu Ogasa (takaharuogasa)	Arkadii Yakovets (arkid15r)	Motoyasu Saburi (motoyasu-saburi)	leirn
wet-certitude	timhemel	RL Thornton (thornshadow99)	Thomas Bandt (aspnetde)

Roel Storms (roelstorms)	Jeroen Willemsen (commjoen)	anonymous-31	Kamran Saifullah (deFr0ggy)
Steve Springett (stevespringett)	Spyros (northdpole)	Hans Herrera (hansphp)	Marx314
CarlosAllendes	Yonah Russ (yruss972)	Sander Maijers (sanmai-NL)	Luboš Bretschneider (bretik)
Eva Sarafianou (esarafianou)	Ata Seren ataseren	Steve Thomas (Sc00bz)	Dominique RIGHETTO (righettod)
Steven van der Baan (svdb-ncc)	Michael Vacarella (Aif4thah)	Tonimir Kisasondi (tkisason)	Stefan Streichsbier (streichsbaer)
hi-unc1e	sb3k (starbuck3000)	mario-platt	Devdatta Akhawe (devd)
Michael Gissing (scolytus)	Jet Anderson (thatsjet)	Dave Wichers (davewichers)	Jonny Schnittger (JonnySchnittger)
Silvia Väli (silviavali)	jackgates73	1songb1rd	Timur - (timurozkul)
Gareth Heyes (hackvertor)	appills	suvikaartinen	chaals (chaals)
DanielPharos (AtlasHackert)	will Farrell (willfarrell)	Alina Vasiljeva (avasiljeva)	Paul McCann (ismisepaul)
Sage (SajjadPourali)	rbsec	Benedikt Bauer (mastacheata)	James Jardine (jamesjardine)
Mark Burnett (m8urnett)	dschwarz91	Cyber-AppSec (Cyber-AppSec)	Tib3rius
BitnessWise (bitnesswise)	damienbod (damienbod)	Jared Meit (jmeit-fwdsec)	Stefan Seelmann (sseelmann)
Brendan O’Connor (ussjoin)	Andrei Titov (andrettv)	Hans-Petter Fjeld (atluxity)	markehack
Neil Madden (NeilMadden)	Michael Geramb (mgeramb)	Osama Elnaggar (ossie-git)	mackowski
Ravi Balla (raviballa)	Hazana (hazanasec)	David Means (dmeans82)	Alexander Stein (tohch4)
BaeSenseii (baesenseii)	Vincent De Schutter (VincentDS)	S Bani (sbani)	Mitsuaki Akiyama (mak1yama)
Christopher Loessl (hashier)	victorxm	Michal Rada (michalradacz)	Veeresh Devireddy (drveresh)

MaknaSEO	darkzero2022	Liam (LiamDobbelaere)	Frank Denis (jedisct1)
Otto Sulin (ottosulin)	carllaw6885	Anders Johan Holmefjord (aholmis)	Richard Fritsch (rfricz)
mesutgungor	Scott Helme (ScottHelme)	Carlo Reggiani (carloreggiani)	Suyash Srivastava (suyash5053)
Mark Potter (markonweb)	Arjan Lamers (alamers)	Gøran Breivik (gobrtg)	flo-blg
Guillaume Déflache (guillaume-d)	Toufik Airane (toufik-airane)	Keith Hoodlet (securingdev)	Sinner (SoftwareSinner)
iloving	Jeroen Beckers (TheDauntless)	Joubin Jabbari (joubin)	yu fujioka (fujiokayu)
execjosh (execjosh)	Alicja Kario (tomato42)	Sidney Ribeiro (srjsoftware)	Gabriel Marquet (Gby56)
Drew Schulz (drschulz)	bedirhan	muralito	Ronnie Flathers (ropnop)
Philippe De Ryck (philippederyck)	Malte (mal33)	MazeOfThoughts	Andreas Falk (andifalk)
Javi (javixeneize)	Daniel Hahn (averell23)	borislav-c	Robin Wood (digininja)
miro2ns	Jan Dockx (jandockx)	vipinsaini434	priyanshukumar397
Nat Sakimura (sakimura)	Benjamin Häublein (BenjaminHae)	unknown-user-from	Ali Ramazan TAŞDELEN (alitasdln)
Pedro Escaleira (oEscal)	Josh (josh-hemphill)	Tim Würtele (SECtim)	AviD (avidouglen)
SheHacksPurple (shehackspurple)	fcerullo-cycubix	Hector Eryx Paredes Camacho (heryxpc)	Irene Michlin (irene221b)
Jonah Y-M (TG-Techie)	Dhiraj Bahroos (bahroos)	Jef Meijvis (jefmeijvis)	IzmaDoesItbeta
Abdessamad TEMMAR (TmmmmmR)	sectroyer	Soh Satoh (sohsatoh)	regoravalaz
james-t (james-bitherder)	Aram Hovsepyan (aramhovsepyan)	JaimeGomezGarciaSan	ValdiGit01
iwatachan (ishowta)	Vinod Anandan (VinodAnandan)	Kevin Kien (KevinKien)	paul-williamson-swoop

endergzr	Radhwan Alshamamri (Rado0z)	Grant Ongers (rewtd)	Cure53 (cure53)
AliR2Linux	Ads Dawson (Gang-GreenTemperTatum)	William Reyor (BillReyor)	gabe (gcrow)
mascotter	luissaiz	Suren Manukyan (vx-sec)	Piotr Gliźniewicz (pglizniewicz)
Tadeusz Wachowski (tadeuszwachowski)	Nasir aka Nate (andesec)	settantasette	Lars Haulin (LarsH)
Terence Eden (edent)	JasmineScholz	Arun Sivadasan (teavanist)	Yusuf GÜR (yusuffgur)
Troy Marshall (troymarshall)	Tanner Prynn (tprynn)	Nick K. (nickific)	raoul361
Azeem Ilyas (TheAxZim)	Evo Stamatov (avioli)	Tim Potter (timpotter87)	Gavin Ray (GavinRay97)
monis (demideus)	Marcin Hoppe (MarcinHoppe)	Grambulf (ramshazar)	Jordan Pike (computersarebad)
Jason Rogers (jason-invision)	Ben Hall (benbhall)	JamesPoppyCock (jamesly123)	WhiteHackLabs (whitehacklabs)
Alex Gaynor (alex)	Filip van Laenen (filipvanlaenen)	jeurgen	GraoMelo
Andreas Kurtz (ay-kay)	Tom Tervoort (TomTervoort)	old man (deveras)	Marco Schnüriger (marcortw)
stiiin	infoseclearn (teaminfoseclearn)	hljupkij	Noe (nmarher)
Lyz (lyz-code)	Martin Riedel (mrtnrdl)	KIM Jaesuck (tcaesvk)	Barbara Schachner (bschach)
René Reuter (AresSec)	carhackpils	Tyler (tyler2cr)	Hugo (hasousa)
Wouter Bloeyaert (Someniak)	Mark de Rijk (markderijkinfosec)	Ramin (picohub)	Philip D. Turner (philipdturner)
Will Chatham (willc)			