Bezpieczeństwo Aplikacji Webowych - projekt

Temat: OWASP Nodegoat

Autorzy: Nikodem Moroz (249401), Tomasz Garus (249073)

Prowadzący: mgr inż. Przemysław Świercz

Spis treści

[Wstęp 3](#_Toc169293763)

[Injection (Wstrzyknięcie) 3](#_Toc169293764)

[Broken Authentication and Session Management 8](#_Toc169293765)

[Skan Nessus 10](#_Toc169293766)

[Cross-Site Scripting (XSS) 12](#_Toc169293767)

[Insecure Direct Object References (Niezabezpieczone bezpośrednie odwołania do obiektów) 15](#_Toc169293768)

[Sensitive Data Exposure 19](#_Toc169293769)

[Missing Function Level Access Control (Brak kontroli dostępu na poziomie funkcji) 20](#_Toc169293770)

[Cross-Site Request Forgery (CSRF) 20](#_Toc169293771)

[Używanie komponentów ze znanymi lukami w zabezpieczeniach 22](#_Toc169293772)

[Unvalidated Redirects and Forwards (Nieważne przekierowania i przekazywanie dalej) 23](#_Toc169293773)

[Server-Side Request Forgery (SSRF) 25](#_Toc169293774)

[Regular Expressions DoS 29](#_Toc169293775)

# Wstęp

4.OWASPNodeGoat(limit:2grupy,grupa3-osobowa)  
  
Celem zadania jest znalezienie oraz exploitacja jak największej liczby podatności. W opisie repozytorium można znaleźć instrukcje dotyczące instalacji oraz podstawowych podatności. Proszę przygotować dokładny opis każdej znalezionej podatności wraz z techniką jej exploitacji. W miarę możliwości proszę przygotować skrypt lub grupę skryptów do automatycznej ekspoitacji znalezionych podatności.

Aplikacja NodeGoat została uruchomiona na Dockerze. Repozytorium https://github.com/OWASP/NodeGoat zawiera plik Dockerfile i docker-compose.yml niezbędne do skonfigurowania aplikacji i instancji db, a następnie połączenia ich ze sobą.

Kolejne kroki wykonane w celu uruchomienia aplikacji wyglądały następująco:

* Instalacja docker i docker compose
* Sklonowanie repozytorium github
  + git clone https://github.com/OWASP/NodeGoat.git
* Przejście do katalogu NodeGoat
  + cd NodeGoat
* Zbudowanie obrazu
  + docker-compose build
* Uruchomienie aplikacji pod adresem <http://localhost:4000/>
  + docker-compose up

# Injection (Wstrzyknięcie)

Wstrzyknięcia to poważne luki bezpieczeństwa, które występują, gdy niezaufane dane są wysyłane do interpretera jako część polecenia lub zapytania. Atakujący mogą wstrzyknąć złośliwe dane, które zmuszą interpreter do wykonania niezamierzonych poleceń lub uzyskania dostępu do danych bez odpowiedniej autoryzacji.

Gdy funkcje eval(), setTimeout(), setInterval(), Function() są używane do przetwarzania danych wejściowych podanych przez użytkownika, może to zostać wykorzystane przez atakującego do wstrzyknięcia i wykonania złośliwego kodu JavaScript na serwerze.

Aplikacje internetowe wykorzystujące funkcję JavaScript eval() do analizowania przychodzących danych bez żadnego rodzaju walidacji danych wejściowych są podatne na ten atak. Atakujący może wstrzyknąć dowolny kod JavaScript, który zostanie wykonany na serwerze. Podobnie funkcje setTimeout() i setInterval() mogą przyjmować kod w formacie ciągu znaków jako pierwszy argument, powodując takie same problemy jak eval().

Luka ta może być bardzo krytyczna i szkodliwa, umożliwiając atakującemu wysyłanie różnego rodzaju poleceń. Nodegoat korzysta właśnie z funkcji eval(). Pozwala to **przykładowo, za pomocą polecenia „**process.kill(process.pid)**” na przeprowadzenie ataku Denial of Service.**

W zakładce „Contributions” użytkownik może wprowadzić Nowy procent składki płacowej

za okres rozliczeniowy. Gdy zamiast liczby wprowadzimy tam komendę „process.kill(process.pid)” jesteśmy w stanie zastopować działanie aplikacji.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, numer

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

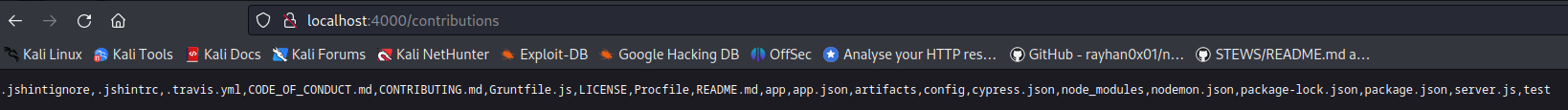
Opis wygenerowany automatycznie

Innym potencjalnym celem atakującego może być odczytanie zawartości plików z serwera. Przykładowo poniższe dwa polecenia wyświetlają odpowiednio zawartość bieżącego katalogu i katalogu nadrzędnego:

* res.end(require('fs').readdirSync('.').toString())
* res.end(require('fs').readdirSync('..').toString())

Gdy wprowadzimy te komendy w pole pokazane w przykładzie z atakiem DOS otrzymamy następujące rezultaty:

* Wyświetlenie katalogu bieżącego



* Wyświetlenie katalogu nadrzędnego:  
  Obraz zawierający tekst, zrzut ekranu, Czcionka, Oprogramowanie multimedialne

  Opis wygenerowany automatycznie

Wyświetlanie zawartości pliku „/etc/passwd” za pomocą komendy:

* res.end(require('fs').readFileSync('../../../../../../../etc/passwd').toString())

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

Kolejnym przykładem wstrzyknięcia są wstrzyknięcia SQL i NoSQL, które umożliwiają atakującemu wstrzyknięcie kodu do zapytania, które zostanie wykonane przez bazę danych. Luki te pojawiają się, gdy twórcy oprogramowania tworzą dynamiczne zapytania do bazy danych, które zawierają dane wejściowe dostarczone przez użytkownika.

