УДК 004

Алекперов З.А.,
магистр 2 курс,
факультет «программная инженерия
и компьютерная техника»
Национальный Исследовательский Университет
Информационных Технологий Механики и Оптики
Россия, г. Санкт-Петербург

ОБЗОР ПОПУЛЯРНЫХ УЯЗВИМОСТЕЙ ВЕБ-ПРИЛОЖЕНИЙ И ИХ РЕШЕНИЙ

Аннотация: *Статья посвящена обзору популярных уязвимостей веб-приложений и их решений. В статье рассматривается существующие проблемы безопасности, топ 10 угроз и методы решения этих проблем.*

Ключевые слова: *Безопасность, угрозы, веб-приложения, атака, OWASP, XSS, уязвимости, SQL-инъекции.*

Annotation: *The article is devoted to the review of popular vulnerabilities of web applications and their solutions. The article discusses the existing security problems, the top 10 threats and methods for solving these problems.*

Key words: *Security, threats, web applications, attack, OWASP, XSS, vulnerabilities, SQL-injection.*

ОСНОВНАЯ ЧАСТЬ

Небезопасное веб-приложение подрывает нашу финансовую, медицинскую, оборонную, энергетическую и другую критически важную инфраструктуру. По мере того, как наше приложение становится все более сложным и связанным, сложность обеспечения безопасности приложения увеличивается в геометрической прогрессии. Быстрые темпы современных процессов разработки веб-приложений делают наиболее распространенные риски существенными для

быстрого и точного обнаружения и устранения. Мы больше не можем мириться с относительно простыми проблемами безопасности, подобными тем, которые представлены в этой первой десятке OWASP [1].

1.1 Угрозы безопасности приложения

Злоумышленники могут использовать множество различных путей через приложение, чтобы нанести вред бизнесу или организации. Каждый из этих путей представляет риск, который может быть или не быть достаточно серьезным, чтобы заслуживать внимания [13].

Распространенные уязвимости веб-приложений можно разделить на три типа: уязвимости внедрения, уязвимости бизнес-логики и уязвимости управления сеансами. Следующая диаграмма показывает классификацию нескольких типов атак, которые обычно используются для использования каждого типа уязвимости [12].

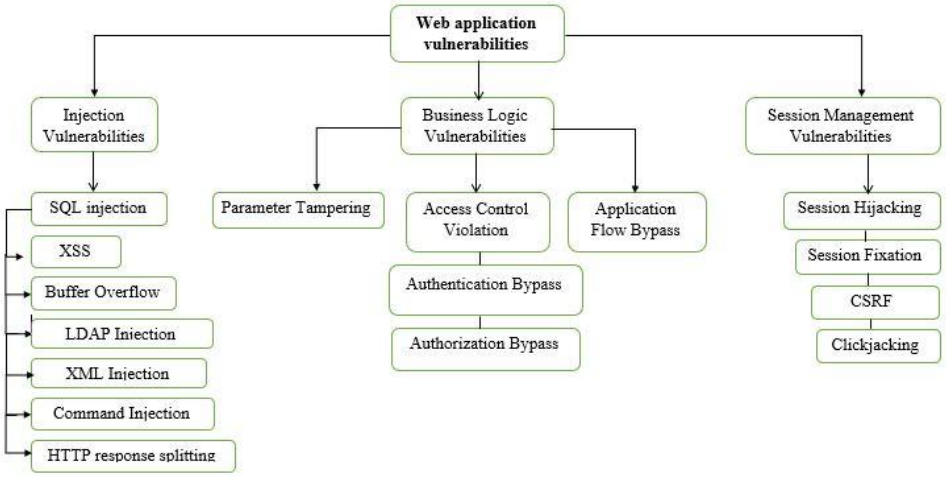

Рисунок 1: Классификация уязвимостей веб-приложений

Уязвимость инъекций.

Инъекция происходит, когда злоумышленник отправляет 5 ненадежных данных как часть явно допустимой команды или запроса, чтобы обмануть интерпретатор приложения и выполнить непреднамеренные команды. Наиболее распространенные типы внедрения — это внедрение SQL, межсайтовый скриптинг (XSS) и внедрение LDAP.

Уязвимости бизнес-логики.

Уязвимости бизнес-логики позволяют злоумышленникам манипулировать законной логикой приложения и выполнять незаконные транзакции. Эти уязвимости обычно используются путем изменения параметров, обхода ограничений аутентификации и авторизации и нарушения нормального потока приложения.

Уязвимости управления сеансами

Уязвимости управления сеансами позволяют злоумышленникам читать или манипулировать переменными сеанса, которые используются для поддержания состояния веб-приложений (например, состояния, когда пользователь вошел в приложение и имеет определенные права авторизации). Атаки захвата и фиксации сеанса нацелены на идентификатор сеанса пользователя, в то время как подделка межсайтовых запросов (CSRF) и взлом кликов нацеливают браузер клиента на отправку запросов, которые законный пользователь не хочет отправлять.

