Дипломная работа по профессии «Специалист по информационной безопасности»

1. OSINT

Согласно задания необходимо просканировать приложение находящиеся по адресу **87.249.53.182**.

Был произведен поиск информации в сети интернет указанного адреса. Удалось выяснить адрес имеет 2 открытых порта **22** и **8060**. При сканировании с помощью **Nmap** удалось найти **22** порт.

Было проведено сканирование с помощью **shodan.io.** По его данным, было найдено:

* **Hostnames**: 608477-cg36175.tmweb.ru
* **Domains**: TMWEB.RU
* **Country**: Russian Federation
* **City**: Saint Petersburg
* **Organization**: TimeWeb Ltd.
* **ISP**: TimeWeb Ltd.
* **ASN**: AS9123
* **Operating System**: Ubuntu
* **22 port**: SSH-2.0-OpenSSH\_8.2p1 Ubuntu-4ubuntu0.2

С помощью **wireshark** удалось выяснить сто приложение взаимодействует с адресами **10.0.2.15**, **216.58.210.163 (Скрин 31)**

Приложение по адресу **87.249.53.182:8060** взаимодействует с приложением **87.249.53.182:7799**.

1. **Scanning and Testing**

С помощью программы **XSpider** сканировал адрес **87.249.53.182**. Программа показала следующие уязвимости:

1. В Режиме Safe scan (Скрин 32, 33, 34, 35, 36)
2. В режиме PCI DSS (Скрин 38-96)

С помощью **Owasp ZAP** было просканировано приложение **NetologyVuinApp.com (87.249.53.182:8060)**. По данным уязвимостям была сделана проверка на False Positive срабатывания и проведен пентест. Удалось обнаружить уязвимости:

1. **Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (CWE-79) (скрин 1) (proof-of-concept скрины 114 и 115) Критичность: 6.3 (CVSS v3)**
   * Решение:
     1. Этап: Архитектура и дизайн.
        1. Используйте проверенную библиотеку или платформу, которая не позволяет этому недостатку возникать или предоставляет конструкции, которые делают эту слабость легче избежать. Примеры библиотек и фреймворков, которые упрощают создание правильно закодированного вывода, включают библиотеку Microsoft Anti-XSS, модуль кодирования OWASP ESAPI и Apache Wicket.
     2. Этапы: реализация; Архитектура и дизайн.
        1. Поймите контекст, в котором будут использоваться ваши данные, и ожидаемую кодировку. Это особенно важно при передаче данных между различными компонентами или при генерации выходных данных, которые могут содержать несколько кодировок одновременно, например, веб-страницы или составные почтовые сообщения.
        2. Изучите все ожидаемые протоколы связи и представления данных, чтобы определить необходимые стратегии кодирования. Для любых данных, которые будут выводиться на другую веб-страницу, особенно для любых данных, полученных от внешних входов, используйте соответствующую кодировку для всех не буквенно-цифровых символов.
        3. Обратитесь к шпаргалке по предотвращению XSS для получения дополнительных сведений о необходимых типах кодирования и экранирования.
     3. Этап: архитектура и дизайн.
        1. Для любых проверок безопасности, которые выполняются на стороне клиента, убедитесь, что эти проверки дублируются на стороне сервера, чтобы избежать CWE-602.
        2. Злоумышленники могут обойти проверки на стороне клиента, изменив значения после того, как проверки были выполнены, или изменив клиента, чтобы полностью удалить проверки на стороне клиента. Затем эти измененные значения будут отправлены на сервер.
        3. Если возможно, используйте структурированные механизмы, которые автоматически обеспечивают разделение данных и кода. Эти механизмы могут обеспечивать соответствующее цитирование, кодирование и проверку автоматически вместо того, чтобы полагаться на разработчика, предоставляющего эту возможность в каждой точке, где генерируются выходные данные.
     4. Этап: реализация.
        1. Для каждой создаваемой веб-страницы используйте и укажите кодировку символов, такую как ISO-8859-1 или UTF-8.
        2. Если кодировка не указана, веб-браузер может выбрать другую кодировку, угадав, какая кодировка фактически используется веб-страницей. Это может привести к тому, что веб-браузер будет рассматривать определенные последовательности как особые, открывая клиента для тонких XSS-атак. Смотри CWE-116 для дополнительных мер, связанных с кодированием / экранированием.
        3. Чтобы предотвратить атаки XSS на файл cookie сеанса пользователя, установите для файла cookie сеанса значение HttpOnly.