Strona „Allocations” aplikacji Nodegoat jest podatna na atak NoSQL Injection. W pole filtrowania aktywów na podstawie wyników akcji możemy wprowadzić komendę „1'; return 1 == '1”.

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie  
Po wprowadzeniu tej komendy możemy wyświetlić alokacje aktywów wszystkich użytkowników.

Obraz zawierający tekst, zrzut ekranu, numer, oprogramowanie

Opis wygenerowany automatycznie

**Jak zapobiec atakom wstrzykiwania JavaScript (SSJS) po stronie serwera:**

1. Weryfikacja danych wejściowych:
   * Weryfikować i walidować wszystkie dane wejściowe użytkownika po stronie serwera przed ich przetworzeniem. Sprawdzać typy danych, zakresy wartości oraz formaty, aby upewnić się, że spełniają one oczekiwane kryteria.
2. Unikanie funkcji eval():
   * Nie używać funkcji eval() do analizowania danych wejściowych użytkownika, ponieważ eval() wykonuje kod, który może być potencjalnie złośliwy. Unikać używania innych funkcji wykonujących kod, takich jak setTimeout(), setInterval() oraz Function().
3. Bezpieczne parsowanie danych JSON:
   * Zamiast eval() do parsowania danych JSON, używać bezpieczniejszej alternatywy, takiej jak JSON.parse(). Do konwersji typów używać odpowiednich metod, takich jak parseInt(), parseFloat(), itp.
4. Tryb ścisły ("use strict"):
   * Dołączać "use strict" na początku każdej funkcji lub skryptu, aby włączyć tryb ścisły JavaScript. Tryb ścisły zmniejsza liczbę błędów i pomaga unikać niebezpiecznych konstrukcji.

**Jak zapobiec atakom SQL/NoSQL Injection:**

1. Przygotowane instrukcje (prepared statements):
   * Używać przygotowanych instrukcji (prepared statements) do wykonywania zapytań SQL. Unikać dynamicznego budowania zapytań SQL przy użyciu konkatenacji ciągów znaków.
2. Walidacja danych wejściowych:
   * Walidować wszystkie dane wejściowe, aby wykryć złośliwe wartości. W przypadku baz danych NoSQL, również weryfikować typy danych wejściowych, aby upewnić się, że są zgodne z oczekiwanymi typami.
3. Najmniejsze uprawnienia:
   * Minimalizować uprawnienia kont aplikacji, aby ograniczyć potencjalne szkody związane z udanym atakiem wstrzyknięcia. Nie przypisywać praw dostępu typu DBA lub administratora do kont aplikacji. Minimalizować także uprawnienia konta systemu operacyjnego, pod którym działa proces bazy danych.

**Przykład kodu źródłowego w aplikacji**

W pliku routes/contributions.js, funkcja handleContributionsUpdate() używa eval() do konwersji podanych przez użytkownika kwot składek na liczbę całkowitą.

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

To sprawia, że aplikacja jest podatna na atak. Można to naprawić po prostu używając funkcji parseInt() zamiast tego.

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

# Broken Authentication and Session Management

W tym ataku atakujący (który może być anonimowym atakującym z zewnątrz, użytkownikiem z własnym kontem, który może próbować wykraść dane z kont lub osobą z wewnątrz, która chce ukryć swoje działania) wykorzystuje nieszczelności lub błędy w funkcjach uwierzytelniania lub zarządzania sesjami, aby podszyć się pod innych użytkowników. Funkcje aplikacji związane z uwierzytelnianiem i zarządzaniem sesjami często nie są poprawnie zaimplementowane, umożliwiając atakującym złamanie haseł, kluczy lub tokenów sesji lub wykorzystanie innych błędów w implementacji w celu przejęcia tożsamości innych użytkowników.

Programiści często tworzą niestandardowe schematy uwierzytelniania i zarządzania sesjami, ale ich poprawne tworzenie jest trudne. W rezultacie te niestandardowe schematy często mają wady w obszarach takich jak wylogowanie, zarządzanie hasłami, limit czasu, zapamiętaj mnie, tajne pytanie, aktualizacja konta itp. Znalezienie takich błędów może być czasami trudne, ponieważ każda implementacja jest unikalna.

W aplikacji NodeGoat można zauważyć między innymi brak polityki haseł przy tworzeniu nowych kont:

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

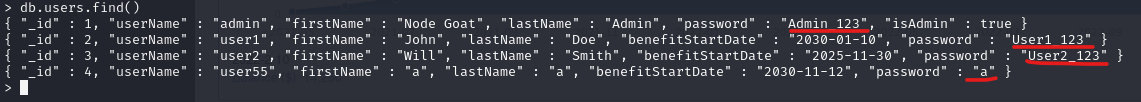
Opis wygenerowany automatycznie

Brak zabezpieczenia haseł funkcjami skrótów i/lub solami przy tworzeniu konta:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Hasła przechowywane są w plaintext w bazie danych:



# Skan Nessus

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

**Jak zapobiec problemom bezpieczeństwa związanym z hasłami i sesjami:**

Długość hasła:

* Ustawić minimalną długość hasła na co najmniej osiem (8) znaków. Połączyć tę długość ze złożonością, aby hasło było trudne do odgadnięcia lub złamania.

Złożoność hasła:

* Użyć kombinacji znaków alfanumerycznych w hasłach. Znaki alfanumeryczne składają się z liter, cyfr, znaków interpunkcyjnych, symboli matematycznych i innych konwencjonalnych symboli.

Wyliczanie nazwy użytkownika/hasła:

* Nie wskazywać, która część danych uwierzytelniających była nieprawidłowa. Używać komunikatu "Nieprawidłowa nazwa użytkownika i/lub hasło" zarówno dla błędów związanych z nazwą użytkownika, jak i hasłem. Odpowiedzi na błędy muszą być identyczne zarówno w wyświetlaniu, jak i kodzie źródłowym.