В 10 ведущих документах OWASP за 2013 и 2017 годы самые распространенные недостатки классифицируются по 10 категориям, которые приведены ниже.

OWASP Top 10 2013	±	OWASP Top 10 2017
A1 – Injection	→	A1:2017 – Injection
A2 – Broken Authentication and Session Management	→	A2:2017 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)	↘	A3:2013 – Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017 – XML External Entity (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017 – Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017 – Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017 – Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017 – Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017 – Insufficient Logging & Monitoring [NEW, Comm.]

Рисунок 2: OWASP Top 10 2017 недостатки безопасности

Обнаружение конфиденциальных данных

Прежде всего необходимо определить потребности в защите данных при передаче и в состоянии покоя [4]. Например, пароли, номера кредитных карт, медицинские карты, личная информация и коммерческая тайна требуют дополнительной защиты, особенно если эти данные подпадают под действие законов о конфиденциальности, например, Общее регулирование защиты данных ЕС (GDPR) или нормативные акты, например, защита финансовых данных, таких как стандарт безопасности данных PCI (PCI DSS). Для всех таких данных:

- Передаются ли какие-либо данные в виде открытого текста? Это касается таких протоколов, как HTTP, SMTP и FTP. Внешний интернет-трафик особенно опасен. Проверьте весь внутренний трафик, например между балансировщиками нагрузки, веб-серверами или внутренними системами.
- Хранятся ли конфиденциальные данные в виде открытого текста, включая резервные копии?
- Используются ли старые или слабые криптографические алгоритмы по умолчанию или в более старом коде?
- Используются ли крипто-ключи по умолчанию, генерируются или повторно используются слабые крипто-ключи, или отсутствует правильное управление ключами или ротация?
- Не применяется ли шифрование, например отсутствуют какие-либо директивы или заголовки безопасности агента пользователя (браузера)?
- Не проверяет ли пользовательский агент (например, приложение, почтовый клиент), действителен ли полученный сертификат сервера?

Пример:

Сценарий № 1: приложение шифрует номера кредитных карт в базе данных с использованием автоматического шифрования базы данных. Тем не менее, эти данные автоматически расшифровываются при получении, что позволяет недостатку SQL-инъекции получить номера кредитных карт в виде открытого текста.

Сценарий № 2. Сайт не использует и не применяет TLS для всех страниц или поддерживает слабое шифрование. Злоумышленник отслеживает сетевой трафик (например, в небезопасной беспроводной сети), понижает количество соединений с HTTPS до HTTP, перехватывает запросы и похищает файл cookie сеанса пользователя. Затем злоумышленник воспроизводит этот файл cookie и захватывает сеанс пользователя (прошедший проверку подлинности), получая доступ или изменяя личные данные пользователя. Вместо вышесказанного они могут изменить все транспортируемые данные, например, получатель денежного перевода.

Сценарий № 3: база данных паролей использует несоленые или простые хэши для хранения паролей всех пользователей. Ошибка загрузки файла позволяет злоумышленнику получить базу данных паролей. Все несоленые хэши могут быть выставлены с помощью радужного стола предварительно рассчитанных хэшей. Хэши, созданные простыми или быстрыми хэш-функциями, могут быть взломаны графическими процессорами, даже если они были засолены.

Возможные решения:

Выполните, как минимум, следующее и ознакомьтесь со ссылками:

- Классифицировать данные, обрабатываемые, хранящиеся или передаваемые приложением. Определите, какие данные являются конфиденциальными в соответствии с законами о конфиденциальности, нормативными требованиями или потребностями бизнеса.
- Применять элементы управления в соответствии с классификацией.
- Не храните конфиденциальные данные без необходимости. Откажитесь от него как можно скорее или используйте токены, совместимые с PCI DSS, или даже усечения. Данные, которые не были сохранены, не могут быть украдены.
- Обязательно шифруйте все конфиденциальные данные в состоянии покоя.
- Обеспечить наличие современных и надежных стандартных алгоритмов, протоколов и ключей; используйте правильное управление ключами.
- Зашифруйте все передаваемые данные с помощью безопасных протоколов, таких как TLS, с использованием шифров с идеальной прямой секретностью

(PFS), приоритезацией шифров сервером и безопасными параметрами. Обеспечить шифрование с использованием таких директив, как HTTP Strict Transport Security (HSTS).

• Отключите кэширование для ответов, которые содержат конфиденциальные данные.

• Храните пароли, используя сильные адаптивные и соленые функции хэширования с рабочим коэффициентом (фактором задержки), такие как Argon2, scrypt, bcrypt или PBKDF2.

• Проверять самостоятельно эффективность конфигурации и настроек.

Внешние сущности XML (XXE).