        4. В браузерах, поддерживающих функцию HttpOnly (например, в более поздних версиях Internet Explorer и Firefox), этот атрибут может предотвратить доступ к cookie сеанса пользователя для вредоносных клиентских скриптов, использующих document.cookie. Это не полное решение, поскольку HttpOnly поддерживается не всеми браузерами.
        5. Что еще более важно, XMLHTTPRequest и другие мощные браузерные технологии предоставляют доступ для чтения к заголовкам HTTP, включая заголовок Set-Cookie, в котором установлен флаг HttpOnly.
        6. Предположим, что весь ввод злонамерен.
           1. Используйте стратегию проверки входных данных «принять заведомо исправные», т. е. Используйте разрешенный список допустимых входных данных, которые строго соответствуют спецификациям.
           2. Отклоняйте любой ввод, который не строго соответствует спецификациям, или преобразуйте его во что-то, что соответствует.
           3. Не полагайтесь исключительно на поиск вредоносных или искаженных входных данных (т. е. Не полагайтесь на список запрещенных). Однако списки запрета могут быть полезны для обнаружения потенциальных атак или определения того, какие входные данные настолько искажены, что их следует сразу отклонить.
           4. При выполнении проверки ввода учитывайте все потенциально важные свойства, включая длину, тип ввода, полный диапазон допустимых значений, отсутствующие или дополнительные вводы, синтаксис, согласованность между связанными полями и соответствие бизнес-правилам. В качестве примера логики бизнес-правила «лодка» может быть синтаксически допустимой, потому что она содержит только буквенно-цифровые символы, но недействительна, если вы ожидаете такие цвета, как «красный» или «синий».
           5. Убедитесь, что вы выполняете проверку ввода на четко определенных интерфейсах в приложении. Это поможет защитить приложение, даже если компонент будет повторно использован или перемещен в другое место.
2. **Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (CWE-89) (Скрин 4, Скрин 5) (proof-of-concept скрины 116 и 117) Критичность: 7.6 (CVSS v3)**
   * Решение:
     1. Не доверяйте набору ввода клиента, даже если на месте есть проверка на стороне клиента. Как правило, введите все данные на стороне сервера.
     2. Если приложение использует JDBC, используйте PreparedStatement или CallableStatement, с параметрами, переданными «?».
     3. Если приложение использует ASP, используйте объекты команды ADO (Command Objects) с проверкой и параметризованными запросами типа (strong type).
     4. Если можно использовать хранимые процедуры в базе данных, используйте их.
     5. Делайте \* не \* объединенные строки в запросы в хранимой процедуре или используйте 'exec', 'exec immediate' или эквивалентную функциональность!
     6. Не создавайте динамические запросы SQL (create dynamic SQL), используя простую строковую (string) конкатенацию.
     7. Избежать всех данных (data), полученных от клиента (client).
     8. Примените «Разрешить список» 'allow list' из разрешенных символов, или «Список запрещений» (deny list') запрещенных символов пользовательского ввода.
     9. Примените привилегию из наименьших привилегий, используя наименее привилегированную базу данных (database user). В частности, избегайте пользователей базы данных «SA» ('sa') или «DB-владельца»('db-owner'). Это не устраняет SQL Инъекцию, но минимизирует её влияние.
     10. Предоставить минимальный доступ к базе данных (minimum database access), который необходимо для приложения.
3. **Missing Encryption of Sensitive Data (CWE-311) (Скрин и proof-of-concept 6) Критичность: 9.6 (CVSS v3)**
   * Решение:
     1. Настройте свой веб-сервер или сервер приложений для использования SSL (HTTPS).
4. **XML Injection (aka Blind XPath Injection) (CWE-91) (Скрин 7) (proof-of-concept скрины 119) Критичность: 7.2 (CVSS v3)**
   * Решение:
     1. Дезинфицируйте и анализируйте каждый пользовательский ввод, поступающий с любой стороны клиента.
5. **Protection Mechanism Failure (CWE-693) (proof-of-concept Скрин 8 Скрин 18, Скрин 20 ) Критичность: 7.1 (CVSS v3)**
   * Решение:
     1. Убедитесь, что ваш веб-сервер, сервер приложений, балансировщик нагрузки и т. д. настроены для установки заголовка Content-Security-Policy для достижения оптимальной поддержки браузера: «Content-Security-Policy» для Chrome 25+, Firefox 23+ и Safari 7. +, «X-Content-Security-Policy» для Firefox 4.0+ и Internet Explorer 10+ и «X-WebKit-CSP» для Chrome 14+ и Safari 6+.