Dodatkowe środki:

1. Ochrona przed atakiem brute-force:
   * Wyłączać konto po ustalonej liczbie nieprawidłowych prób logowania (np. pięć prób). Konto powinno być wyłączone na okres wystarczający do zniechęcenia do zgadywania poświadczeń metodą brute force, ale nie na tyle długo, aby umożliwić przeprowadzenie ataku typu "odmowa usługi".
2. Tymczasowe hasła:
   * Wysyłać hasła tymczasowe wyłącznie za pośrednictwem szyfrowanego połączenia lub jako zaszyfrowane dane (np. w zaszyfrowanej wiadomości e-mail). Wymuszać zmianę haseł tymczasowych przy następnym użyciu. Hasła tymczasowe i linki powinny mieć krótki czas wygaśnięcia.

**Środki bezpieczeństwa związane z zarządzaniem sesjami:**

1. Ochrona danych uwierzytelniających:
   * Chronić dane uwierzytelniające użytkownika przy użyciu haszowania lub szyfrowania.
2. Ukrywanie identyfikatorów sesji:
   * Nie ujawniać identyfikatorów sesji w adresie URL (np. unikać przepisywania adresów URL).
3. Zarządzanie czasem trwania sesji:
   * Ustawić limit czasu trwania sesji. Unieważniać sesje użytkownika lub tokeny uwierzytelniające podczas wylogowywania.
4. Odtwarzanie identyfikatorów sesji:
   * Odtwarzać identyfikatory sesji po pomyślnym zalogowaniu.
5. Szyfrowanie danych w tranzycie:
   * Nie przesyłać haseł, identyfikatorów sesji ani innych danych uwierzytelniających przez niezaszyfrowane połączenia.

**Przykład kodu źródłowego w aplikacji**

W aplikacji NodeGoat hasło jest przechowywane w bazie danych w postaci zwykłego tekstu. Oto powiązany kod w metodzie data/user-dao.js addUser():

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Aby to zabezpieczyć, należy obsługiwać przechowywanie haseł w bezpieczniejszy sposób, stosując szyfrowanie jednokierunkowe przy użyciu hashowania:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Za generowanie skrótu hasła są odpowiedzialne funkcje:

var salt = bcrypt.genSaltSync();

var passwordHash = bcrypt.hashSync(password, salt);

Hash hasła nie można odszyfrować, dlatego jest bezpieczniejszy. Aby porównać hasło podczas logowania użytkownika, hasło wprowadzone przez użytkownika jest konwertowane na hash i porównywane z hashem w pamięci.

Moża to wykonać w następujący sposób:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

# Cross-Site Scripting (XSS)

Błędy Cross-Site Scripting (XSS) występują, gdy aplikacja internetowa pobiera niezaufane dane i wysyła je do przeglądarki internetowej bez odpowiedniej walidacji lub ucieczki (escaping). Tego rodzaju luki bezpieczeństwa pozwalają atakującym na wstrzyknięcie i wykonanie złośliwych skryptów w przeglądarce ofiary. Skrypty te mogą uzyskać dostęp do plików cookie, tokenów sesji lub innych poufnych informacji przechowywanych przez przeglądarkę, a także przekierować użytkownika na złośliwe strony.

Istnieją dwa rodzaje ataków XSS:

* Reflected XSS (XSS odbite) - ma miejsce, gdy złośliwe dane są zwracane przez serwer w natychmiastowej odpowiedzi na żądanie HTTP od ofiary. Atakujący może skonstruować specjalny link zawierający złośliwy skrypt, który zostanie wykonany, gdy ofiara kliknie ten link.
* Stored XSS (XSS przechowywane) - ma miejsce, gdy złośliwe dane są przechowywane na serwerze i później osadzane na stronie HTML dostarczanej ofierze. Ten rodzaj ataku jest szczególnie groźny, ponieważ złośliwe dane mogą być wielokrotnie wyświetlane wielu użytkownikom.

Każdy z odbitych i przechowywanych XSS może wystąpić na serwerze lub na kliencie (co jest również znane jako XSS oparty na DOM), w zależności od tego, kiedy złośliwe dane zostaną wstrzyknięte do znaczników HTML.

Aplikacja Nodegoat jest podatna na atak XSS w formularzu profilu. Podczas przesyłania formularza, wartości pól Imię oraz Nazwisko są przesyłane do serwera, bez żadnej walidacji i następnie są zapisywane w bazie danych. Wartości te są następnie wysyłane z powrotem do przeglądarki bez odpowiedniej funkcji escapingu i są wyświetlane.

Aby pokazać działanie ataku XSS, zalogowano się na konto użytkownika user1.

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie

Następnie jako jego nazwisko wprowadzono następujący skrypt „<script>alert(document.cookie)</script>”. Za jego pomocą można wyświetlić wartości ciasteczek.

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

Następnie zalogowano się na profil administratora.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Po zalogowaniu się, wykonywany jest wprowadzony przez nas skrypt i wyświetla się zawartość ciasteczek.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

**Jak zapobiec atakom XSS:**

Walidacja i sanityzacja danych wejściowych:

* Walidacja danych wejściowych: Stosować walidację białej listy, aby upewnić się, że tylko oczekiwane dane są akceptowane.
* Sanityzacja danych: Oczyszczać dane wejściowe, aby usunąć lub zneutralizować złośliwe treści.

Kodowanie danych wyjściowych:

* Kodowanie danych wyjściowych: Kodować dane wyjściowe zgodnie z kontekstem, w jakim będą używane (HTML, CSS, JavaScript, itd.), aby ograniczyć ryzyko XSS.

Flaga plików cookie HTTPOnly:

* Ustawienie flagi HTTPOnly: Ustawiać flagę HTTPOnly na plikach cookie sesji i niestandardowych plikach cookie, aby uniemożliwić dostęp do nich przez JavaScript.

Wdrożenie polityki bezpieczeństwa treści (CSP):

* Polityka bezpieczeństwa treści (CSP): Używać nagłówka HTTP Content-Security-Policy, aby tworzyć białe listy zasobów po stronie klienta.

# Insecure Direct Object References (Niezabezpieczone bezpośrednie odwołania do obiektów)

Bezpośrednie odwołanie do obiektu występuje, gdy programista ujawnia odwołanie do wewnętrznego obiektu implementacji, takiego jak plik, katalog lub klucz bazy danych. Bez kontroli dostępu lub innej ochrony atakujący mogą manipulować tymi odniesieniami, aby uzyskać dostęp do nieautoryzowanych danych.