Приложения и, в частности, xmlbased веб-службы или нисходящие интеграции могут быть уязвимы для атаки, если[5]:

• Приложение принимает XML непосредственно или XML-загрузки, особенно из ненадежных источников, или вставляет ненадежные данные в XML-документы, которые затем анализируются обработчиком XML.

• Любой из XML-процессоров в приложении или веб-службах на основе SOAP имеет включенные определения типов документов (DTDs). Поскольку точный механизм отключения обработки DTD зависит от процессора, рекомендуется проконсультироваться со ссылкой, такой как OWASP Cheat Sheet"XXE Prevention".

• Если приложение использует SAML для обработки удостоверений в целях Федеративной безопасности или единого входа (SSO). SAML использует XML для утверждений идентификаторов и может быть уязвимым.

• Если приложение использует SOAP до версии 1.2, оно, вероятно, подвержено атакам XXE, если XML-объекты передаются в SOAP framework.

• Вероятность уязвимости к атакам XXE означает, что приложение уязвимо для атак отказа в обслуживании, включая атаку Billion Laughs.

Пример: обнаружены многочисленные проблемы public XXE, в том числе атакующие встроенные устройства. XXE встречается во многих неожиданных

местах, включая глубоко вложенные зависимости. Самый простой способ-загрузить вредоносный XML-файл, если он принят:

Сценарий № 1: злоумышленник пытается извлечь данные с сервера:

<?xml version="1.0" encoding="ISO8859-1"?>

<!DOCTYPE foo [

<!ELEMENT foo ANY >

<!ENTITY xxe

SYSTEM "file:///etc/passwd" >]>

<foo>&xxe;</foo> ;

Сценарий № 2. Злоумышленник проверяет частную сеть сервера, изменив приведенную выше строку ENTITY на:

<!ENTITY xxe SYSTEM

"https://192.168.1.1/private" >]>

Scenario #3: An attacker attempts a denial-of-service attack by including a potentially endless file:

<!ENTITY xxe SYSTEM

"file:///dev/random" >]>

Возможные решения:

Обучение разработчиков имеет важное значение для выявления и смягчения последствий XXE. Кроме того, предотвращение XXE требует:

- По возможности используйте менее сложные форматы данных, такие как JSON, и избегайте сериализации конфиденциальных данных.
- Исправлять или обновлять все процессоры и библиотеки XML, используемые приложением или в базовой операционной системе. Используйте проверки зависимостей. Обновите SOAP до SOAP 1.2 или выше.
- Отключите внешнюю сущность XML и обработку DTD во всех синтаксических анализаторах XML в приложении в соответствии с таблицей OWASP «Предотвращение XXE».

• Внедрить положительную («белый список») проверку, фильтрацию или очистку входных данных на стороне сервера, чтобы предотвратить враждебные данные в документах, заголовках или узлах XML.

• Убедитесь, что функция загрузки файлов XML или XSL проверяет входящий XML с использованием проверки XSD или аналогичной.

• Инструменты SAST могут помочь обнаружить XXE в исходном коде, хотя ручная проверка кода является лучшей альтернативой в больших и сложных приложениях с множеством интеграций. Если эти элементы управления невозможны, рассмотрите возможность использования виртуальных исправлений, шлюзов безопасности API или брандмауэров веб-приложений (WAF) для обнаружения, мониторинга и блокировки атак XXE.

Broken Access Control

Контроль доступа обеспечивает соблюдение политики, так что пользователи не могут действовать за пределами своих предполагаемых разрешений [6]. Сбои обычно приводят к несанкционированному раскрытию информации, изменению или уничтожению всех данных или выполнению бизнес-функций вне пределов пользователя. Общие уязвимости контроля доступа включают в себя:

• Обход проверок контроля доступа путем изменения URL-адреса, внутреннего состояния приложения или страницы HTML или просто с помощью специального инструмента атаки API.

• Разрешение замены первичного ключа на запись других пользователей, разрешение просмотра или редактирования чужой учетной записи.

• Повышение привилегий. Выступаете в роли пользователя, не входя в систему, или действуя как администратор, когда вы входите в систему как пользователь.

• манипулирование метаданными, например, воспроизведение или фальсификация с помощью токена контроля доступа JSON Web Token (JWT), файла cookie или скрытого поля, которые используются для повышения привилегий, или злоупотребление аннулированием JWT

- Неправильная настройка CORS допускает несанкционированный доступ к API.
- Принудительный просмотр страниц, прошедших проверку подлинности, как пользователя, не прошедшего проверку подлинности, или привилегированных страниц как обычный пользователь. Доступ к API с отсутствующими элементами управления доступом для POST, PUT и DELETE.