     2. Убедитесь, что ваш веб-сервер, сервер приложений, балансировщик нагрузки и т. д. настроены для установки заголовка Permissions-Policy.
6. **Improper Restriction of Rendered UI Layers or Frames (CWE-1021) (Скрин и proof-of-concept 9) Критичность: 6.8 (CVSS v3)**
   * Решение
     1. Современные веб-браузеры поддерживают Content-Security-Policy и заголовки HTTP X-Frame-Options. Убедитесь, что один из них установлен на всех веб-страницах, возвращаемых вашим сайтом/приложением.
     2. Если вы ожидаете, что страница будет обрамлена только страницами на вашем сервере (например, это часть FRAMESET), вам следует использовать SAMEORIGIN, в противном случае, если вы никогда не ожидаете, что страница будет обрамлена, вам следует использовать DENY. В качестве альтернативы рассмотрите возможность реализации директивы Content Security Policy «frame-ancestors».
7. **Cross-Site Request Forgery (CSRF) (CWE-352) (Скрин 10, Скрин 11) Критичность: 8.3 (CVSS v3)**
   * Решение:
     1. Этап: Архитектура и дизайн.
        1. Используйте проверенную библиотеку или платформу, которая не позволяет этому недостатку возникать или предоставляет конструкции, которые делают эту слабость легче избежать. Например, используйте пакеты анти-CSRF, такие как OWASP CSRFGuard.
     2. Этап: реализация.
        1. Убедитесь, что в вашем приложении нет проблем с межсайтовыми сценариями, потому что большинство средств защиты CSRF можно обойти с помощью сценария, управляемого злоумышленником.
     3. Этап: архитектура и дизайн.
        1. Создайте уникальный одноразовый номер для каждой формы, поместите одноразовый номер в форму и проверьте одноразовый номер после получения формы.
        2. Убедитесь, что одноразовый номер непредсказуем (CWE-330). Обратите внимание, что это можно обойти с помощью XSS.
        3. Определите особо опасные операции.
        4. Когда пользователь выполняет опасную операцию, отправьте отдельный запрос подтверждения, чтобы убедиться, что пользователь намеревался выполнить эту операцию.
        5. Обратите внимание, что это можно обойти с помощью XSS.
        6. Используйте элемент управления ESAPI Session Management. Этот элемент управления включает компонент для CSRF.
        7. Не используйте метод GET для любого запроса, который вызывает изменение состояния.
     4. Этап: реализация.
        1. Проверьте заголовок HTTP Referer, чтобы узнать, исходит ли запрос от ожидаемой страницы. Это может нарушить законную функциональность, поскольку пользователи или прокси-серверы могли отключить отправку Referer по соображениям конфиденциальности.
8. **Exposure of Information Through Directory Listing (CWE-548) (Скрин 12) (proof-of-concept скрины 118) Критичность: 7.5 (CVSS v3)**
   * Решение:
     1. Отключить просмотр каталогов. Если это необходимо, убедитесь, что перечисленные файлы не представляют опасности.
9. **Improper Input Validation (CWE-20) (Скрин 13, Скрин 27) (proof-of-concept скрины 114 и 115) Критичность: 7.3 (CVSS v3)**
   * Решение:
     1. Веб-серверы и фреймворки должны быть настроены так, чтобы они не обслуживали ответы на неоднозначные URL-адреса таким образом, чтобы относительный путь таких URL-адресов мог быть неправильно интерпретирован компонентами либо на стороне клиента, либо на стороне сервера.
     2. В приложении правильное использование HTML-тега «<base>» в HTTP-ответе однозначно определит базовый URL-адрес для всех относительных URL-адресов в документе.
     3. Используйте заголовок HTTP-ответа Content-Type, чтобы злоумышленнику было сложнее заставить веб-браузер неверно интерпретировать тип содержимого ответа.
     4. Используйте заголовок HTTP-ответа «X-Content-Type-Options: nosniff», чтобы веб-браузер не «обнюхивал» тип содержимого ответа.
     5. Используйте современный DOCTYPE, такой как «<! Doctype html>», чтобы предотвратить отображение страницы в веб-браузере с использованием «Quirks Mode», поскольку это приводит к тому, что тип контента игнорируется веб-браузером.
     6. Укажите заголовок HTTP-ответа «X-Frame-Options», чтобы предотвратить включение режима Quirks Mode в веб-браузере с использованием фрейминговых атак.
     7. Проверяйте все входные данные и очищайте выходные данные перед записью в какие-либо атрибуты HTML.