Jeśli aplikacje używają rzeczywistej nazwy lub klucza obiektu podczas generowania stron internetowych i nie sprawdzają, czy użytkownik jest autoryzowany do obiektu docelowego, może to skutkować niezabezpieczonym bezpośrednim odwołaniem do obiektu. Atakujący może wykorzystać takie błędy, manipulując wartościami parametrów. O ile odniesienia do obiektów nie są nieprzewidywalne, atakujący może łatwo uzyskać dostęp do wszystkich dostępnych danych tego typu.

W aplikacji Nodegoat w zakładce „Allocations” userid jest widoczny jako części adresu url. Atakujący może manipulować wartością id i uzyskać dostęp do informacji o alokacjach innych użytkowników. Przykładowo dla użytkownika admin userid wynosi 1.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznie

Po zmianie user id na 2 <http://localhost:4000/allocations/2> możemy wyświetlić dane innego użytkownika.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznie

**Jak zapobiec problemom związanych z bezpośrednim odwołaniem do obiektów:**

Sprawdzanie dostępu:

* Kontrola dostępu: Każde użycie bezpośredniego odwołania do obiektu z niezaufanego źródła musi obejmować sprawdzenie, czy użytkownik jest uprawniony do żądanego obiektu.

Używanie pośrednich odwołań do obiektów:

* Pośrednie odwołania do obiektów: Zamiast ujawniać rzeczywiste klucze bazy danych jako część linków dostępu, stosować tymczasowe pośrednie odniesienia dla użytkownika lub sesji. Na przykład, zamiast używać klucza bazy danych zasobu, użyć liczb sekwencyjnych lub unikalnych liczb losowych do wskazania wybranego zasobu. Aplikacja musi zmapować pośrednie odniesienie na rzeczywisty klucz bazy danych na serwerze.

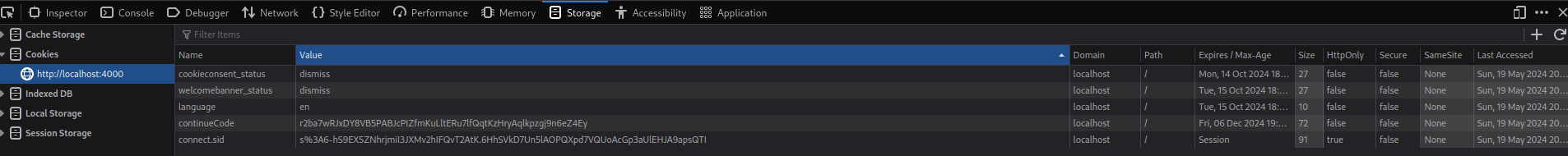
Testowanie i analiza kodu:

* Manipulacja wartościami parametrów: Testerzy mogą manipulować wartościami parametrów, aby wykryć błędy autoryzacji.
* Analiza kodu: Regularna analiza kodu może szybko wykazać, czy autoryzacja jest prawidłowo weryfikowana.

Security Misconfiguration (Błędna Konfiguracja Zabezpieczeń)

Luka ta umożliwia atakującemu dostęp do domyślnych kont, nieużywanych stron, niezałatanych błędów, niezabezpieczonych plików i katalogów itp. w celu uzyskania nieautoryzowanego dostępu do systemu lub wiedzy o nim. Błędna konfiguracja zabezpieczeń może wystąpić na dowolnym poziomie stosu aplikacji, w tym na platformie, serwerze WWW, serwerze aplikacji, bazie danych, frameworku i niestandardowym kodzie.

W aplikacji Nodegoat można zauważyć domyślną nazwę pliku cookie – „connect.sid”.



Domyślna nazwa pliku cookie sesji dla sesji Express.js to connect.sid. Można ją zmienić, ustawiając atrybut key podczas tworzenia sesji. Pozwoli to uniknąć ujawnienia domyślnej konfiguracji, co utrudnia atakującym przeprowadzenie ataków.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Domyślny nagłówek HTTP x-powered-by może ujawnić atakującemu szczegóły implementacji. Można go usunąć, umieszczając ten kod w pliku server.js.

**Jak zapobiec problemom bezpieczeństwa w Node.js i Express:**

Aktualizacja i monitorowanie:

* Używanie najnowszej stabilnej wersji Node.js i Express: Zawsze używać najnowszej stabilnej wersji Node.js i Express, aby korzystać z najnowszych poprawek zabezpieczeń.
* Monitorowanie luk w zabezpieczeniach: Regularnie sprawdzać opublikowane luki w zabezpieczeniach dla Node.js i Express.

Zarządzanie uprawnieniami:

* Nie uruchamianie aplikacji z uprawnieniami roota: Uruchomić serwer jako root tylko w celu uzyskania dostępu do uprzywilejowanych portów (np. 80), a następnie obniżyć uprawnienia do użytkownika nieuprzywilejowanego po ustanowieniu nasłuchiwania na porcie.
* Używanie serwera proxy lub mapowania portów: Alternatywnie można użyć oddzielnego serwera proxy lub mapowania portów.

Bezpieczne nagłówki i pliki cookie:

* Przeglądanie domyślnych nagłówków HTTP: Usunąć lub zmodyfikować domyślne nagłówki, które mogą ujawniać wewnętrzne szczegóły implementacji.
* Używanie ogólnych nazw plików cookie sesji: Stosować ogólne nazwy plików cookie, aby nie ujawniać szczegółów implementacji sesji.

Ograniczenie rozmiaru żądań HTTP:

* Ograniczenie rozmiaru ciała żądania HTTP: Ustawiać rozsądne limity rozmiaru dla każdego typu zawartości (urlencoded, JSON, multipart) zamiast używać zagregowanego limitu middleware.
* Usuwanie zbędnego oprogramowania pośredniczącego: Dołączać tylko te middleware, które są absolutnie niezbędne dla aplikacji. Na przykład, jeśli aplikacja nie obsługuje przesyłania plików, nie dołączać middleware multipart.