Пример:

Сценарий № 1. Приложение использует непроверенные данные в вызове SQL, который обращается к информации об учетной записи:

```
pstmt.setString(1,
request.getParameter("acct"));
ResultSet results = pstmt.executeQuery();
```

Злоумышленник просто изменяет параметр «acct» в браузере, чтобы отправить любой номер учетной записи, который он хочет. Если не проверено должным образом, злоумышленник может получить доступ к любой учетной записи пользователя.

http://example.com/app/accountInfo?acc t=notmyacct

Сценарий № 2. Злоумышленник просто заставляет просматривать целевые URL-адреса. Права администратора требуются для доступа к странице администратора.

http://example.com/app/getappInfo

http://example.com/app/admin_getappInfo

Если неаутентифицированный пользователь может получить доступ к любой странице, это недостаток. Если не администратор может получить доступ к странице администратора, это недостаток.

Предупреждение. Контроль доступа эффективен только в том случае, если он применяется в доверенном серверном коде или в бессерверном API, где злоумышленник не может изменить проверку контроля доступа или метаданные.

- За исключением общедоступных ресурсов, по умолчанию запретите.

- Реализуйте механизмы контроля доступа один раз и повторно используйте их во всем приложении, включая минимизацию использования CORS.
- Элементы управления доступом к модели должны обеспечивать владение записями, а не признавать, что пользователь может создавать, читать, обновлять или удалять любые записи.
- Уникальные требования к бизнес-ограничениям приложений должны соблюдаться моделями доменов.
- Отключите список каталогов веб-сервера и убедитесь, что метаданные файлов (например, .git) и файлы резервных копий отсутствуют в корнях веб-сайтов.
- Журнал ошибок контроля доступа, при необходимости предупредить администраторов (например, повторные сбои).
- Ограничение скорости API и доступ к контроллеру, чтобы минимизировать вред от автоматизированного инструментария атаки.
- Токены JWT должны быть признаны недействительными на сервере после выхода из системы. Разработчики и сотрудники QA должны включать функциональный блок контроля доступа и интеграционные тесты.

Security Misconfiguration

Приложение может быть уязвимо, если [7]:

- Отсутствует соответствующее усиление безопасности в любой части стека приложений или неправильно настроенные разрешения для облачных сервисов.
- Ненужные функции включены или установлены (например, ненужные порты, службы, страницы, учетные записи или привилегии).
- Учетные записи по умолчанию и их пароли по-прежнему включены и не изменены.
- Обработка ошибок выявляет следы стека или другие чрезмерно информативные сообщения об ошибках для пользователей.
- Для модернизированных систем новейшие функции безопасности отключены или настроены ненадежно.

• Параметры безопасности на серверах приложений, в платформах приложений (например, Struts, Spring, ASP.NET), библиотеках, базах данных и т. Д. Не установлены на безопасные значения.

• Сервер не отправляет заголовки безопасности или директивы, или они не установлены в безопасные значения.

• Программное обеспечение устарело или уязвимо (см. A9: 2017 - Использование компонентов с известными уязвимостями). Без согласованного, повторяемого процесса настройки безопасности приложений системы подвергаются более высокому риску.

Пример:

Сценарий № 1: Сервер приложений поставляется с примерами приложений, которые не удаляются с рабочего сервера. Эти примеры приложений имеют известные уязвимости, которые злоумышленники используют для взлома сервера. Если одним из этих приложений является консоль администратора, и учетные записи по умолчанию не были изменены, злоумышленник входит в систему с паролями по умолчанию и вступает во владение.

Сценарий № 2: Список каталогов не отключен на сервере. Злоумышленник обнаруживает, что он может просто перечислить каталоги. Злоумышленник находит и загружает скомпилированные классы Java, которые они декомпилируют и перепроектируют для просмотра кода. Затем злоумышленник обнаруживает серьезный недостаток контроля доступа в приложении.

Сценарий № 3: Конфигурация сервера приложений допускает подробные сообщения об ошибках, например, следы стека, которые будут возвращены пользователям. Это потенциально может раскрыть конфиденциальную информацию или лежащие в ее основе недостатки, такие как версии компонентов, которые, как известно, уязвимы. Сценарий № 4. У поставщика облачных услуг есть разрешения на совместное использование по умолчанию, открытые для Интернета другими пользователями CSP. Это позволяет получить доступ к конфиденциальным данным, хранящимся в облачном хранилище.

Возможные решения:

Должны быть внедрены безопасные процессы установки, в том числе:

• Повторяемый процесс укрепления, который позволяет быстро и легко развернуть другую среду, которая должным образом заблокирована. Среды разработки, контроля качества и производства должны быть настроены одинаково, с разными учетными данными, используемыми в каждой среде. Этот процесс должен быть автоматизирован, чтобы минимизировать усилия, необходимые для настройки новой безопасной среды.

• Минимальная платформа без лишних функций, компонентов, документации и примеров. Удалите или не устанавливайте неиспользуемые функции и рамки.