10. **Sensitive Cookie Without 'HttpOnly' Flag (CWE-1004) (Скрин и proof-of-concept 15) Критичность: 6.5 (CVSS v3)**
    * Решение:
      1. Убедитесь, что для всех файлов cookie установлен флаг HttpOnly.
11. **Sensitive Cookie with Improper SameSite Attribute (CWE-1275) ) (Скрин и proof-of-concept 16) Критичность: 5.3 (CVSS v3)**
    * Решение:
      1. Убедитесь, что для атрибута SameSite установлено значение «слабый» ('lax') или в идеальном случае «строгий» (strict') для всех файлов cookie.
12. **Exposure of Sensitive Information to an Unauthorized Actor (CWE-200) (proof-of-concept Скрин 17, Скрин 19, Скрин 21, Скрин 22, Скрин 23, Скрин 28, Скрин 29) Критичность: 8.2 (CVSS v3)**
    * Решение:
      1. Настройте сервер для предотвращения таких утечек информации. Например: В Tomcat это делается с помощью директивы «сервер» и реализации пользовательских страниц ошибок. В Apache это делается с помощью директив ServerSignature и ServerTokens.
      2. Вручную подтвердите, что данные отметки времени не являются конфиденциальными и что данные не могут быть агрегированы для выявления шаблонов, которые можно использовать.
      3. Удалите частный IP-адрес из тела ответа HTTP. Для комментариев используйте комментарий JSP / ASP / PHP вместо комментария HTML / JavaScript, который могут видеть клиентские браузеры.
      4. Убедитесь, что ваш веб-сервер, сервер приложений, балансировщик нагрузки и т. д. настроены на подавление заголовков «X-Powered-By».
      5. Не передавайте конфиденциальную информацию в URI.

При сканировании приложения **87.249.53.182:7799** с помощью **OWASP ZAP** были выявлены уязвимости. По данным уязвимостям была сделана проверка на False Positive срабатывания и проведен пентест:

1. **Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (CWE-89) (Скрин 98) (proof-of-concept Скрины 121 и 122) Критичность: 8.2 (CVSS v3)**
   1. Решение:
      1. Не доверяйте вводу на стороне клиента, даже если есть проверка на стороне клиента. В общем, проверьте type всеx данныx на стороне сервера.
      2. Если приложение использует JDBC, используйте PreparedStatement или CallableStatement с параметрами, передаваемыми через '?'
      3. Если приложение использует ASP, используйте объекты команд ADO (ADO Command Objects) со строгой проверкой типов (strong type) и параметризованными запросами.
      4. Если можно использовать хранимые процедуры базы данных (database Stored), используйте их.
      5. \* Не \* объединяйте строки в запросы в хранимой процедуре и не используйте 'exec', 'exec немедленно' или аналогичные функции!
      6. Не создавайте динамические запросы SQL (create dynamic SQL), используя простую конкатенацию строк.
      7. Экранировать все данные, полученные от клиента.
      8. Примените «разрешенный список» ('allow list') разрешенных символов или «запрещающий список» ('deny list') запрещенных символов при вводе пользователем.
      9. Примените минимальную привилегию, используя минимально возможного пользователя базы данных.
      10. В частности, избегайте использования пользователей базы данных «sa» или «db-owner». Это не устраняет SQL-инъекцию, но сводит к минимуму ее влияние.
      11. Предоставьте приложению минимальный доступ к базе данных.
2. **Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (CWE-78) (Скрин 99**) **(proof-of-concept Скрин 120) Критичность: 9.1 (CVSS v3)**
   1. По возможности используйте вызовы библиотеки, а не внешние процессы, чтобы воссоздать желаемую функциональность.
   2. Запустите свой код в «тюрьме» или подобной среде песочницы, которая устанавливает строгие границы между процессом и операционной системой. Это может эффективно ограничить, к каким файлам можно получить доступ в конкретном каталоге или какие команды могут выполняться вашим программным обеспечением.
   3. Примеры уровня ОС включают chroot jail Unix, AppArmor и SELinux. В общем, управляемый код может обеспечить некоторую защиту. Например, java.io.FilePermission в Java SecurityManager позволяет вам указывать ограничения на файловые операции.
   4. Это может быть невыполнимым решением, и оно только ограничивает влияние на операционную систему; остальная часть вашего приложения все еще может быть скомпрометирована.
   5. Для любых данных, которые будут использоваться для генерации команды, которая должна быть выполнена, держите как можно большую часть этих данных вне внешнего контроля. Например, в веб-приложениях это может потребовать сохранения команды локально в состоянии сеанса вместо отправки ее клиенту в скрытом поле формы.
