УДК 004.056

Студ. Н.В. Филон
Науч. рук. ст. преп. Я.А. Игнаткова
(кафедра информатики и веб-дизайна, БГТУ)

СПОСОБЫ ХРАНЕНИЯ ДАННЫХ ПРИ АВТОРИЗАЦИИ

Авторизация – важный и необходимый элемент для большинства современных веб-сайтов, определяющий доступ пользователей к функционалу ресурса. Современные методы хранения статуса авторизации, а именно PHP-сессии, cookies и localstorage предлагают различные компромиссы между безопасностью, удобством и масштабируемостью.

PHP-сессии хранят данные на сервере, что минимизирует риски утечки. Сервер генерирует уникальный идентификатор сессии, который передается клиенту через cookie. Это делает метод идеальным для банкинга и админ-панелей, где безопасность приоритетна. Однако сессии требуют синхронизации между серверами и создают нагрузку при масштабировании. Время жизни сессии можно настроить, завершая её после неактивности пользователя.

Cookies автоматически отправляются с каждым HTTP-запросом, обеспечивая долгую сессию. Флаги HttpOnly и Secure (снижают риск XSS и перехвата данных. Однако cookies уязвимы к CSRF-атакам. Злоумышленник может подделать запрос от имени пользователя. Атрибут SameSite частично решает эту проблему. Объём ограничен 4 КБ, что делает их пригодными для хранения токенов, но не больших данных. Популярны в соцсетях и блогах.

LocalStorage предоставляет до 10 МБ памяти на клиенте, что удобно для одностраничных сайтов, таких как React App. Однако localStorage крайне уязвим к XSS-атакам. Любой скрипт на странице может получить доступ к данным. Для защиты используют шифрование токенов, короткое время их жизни и Content Security Policy. Комбинированный метод (сессии + cookies) часто используется в интернет-магазинах: корзина хранится в сессии для безопасности, а авторизация – в cookies для удобства. Такой подход требует защиты от CSRF и усложняет синхронизацию времени жизни компонентов веб-сайта.

Оптимальный метод авторизации зависит от целей приложения. PHP-сессии обеспечивают максимальную безопасность, cookies – удобство, localStorage – гибкость для SPA. Комбинированный подход требует тщательной реализации, но объединяет преимущества используемых методов. Независимо от выбора, желательно использование HTTPS, хеширование паролей и регулярный аудит кода.