• Задача по проверке и обновлению конфигураций, подходящих для всех заметок по безопасности, обновлений и исправлений в рамках процесса управления исправлениями (см. Использование компонентов с известными уязвимостями). В частности, проверьте разрешения облачного хранилища (например, разрешения S3 Bucket).

• Сегментированная прикладная архитектура, которая обеспечивает эффективное и безопасное разделение между компонентами или арендаторами с сегментацией, контейнеризацией или облачными группами безопасности.

• Отправка директив по безопасности клиентам, например, Заголовки безопасности.

• Автоматизированный процесс проверки эффективности конфигураций и настроек во всех средах.

Межсайтовый скриптинг (XSS)

Существуют три формы XSS, обычно предназначенные для браузеров пользователей [8]:

Отраженный XSS. Приложение или API включают в себя не проверенный и не экранированный ввод пользователя как часть вывода HTML. Успешная атака может позволить злоумышленнику выполнить произвольный HTML и JavaScript в браузере жертвы. Как правило, пользователю необходимо взаимодействовать с какой-либо вредоносной ссылкой, указывающей на контролируемую

злоумышленником страницу, например вредоносными веб-сайтами, рекламирующими сайты, или аналогичными.

Хранимый XSS. Приложение или API хранит неанизированный пользовательский ввод, который позже просматривается другим пользователем или администратором. Сохраненный XSS часто считается высоким или критическим риском.

DOM XSS. Платформы JavaScript, одностраничные приложения и API, которые динамически включают контролируемые злоумышленником данные на страницу, уязвимы для DOM XSS. В идеале приложение не должно отправлять контролируемые злоумышленником данные в небезопасные API-интерфейсы JavaScript. Типичные атаки XSS включают в себя кражу сеанса, захват учетной записи, обход MFA, замену или удаление узла DOM (например, панели входа в троян), атаки на браузер пользователя, такие как загрузка вредоносного программного обеспечения, регистрация ключей и другие атаки на стороне клиента.

Пример:

Сценарий 1. Приложение использует ненадежные данные при создании следующего фрагмента HTML-кода без проверки или экранирования:

(String)page+="<input name='creditcard' type='TEXT' value='" + request.getParameter("CC") + "'>";

Атакующий изменяет параметр «CC» в браузере, чтобы:

'><script>document.location='http://www.attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>'.

Эта атака приводит к отправке идентификатора сеанса жертвы на веб-сайт злоумышленника, позволяя злоумышленнику захватить текущий сеанс пользователя. Примечание. Злоумышленники могут использовать XSS для победы над любой автоматической защитой CrossSite Request Forgery (CSRF), которую может использовать приложение.

Возможные решения:

Для предотвращения XSS необходимо отделить ненадежные данные от активного содержимого браузера. Это может быть достигнуто путем:

• Использование каркасов, которые автоматически выходят из XSS, таких как последняя версия Ruby on Rails, React JS. Изучите ограничения XSS-защиты каждого фреймворка и соответствующим образом обработайте варианты использования, которые не охвачены.

• Выход из ненадежных данных HTTP-запроса на основе контекста в выводе HTML (тело, атрибут, JavaScript, CSS или URL) устранит уязвимости отраженного и сохраненного XSS. Шпаргалка OWASP «XSS Prevention» содержит подробную информацию о необходимых методах экранирования данных.

• Применение контекстно-зависимой кодировки при изменении документа браузера на стороне клиента действует против DOM XSS. Когда этого нельзя избежать, к интерфейсам API браузера можно применить аналогичные контекстно-зависимые методы экранирования, как описано в Шпаргалке OWASP «Предотвращение XSS на основе DOM».

• Включение политики безопасности контента (CSP) - это глубоко уравновешивающий смягчающий контроль против XSS. Это эффективно, если не существует других уязвимостей, которые позволили бы размещать вредоносный код через локальные файлы (например, перезапись пути или уязвимые библиотеки из разрешенных сетей доставки контента). Небезопасная десериализация: [9] Приложения и API будут уязвимы, если они десериализуют враждебные или взломанные объекты, предоставленные злоумышленником. Это может привести к двум основным типам атак:

• Атаки, связанные с объектами и структурой данных, когда злоумышленник изменяет логику приложения или достигает произвольного удаленного выполнения кода, если приложению доступны классы, которые могут изменить поведение во время или после десериализации.

• Типичные атаки с фальсификацией данных, такие как атаки, связанные с контролем доступа, когда используются существующие структуры данных, но

содержимое изменяется. Сериализация может использоваться в приложениях для: • Удаленного и межпроцессного взаимодействия (RPC / IPC)

• Проводные протоколы, веб-сервисы, брокеры сообщений

• Кеширование / Постоянство

• Базы данных, кеш-серверы, файловые системы

• HTTP-куки, параметры HTML-формы, токены аутентификации API

Пример:

Сценарий № 1. Приложение React вызывает набор микросервисов Spring Boot. Будучи функциональными программистами, они старались обеспечить неизменность своего кода. Решение, которое они придумали, заключается в сериализации состояния пользователя и передаче его туда и обратно с каждым запросом. Злоумышленник замечает сигнатуру Java-объекта «R00» и использует инструмент Java Serial Killer для удаленного выполнения кода на сервере приложений.