   6. Используйте проверенную библиотеку или фреймворк, которые не допускают появления этой уязвимости или предоставляют конструкции, которые упрощают ее устранение. Например, рассмотрите возможность использования элемента управления кодированием ESAPI или аналогичного инструмента, библиотеки или инфраструктуры. Это поможет программисту кодировать выходные данные менее подверженным ошибкам.
   7. Если вам необходимо использовать динамически генерируемые строки запроса или команды, несмотря на риск, правильно заключите аргументы в кавычки и избегайте любых специальных символов в этих аргументах. Самый консервативный подход - экранировать или фильтровать все символы, которые не проходят чрезвычайно строгий список разрешений (например, все, что не является буквенно-цифровым или пробельным). Если некоторые специальные символы все еще необходимы, например, пробел, заключите каждый аргумент в кавычки после этапа экранирования / фильтрации. Будьте осторожны с инъекцией аргументов.
   8. Если выполняемая программа позволяет указывать аргументы во входном файле или из стандартного ввода, рассмотрите возможность использования этого режима для передачи аргументов вместо командной строки.
   9. Если возможно, используйте структурированные механизмы, которые автоматически обеспечивают разделение данных и кода. Эти механизмы могут обеспечивать соответствующее цитирование, кодирование и проверку автоматически вместо того, чтобы полагаться на разработчика, предоставляющего эту возможность в каждой точке, где генерируются выходные данные.
   10. Некоторые языки предлагают несколько функций, которые можно использовать для вызова команд. По возможности идентифицируйте любую функцию, которая вызывает командную оболочку, используя одну строку, и замените ее функцией, требующей отдельных аргументов. Эти функции обычно выполняют соответствующие кавычки и фильтрацию аргументов. Например, в C функция system () принимает строку, содержащую всю команду, которая должна быть выполнена, тогда как для execl (), execve () и других требуется массив строк, по одной для каждого аргумента. В Windows CreateProcess () принимает только одну команду за раз. В Perl, если system () предоставлен с массивом аргументов, он будет заключать в кавычки каждый из аргументов.
   11. Предположим, что весь ввод злонамерен. Используйте стратегию проверки входных данных «принять заведомо исправные», т. е. Используйте разрешенный список допустимых входных данных, которые строго соответствуют спецификациям. Отклоняйте любой ввод, который не строго соответствует спецификациям, или преобразуйте его во что-то, что соответствует. Не полагайтесь исключительно на поиск вредоносных или искаженных входных данных (т. е. Не полагайтесь на список запрещенных). Однако списки запрета могут быть полезны для обнаружения потенциальных атак или определения того, какие входные данные настолько искажены, что их следует сразу отклонить.
   12. При выполнении проверки ввода учитывайте все потенциально важные свойства, включая длину, тип ввода, полный диапазон допустимых значений, отсутствующие или дополнительные вводы, синтаксис, согласованность между связанными полями и соответствие бизнес-правилам. В качестве примера логики бизнес-правила «лодка» может быть синтаксически допустимой, потому что она содержит только буквенно-цифровые символы, но недействительна, если вы ожидаете такие цвета, как «красный» или «синий».
   13. При построении командных строк ОС используйте строгие разрешающие списки, которые ограничивают набор символов на основе ожидаемого значения параметра в запросе. Это косвенно ограничит масштаб атаки, но этот метод менее важен, чем правильное кодирование вывода и экранирование.
   14. Обратите внимание, что правильное кодирование вывода, экранирование и цитирование являются наиболее эффективным решением для предотвращения внедрения команд ОС, хотя проверка ввода может обеспечить некоторую глубинную защиту. Это потому, что он эффективно ограничивает то, что будет отображаться на выходе. Проверка ввода не всегда предотвращает внедрение команд ОС, особенно если требуется поддерживать текстовые поля произвольной формы, которые могут содержать произвольные символы. Например, при вызове почтовой программы вам может потребоваться разрешить поле темы содержать опасные в противном случае вводы, такие как ";" и символы ">", которые необходимо экранировать или обрабатывать иным образом. В этом случае удаление символа может снизить риск внедрения команды ОС, но приведет к некорректному поведению, поскольку поле темы не будет записано так, как задумал пользователь. Это может показаться незначительным неудобством, но это может быть более важным, когда программа полагается на хорошо структурированные строки темы для передачи сообщений другим компонентам.