Zarządzanie plikami tymczasowymi:

* Czyszczenie plików tymczasowych: Opracować strategię czyszczenia plików tymczasowych generowanych przez wieloczęściowe middleware, aby zapobiec zapełnieniu dysku.

Vetowanie i zarządzanie pakietami npm:

* Vetowanie pakietów npm: Przeglądać i zatwierdzać pakiety npm używane przez aplikację, aby upewnić się, że nie zawierają znanych luk.
* Blokowanie wersji pakietów npm: Używać npm shrinkwrap lub package-lock.json, aby zablokować wersje pakietów npm i mieć pełną kontrolę nad aktualizacjami.

Ustawianie nagłówków HTTP specyficznych dla zabezpieczeń:

* Ustawianie bezpiecznych nagłówków HTTP: Skonfigurować nagłówki HTTP, aby zwiększyć bezpieczeństwo, np. Content-Security-Policy, X-Content-Type-Options, X-Frame-Options, Strict-Transport-Security.

# Sensitive Data Exposure

Luka ta umożliwia atakującemu uzyskanie dostępu do poufnych danych, takich jak karty kredytowe, identyfikatory podatkowe, dane uwierzytelniające itp. w celu przeprowadzenia oszustwa związanego z kartami kredytowymi, kradzieży tożsamości lub innych przestępstw. Utrata takich danych może mieć poważne konsekwencje biznesowe i zaszkodzić reputacji firmy. Wrażliwe dane zasługują na dodatkową ochronę, taką jak szyfrowanie w spoczynku lub podczas przesyłania, a także specjalne środki ostrożności podczas wymiany z przeglądarką.

Mechanika ataku

Jeśli witryna nie używa SSL/TLS dla wszystkich uwierzytelnionych stron, atakujący może monitorować ruch sieciowy (np. w otwartej sieci bezprzewodowej) i wykraść plik cookie sesji użytkownika. Atakujący może następnie odtworzyć ten plik cookie i przejąć sesję użytkownika, uzyskując dostęp do jego prywatnych danych.

Jeśli atakujący uzyska dostęp do bazy danych aplikacji, może wykraść poufne informacje, które nie zostały zaszyfrowane lub zaszyfrowane słabym algorytmem szyfrowania

**Jak temu zapobiec?**

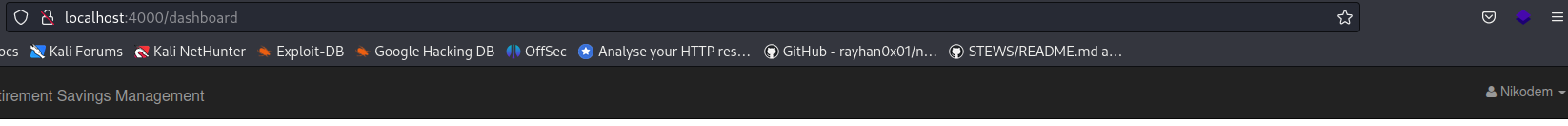
* Implementacja HTTPS: Używanie protokołu HTTPS zamiast HTTP zapewnia szyfrowanie komunikacji między klientem a serwerem. Dzięki temu dane przesyłane między użytkownikiem a aplikacją są zabezpieczone przed podsłuchiwaniem i modyfikacją przez nieupoważnione osoby.
* Szyfrowanie danych
  + Szyfrowanie danych w spoczynku i podczas przesyłania:
    - Używanie silnych algorytmów szyfrowania: Zapewnienie, że wszystkie poufne dane są przechowywane i przesyłane w formie zaszyfrowanej.
    - Zarządzanie kluczami: Używanie silnych, standardowych algorytmów i odpowiednie zarządzanie kluczami szyfrującymi.
* Zarządzanie danymi
  + Minimalizacja przechowywania wrażliwych danych:
    - Usuwanie danych po ich użyciu: W miarę możliwości nie przechowywać wrażliwych danych dłużej niż jest to konieczne.
* Dodatkowe środki bezpieczeństwa
  + Wyłączenie autouzupełniania formularzy: Wyłączenie funkcji autouzupełniania w formularzach, szczególnie tych zawierających dane wrażliwe, aby zapobiec nieautoryzowanemu dostępowi do danych.
  + Wyłączenie buforowania stron zawierających dane wrażliwe: Zastosowanie odpowiednich nagłówków HTTP (np. Cache-Control: no-store, no-cache, must-revalidate) aby wyłączyć buforowanie stron zawierających dane wrażliwe w przeglądarkach i serwerach proxy.

# Missing Function Level Access Control (Brak kontroli dostępu na poziomie funkcji)

Brak kontroli dostępu na poziomie funkcji to poważna luka bezpieczeństwa, która może prowadzić do nieautoryzowanego dostępu do funkcji aplikacji internetowej. Wiele aplikacji internetowych weryfikuje prawa dostępu na poziomie funkcji jedynie na poziomie interfejsu użytkownika, co oznacza, że wyświetlają lub ukrywają pewne elementy interfejsu w zależności od uprawnień użytkownika. Jednak równie ważne jest, aby te same kontrole dostępu były przeprowadzane po stronie serwera, gdzie rzeczywiście dochodzi do przetwarzania żądań. W przypadku braku kontroli dostępu na poziomie funkcji, atakujący może fałszować żądania HTTP, aby uzyskać dostęp do funkcji, do których nie jest uprawniony. Dzieje się tak, ponieważ serwer nie weryfikuje odpowiednio, czy użytkownik ma prawo do wykonania danej operacji.

W aplikacji Nodegoat luka ta występuje w module świadczeń (Benefits), który umożliwia zmianę daty rozpoczęcia świadczenia dla pracowników. Link do modułu świadczeń jest widoczny tylko dla użytkownika admin. Jednak serwer nie weryfikuje, czy użytkownik faktycznie ma uprawnienia do korzystania z tego modułu, co umożliwia atakującemu uzyskanie dostępu do tego modułu poprzez bezpośrednie odwołanie się do jego URL.

Na obrazku widocznym niżej widać, że użytkownik Nikodem, może wejść na panel benefitów i zmienić datę świadczeń dowolnemu użytkownikowi.