Сценарий № 2. Форум PHP использует сериализацию объектов PHP для сохранения «супер» cookie, содержащего идентификатор пользователя, роль, хэш пароля и другое состояние:

a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}

Злоумышленник изменяет сериализованный объект, чтобы предоставить ему права администратора:

a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}

Возможные решения:

Единственный безопасный архитектурный шаблон - не принимать сериализованные объекты из ненадежных источников или использовать среды сериализации, которые допускают только примитивные типы данных. Если это невозможно, рассмотрите одно из следующих:

- Реализация проверок целостности, таких как цифровые подписи, на любых сериализованных объектах для предотвращения создания враждебных объектов или подделки данных.
- Обеспечение строгих ограничений типов во время десериализации перед созданием объекта, так как код обычно ожидает определяемый набор классов. Обходные пути для этой техники были продемонстрированы, поэтому полагаться исключительно на это не рекомендуется.
- Выделение и запуск кода, который по возможности десериализуется в средах с низким уровнем привилегий.
- Регистрация исключений и сбоев десериализации, например, когда входящий тип не является ожидаемым типом или десериализация генерирует исключения.
- Ограничение или мониторинг входящих и исходящих сетевых подключений от контейнеров или серверов, которые десериализуются.
- Мониторинг десериализации, оповещение о постоянной десериализации пользователя.

Использование компонентов с известными уязвимостями:

Вы, вероятно, уязвимы [10]:

- Если вы не знаете версии всех используемых вами компонентов (как на стороне клиента, так и на стороне сервера). Это включает в себя компоненты, которые вы непосредственно используете, а также вложенные зависимости.
- Если программное обеспечение уязвимо, не поддерживается или устарело. Это включает в себя ОС, веб-сервер / сервер приложений, систему управления базами данных (СУБД), приложения, API и все компоненты, среды выполнения и библиотеки.
- Если вы регулярно не сканируете уязвимости и подписываетесь на бюллетени по безопасности, связанные с компонентами, которые вы используете.

- Если вы не исправляете или не обновляете базовую платформу, платформы и зависимости своевременно и на основе рисков. Это обычно происходит в средах, когда исправление является ежемесячной или ежеквартальной задачей под контролем изменений, что оставляет организации открытыми для многих дней или месяцев ненужного воздействия фиксированных уязвимостей.
- Если разработчики программного обеспечения не проверяют совместимость обновленных, обновленных или исправленных библиотек.
- Если вы не защищаете конфигурации компонентов (см. A6: 2017 - Неправильная настройка безопасности).

Пример:

Сценарий № 1. Компоненты обычно работают с теми же привилегиями, что и само приложение, поэтому недостатки любого компонента могут привести к серьезным последствиям. Такие недостатки могут быть случайными (например, ошибка кодирования) или преднамеренными (например, бэкдор в компоненте). Вот некоторые обнаруженные уязвимости компонента:

- CVE-2017-5638, уязвимость удаленного выполнения кода в Struts 2, которая позволяет выполнять произвольный код на сервере, обвиняется в существенных нарушениях.
- Хотя интернет вещей (IoT) часто трудно или невозможно исправить, важность исправления может быть велика (например, биомедицинские устройства).

Существуют автоматизированные инструменты, помогающие злоумышленникам находить не исправленные или неправильно настроенные системы. Например, поисковая система Shodan IoT может помочь вам найти устройства, которые по-прежнему страдают от уязвимости Heartbleed, исправленной в апреле 2014 года.

Возможные решения. Должен быть процесс управления исправлениями, чтобы:

- Удалить неиспользуемые зависимости, ненужные функции, компоненты, файлы и документацию.

- Постоянно проводить инвентаризацию версий как клиентских, так и серверных компонентов (например, каркасов, библиотек) и их зависимостей, используя такие инструменты, как версии, DependencyCheck, retire.js и т. Д. Постоянно отслеживайте источники, такие как CVE и NVD, на наличие уязвимостей в компонентах. Используйте инструменты анализа состава программного обеспечения для автоматизации процесса. Подпишитесь на уведомления по электронной почте об уязвимостях безопасности, связанных с компонентами, которые вы используете.
- Получать компоненты только из официальных источников по защищенным ссылкам. Предпочитайте подписанные пакеты, чтобы уменьшить вероятность включения измененного вредоносного компонента.
- Мониторинг библиотек и компонентов, которые не поддерживаются или не создают исправлений безопасности для более старых версий. Если исправление невозможно, рассмотрите возможность развертывания виртуального исправления для мониторинга, обнаружения или защиты от обнаруженной проблемы.