   15. Даже если вы допустили ошибку при проверке (например, забыли одно из 100 полей ввода), соответствующая кодировка все равно может защитить вас от атак на основе инъекций. Пока это не выполняется изолированно, проверка входных данных по-прежнему является полезным методом, поскольку она может значительно уменьшить поверхность атаки, позволить вам обнаруживать некоторые атаки и обеспечить другие преимущества безопасности, которые не учитываются при правильном кодировании.
3. **Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (CWE-79) (Скрин 100)** **(proof-of-concept Скрины 123 и 124) Критичность: 6.3 (CVSS v3)**
   1. Решение:
      1. Этап: Архитектура и дизайн
         1. Используйте проверенную библиотеку или платформу, которая не позволяет этому недостатку возникать или предоставляет конструкции, которые делают эту слабость легче избежать.
         2. Примеры библиотек и фреймворков, которые упрощают создание правильно закодированного вывода, включают библиотеку Microsoft Anti-XSS, модуль кодирования OWASP ESAPI и Apache Wicket.
      2. Этапы: реализация; Архитектура и дизайн
         1. Поймите контекст, в котором будут использоваться ваши данные, и ожидаемую кодировку. Это особенно важно при передаче данных между различными компонентами или при генерации выходных данных, которые могут содержать несколько кодировок одновременно, например, веб-страницы или составные почтовые сообщения. Изучите все ожидаемые протоколы связи и представления данных, чтобы определить необходимые стратегии кодирования.
         2. Для любых данных, которые будут выводиться на другую веб-страницу, особенно для любых данных, полученных от внешних входов, используйте соответствующую кодировку для всех не буквенно-цифровых символов.
         3. Обратитесь к шпаргалке по предотвращению XSS для получения дополнительных сведений о необходимых типах кодирования и экранирования.
      3. Этап: архитектура и дизайн
         1. Для любых проверок безопасности, которые выполняются на стороне клиента, убедитесь, что эти проверки дублируются на стороне сервера, чтобы избежать CWE-602. Злоумышленники могут обойти проверки на стороне клиента, изменив значения после того, как проверки были выполнены, или изменив клиента, чтобы полностью удалить проверки на стороне клиента. Затем эти измененные значения будут отправлены на сервер.
         2. Если возможно, используйте структурированные механизмы, которые автоматически обеспечивают разделение данных и кода. Эти механизмы могут обеспечивать соответствующее цитирование, кодирование и проверку автоматически, вместо того, чтобы полагаться на разработчика, предоставляющего эту возможность в каждой точке, где генерируются выходные данные.
      4. Этап: реализация
         1. Для каждой создаваемой веб-страницы используйте и укажите кодировку символов, такую как ISO-8859-1 или UTF-8. Если кодировка не указана, веб-браузер может выбрать другую кодировку, угадав, какая кодировка фактически используется веб-страницей. Это может привести к тому, что веб-браузер будет рассматривать определенные последовательности как особые, открывая клиента для тонких XSS-атак. Смотри CWE-116 для дополнительных мер, связанных с кодированием / экранированием.
         2. Чтобы предотвратить атаки XSS на файл cookie сеанса пользователя, установите для файла cookie сеанса значение HttpOnly. В браузерах, поддерживающих функцию HttpOnly (например, в более поздних версиях Internet Explorer и Firefox), этот атрибут может предотвратить доступ к cookie сеанса пользователя для вредоносных клиентских скриптов, использующих document.cookie. Это не полное решение, поскольку HttpOnly поддерживается не всеми браузерами. Что еще более важно, XMLHTTPRequest и другие мощные браузерные технологии предоставляют доступ для чтения к заголовкам HTTP, включая заголовок Set-Cookie, в котором установлен флаг HttpOnly.
         3. Предположим, что весь ввод злонамерен. Используйте стратегию проверки входных данных «принять заведомо исправные», т. е. Используйте разрешенный список допустимых входных данных, которые строго соответствуют спецификациям. Отклоняйте любой ввод, который не строго соответствует спецификациям, или преобразуйте его во что-то, что соответствует. Не полагайтесь исключительно на поиск вредоносных или искаженных входных данных (т. е. Не полагайтесь на список запрещенных). Однако списки запрета могут быть полезны для обнаружения потенциальных атак или определения того, какие входные данные настолько искажены, что их следует сразу отклонить.
         4. При выполнении проверки ввода учитывайте все потенциально важные свойства, включая длину, тип ввода, полный диапазон допустимых значений, отсутствующие или дополнительные вводы, синтаксис, согласованность между связанными полями и соответствие бизнес-правилам. В качестве примера логики бизнес-правила «лодка» может быть синтаксически допустимой, потому что она содержит только буквенно-цифровые символы, но недействительна, если вы ожидаете такие цвета, как «красный» или «синий».