Obraz zawierający zrzut ekranu, oprogramowanie, tekst, Ikona komputerowa

Opis wygenerowany automatycznie

**Jak temu zapobiec?**

Większość aplikacji internetowych nie wyświetla linków i przycisków do nieautoryzowanych funkcji, ale ta "kontrola dostępu w warstwie prezentacji" w rzeczywistości nie zapewnia ochrony. Należy również wykorzystać system do zarządzania uprawnieniami opartego na rolach (Role-Based Access Control), aby nadawać użytkownikom odpowiednie role i przydzielać im dostęp do odpowiednich funkcji aplikacji w logice biznesowej aplikacji.

# Cross-Site Request Forgery (CSRF)

Atak Cross-Site Request Forgery (CSRF) zmusza zalogowaną przeglądarkę ofiary do wysłania fałszywego żądania HTTP do podatnej aplikacji internetowej. Żądanie to zawiera plik cookie sesji ofiary oraz inne automatycznie dołączane informacje uwierzytelniające. Dzięki temu atakujący może zmusić przeglądarkę ofiary do wykonania działań, które aplikacja przetwarza jako legalne żądania od autoryzowanego użytkownika. Przeglądarki internetowe automatycznie wysyłają dane uwierzytelniające, takie jak pliki cookie sesji, wraz z każdym żądaniem HTTP wysyłanym do serwera, z którego pochodzą te pliki cookie. Atakujący może stworzyć złośliwą stronę internetową, która generuje takie żądania, wyglądające na legalne żądania wysyłane przez przeglądarkę ofiary.

Aby wykonać ten atak na aplikację Nodegoat skorzystano z repozytorium dostępnym pod adresem <https://github.com/ckarande/nodegoat-csrf-attack>.

Za jej pomocą stworzoną złośliwą stronę hostowaną lokalnie na porcie 3000.

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Za pomocą tej strony można zmienić wartości w polach „Bank Account” oraz „Bank Routing” na dowolne wskazane przez atakującego. Jedynym warunkiem jest kliknięcie przez użytkownika przycisku „WIN”

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

**Jak temu zapobiec?**

Należy przede wszystkim:

* Używać oprogramowania pośredniczącego CSRF: W aplikacjach opartych na Express.js należy używać oprogramowania pośredniczącego CSRF, które automatycznie generuje i weryfikuje token CSRF dla żądań modyfikujących stan (PUT, POST, DELETE).
* Weryfikować CSRF w każdym żądaniu modyfikującym stan: Upewnić się, że każde żądanie, które zmienia stan serwera (np. dodawanie, aktualizacja, usuwanie danych), zawiera poprawnie przekazany token CSRF.
* Unikać niebezpiecznych metod HTTP: Metody takie jak GET nie powinny zmieniać stanu serwera, aby zmniejszyć ryzyko ataków CSRF. Zaleca się używanie metod POST, PUT, DELETE do modyfikowania stanu.
* Ograniczać czas życia tokenu CSRF: Tokeny CSRF powinny być ważne tylko przez krótki czas (np. kilka minut), aby zwiększyć bezpieczeństwo.
* Zabezpieczać ciasteczka: Ciasteczka zawierające token CSRF powinny mieć ustawioną flagę HTTPOnly, aby nie były dostępne z poziomu JavaScriptu w przeglądarce.
* Regularnie aktualizować zabezpieczenia: Regularnie aktualizować oprogramowanie pośredniczącego CSRF oraz inne zależności aplikacji, aby korzystać z najnowszych zabezpieczeń i poprawek.

# Używanie komponentów ze znanymi lukami w zabezpieczeniach

Komponenty, takie jak biblioteki, frameworki i inne moduły oprogramowania, prawie zawsze działają z pełnymi uprawnieniami. Jeśli podatny komponent zostanie wykorzystany, taki atak może ułatwić poważną utratę danych lub przejęcie serwera. Aplikacje korzystające z komponentów ze znanymi lukami w zabezpieczeniach mogą osłabić obronę aplikacji i umożliwić szereg możliwych ataków i skutków.

Korzystanie z niezabezpieczonych pakietów npm może prowadzić do takiej podatności. Niektóre projekty pomagają obecnie testować i ostrzegać o niezabezpieczonych zależnościach:

* npm audit to skaner podatności wbudowany w npm CLI (wersja 6 lub nowsza)
* Aktualizacje zabezpieczeń Dependabot mogą automatycznie tworzyć żądania ściągnięcia GitHub w celu zaktualizowania podatnych zależności.
* Snyk.io to narzędzie Node.js CLI i platforma do skanowania i wykrywania podatnych pakietów.

Powyższe narzędzia korzystają z list podatności, które można również przeglądać bezpośrednio lub przeszukiwać tutaj:

* NPM Security Advisories
* GitHub Advisory Database
* Snyk Vulnerability DB

Istnieje kilka innych narzędzi, które mogą wykrywać i aktualizować nieaktualne pakiety:

* npm outdated i yarn outdated to narzędzia wiersza poleceń, które pokazują potencjalnie nieaktualne zależności.
* Aktualizacje wersji Dependabot mogą automatycznie tworzyć żądania ściągnięcia GitHub w celu aktualizacji nieaktualnych zależności.
* David DM daje ci przegląd zależności twojego projektu, używanej wersji i najnowszej dostępnej, dzięki czemu możesz szybko zobaczyć, co dryfuje.
* npm-check Sprawdza nieaktualne, nieprawidłowe i nieużywane zależności.