Каждая организация должна убедиться, что существует постоянный план мониторинга, сортировки и применения обновлений или изменений конфигурации в течение всего жизненного цикла приложения или портфеля.

Недостаточная регистрация и мониторинг

Недостаточное ведение журнала, обнаружение, мониторинг и активный ответ происходят в любое время[11]:

- Аудиторские события, такие как входы в систему, неудачные входы в систему и транзакции с высокой стоимостью, не регистрируются.
- Предупреждения и ошибки не генерируют, неадекватные или нечеткие сообщения журнала.
- Журналы приложений и API не отслеживаются на предмет подозрительной активности.
- Журналы хранятся только локально.

- Соответствующие пороги оповещения и процессы эскалации реагирования отсутствуют или не эффективны.
- Тестирование на проникновение и сканирование с помощью инструментов DAST (таких как OWASP ZAP) не вызывают оповещения.
- Приложение не может обнаруживать, расширять или оповещать об активных атаках в режиме реального времени или почти в реальном времени. Вы уязвимы для утечки информации, если делаете запись и оповещение о событиях видимыми для пользователя или злоумышленника (см. Обнаружение конфиденциальной информации).

Пример:

Сценарий № 1: Программное обеспечение для форумов с открытым исходным кодом, работающее небольшой группой, было взломано с использованием недостатка в его программном обеспечении. Злоумышленникам удалось уничтожить внутренний репозиторий исходного кода, содержащий следующую версию, и все содержимое форума. Хотя источник может быть восстановлен, отсутствие мониторинга, регистрации или оповещения привело к гораздо худшему нарушению. В результате этой проблемы проект программного обеспечения форума больше не активен.

Сценарий № 2. Злоумышленник использует сканирование пользователей, использующих общий пароль. Они могут взять на себя все учетные записи, используя этот пароль. Для всех остальных пользователей это сканирование оставляет только один ложный логин. Через несколько дней это можно повторить с другим паролем.

Сценарий № 3: у крупного американского ритейлера, по сообщениям, была внутренняя песочница для анализа вредоносных программ, анализирующая вложения. Программное обеспечение «песочницы» обнаружило потенциально нежелательное программное обеспечение, но никто не ответил на это обнаружение. Песочница в течение некоторого времени выдавала предупреждения, прежде чем было обнаружено нарушение из-за мошеннических операций с картами со стороны внешнего банка.

Возможные решения:

- Убедитесь, что все ошибки входа в систему, контроля доступа и проверки входных данных на стороне сервера могут регистрироваться с достаточным контекстом пользователя для выявления подозрительных или злонамеренных учетных записей и храниться в течение достаточного времени для проведения отложенного судебного анализа.
- Убедитесь, что журналы создаются в формате, который можно легко использовать для решений централизованного управления журналами.
- Убедитесь, что у транзакций с высокой стоимостью есть контрольный журнал с элементами управления целостностью, чтобы предотвратить подделку или удаление, такие как таблицы базы данных только для добавления или аналогичные.
- Установить эффективный мониторинг и оповещение таким образом, чтобы подозрительные действия обнаруживались и своевременно реагировались.
- Разработать или принять план реагирования на инциденты и восстановления, такой как NIST 800-61 rev 2 или более поздний. Существуют коммерческие и открытые среды защиты приложений, такие как OWASP AppSensor, брандмауэры веб-приложений, такие как ModSecurity с базовым набором правил OWASP ModSecurity, и программное обеспечение для корреляции журналов с настраиваемыми информационными панелями и оповещениями.

В следующем рисунке приведена сводка уязвимостей веб-приложений, упомянутых в этой главе, включая некоторые их характеристики и связанные с ними атаки.

RISK	Threat Agents	Attack Vectors: Exploitability	Security Weakness: Prevalence	Security Weakness: Detectability	Impacts: Technical	Impacts: Business	Score
A1:2017-Injection	App Specific	EASY: 3	COMMON: 2	EASY: 3	SEVERE: 3	App Specific	8.0
A2:2017-Authentication	App Specific	EASY: 3	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	7.0
A3:2017-Sens. Data Exposure	App Specific	AVERAGE: 2	WIDESPREAD: 3	AVERAGE: 2	SEVERE: 3	App Specific	7.0
A4:2017-XML External Entities (XXE)	App Specific	AVERAGE: 2	COMMON: 2	EASY: 3	SEVERE: 3	App Specific	7.0
A5:2017-Broken Access Control	App Specific	AVERAGE: 2	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	6.0
A6:2017-Security Misconfiguration	App Specific	EASY: 3	WIDESPREAD: 3	EASY: 3	MODERATE: 2	App Specific	6.0
A7:2017-Cross-Site Scripting (XSS)	App Specific	EASY: 3	WIDESPREAD: 3	EASY: 3	MODERATE: 2	App Specific	6.0
A8:2017-Insecure Deserialization	App Specific	DIFFICULT: 1	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	5.0
A9:2017-Vulnerable Components	App Specific	AVERAGE: 2	WIDESPREAD: 3	AVERAGE: 2	MODERATE: 2	App Specific	4.7
A10:2017-Insufficient Logging&Monitoring	App Specific	AVERAGE: 2	WIDESPREAD: 3	DIFFICULT: 1	MODERATE: 2	App Specific	4.0