         5. Убедитесь, что вы выполняете проверку ввода на четко определенных интерфейсах в приложении. Это поможет защитить приложение, даже если компонент будет повторно использован или перемещен в другое место.
4. **Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (CWE-22) (Скрин 101)** **(proof-of-concept Скрин 125) Критичность: 8.2 (CVSS v3)**
   1. Решение:
      1. Предположим, что весь ввод злонамерен. Используйте стратегию проверки входных данных «принять заведомо исправные», т. е. Используйте разрешенный список допустимых входных данных, которые строго соответствуют спецификациям. Отклоняйте любой ввод, который не строго соответствует спецификациям, или преобразуйте его во что-то, что соответствует. Не полагайтесь исключительно на поиск вредоносных или искаженных входных данных (т. е. Не полагайтесь на список запрещенных). Однако списки запрета могут быть полезны для обнаружения потенциальных атак или определения того, какие входные данные настолько искажены, что их следует сразу отклонить.
      2. При выполнении проверки ввода учитывайте все потенциально важные свойства, включая длину, тип ввода, полный диапазон допустимых значений, отсутствующие или дополнительные вводы, синтаксис, согласованность между связанными полями и соответствие бизнес-правилам. В качестве примера логики бизнес-правила «лодка» может быть синтаксически допустимой, потому что она содержит только буквенно-цифровые символы, но недействительна, если вы ожидаете такие цвета, как «красный» или «синий».
      3. Для имен файлов используйте строгие списки разрешений, которые ограничивают используемый набор символов. Если возможно, разрешите только один "." в имени файла, чтобы избежать слабых мест, и исключить разделители каталогов, такие как «/». Используйте разрешенный список допустимых расширений файлов.
      4. Предупреждение: если вы попытаетесь очистить свои данные, сделайте так, чтобы конечный результат не был в форме, которая может быть опасной. Механизм очистки может удалить такие символы, как "." а также ';' который может потребоваться для некоторых эксплойтов. Злоумышленник может попытаться обмануть механизм очистки, чтобы «очистить» данные до опасной формы. Предположим, злоумышленник вводит '.' внутри имени файла (например, "sensi.tiveFile"), а механизм очистки удаляет символ, в результате чего получается допустимое имя файла, "sensitiveFile". Если теперь предполагается, что входные данные безопасны, файл может быть скомпрометирован.
      5. Перед проверкой входные данные должны быть декодированы и канонизированы в текущее внутреннее представление приложения. Убедитесь, что ваше приложение не декодирует один и тот же ввод дважды. Такие ошибки можно использовать для обхода схем разрешенных списков путем введения опасных входных данных после их проверки.
      6. Используйте встроенную функцию канонизации пути (например, realpath () в C), которая создает каноническую версию имени пути, которая эффективно удаляет последовательности «..» и символические ссылки.
      7. Запустите свой код, используя самые низкие привилегии, необходимые для выполнения необходимых задач. Если возможно, создайте изолированные учетные записи с ограниченными правами, которые используются только для одной задачи. Таким образом, успешная атака не предоставит злоумышленнику немедленный доступ к остальному программному обеспечению или его среде. Например, приложениям баз данных редко требуется запускать от имени администратора базы данных, особенно в повседневных операциях.
      8. Когда набор допустимых объектов, таких как имена файлов или URL-адресов, ограничен или известен, создайте сопоставление из набора фиксированных входных значений (например, числовых идентификаторов) с фактическими именами файлов или URL-адресами и отклоните все остальные входные данные.
      9. Запустите свой код в «тюрьме» или подобной среде песочницы, которая устанавливает строгие границы между процессом и операционной системой. Это может эффективно ограничить, к каким файлам можно получить доступ в конкретном каталоге или какие команды могут выполняться вашим программным обеспечением.
      10. Примеры уровня ОС включают chroot jail Unix, AppArmor и SELinux. В общем, управляемый код может обеспечить некоторую защиту. Например, java.io.FilePermission в Java SecurityManager позволяет вам указывать ограничения на файловые операции.
      11. Это может быть невыполнимым решением, и оно только ограничивает влияние на операционную систему; остальная часть вашего приложения все еще может быть скомпрометирована.
5. **Protection Mechanism Failure (CWE-693) (Скрин и proof-of-concept 102, 109, 110) Критичность: 7.1 (CVSS v3)**
   1. Решение:
      1. Убедитесь, что ваш веб-сервер, сервер приложений, балансировщик нагрузки и т. д. настроены для установки заголовка Content-Security-Policy для достижения оптимальной поддержки браузера: «Content-Security-Policy» для Chrome 25+, Firefox 23+ и Safari 7. +, «X-Content-Security-Policy» для Firefox 4.0+ и Internet Explorer 10+ и «X-WebKit-CSP» для Chrome 14+ и Safari 6+.
      2. Убедитесь, что ваш веб-сервер, сервер приложений, балансировщик нагрузки и т. д. настроены для установки заголовка Permissions-Policy.
      3. Убедитесь, что приложение / веб-сервер правильно задает заголовок Content-Type и что он устанавливает заголовок X-Content-Type-Options равным «nosniff» для всех веб-страниц.
      4. Если возможно, убедитесь, что конечный пользователь использует современный веб-браузер, соответствующий стандартам, который вообще не выполняет сниффинг MIME или который может быть направлен веб-приложением / веб-сервером, чтобы не выполнять сниффинг MIME.
6. **Improper Restriction of Rendered UI Layers or Frames (CWE-1021) (Скрин 103 и proof-of-concept) Критичность: 8.1 (CVSS v3)**
   1. Решение:
      1. Современные веб-браузеры поддерживают Content-Security-Policy и заголовки HTTP X-Frame-Options.Убедитесь, что один из них установлен на всех веб-страницах, возвращаемых вашим сайтом/приложением. Если вы ожидаете, что страница будет обрамлена только страницами на вашем сервере (например, это часть FRAMESET), вам следует использовать SAMEORIGIN, в противном случае, если вы никогда не ожидаете, что страница будет обрамлена, вам следует использовать DENY.В качестве альтернативы рассмотрите возможность реализации директивы Content Security Policy «frame-ancestors».
7. **Cross-Site Request Forgery (CSRF) (CWE-352) (Скрин 104, Скрин 105) Критичность: 8.3 (CVSS v3)**
   1. Решение:
      1. Этап: Архитектура и дизайн
         1. Используйте проверенную библиотеку или платформу которая не позволяет этому недостатку возникать или предоставляет конструкции которые делают эту слабость легче избежать. Например, используйте пакеты анти-CSRF, такие как OWASP CSRFGuard.
      2. Этап: реализация
         1. Убедитесь, что в вашем приложении нет проблем с межсайтовыми сценариями, потому что большинство средств защиты CSRF можно обойти с помощью сценария, управляемого злоумышленником.
      3. Этап: архитектура и дизайн
         1. Создайте уникальный одноразовый номер для каждой формы, поместите одноразовый номер в форму и проверьте одноразовый номер после получения формы. Убедитесь, что одноразовый номер непредсказуем (CWE-330).
         2. Обратите внимание, что это можно обойти с помощью XSS.
         3. Определите особо опасные операции. Когда пользователь выполняет опасную операцию, отправьте отдельный запрос подтверждения, чтобы убедиться, что пользователь намеревался выполнить эту операцию.
         4. Обратите внимание, что это можно обойти с помощью XSS.
         5. Используйте элемент управления ESAPI Session Management.
         6. Этот элемент управления включает компонент для CSRF.
         7. Не используйте метод GET для любого запроса, который вызывает изменение состояния.
      4. Этап: реализация
         1. Проверьте заголовок HTTP Referer, чтобы узнать, исходит ли запрос от ожидаемой страницы. Это может нарушить законную функциональность, поскольку пользователи или прокси-серверы могли отключить отправку Referer по соображениям конфиденциальности.
8. **Inclusion of Functionality from Untrusted Control Sphere (CWE-829) (Скрин и proof-of-concept 106) Критичность: 9 (CVSS v3)**
   1. Решение:
      1. Обновите до последней версии jquery.
9. **Exposed Dangerous Method or Function (CWE-749) (Скрин и proof-of-concept 108) Критичность: 8.9 (CVSS v3)**
   1. Решение:
      1. <https://angular.io/guide/security>
10. **Exposure of Sensitive Information to an Unauthorized Actor (CWE-200) (Скрин и proof-of-concept 111, 112) Критичность: 8.2 (CVSS v3)**
    1. Решение:
       1. Убедитесь, что ваш веб-сервер, сервер приложений, балансировщик нагрузки и т. д. настроены на подавление заголовка «Сервер» или предоставление общих сведений.
       2. Просмотрите исходный код этой страницы.
       3. Реализуйте настраиваемые страницы ошибок.
       4. Подумайте о реализации механизма для предоставления уникальной ссылки / идентификатора ошибки клиенту (браузеру), регистрируя при этом детали на стороне сервера и не раскрывая их пользователю.