Mechanika ataku

Pakiety npm są istotną częścią naszej aplikacji node. Pakiety te mogą przypadkowo lub złośliwie zawierać niezabezpieczony kod. Poprzez niezabezpieczone pakiety atakujący może:

* Tworzyć i uruchamiać skrypty na różnych etapach instalacji lub użytkowania pakietu.
* odczytywać, zapisywać, aktualizować i usuwać pliki w systemie
* Zapisywać i wykonywać pliki binarne
* Zbierać poufne dane i wysyłać je zdalnie

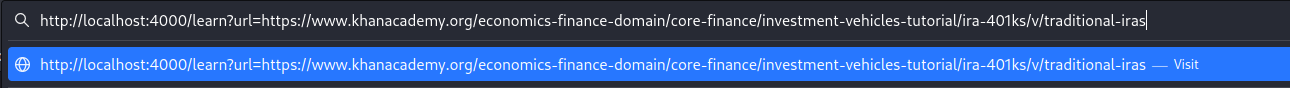
Oto kilka środków, które można podjąć, aby chronić się przed złośliwymi pakietami npm:

1. Nie uruchamiać aplikacji z uprawnieniami roota: Aby zminimalizować ryzyko dostępu do systemowych zasobów przez potencjalnie złośliwe pakiety, aplikacje nie powinny być uruchamiane z uprawnieniami roota.
2. Preferować pakiety z statyczną analizą kodu: Wybieranie pakietów, które zawierają statyczną analizę kodu (np. JSHint, JSLint), pomaga w wykrywaniu potencjalnych zagrożeń bezpieczeństwa i problemów związanych z jakością kodu.
3. Preferować pakiety z kompleksowymi testami: Korzystanie z pakietów, które posiadają solidne testy jednostkowe i przeglądowe, pomaga w zapewnieniu mniejszej podatności na błędy i ataki.
4. Przejrzeć kod pod kątem dostępu do plików i baz danych: Przed wdrożeniem, należy dokładnie przeglądać kod źródłowy pakietów, szczególnie tych, które mają dostęp do systemowych zasobów lub bazy danych, aby upewnić się, że nie zawierają nieoczekiwanych dostępów lub manipulacji.
5. Zbadać popularność i recenzje pakietu: Badanie popularności pakietu w społeczności npm oraz czy inne projekty go używają, może dostarczyć informacji na temat jego jakości i bezpieczeństwa.
6. Zablokować wersje używanych pakietów: Zaleca się blokowanie wersji pakietów npm, aby uniknąć automatycznych aktualizacji, które mogą wprowadzać nieoczekiwane zmiany w funkcjonalnościach lub bezpieczeństwie.
7. Obserwować repozytoria na GitHubie: Subskrybowanie repozytoriów, które są używane w projekcie, pozwala na śledzenie nowych wersji i aktualizacji, co umożliwia szybkie reagowanie na odkryte luki bezpieczeństwa.

# Unvalidated Redirects and Forwards (Nieważne przekierowania i przekazywanie dalej)

Niezatwierdzone przekierowania i przekazywanie (forwards) to luki bezpieczeństwa, które mogą być wykorzystane przez atakujących do przekierowywania użytkowników na złośliwe strony internetowe lub omijania zabezpieczeń. Atakujący może manipulować niezabezpieczonymi danymi, aby skierować użytkownika na phishingowe strony lub strony zawierające złośliwe oprogramowanie. Aplikacje internetowe często używają przekierowań, aby skierować użytkowników do innych stron, w tym także zewnętrznych witryn. Kiedy aplikacja nie weryfikuje poprawności adresów URL, atakujący może zmodyfikować te adresy, aby przekierować użytkowników do niepożądanych miejsc. Przeglądarki użytkowników, ufając aplikacji, automatycznie wykonują te przekierowania, co może prowadzić do poważnych konsekwencji, takich jak kradzież danych uwierzytelniających, infekcja złośliwym oprogramowaniem lub inne formy oszustw.

W aplikacji Nodegoat jest moduł zatytułowany „Learning Resources”, który przekierowuje do innej strony internetowej bez walidacji adresu URL.



Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

Można podmienić ten link na przykładowo: http://localhost:4000/learn?url=https://[www.gambling.com](http://www.gambling.com) i zostać przekierowanym na złośliwą stronę. Ofiary są bardziej skłonne do kliknięcia, ponieważ początkowa część linku (przed parametrami zapytania) wskazuje na zaufaną witrynę.

Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, Strona internetowa

Opis wygenerowany automatycznie

**Jak temu zapobiec?**

Aby zapobiec niebezpieczeństwom związanym z niebezpiecznymi przekierowaniami, można podjąć następujące środki ostrożności:

1. Unikanie używania przekierowań, jeśli to możliwe: Najlepszym sposobem na zapobieżenie problemom związanym z przekierowaniami jest unikanie ich stosowania. Wiele aplikacji może być zaprojektowanych w taki sposób, że nie wymagają bezpośrednich przekierowań na inne strony.
2. Nie angażowanie parametrów użytkownika w obliczanie miejsca docelowego: Jeśli konieczne jest użycie przekierowań, należy unikać umieszczania parametrów kontrolowanych przez użytkownika bezpośrednio w adresie URL przekierowania. Zamiast tego, należy używać wewnętrznych identyfikatorów lub nazw symbolicznych, które są mapowane na właściwe adresy URL po stronie serwera.
3. Weryfikacja i autoryzacja parametrów docelowych: Jeśli używanie parametrów docelowych jest nieuniknione, należy upewnić się, że wartość przekierowania jest prawidłowa i autoryzowana przez aplikację. Można to zrobić poprzez sprawdzenie, czy dana wartość jest na przykład częścią mapowania wewnętrznego.
4. Używanie mapowania zamiast bezpośrednich adresów URL: Zamiast przekazywać użytkownikowi bezpośredni adres URL, który może być manipulowany, lepiej jest użyć mapowania lub identyfikatorów wewnętrznych. Kod po stronie serwera powinien przetłumaczyć te mapowania na rzeczywisty adres URL docelowy, co zwiększa bezpieczeństwo aplikacji.

# Server-Side Request Forgery (SSRF)

Atak Server-Side Request Forgery (SSRF) pozwala atakującemu na wykorzystanie serwera aplikacji do wysyłania złośliwych żądań HTTP do zasobów wewnętrznych lub zewnętrznych. Atakujący może manipulować adresami URL używanymi przez serwer do odczytu lub przesyłania danych, co może prowadzić do wycieku poufnych informacji, takich jak konfiguracja serwera, metadane w środowiskach chmurowych (np. AWS), a także do interakcji z wewnętrznymi usługami (np. bazami danych) lub wykonywania żądań HTTP POST do nieautoryzowanych zasobów. Atak SSRF polega na zmodyfikowaniu adresu URL, który serwer wykorzystuje do wykonywania żądań HTTP. W aplikacjach podatnych na ten atak, serwer może realizować złośliwe żądania przygotowane przez atakującego, które są traktowane jako zaufane.