Рисунок 3: Сводка уязвимостей веб-приложений и связанных с ними атак

ЗАКЛЮЧЕНИЕ

Веб-приложения становятся популярными и имеют широкое распространение в нашей повседневной жизни. Но в то же время, используя уязвимости, конфиденциальные данные также регулярно публикуются. В этом документе рассматриваются различные уязвимости веб-приложений на основе свойств безопасности, которые веб-приложение должно сохранять. Однако мы также обсудили возможные решения, которые снижают вероятность появления уязвимостей. Однако внедрение решений по снижению уязвимости защищает веб-приложения от современных угроз. В конце концов, всегда появляются новые расширенные атаки безопасности, требующие, чтобы специалист по безопасности имел позитивное решение безопасности, не подвергая риску огромное количество веб-приложений.

СПИСОК ЛИТЕРАТУРЫ

1. OWASP Top Ten Project. [Электронный ресурс]. – Режим доступа: https://www.owasp.org/index.php/Categ ory:OWASP_Top_Ten_Project (дата обращения: 05.04.2019).
2. Injection. [Электронный ресурс]. – Режим доступа: https://www.owasp.org/index.php/Top_1 0-2017_A1-Injection (дата обращения: 05.04.2019).
3. Broken Authentication. [Электронный ресурс]. – Режим доступа: https://www.owasp.org/index.php/Top_1 0-2017_A2-Broken_Authentication (дата обращения: 05.04.2019).
4. Sensitive Data Exposure. [Электронный ресурс]. – Режим доступа: https://www.owasp.org/index.php/Top_1 0-2017_A3-Sensitive_Data_Exposure (дата обращения: 28.03.2019).
5. XML External Entities (XXE) . [Электронный ресурс]. – Режим доступа: https://www.owasp.org/index.php/Top_10-2017_A4XML_External_Entities_(XXE) (дата обращения: 30.03.2019).
6. Broken Access Control. [Электронный ресурс]. – Режим доступа: https://www.owasp.org/index.php/Top_1 0-2017_A5-Broken_Access_Control (дата обращения: 02.04.2019).
7. Security Misconfiguration. [Электронный ресурс]. – Режим доступа: https://www.owasp.org/index.php/Top_1 0-2017_A6-Security_Misconfiguration (дата обращения: 04.04.2019).
8. Cross-Site Scripting (XSS) . [Электронный ресурс]. – Режим доступа: https://www.owasp.org/index.php/Top_1 0-2017_A7-Cross-Site_Scripting_(XSS) (дата обращения: 06.04.2019).
9. Insecure Deserialization. [Электронный ресурс]. – Режим доступа: https://www.owasp.org/index.php/Top_1 0-2017_A8-Insecure_Deserialization (дата обращения: 07.04.2019).
10. Using Components with Known Vulnerabilities. [Электронный ресурс]. – Режим доступа: https://www.owasp.org/index.php/Top_10-

2017_A9Using_Components_with_Known_Vulnerabilities (дата обращения: 09.04.2019).

11. Insufficient Logging & Monitoring. [Электронный ресурс]. – Режим доступа: https://www.owasp.org/index.php/Top_1 0-2017_A10Insufficient_Logging%26Monitoring (дата обращения: 10.04.2019).

12. G. Deepa and P. S. Thilagam, "Securing web applications from injection and logic vulnerabilities: Approaches and challenges," Information and Software Technology, no. 74, pp. 160-180, 2016.

13. Application Security Risks. [Электронный ресурс]. – Режим доступа: https://www.owasp.org/index.php/Top_1 0-2017_Application_Security_Risks (дата обращения: 10.04.2019).

14. OWASP Foundation, "OWASP Top10 2013," 2013. [Электронный ресурс]. – Режим доступа: https://www.owasp.org/images/f/f8/OWA SP_Top_10_-_2013.pdf. (дата обращения: 12.04.2019).

15. OWASP Foundation, "OWASP Top10 2017 (Release Candidate 1)," 2017. [Электронный ресурс]. – Режим доступа: ttps://github.com/OWASP/Top10/raw/ master/2017/OWASP%20Top%2010%2 0%202017%20RC1-English.pdf. (дата обращения: 12.04.2019).