W aplikacji Nodegoat na stronie „Research” użytkownik może przesłać symbolu akcji. Symbol ten jest następnie łączony z adresem URL Yahoo Finance, a serwer pobiera odpowiedź i wyświetla ją użytkownikowi.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

Za pomocą narzędzia burp możemy przejąć żądanie skierowane do Yahoo, a następnie je zmodyfikować w zakładce Repeater.

Obraz zawierający tekst, zrzut ekranu, linia, numer

Opis wygenerowany automatycznie

Po modyfikacji możemy skopiować link i wyświetlić go w przeglądarce.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

<http://burpsuite/show/2/lfmip5gn0yp102ulbx7teokgmlrmtnhh>

Możemy modyfikować request zależnie od naszych potrzeb. Tutaj zdecydowaliśmy się na wykorzystanie luki SSRF jako sposób na zebranie informacji o serwerze i sieci lokalnej. Co było możliwie dzięki modyfikacji GET’a na”

GET /research?url=http%3A%2F%2Fifconfig.p&symbol=ro HTTP/1.1

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

**Jak temu zapobiec?**

Aby skutecznie zapobiegać lukom w zabezpieczeniach SSRF (Server-Side Request Forgery) w aplikacjach internetowych, warto stosować następujące środki ostrożności:

1. Używanie białej listy dozwolonych domen, zasobów i protokołów:
   * Ogranicz, które domeny, zasoby (np. IP, hostname) i protokoły (np. HTTP, HTTPS) są dozwolone przez Twoją aplikację do pobierania zasobów z zewnętrznych serwerów. Użyj białej listy, aby upewnić się, że tylko te zasoby są dostępne dla Twojej aplikacji.
2. Walidacja danych wejściowych:
   * Wszystkie dane wejściowe akceptowane przez użytkownika, które mają wpływ na decyzje dotyczące pobierania zasobów, powinny być dokładnie walidowane i sprawdzane. Upewnij się, że dane wejściowe są zgodne z oczekiwaną specyfikacją (np. adresem URL zgodnym z dozwolonymi wzorcami).
3. Unikanie przyjmowania danych wejściowych w funkcjach kontrolujących pobieranie zasobów:
   * Jeśli to możliwe, staraj się unikać sytuacji, w których dane wejściowe bezpośrednio kontrolują, skąd serwer WWW pobiera zasoby. Zamiast tego, używaj mapowania wewnętrznego lub uprzednio zdefiniowanych reguł do określania, które zasoby mogą być pobierane.

# Regular Expressions DoS

Atak ReDoS (Regular Expression Denial of Service) polega na wykorzystaniu podatności w implementacjach wyrażeń regularnych, które mogą prowadzić do ekstremalnie długiego czasu przetwarzania danych wejściowych. Atakujący dostarcza specjalnie spreparowane dane, które powodują, że wyrażenie regularne wykonuje wykładniczo złożone obliczenia, co może spowodować zawieszenie lub znaczne spowolnienie działania serwera. Większość implementacji wyrażeń regularnych, w tym te używane w Node.js, jest podatna na ataki ReDoS. Atakujący może wprowadzić długi ciąg znaków, który spowoduje, że wyrażenie regularne będzie wymagało ogromnej ilości czasu na przetworzenie. W kontekście Node.js, który działa na jednowątkowej pętli zdarzeń, taka podatność może zablokować główny wątek, uniemożliwiając obsługę innych żądań.

W aplikacji NodeGoat można zauważyć, że używa ona następującego wyrażenia regularnego do walidacji formatu tekstu od użytkownika:

* var regexPattern = /([0-9]+)+\#/;
* var testComplyWithRequirements = regexPattern.test(bankRouting)

Atakujący może wprowadzić specjalnie spreparowany ciąg znaków, taki jak:

91762612117612121123123123123121

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

Ten ciąg spowoduje, że wyrażenie regularne wejdzie w sytuację, w której będzie potrzebowało ogromnej ilości czasu na przetworzenie, co może prowadzić do wykorzystania 100% procesora i zawieszenia procesu Node.js.

**Jak temu zapobiec?**

Aby zapobiegać atakom ReDoS (Regular Expression Denial of Service) wykorzystując niebezpieczne wyrażenia regularne, należy podjąć następujące środki ostrożności:

1. Unikaj pisania własnych wyrażeń regularnych:
   * Pisanie własnych wyrażeń regularnych może prowadzić do napisania nieoptymalnych lub podatnych na ataki ReDoS. Zamiast tego, rozważ użycie gotowych rozwiązań lub narzędzi zaprojektowanych do bezpiecznej walidacji danych, takich jak pakiet validator.js.
2. Używaj pakietu validator.js:
   * validator.js to biblioteka do walidacji i obróbki danych, która zawiera wbudowane funkcje do sprawdzania poprawności danych, takich jak adresy URL, adresy email, numery telefonów, numery bankowe itp. Jest ona zaprojektowana z myślą o bezpieczeństwie i minimalizacji ryzyka ataków ReDoS.
3. Używaj pakietu safe-regex:
   * Pakiet safe-regex jest narzędziem, które pomaga w analizie i ocenie, czy dane wyrażenie regularne jest bezpieczne i czy jest podatne na ataki ReDoS. Może ono wykryć, czy wyrażenie regularne jest podatne na katastrofalne cofanie, które mogłoby spowodować znaczne obciążenie procesora.

**Przykład kodu źródłowego**

Aktualny kod:  
  
Aby zabezpieczyć aplikację można użyć funkcji isSafeRegex, którea używa safe-regex do oceny, czy wzorzec regexPattern jest bezpieczny. Następnie sprawdzamy, czy ten wzorzec jest bezpieczny przed użyciem go do testowania wartości bankRouting. Dzięki temu ograniczamy ryzyko ReDoS poprzez kontrolowanie złożoności używanych wzorców regularnych.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie