

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

ZARZĄDZANIE INFRASTRUKTURĄ TELEINFORMATYCZNĄ
[W04CBE-SI0013G]

Implementacja serwisu webowego podatnego na ataki OWASP top 10



Politechnika
Wrocławska

Autorzy

Tomasz PRZYBYSZEWSKI, 264149

Olivier STRACHOTA-GORZYCKI, 264072

12 stycznia 2024

Spis treści

1	Instalacja Programu	3
2	OWASP TOP 10	3
2.1	Broken Access Control	3
2.1.1	Omówienie Ataku	3
2.1.2	Przykład Ataku	4
2.1.3	Implementacja Podatności	4
2.1.4	Jak temu zapobiec?	5
2.2	Cryptographic Failures	7
2.2.1	Omówienie Ataku	7
2.2.2	Przykład Ataku	7
2.2.3	Implementacja Podatności	8
2.2.4	Jak temu zapobiec?	9
2.3	Injection	10
2.3.1	Omówienie Ataku	10
2.3.2	Przykład Ataku	11
2.3.3	Implementacja Podatności	11
2.3.4	Jak temu zapobiec?	12
2.4	Insecure Design	13
2.4.1	Omówienie Ataku	13
2.4.2	Przykład Ataku	13
2.4.3	Implementacja Podatności	13
2.4.4	Jak temu zapobiec?	14
2.5	Security Misconfiguration	15
2.5.1	Omówienie Ataku	15
2.5.2	Przykład Ataku	15
2.5.3	Implementacja Podatności	16
2.5.4	Jak temu zapobiec?	17
2.6	Vulnerable and Outdated Components	18
2.6.1	Omówienie Ataku	18
2.6.2	Przykład Ataku	18
2.6.3	Jak temu zapobiec?	19
2.7	Identification and Authentication Failures	20
2.7.1	Omówienie Ataku	20
2.7.2	Przykład Ataku	20
2.7.3	Implementacja Podatności	21
2.7.4	Jak temu zapobiec?	22
2.8	Software and Data Integrity Failures	23
2.8.1	Omówienie Ataku	23
2.8.2	Przykład Ataku	23
2.8.3	Jak temu zapobiec?	24
2.9	Security Logging and Monitoring Failures	25
2.9.1	Omówienie Ataku	25

2.9.2	Przykład Ataku	25
2.9.3	Jak temu zapobiec?	26
2.10	Server-Side Request Forgery (SSRF)	27
2.10.1	Omówienie Ataku	27
2.10.2	Przykład Ataku	27
2.10.3	Implementacja Podatności	27
2.10.4	Jak temu zapobiec?	28

1 Instalacja Programu

1. Pobrać i zainstalować XAMPP, np. ze strony [apachefriends](http://apachefriends.org)
2. Pobrać pliki z githuba
3. Wypakować foldery z archiwów
4. Przenieść wypakowany folder `owasp-top-10` do `C:\xampp\htdocs` (Podana lokalizacja jest domyślna i może się różnić w zależności od miejsca instalacji)
5. Skopiować zawartość folderu `Data` i wkleić ją w `C:\xampp\mysql\data` (Podana lokalizacja jest domyślna i może się różnić w zależności od miejsca instalacji)
6. Uruchomić program XAMPP i wybrać 2 pierwsze moduły, Apache oraz MySQL, kliknąć `Start`
7. W przeglądarce internetowej w pasku URL wpisać `http://localhost/owasp-top-10/`

2 OWASP TOP 10

Podatności zaimplementowane w naszym projekcie oraz opisane w poniższej dokumentacji zostały zaczerpnięte ze strony owasp.org. Podatności są zgodne z listą OWASP Top 10 - 2021.

2.1 Broken Access Control

2.1.1 Omówienie Ataku

Kontrola dostępu ogranicza użytkownika do działania w ramach zamierzonych przez twórcę uprawnień. Awaria kontroli dostępu będzie skutkować brakiem ograniczeń dostępu dla zwykłych użytkowników, może to doprowadzić do ujawnienia informacji poufnych, nieuprawnionej modyfikacji lub usunięcia danych.

Do najczęstszych podatności **Broken Access Control** należą:

- Naruszenie zasady ***principle of least privilege*** - dostęp powinien być przyznawany tylko w określonych przypadkach, dla określonych ról lub użytkowników, ale jest przyznawany każdemu
- Omijanie kontroli dostępu poprzez modyfikację **adresu URL** - zamiana parametrów lub wymuszenie wyszukiwania
- Zezwolenie na przeglądanie lub edycję konta poprzez podanie jej unikalnego identyfikatora - bezpośrednie odniesienia do obiektów
- Dostęp do **API** z brakującymi kontrolami dostępu dla **POST**, **PUT** i **DELETE**.
- Eskalacja uprawnień - działanie jako użytkownik bez zalogowania się lub działanie jako administrator po zalogowaniu się jako użytkownik

- Manipulacja metadanymi - manipulowanie tokenem kontroli dostępu **JSON Web Token(JWT)** lub ukrytym polem w celu eskalacji uprawnień lub unieważnienia **JWT**
- Błędna konfiguracja **Cross-Origin Resource Sharing** - umożliwia dostęp do **API** z nieautoryzowanych/niezaufanych źródeł.

2.1.2 Przykład Ataku

Przykładowy atak wykorzystujący **Broken Access Control** może zostać przeprowadzony w kilku krokach:

1. **Identyfikacja słabego punktu** - Atakujący identyfikuje aplikację, która wykorzystuje niezweryfikowane dane w wywołaniu SQL do dostępu do informacji o kontach użytkowników. Przykładem może być fragment kodu, gdzie aplikacja przyjmuje parametr 'acct' z żądania przeglądarki do przygotowania zapytania SQL:

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

2. **Manipulacja parametru** - Atakujący modyfikuje parametr 'acct' w adresie URL przeglądarki, aby przesłać dowolny numer konta. Na przykład, zmieniając URL na: *https://example.com/app/accountID?acct=notmyacct*, w ten sposób, jeśli aplikacja nie sprawdza poprawności tego parametru, atakujący może próbować uzyskać dostęp do różnych kont użytkowników.
3. **Wykorzystanie słabego punktu** - Przez modyfikację parametru 'acct', atakujący przesyła różne numery kont w nadziei na uzyskanie dostępu do informacji innych użytkowników. Jeśli aplikacja nie weryfikuje, czy użytkownik ma uprawnienia do dostępu do konta o podanym numerze, atakujący może uzyskać dostęp do danych nieautoryzowanych kont.
4. **Zbieranie danych** - W przypadku powodzenia ataku, atakujący może przeglądać informacje z różnych kont, włącznie z danymi osobowymi, informacjami finansowymi lub innymi wrażliwymi danymi.
5. **Eskalacja ataku** - Wykorzystując uzyskane informacje, atakujący może podejmować dalsze działania, takie jak kradzież tożsamości, oszustwa finansowe lub dalsze ataki na system.

2.1.3 Implementacja Podatności

W naszym projekcie zaimplementowaliśmy **Broken Access Control** poprzez brak kontroli adresu URL. Jest to jeden z najprostrzych sposobów uzyskania dostępu do bazy danych ponieważ nie wymaga od użytkownika żadnej znajomości loginów i haseł się tam znajdujących.

ID:

Login:

Submit

Rysunek 1: Okienko logowania

Standardowe okienko wyświetlenia danych wygląda jak na powyższym obrazku. Po wpisaniu poprawnego loginu oraz hasła wyświetlą się dane użytkownika. Jednak można to w bardzo prosty sposób obejść. Wystarczy na samym końcu adresu URL dopisać np. `?id=3` w celu wyświetlenia danych użytkownika o podanym id. Otrzymamy wtedy dostęp do strony do której normalnie nie moglibyśmy mieć dostępu.

User Details

ID: 3
Email: andrzej@example.com
Login: andrzej
Password: secret

Rysunek 2: Podstrona otrzymana na skutek Broken Access Control

Jeśli znamy login użytkownika to w tym wypadku również możemy uzyskać dostęp do tej podstrony poprzez dopisanie na końcu adresu URL np. `?login=andrzej`.

2.1.4 Jak temu zapobiec?

- Wprowadzenie zasady *principle of least privilege*
- Kontrola dostępu oparta na rolach - model **RBAC** pozwala na łatwe zarządzanie uprawnieniami dla grup użytkowników zamiast dla pojedynczych użytkowników.
- Autoryzacja na poziomie funkcji - system powinien sprawdzać uprawnienia przy każdym dostępie do funkcji wymagającej autoryzacji, a nie tylko przy logowaniu
- Monitorowanie aktywności - Monitorowanie aktywności użytkowników i zapisywanie dzienników może pomóc w wykrywaniu nieautoryzowanych prób dostępu i innych podejrzanych działań.
- Przeprowadzanie audytów - Regularne audyty i testy penetracyjne pozwolą wykrywać i naprawiać luki w zabezpieczeniach.

- Regularne aktualizacje - Regularna aktualizacja oprogramowania i systemów zapewni ochronę przed znanymi lukami w zabezpieczeniach
- Wprowadzenie zasady *zero trust*

2.2 Cryptographic Failures

2.2.1 Omówienie Ataku

Ataki **Cryptographic Failures** występują kiedy atakujący kierują swoje działania na wrażliwe dane, takie jak hasła, numery kart kredytowych i informacje osobiste, gdy nie są one odpowiednio chronione. Jest to podstawowa przyczyna narażenia wrażliwych danych na ryzyko.

Aby zapewnić ochronę danych w transmisji i przechowywanych, należy wziąć pod uwagę następujące aspekty:

- Stare lub słabe algorytmy kryptograficzne - Istotne jest unikanie stosowania starych lub słabych algorytmów lub protokołów kryptograficznych, zarówno domyślnie, jak i w starszym kodzie.
- Wymuszenie szyfrowania - Ważne jest, aby wymuszać szyfrowanie, na przykład przez stosowanie odpowiednich dyrektyw lub nagłówków bezpieczeństwa w przeglądarkach.
- Losowość w celach kryptograficznych - Ważne jest, aby do celów kryptograficznych wykorzystywać losowość, która została zaprojektowana do spełnienia wymagań kryptograficznych i nie była ziarnowana w przewidywalny sposób lub z niską entropią.
- Przestarzałe funkcje hashowania - Należy unikać stosowania przestarzałych funkcji hashujących, takich jak MD5 czy SHA1, oraz niekryptograficznych funkcji hashujących tam, gdzie potrzebne są kryptograficzne.
- Hasła jako klucze kryptograficzne - Nie należy używać haseł jako kluczy kryptograficznych bez funkcji pochodzenia klucza opartej na hasle.

2.2.2 Przykład Ataku

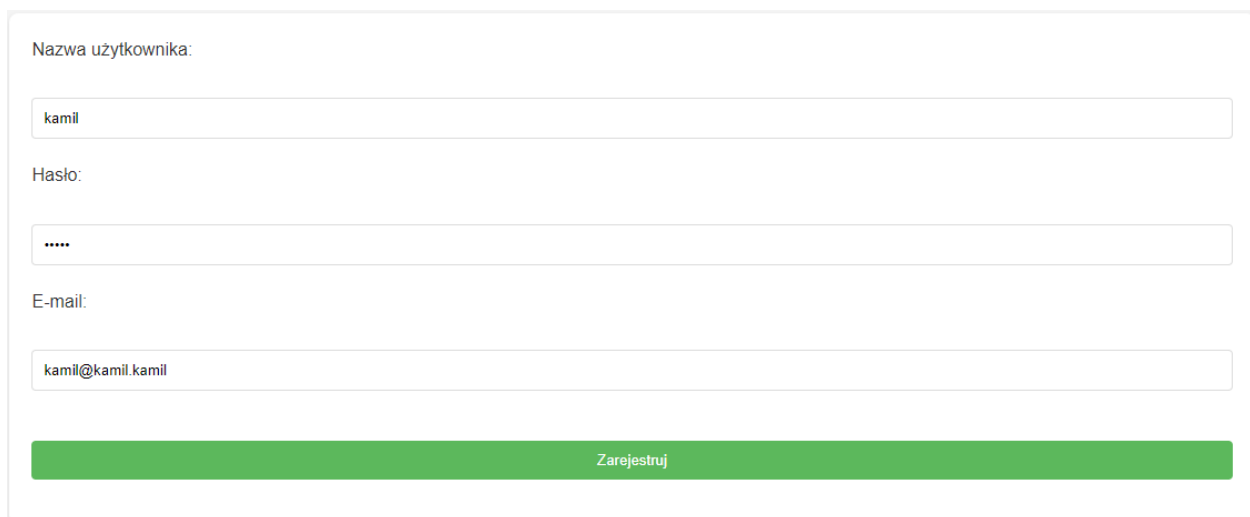
Przykładowy atak wykorzystujący **Cryptographic Failures** może zostać przeprowadzony w kilku krokach:

1. **Wykrycie wady w zabezpieczeniach przesyłania plików** - Atakujący odkrywa lukę w zabezpieczeniach funkcji przesyłania plików na stronie internetowej, która umożliwia wgranie i pobranie dowolnych plików z serwera. Taka luka może wynikać z braku odpowiednich ograniczeń co do rodzajów przesyłanych plików lub braku weryfikacji uprawnień użytkowników.
2. **Wykorzystanie luki do pobrania bazy danych haseł** - Atakujący wykorzystuje lukę, by przesłać i uruchomić skrypt lub narzędzie na serwerze, które pozwala mu na dostęp do i pobranie bazy danych haseł użytkowników. Baza ta zawiera hasła zapisane w formie hashy, ale bez solenia lub używające prostej funkcji hashującej.
3. **Analiza haseł** - Po pobraniu bazy, atakujący analizuje hashe. Zauważa, że hasła są niesolone lub zostały utworzone za pomocą prostych, szybkich algorytmów hashujących, takich jak MD5 czy SHA-1, które są podatne na ataki.

4. **Wykorzystanie tęczowych tablic lub mocy obliczeniowej GPU** - Używając tęczowych tablic, które są dużymi, prekalkulowanymi zbiorami hashy, atakujący próbuje dopasować zapisane w bazie hashe do znanych wartości. Dla niesolonych hashy, ten proces może być bardzo szybki i efektywny. Jeśli hasła były solone, ale z użyciem prostych funkcji hashujących, atakujący może użyć mocy obliczeniowej GPU (kart graficznych), które są efektywne w łamaniu takich hashy.
5. **Odszyfrowanie haseł i dalsze wykorzystanie** - Po zdekodowaniu jednego lub większej liczby haseł, atakujący może wykorzystać tę informację do uzyskania nieautoryzowanego dostępu do kont użytkowników, wykonania ataków podszywając się pod użytkownika, przeprowadzenia innych ataków na infrastrukturę IT lub kradzieży wrażliwych danych.

2.2.3 Implementacja Podatności

W naszym projekcie zaimplementowaliśmy **Cryptographic Failures** poprzez hashowanie haseł przestarzałą funkcją hashującą MD5. W przygotowanym przez nasz przykładzie należy zarejestrować nowego użytkownika.



The image shows a registration form with three input fields and a submit button. The first field is labeled 'Nazwa użytkownika:' and contains the text 'kamil'. The second field is labeled 'Hasło:' and contains five dots. The third field is labeled 'E-mail:' and contains the text 'kamil@kamil.kamil'. Below the fields is a green button with the text 'Zarejestruj'.

Rysunek 3: Rejestracja nowego użytkownika

Po rejestracji w bazie danych będzie widać nowego użytkownika i hasło zahashowane funkcją MD5, którą można bardzo łatwo odszyfrować za pomocą narzędzi dostępnych w internecie.

```
id: 1 - Nazwa użytkownika: jan123 - Hasło: 32250170a0dca92d53ec9624f336ca24 - Email: jan@example.com
id: 2 - Nazwa użytkownika: anna89 - Hasło: d8578edf8458ce06fbc5bb76a58c5ca4 - Email: anna@example.com
id: 3 - Nazwa użytkownika: andrzej - Hasło: 5ebe2294ecd0e0f08eab7690d2a6ee69 - Email: andrzej@example.com
id: 4 - Nazwa użytkownika: maria23 - Hasło: 7c6a180b36896a0a8c02787eeafb0e4c - Email: maria@example.com
id: 5 - Nazwa użytkownika: piotr - Hasło: 0f359740bd1cda994f8b55330c86d845 - Email: piotr@example.com
id: 6 - Nazwa użytkownika: kasia - Hasło: f17d41986c2802c2dc68fd54f39a617a - Email: katarzyna@example.com
id: 7 - Nazwa użytkownika: tom123 - Hasło: bb77d0d3b3f239fa5db73bdf27b8d29a - Email: tomasz@example.com
id: 8 - Nazwa użytkownika: magda - Hasło: 190fd720341a78719364488388416bf9 - Email: magdalena@example.com
id: 9 - Nazwa użytkownika: pawel34 - Hasło: d00f77c7a476ce32060ba7e843deb2b8 - Email: pawel@example.com
id: 10 - Nazwa użytkownika: ewa85 - Hasło: e8f1df8f7db78f677a86ca22a7dd2c8d - Email: ewa@example.com
id: 13 - Nazwa użytkownika: kamil - Hasło: 11d462a4a1b14b00580d8020d6f64998 - Email: kamil@kamil.kamil
```

Rysunek 4: Hasła zahashowane MD5

2.2.4 Jak temu zapobiec?

Przestrzeganie poniższych zasad jest niezbędne w celu zapewnienia bezpieczeństwa danych przetwarzanych, przechowywanych lub transmitowanych przez aplikację.

- Klasyfikacja danych - Należy klasyfikować dane przetwarzane, przechowywane lub przesyłane przez aplikację. Identyfikować, które dane są wrażliwe zgodnie z przepisami o ochronie prywatności, wymogami regulacyjnymi lub potrzebami biznesowymi.
- Minimalizacja przechowywania danych wrażliwych - Unikać niepotrzebnego przechowywania danych wrażliwych. Dane, które nie są zatrzymywane, nie mogą zostać skradzione.
- Szyfrowanie danych w spoczynku - Należy upewnić się, że wszystkie dane wrażliwe są szyfrowane w spoczynku.
- Zastosowanie silnych algorytmów i protokołów.
- Szyfrowanie danych w transmisji - Szyfrowanie wszystkich danych w transmisji przy użyciu bezpiecznych protokołów, takich jak TLS z szyframi zapewniającymi poufność do przodu (FS), priorytetyzacją szyfrów przez serwer i bezpiecznymi parametrami. Wymuszanie szyfrowania za pomocą dyrektyw, takich jak HTTP Strict Transport Security (HSTS).
- Wyłączenie pamięci podręcznej dla odpowiedzi zawierających dane wrażliwe.
- Unikanie przestarzałych protokołów.
- Używanie szyfrowania z uwierzytelnieniem -Używać szyfrowania z uwierzytelnieniem zamiast samego szyfrowania.
- Unikanie przestarzałych funkcji i schematów kryptograficznych.
- Generowanie i przechowywanie kluczy - Klucze powinny być generowane kryptograficznie losowo i przechowywane w pamięci jako tablice bajtów. Jeśli używane jest hasło, musi ono zostać przekształcone na klucz za pomocą odpowiedniej funkcji pochodzenia klucza bazującej na hasle.

2.3 Injection

2.3.1 Omówienie Ataku

Podatność na ataki typu **Injection** pozwala atakującym wprowadzić złośliwe dane wejściowe do aplikacji lub przekazywać złośliwy kod przez aplikację do innego systemu. Podczas tego ataku niezaufane dane wejściowe lub nieautoryzowany kod są "wstrzykiwane" do programu i interpretowane jako część zapytania lub polecenia. Rezultatem jest zmiana działania programu, przekierowująca go do realizacji szkodliwych celów.

Aplikacja jest podatna na ataki typu **Injection** gdy:

- Dane dostarczone przez użytkownika nie są weryfikowane, filtrowane lub oczyszczane przez aplikację.
- Używane są dynamiczne zapytania lub niestandardowe wywołania bez uwzględniania kontekstu, które są bezpośrednio interpretowane.
- Dane są wykorzystywane w parametrach wyszukiwania mapowania obiektowo-relacyjnego (**Object-relational mapping**) do wydobywania dodatkowych, wrażliwych rekordów.
- Dane są używane bezpośrednio lub są konkatelowane - Zapytanie SQL lub inna komenda zawiera strukturę i złośliwe dane w dynamicznych zapytaniach, poleceniach lub procedurach składowanych.

Niektóre z najpopularniejszych ataków typu **Injection** to:

- SQL Injection
- NoSQL Injection
- OS Command Injection (Shell Injection)
- ORM Injection
- LDAP Injection
- Expression Language Injection
- Object Graph Navigation Library Injection

Idea tych ataków jest podobna dla wszystkich interpreterów. Przegląd kodu źródłowego to najlepszy sposób na wykrycie podatności aplikacji na ten typ ataku. Zaleca się automatyzację testów wszystkich elementów wejściowych aplikacji, takich jak parametry, nagłówki, adresy URL, ciasteczka, JSON, SOAP i XML.

2.3.2 Przykład Ataku

Przykładowy atak wykorzystujący **Injection** może zostać przeprowadzony w kilku krokach:

1. **Identyfikacja zapytania SQL** - Atakujący rozpoznaje, że aplikacja tworzy zapytanie SQL poprzez bezpośrednie dołączenie danych wejściowych użytkownika:

```
1 String query = "SELECT * FROM accounts WHERE custID='" + request.  
2 getParameter("id") + "'";
```

2. **Przygotowanie danych wejściowych do ataku** - Atakujący przygotowuje specjalnie skonstruowany ciąg wejściowy, który, gdy zostanie dołączony do zapytania SQL, zmieni jego strukturę i zachowanie. Na przykład, jeśli atakujący dostarczy wartość *id* jako:

```
1 ' OR '1'='1
```

Zapytanie SQL zmieni się w następujący sposób:

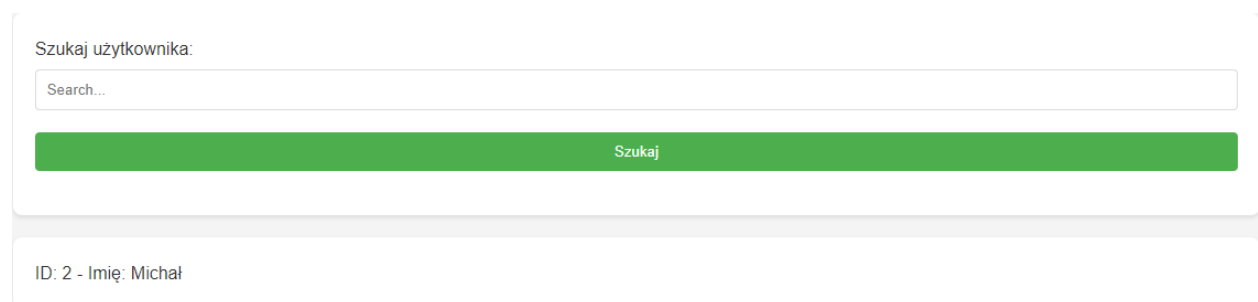
```
1 SELECT * FROM accounts WHERE custID='' OR '1'='1';
```

Warunek `'1'='1'` jest zawsze prawdziwy, więc zapytanie zwróci wszystkie rekordy z tabeli `accounts`.

3. **Wykorzystanie podatności** - Aplikacja, nieświadoma, że dane wejściowe zostały zmodyfikowane, wykonuje zmodyfikowane zapytanie SQL. W efekcie, atakujący może uzyskać dostęp do wrażliwych danych z bazy, które w normalnych warunkach nie byłyby dla niego dostępne.
4. **Ekstrakcja danych** - Wykorzystując uzyskany dostęp, atakujący może wykraść dane użytkowników, takie jak numery kont, dane osobowe, informacje finansowe itp.

2.3.3 Implementacja Podatności

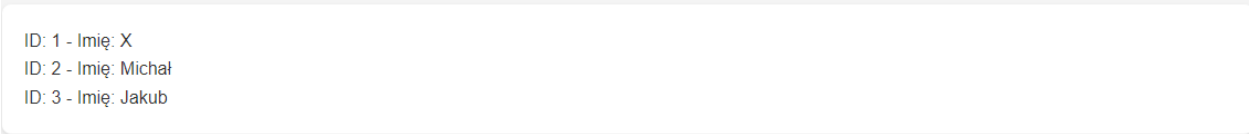
W naszym projekcie zaimplementowaliśmy **Injection** poprzez brak filtracji danych wejściowych. Domyślnie w tej podstronie możemy wyszukać użytkownika poprzez wpisanie jego nazwy w przeznaczonym do tego okienku. Przykładowo możemy znaleźć użytkownika o imieniu Michał. Otrzymamy poniższy efekt.



The screenshot shows a web interface for searching users. At the top, there is a label "Szukaj użytkownika:" followed by a text input field containing the placeholder "Search...". Below the input field is a prominent green button with the text "Szukaj". Underneath the button is a light gray rectangular box that displays the search results. The results shown are "ID: 2 - Imię: Michał".

Rysunek 5: Okienko do wyszukiwania użytkowników z wynikiem wyszukiwania

Jednak wpisując w tym okienku kwerendę SQL np. `' OR 1=1#` możemy dostać dostęp do całej bazy danych, ponieważ dopiszemy warunek, który jest zawsze prawdziwy i otrzymamy poniższy wynik.



```
ID: 1 - Imię: X
ID: 2 - Imię: Michał
ID: 3 - Imię: Jakub
```

Rysunek 6: Wyświetlona baza danych po wpisaniu kwerendy

2.3.4 Jak temu zapobiec?

Aby zapobiec atakom typu **Injection**, należy odzielić dane od zapytań użytkownika.

- Użycie bezpiecznego API - Najlepszą opcją jest użycie bezpiecznego API, które całkowicie unika korzystania z interpretera, zapewnia interfejs parametryzowany lub przeprowadza migrację do narzędzi mapowania obiektowo-relacyjnego (ORM).
- Pozytywna walidacja danych wejściowych po stronie serwera - Samo w sobie nie jest kompletną obroną, ponieważ wiele aplikacji wymaga specjalnych znaków, takich jak obszary tekstowe czy API dla aplikacji mobilnych.
- Dla wszelkich pozostałych dynamicznych zapytań - Ucieczka (escape) specjalnych znaków przy użyciu specyficznej składni ucieczki dla danego interpretera.
- Użycie LIMIT i innych kontroli SQL w zapytaniach - Ma to na celu zapobieganie masowemu ujawnianiu rekordów w przypadku **SQL Injection**.

2.4 Insecure Design

2.4.1 Omówienie Ataku

Ataki typu **Insecure Design** to obszerna kategoria obejmująca różne słabości, określane jako "brakujące lub nieskuteczne projektowanie kontroli". Istnieje różnica między niebezpiecznym projektowaniem a niebezpieczną implementacją. Rozróżniamy wady projektowania i wady implementacji z ważnego powodu - mają one różne przyczyny i wymagają odmiennych środków zaradczych. Bezpieczny projekt może nadal posiadać wady implementacyjne prowadzące do podatności, które mogą zostać wykorzystane. Niebezpieczny projekt nie może zostać naprawiony przez perfekcyjną implementację, ponieważ niezbędne kontrole bezpieczeństwa nigdy nie zostały stworzone, aby bronić przed konkretnymi atakami. Jednym z czynników przyczyniających się do niebezpiecznego projektowania jest brak profilowania ryzyka w oprogramowaniu lub systemie, który jest rozwijany, a co za tym idzie brak zdolności do ustalenia wymaganego poziomu projektowania bezpieczeństwa.

2.4.2 Przykład Ataku

Przykładowy atak wykorzystujący **Insecure Design** może zostać przeprowadzony w kilku krokach:

1. **Analiza systemu** - Atakujący zauważa, że sieć kin oferuje zniżki grupowe i pozwala na rezerwację do 15 miejsc na jedną grupę bez konieczności wpłaty zaliczki. Atakujący analizuje ten proces rezerwacji i modeluje potencjalne zagrożenia.
2. **Wyszukanie luki w procesie rezerwacji** - Atakujący odkrywa, że system rezerwacji nie ma mechanizmów ograniczających liczbę jednoczesnych rezerwacji dokonywanych przez jednego użytkownika lub z jednego adresu IP.
3. **Przeprowadzenie ataku** - Wykorzystując tę lukę, atakujący opracowuje skrypt lub program, który automatycznie dokonuje wielu rezerwacji na maksymalną liczbę osób jednocześnie w wielu kinach. Na przykład, atakujący może zarezerwować 600 miejsc w różnych kinach, wypełniając wszystkie dostępne seanse, nie ponosząc przy tym żadnych kosztów zaliczki.
4. **Efekt ataku** - W wyniku tego ataku, prawdziwi klienci nie mogą dokonać rezerwacji, co prowadzi do znaczących strat finansowych dla sieci kin. Ponadto, ze względu na brak zaliczki, atakujący nie ponosi żadnego ryzyka finansowego.

2.4.3 Implementacja Podatności

W naszym projekcie zaimplementowaliśmy **Insecure Design** poprzez umożliwienie resetowania hasła tylko dzięki poprawnej odpowiedzi na pytanie kontrolne. Na podstronie widoczny jest formularz zmiany hasła do konta, potrzeba jedynie loginu i odpowiedzi na pytanie kontrolne, obydwa można zdobyć w sposób siłowy, czyli wpisywanie przykładowych loginów i najpopularniejszych zwierząt, aż uda się zresetować hasło. My dla celów prezentacyjnych umożliwiliśmy wgląd do bazy danych.

```
id: 1 - Nazwa użytkownika: Jakub - Hasło: qwerty - Odpowiedź: kot
id: 2 - Nazwa użytkownika: Marta - Hasło: tajnehaslo - Odpowiedź: pies
id: 3 - Nazwa użytkownika: Kamil - Hasło: qwerty - Odpowiedź: kot
id: 4 - Nazwa użytkownika: Anna - Hasło: 12345 - Odpowiedź: ryba
id: 5 - Nazwa użytkownika: Zofia - Hasło: haslo2023 - Odpowiedź: krowa
id: 6 - Nazwa użytkownika: Paweł - Hasło: admin123 - Odpowiedź: królik
id: 7 - Nazwa użytkownika: Magdalena - Hasło: password - Odpowiedź: kura
id: 8 - Nazwa użytkownika: Tomasz - Hasło: 123abc - Odpowiedź: pies
id: 9 - Nazwa użytkownika: Karolina - Hasło: solniczka - Odpowiedź: pies
id: 10 - Nazwa użytkownika: Michał - Hasło: michal1 - Odpowiedź: chomik
```

Rysunek 7: Stworzona przez nas baza danych

Widzimy w bazie użytkownika o nazwie Jakub, hasło qwerty oraz, że jego ulubione zwierze to kot. Znając login i odpowiedź na pytanie kontrolne możemy zmienić hasło do tego konta na np. `test`, wtedy zyskamy pełen dostęp do konta i będziemy mogli zdobyć dostępne na nim informacje, albo wykorzystać je w inny sposób.

```
id: 1 - Nazwa użytkownika: Jakub - Hasło: test - Odpowiedź: kot
id: 2 - Nazwa użytkownika: Marta - Hasło: tajnehaslo - Odpowiedź: pies
id: 3 - Nazwa użytkownika: Kamil - Hasło: qwerty - Odpowiedź: kot
id: 4 - Nazwa użytkownika: Anna - Hasło: 12345 - Odpowiedź: ryba
id: 5 - Nazwa użytkownika: Zofia - Hasło: haslo2023 - Odpowiedź: krowa
id: 6 - Nazwa użytkownika: Paweł - Hasło: admin123 - Odpowiedź: królik
id: 7 - Nazwa użytkownika: Magdalena - Hasło: password - Odpowiedź: kura
id: 8 - Nazwa użytkownika: Tomasz - Hasło: 123abc - Odpowiedź: pies
id: 9 - Nazwa użytkownika: Karolina - Hasło: solniczka - Odpowiedź: pies
id: 10 - Nazwa użytkownika: Michał - Hasło: michal1 - Odpowiedź: chomik
```

Rysunek 8: Baza ze zmienionym hasłem

2.4.4 Jak temu zapobiec?

Aby zapewnić bezpieczeństwo w procesie tworzenia oprogramowania, należy wdrożyć i stosować następujące praktyki:

- Tworzenie i korzystanie z biblioteki bezpiecznych wzorców projektowych lub gotowych do użycia komponentów:
- Stosowanie modelowania zagrożeń dla krytycznych elementów jak uwierzytelnianie, kontrola dostępu, logika biznesowa i kluczowe procesy:
- Ustanowienie i wykorzystanie bezpiecznego cyklu życia rozwoju oprogramowania
- Wprowadzenie kontroli wiarygodności na każdym poziomie aplikacji (od frontendu do backendu):
- Ograniczenie zużycia zasobów przez użytkownika lub usługę:

2.5 Security Misconfiguration

2.5.1 Omówienie Ataku

Security Misconfiguration występuje, gdy opcje bezpieczeństwa nie są zdefiniowane w sposób maksymalizujący bezpieczeństwo, lub gdy usługi są wdrażane z niebezpiecznymi ustawieniami domyślnymi. Problem ten może wystąpić w każdym systemie komputerowym, aplikacji oprogramowania, jak również w infrastrukturze chmurowej i sieciowej.

Aplikacja może być podatna na zagrożenia z następujących powodów:

- Brak wzmocnienia bezpieczeństwa - W aplikacji brakuje odpowiedniego wzmocnienia bezpieczeństwa lub nieprawidłowo skonfigurowano uprawnienia w usługach chmurowych.
- Włączenie niepotrzebnych funkcji - Aktywowano niepotrzebne funkcje lub zainstalowano dodatkowe komponenty, takie jak zbędne porty, usługi, strony, konta lub uprawnienia.
- Domyślne konta i hasła - Domyślne konta i hasła pozostają aktywne i niezmienione.
- Nieodpowiednia obsługa błędów - Obsługa błędów ujawnia stopy wywołań lub inne nadmiernie informacyjne komunikaty o błędach użytkownikom.
- Brak nagłówków bezpieczeństwa - Serwer nie wysyła nagłówków bezpieczeństwa lub dyrektyw, lub nie są one ustawione na wartości zapewniające bezpieczeństwo.
- Wyłączenie najnowszych funkcji bezpieczeństwa w zaktualizowanych systemach - W zaktualizowanych systemach najnowsze funkcje bezpieczeństwa są wyłączone lub nie są skonfigurowane w sposób bezpieczny.

2.5.2 Przykład Ataku

Przykładowy atak wykorzystujący **Security Misconfiguration** może zostać przeprowadzony w kilku krokach:

1. **Wykrycie słabości w konfiguracji** - Atakujący wchodzi z aplikacją w interakcję, która powoduje błąd (np. wprowadzając nieprawidłowe dane wejściowe lub wywołując nieistniejące adresy URL). Zauważa, że serwer aplikacji zwraca szczegółowe komunikaty o błędach.
2. **Analiza zwróconych informacji** - Komunikaty o błędach mogą zawierać wrażliwe informacje, takie jak szczegóły wewnętrznej struktury aplikacji, nazwy komponentów, ich wersje, informacje o strukturze baz danych czy algorytmach używanych w aplikacji. Atakujący analizuje te informacje w poszukiwaniu potencjalnych słabości.
3. **Identyfikacja komponentów podatnych na ataki** - Na podstawie zebranych informacji, atakujący identyfikuje konkretne komponenty lub moduły aplikacji, które są przestarzałe lub znane z podatności. Przykładowo, może odkryć, że aplikacja korzysta z konkretnej wersji frameworka lub biblioteki, która ma znane luki bezpieczeństwa.

4. **Przygotowanie i przeprowadzenie ataku** - Wykorzystując te informacje, atakujący opracowuje atak dostosowany do wykrytych podatności. Może to być np. atak typu **Injection**, jeśli znaleziono podatność w mechanizmie przetwarzania zapytań, czy atak typu DoS, jeśli znaleziono podatność, która pozwala na przeciążenie serwera.
5. **Wykorzystanie wykrytych słabości** - Atakujący wykorzystuje zidentyfikowane słabości do uzyskania nieautoryzowanego dostępu, kradzieży danych, zakłócenia działania aplikacji lub innych szkodliwych działań.

2.5.3 Implementacja Podatności

W naszym projekcie zaimplementowaliśmy **Security Misconfiguration** poprzez dodanie dodatkowego zbędnego komponentu jakim jest zły nagłówek strony. Domyślny adres tej podstrony postawionej na serwerze lokalnym to: `http://localhost/owasp-top-10/vulnerabilities/security-misconfiguration/index.html`.

OWASP Top 10 - Podatność 5: Security Misconfiguration

Omówienie Ataku

Security Misconfiguration występuje, gdy opcje bezpieczeństwa nie są zdefiniowane w sposób maksymalizujący bezpieczeństwo, lub gdy usługi są wdrażane z niebezpiecznymi ustawieniami domyślnymi. Problem ten może wystąpić w każdym systemie komputerowym, aplikacji oprogramowania, jak również w infrastrukturze chmurowej i sieciowej.




Do najczęstszych podatności na ataki typu Security Misconfiguration należą:

- Brak wzmocnienia bezpieczeństwa - W aplikacji brakuje odpowiedniego wzmocnienia bezpieczeństwa lub nieprawidłowo skonfigurowano uprawnienia w usługach chmurowych.
- Włączenie niepotrzebnych funkcji - Aktywowano niepotrzebne funkcje lub zainstalowano dodatkowe komponenty, takie jak zbędne porty, usługi, strony, konta lub uprawnienia.
- Domyślne konta i hasła - Domyślne konta i hasła pozostają aktywne i niezmienione.
- Nieodpowiednia obsługa błędów - Obsługa błędów ujawnia stopy wywołań lub inne nadmiernie informacyjne komunikaty o błędach użytkownikom.
- Brak nagłówków bezpieczeństwa - Serwer nie wysyła nagłówków bezpieczeństwa lub dyrektyw, lub nie są one ustawione na wartości zapewniające bezpieczeństwo.
- Wyłączenie najnowszych funkcji bezpieczeństwa w zaktualizowanych systemach - W zaktualizowanych systemach najnowsze funkcje bezpieczeństwa są wyłączone lub nie są skonfigurowane w sposób bezpieczny.

Rysunek 9: Domyślny wygląd podstrony

Domyślnie podstrona wygląda tak jak przedstawiono poniżej, wystarczy jednak zmodyfikować adres URL i usunąć z niego nagłówek strony, żeby zostało `http://localhost/owasp-top-10/vulnerabilities/security-misconfiguration/` i dostaniemy dostęp do plików serwera i będziemy mogli dowolnie zmieniać podstrony bez weryfikacji i będziemy mieli dostęp do plików, które normalnie nie są dostępne.

Index of /owasp-top-10/vulnerabilities/security-misconfiguration

Name	Last modified	Size	Description
 Parent Directory	-	-	-
 indeks.html	2024-01-12 14:16	1.9K	
 test.txt	2024-01-12 16:57	40	

Apache/2.4.58 (Win64) OpenSSL/3.1.3 PHP/8.2.12 Server at localhost Port 80

Rysunek 10: Ujawnione pliki serwera

2.5.4 Jak temu zapobiec?

W celu uniknięcia **Security Misconfiguration** najlepiej zaimplementować:

- Powtarzalny proces wzmacniania - Proces ten umożliwia szybkie i łatwe wdrożenie kolejnego środowiska, które jest odpowiednio zabezpieczone. Środowiska rozwojowe, testowe i produkcyjne powinny być konfigurowane identycznie, z różnymi poświadczeniami użytkowymi w każdym środowisku. Proces ten powinien być zautomatyzowany, aby zminimalizować wysiłek potrzebny do ustawienia nowego bezpiecznego środowiska.
- Minimalizację platformy - Należy usunąć lub nie instalować niepotrzebnych funkcji, frameworków, komponentów, dokumentacji oraz przykładów. Skupienie się na minimalnej, niezbędnej funkcjonalności platformy zmniejsza powierzchnię ataku i ryzyko bezpieczeństwa.
- Regularny przegląd i aktualizacja konfiguracji - Należy regularnie przeglądać i aktualizować konfiguracje zgodnie z wszystkimi notatkami bezpieczeństwa, aktualizacjami i łatkami, jako część procesu zarządzania łatkami.
- Segmentowana architektura aplikacji - Zapewnić skuteczne i bezpieczne oddzielenie między komponentami lub najemcami aplikacji poprzez segmentację, konteneryzację lub stosowanie grup bezpieczeństwa w chmurze (ACL).
- Wysyłanie dyrektyw bezpieczeństwa do klientów - Zastosować wysyłanie nagłówków bezpieczeństwa do klientów, co pomaga w ochronie przed różnymi zagrożeniami internetowymi.
- Automatyzacja procesu weryfikacji konfiguracji i ustawień - Należy automatycznie weryfikować skuteczność konfiguracji i ustawień w każdym środowisku, aby zapewnić ich zgodność z najlepszymi praktykami bezpieczeństwa. Pozwala to na szybkie wykrywanie i korygowanie potencjalnych luk w zabezpieczeniach.

2.6 Vulnerable and Outdated Components

2.6.1 Omówienie Ataku

Vulnerable and Outdated Components odnosi się do bibliotek lub frameworków stron trzecich używanych w aplikacjach internetowych, które mają znane podatności lub nie są już wspierane przez ich twórców. Takie komponenty mogą być wykorzystywane przez atakujących do uzyskania nieautoryzowanego dostępu do wrażliwych danych lub przejęcia kontroli nad systemem.

Istnieje prawdopodobieństwo, że system jest podatny na zagrożenia, jeśli występuje:

- Nieznajomość wersji wszystkich komponentów - Nieznajomość wersji wszystkich używanych komponentów (zarówno po stronie klienta, jak i serwera), włączając w to komponenty bezpośrednio wykorzystywane oraz zagnieżdżone zależności.
- Użycie oprogramowania podatnego, nieobsługiwanego lub przestarzałego - Dotyczy to systemu operacyjnego, serwera internetowego/aplikacji, systemu zarządzania bazą danych (DBMS), aplikacji, API i wszystkich komponentów, środowisk uruchomieniowych oraz bibliotek.
- Brak regularnego skanowania podatności i subskrypcji biuletynów bezpieczeństwa
- Brak naprawy lub aktualizacji podstawowej platformy, frameworków i zależności w odpowiednim czasie - Często występuje w środowiskach, gdzie aktualizacje są realizowane miesięcznie lub kwartalnie w ramach kontroli zmian, pozostawiając organizacje narażone na dni lub miesiące niepotrzebnego ryzyka z powodu niezaktualizowanych podatności.
- Brak testowania kompatybilności przez deweloperów oprogramowania
- Niebezpieczna konfiguracja komponentów - Dotyczy to braku zabezpieczenia konfiguracji komponentów (**Security Misconfiguration**).

2.6.2 Przykład Ataku

Przykładowy atak wykorzystujący **Vulnerable and Outdated Components** może zostać przeprowadzony w kilku krokach:

1. **Identyfikacja podatnego komponentu** - Atakujący przeprowadza analizę aplikacji w celu zidentyfikowania używanych w niej zewnętrznych komponentów, takich jak biblioteki, frameworki czy moduły. Szczególną uwagę zwraca na te komponenty, które nie były aktualizowane od dłuższego czasu lub są znane z historycznych podatności.
2. **Znalezienie słabości** - Po zidentyfikowaniu potencjalnie podatnego komponentu, atakujący szuka informacji o znanych słabościach tego komponentu. Może to zrobić, na przykład, przeszukując publiczne bazy danych podatności takie jak CVE (Common Vulnerabilities and Exposures).
3. **Wykorzystanie znalezionej podatności** - Jeśli atakujący znajdzie słabość, np. błąd w kodzie lub umyślnie umieszczony backdoor, przygotowuje atak, który wykorzysta tę podatność. Na przykład, jeśli komponent ma nierozwiązaną podatność pozwalającą na wykonanie zdalnego kodu, atakujący może stworzyć specjalnie spreparowane żądanie, które, gdy zostanie przetworzone przez aplikację, pozwoli na wykonanie arbitralnego kodu przez atakującego.

4. **Przeprowadzenie ataku** - Atakujący wykonuje przygotowany atak, który może prowadzić do różnych negatywnych konsekwencji, w zależności od rodzaju wykorzystanej podatności. Może to obejmować kradzież wrażliwych danych, instalację złośliwego oprogramowania, przerwanie działania aplikacji, czy uzyskanie nieautoryzowanego dostępu do systemu.
5. **Eskalacja uprawnień** - Ze względu na to, że komponenty często działają z takimi samymi uprawnieniami jak sama aplikacja, wykorzystanie podatności w komponencie może umożliwić atakującemu uzyskanie pełnych uprawnień aplikacji, a nawet całego systemu.

2.6.3 Jak temu zapobiec?

Aby zapewnić odpowiednie zarządzanie łataniami i aktualizacjami, organizacja powinna wdrożyć proces, który obejmuje:

- Usuwanie niepotrzebnych zależności i komponentów - Należy usunąć niepotrzebne zależności, funkcje, komponenty, pliki i dokumentację, które nie są wykorzystywane.
- Ciągła inwentaryzacja wersji komponentów - Regularne sprawdzanie wersji komponentów zarówno po stronie klienta, jak i serwera (np. frameworki, biblioteki) oraz ich zależności, korzystając z narzędzi takich jak versions, OWASP Dependency Check, retire.js itp. Ciągłe monitorowanie źródeł takich jak Common Vulnerability and Exposures (CVE) i National Vulnerability Database (NVD) pod kątem podatności w komponentach.
- Pobieranie Komponentów z Oficjalnych Źródeł - Należy uzyskiwać komponenty wyłącznie z oficjalnych źródeł. Wybierać podpisane pakiety, aby zminimalizować ryzyko dodania zmodyfikowanego lub złośliwego komponentu (**Software and Data Integrity Failures**).
- Monitorowanie komponentów nieobsługiwanych lub nieaktualizowanych - Należy monitorować biblioteki i komponenty, które nie są już utrzymywane lub nie tworzą łańcuchów bezpieczeństwa dla starszych wersji. Jeśli aktualizacja nie jest możliwa, rozważyć wdrożenie wirtualnej łaty, aby monitorować, wykrywać lub chronić przed wykrytym problemem.

Organizacja powinna wdrożyć ciągły plan monitorowania, oceniania statusu bezpieczeństwa i stosowania aktualizacji lub zmian konfiguracji na cały okres życia aplikacji.

2.7 Identification and Authentication Failures

2.7.1 Omówienie Ataku

Identification and Authentication Failures to podatności związane ze schematami uwierzytelniania aplikacji. Takie wady mogą prowadzić do poważnych i szkodliwych naruszeń ochrony danych.

Potwierdzenie tożsamości użytkownika, uwierzytelnienie i zarządzanie sesją są kluczowe dla ochrony przed atakami związanymi z uwierzytelnieniem. Aplikacja może posiadać słabości w uwierzytelnieniu, jeśli:

- Pozwala na automatyczne ataki - Umożliwia ataki typu credential stuffing, gdzie atakujący dysponuje listą prawidłowych nazw użytkowników i haseł.
- Pozwala na ataki brute force lub Inne automatyzowane ataki - Nie zabezpiecza przed atakami metodą prób i błędów lub innymi automatyzowanymi metodami ataku.
- Pozwala na używanie domyślnych, słabych lub powszechnie znanych haseł - Umożliwia używanie słabych lub powszechnie znanych haseł, takich jak "Password1" czy "admin/admin".
- Używa słabych lub nieefektywnych procesów odzyskiwania loginów i haseł - Stosuje niebezpieczne procesy odzyskiwania zapomnianych haseł, takie jak odpowiedzi na pytania oparte na wiedzy ogólnodostępnej.
- Przechowuje hasła w postaci jawnej, szyfrowanej lub słabo zahashowanej - Używa jawnych, szyfrowanych lub słabo zahaszowanych haseł w magazynach danych (**Cryptographic Failures**).
- Brak lub nieefektywne uwierzytelnianie wieloskładnikowe.
- Ujawnia identyfikator sesji w adresie URL.
- Ponowne używanie identyfikatora sesji po pomyślnym logowaniu.
- Nieprawidłowe unieważnianie identyfikatorów sesji - Sesje użytkowników lub tokeny uwierzytelniania (szczególnie w przypadku pojedynczego logowania, SSO) nie są odpowiednio unieważniane podczas wylogowania lub po okresie nieaktywności.

2.7.2 Przykład Ataku

Przykładowy atak wykorzystujący **Identification and Authentication Failures** może zostać przeprowadzony w kilku krokach:

1. **Użycie publicznego komputera przez użytkownika** - Użytkownik loguje się do aplikacji na publicznym komputerze (np. w bibliotece, kawiarni internetowej, uczelni). Po zakończeniu korzystania z aplikacji, użytkownik zamyka tylko kartę przeglądarki, nie wybierając opcji "wyloguj".

2. **Brak wygaśnięcia sesji** - Sesja użytkownika pozostaje aktywna, pomimo zamknięcia karty przeglądarki. Aplikacja nie ma poprawnie skonfigurowanego limitu czasu sesji, co oznacza, że sesja nie wygasa automatycznie po określonym czasie bezczynności.
3. **Atakujący zyskuje dostęp do komputera** - Godzinę później atakujący korzysta z tego samego komputera publicznego. Otwierając przeglądarkę i wracając do strony aplikacji, atakujący odkrywa, że sesja poprzedniego użytkownika jest nadal aktywna.
4. **Nieautoryzowany dostęp** - Atakujący, mając dostęp do sesji uwierzytelnionego użytkownika, może przeglądać, modyfikować lub kopiować wrażliwe dane użytkownika. Może to obejmować dane osobowe, informacje finansowe, korporacyjne lub inne poufne informacje dostępne w aplikacji.
5. **Potencjalne dalsze działania** - Wykorzystując dostęp, atakujący może również podjąć dalsze działania, takie jak zmiana haseł, dokonywanie transakcji w imieniu użytkownika, wysyłanie wiadomości lub innych działań, które mogą prowadzić do kradzieży tożsamości, strat finansowych dla użytkownika lub innych negatywnych konsekwencji.

2.7.3 Implementacja Podatności

W naszym projekcie zaimplementowaliśmy **Identification and Authentication Failures** poprzez brak zabezpieczeń podczas logowania takich jak limit logowań w ograniczonym czasie, lub blokowania automatycznego wpisywania ataków. Na potrzeby prezentacji stworzyliśmy bazę danych z hasłami słownikowymi i umożliwiliśmy jej podejrzenie.

```
id: Jakub - Nazwa użytkownika: Jakub - Hasło: haslo123
id: Marta - Nazwa użytkownika: Marta - Hasło: tajnehaslo
id: Kamil - Nazwa użytkownika: Kamil - Hasło: qwerty
id: Anna - Nazwa użytkownika: Anna - Hasło: 12345
id: Zofia - Nazwa użytkownika: Zofia - Hasło: haslo2023
id: Paweł - Nazwa użytkownika: Paweł - Hasło: admin123
id: Magdalena - Nazwa użytkownika: Magdalena - Hasło: password
id: Tomasz - Nazwa użytkownika: Tomasz - Hasło: 123abc
id: Karolina - Nazwa użytkownika: Karolina - Hasło: solniczka
id: Michał - Nazwa użytkownika: Michał - Hasło: michal1
```

Rysunek 11: Baza z hasłami słownikowymi

Bardzo łatwo można zgadnąć hasła słownikowe, bądź zalogować się na konto z takim hasłem wykorzystując zautomatyzowane logowania, nasz projekt celowo nie uwzględnia takich blokad. Mając dostęp do przykładowej bazy możemy wybrać dowolnego użytkownika i się zalogować jego danymi, po poprawnym logowaniu wyświetli nam się monit informujący nas o udanym logowaniu.

Logowanie udane!

Rysunek 12: Monit po poprawnym logowaniu

2.7.4 Jak temu zapobiec?

Aby zwiększyć bezpieczeństwo uwierzytelniania w aplikacjach, zaleca się wdrożenie następujących praktyk:

- Wdrażanie uwierzytelniania wieloskładnikowego (MFA) - Należy zaimplementować MFA, aby zapobiec atakom typu credential stuffing, atakom typu brute force i wykorzystywaniu skradzionych danych logowania.
- Unikanie domyślnych danych logowania - Unikać wdrażania i dostarczania oprogramowania z domyślnymi danymi logowania, szczególnie dla użytkowników administracyjnych.
- Sprawdzanie słabych haseł - Zaimplementować kontrolę słabych haseł, np. testując nowe lub zmieniane hasła pod względem listy 10000 najpopularniejszych haseł
- Polityka haseł zgodna z wytycznymi NIST 800-63b - Dostosować długość, złożoność i politykę rotacji haseł do wytycznych Instytutu Standardów i Technologii Narodowej (NIST)
- Zabezpieczenie ścieżek rejestracji i odzyskiwania poświadczeń - Zabezpieczyć ścieżki rejestracji, odzyskiwania poświadczeń i API przed atakami polegającymi na wyliczaniu kont, stosując te same komunikaty dla wszystkich wyników.
- Ograniczenie nieudanych prób logowania - Należy ograniczyć lub wprowadzić czasowe opóźnianie nieudanych prób logowania uważając przy tym, aby nie wywołać DoS. Rejestrować wszystkie niepowodzenia w logach i alarmować administratorów, gdy zostaną wykryte ataki typu credential stuffing, brute force lub inne ataki.

2.8 Software and Data Integrity Failures

2.8.1 Omówienie Ataku

Software and Data Integrity Failures odnosi się do kodu i infrastruktury, które nie chronią przed naruszeniem integralności. Przykładem może być sytuacja, w której aplikacja polega na wtyczkach, bibliotekach lub modułach pochodzących ze źródeł, repozytoriów lub sieci dostarczania treści (CDN), które nie są zaufane. Niezabezpieczony proces CI/CD (Continuous Integration/Continuous Deployment) może wprowadzać ryzyko nieautoryzowanego dostępu, złośliwego kodu lub kompromitacji systemu. Ponadto wiele aplikacji obecnie zawiera funkcjonalność automatycznej aktualizacji, gdzie aktualizacje są pobierane bez wystarczającej weryfikacji integralności i stosowane do wcześniej zaufanej aplikacji. Atakujący mogą potencjalnie przesłać własne aktualizacje, które będą dystrybuowane i uruchamiane na wszystkich instalacjach. Kolejnym przykładem jest sytuacja, w której obiekty lub dane są kodowane lub serializowane do struktury, którą atakujący może zobaczyć i modyfikować, co czyni je podatnymi na niebezpieczną deserializację.

2.8.2 Przykład Ataku

Przykładowy atak wykorzystujący **Software and Data Integrity Failures** może zostać przeprowadzony w kilku krokach:

1. **Wykrycie braku weryfikacji podpisu firmware** - Atakujący zdaje sobie sprawę, że wiele domowych routerów i innych urządzeń nie wykonuje weryfikacji podpisu cyfrowego podczas aktualizacji firmware. To oznacza, że urządzenia te będą akceptować i instalować każde oprogramowanie układowe, bez sprawdzania jego autentyczności.
2. **Przygotowanie sfałszowanego firmware** - Atakujący opracowuje sfałszowaną wersję firmware, która może zawierać złośliwy kod, np. backdoory umożliwiające zdalny dostęp, narzędzia do monitorowania ruchu sieciowego, złośliwe oprogramowanie typu ransomware, czy inne składniki umożliwiające wykorzystanie urządzenia do ataków typu botnet.
3. **Rozpowszechnianie sfałszowanego firmware** - Atakujący może następnie rozpowszechnić sfałszowane firmware, wykorzystując różne metody, np. phishing, fałszywe strony internetowe oferujące aktualizacje, ataki typu man-in-the-middle w sieciach niezabezpieczonych, lub nawet fizyczną manipulację urządzeń.
4. **Instalacja sfałszowanego firmware na urządzeniach** - Użytkownicy, nieświadomi zagrożenia, instalują sfałszowane oprogramowanie na swoich urządzeniach. Ponieważ urządzenia nie weryfikują autentyczności oprogramowania, sfałszowane firmware zostaje zainstalowane bez problemów.
5. **Wykorzystanie zainstalowanego złośliwego oprogramowania** - Po zainstalowaniu, złośliwe oprogramowanie może być wykorzystane do różnych celów, w zależności od intencji atakującego. Może to obejmować kradzież danych, monitorowanie aktywności sieciowej, przekształcanie urządzenia w część botnetu do przeprowadzania ataków DDoS, czy zablokowanie dostępu do urządzenia i żądanie okupu (ransomware).

2.8.3 Jak temu zapobiec?

Aby zapewnić integralność oprogramowania i danych, warto wdrożyć następujące praktyki:

- Weryfikacja za pomocą podpisów cyfrowych - Należy używać podpisów cyfrowych lub podobnych mechanizmów do weryfikacji czy oprogramowanie lub dane pochodzą z oczekiwanego źródła i nie zostały zmienione.
- Zaufane repozytoria dla bibliotek i zależności - Należy upewnić się, że biblioteki i zależności zaimplementowane w aplikacji lub serwisie korzystają z zaufanych i zweryfikowanych repozytoriów
- Proces przeglądu kodu i zmian konfiguracyjnych - Zapewnić proces przeglądu kodu i zmian konfiguracyjnych, aby zminimalizować ryzyko wprowadzenia złośliwego kodu lub konfiguracji do łańcucha oprogramowania
- Odpowiednia segregacja i kontrola dostępu w CI/CD - Upewnić się, że proces CI/CD (Continuous Integration/Continuous Deployment) ma odpowiednią segregację, konfigurację i kontrolę dostępu, aby zapewnić integralność kodu przepływającego przez procesy kompilacji i wdrażania.
- Ochrona nieopisanych lub niezaszyfrowanych danych serializowanych - Zapewnić, aby niepodpisane lub niezaszyfrowane serializowane dane nie były wysyłane do niezaufanych klientów bez pewnej formy kontroli integralności lub cyfrowego podpisu, aby wykryć manipulację lub powtórne użycie serializowanych danych.

2.9 Security Logging and Monitoring Failures

2.9.1 Omówienie Ataku

Security Logging and Monitoring Failures to podatności bezpieczeństwa, które mogą wystąpić gdy system bądź aplikacja nie rejestruje lub nie monitoruje odpowiednio zdarzeń związanych z bezpieczeństwem. Może to umożliwić atakującemu uzyskanie nieautoryzowanego dostępu do systemów i danych bez wykrycia.

Bez odpowiedniego logowania i monitorowania, wykrycie naruszeń jest niemożliwe. Niewystarczające logowanie, wykrywanie, monitorowanie i aktywna reakcja mają miejsce, gdy:

- Brak zapisywania do logów wydarzeń wykorzystywanych do audytów - Zdarzenia takie jak logowania, nieudane próby logowania, transakcje o wysokiej wartości nie są rejestrowane.
- Niewystarczające komunikaty logów - Ostrzeżenia i błędy nie generują żadnych, niewystarczających lub niejasnych komunikatów w logach.
- Brak monitorowania logów aplikacji i API - Logi aplikacji i API nie są monitorowane pod kątem podejrzanej aktywności.
- Lokalne przechowywanie logów - Logi są przechowywane tylko lokalnie, bez kopii zdalnych lub backupów.
- Niewystarczające progi alarmowe i procesy eskalacji reakcji - Brak odpowiednich progów alarmowych oraz skutecznych procesów eskalacji reakcji.
- Testy penetracyjne i skany nie wywołują alarmów - Testy penetracyjne i skany narzędziami do dynamicznego testowania bezpieczeństwa aplikacji (DAST), takimi jak OWASP ZAP, nie wyzwalają alarmów.
- Brak wykrywania, eskalacji lub alarmowania o atakach - Aplikacja nie jest w stanie wykryć, eskalować lub alarmować o aktywnych atakach w czasie rzeczywistym.
- Podatność na wyciek informacji - Istnieje ryzyko wycieku informacji poprzez uczynienie zdarzeń logowania i alarmowania widocznymi dla użytkownika lub atakującego (**Broken Access Control**)

2.9.2 Przykład Ataku

Przykładowy atak wykorzystujący **Security Logging and Monitoring Failures** może zostać przeprowadzony w kilku krokach:

1. **Wykrycie braku monitorowania i rejestrowania** - Atakujący identyfikuje cel - organizację, która nie prowadzi skutecznego monitorowania i rejestrowania aktywności w swoich systemach informatycznych. Może to zostać odkryte na różne sposoby, np. poprzez testy penetracyjne, analizę publicznie dostępnych informacji, lub wskazówki od byłego pracownika.

2. **Przygotowanie ataku** - Atakujący opracowuje strategię ataku, wykorzystując fakt, że działania w systemie nie są monitorowane ani rejestrowane. To oznacza, że nawet w przypadku wykrycia nieautoryzowanego dostępu, trudno będzie ustalić, co dokładnie zostało zrobione i kiedy to miało miejsce.
3. **Wykonanie ataku** - Atakujący infiltruje system, wykorzystując różne metody, takie jak phishing, wykorzystanie znanych podatności, czy ataki siłowe na słabe hasła. Bez odpowiedniego monitorowania, te działania mogą pozostać niezauważone przez dłuższy czas.
4. **Działania w systemie** - Po uzyskaniu dostępu, atakujący przeprowadza różnorodne działania, takie jak kradzież danych, instalacja złośliwego oprogramowania, tworzenie backdoorów dla łatwiejszego dostępu w przyszłości, czy sabotowanie systemów.
5. **Utrzymanie dostępu i niewykrywalności** - Atakujący może utrzymywać dostęp do systemu przez długi czas, wykorzystując brak monitorowania. Może regularnie wydobywać dane lub wykonywać inne szkodliwe działania, pozostając niewykrytym.

2.9.3 Jak temu zapobiec?

Deweloperzy powinni wdrożyć niektóre lub wszystkie z poniższych kontroli, w zależności od ryzyka związanego z aplikacją:

- Rejestrowanie Niepowodzeń logowania i uwierzytelniania - Należy upewnić się, że wszystkie nieudane próby logowania, kontroli dostępu i walidacji danych po stronie serwera są rejestrowane z wystarczającym kontekstem użytkownika, aby zidentyfikować podejrzane lub złośliwe konta, i są przechowywane wystarczająco długo, aby umożliwić opóźnioną analizę śledczą.
- Generowanie logów w przystępnym formacie - Zapewnić, aby logi były generowane w formacie, który jest łatwy do przeanalizowania.
- Poprawne kodowanie logów - Upewnić się, że dane logów są kodowane poprawnie, aby zapobiec atakom typu **Injection** lub atakom na systemy logowania lub monitorowania.
- Skuteczne monitorowanie i alarmowanie przez zespoły DevSecOps - Ustanowić efektywne monitorowanie i alarmowanie, tak aby podejrzane działania były szybko wykrywane i na nie reagowano.
- Plan reagowania na incydenty i odtwarzania - Ustanowić lub zaadaptować plan reagowania na incydenty i odtwarzania, taki jak NIST 800-61r2 lub nowszy.
- Korzystanie z frameworków ochrony aplikacji i oprogramowania do korelacji logów - Wykorzystanie komercyjnych i opensourcowych frameworków ochrony aplikacji, takich jak OWASP ModSecurity Core Rule Set, oraz oprogramowania do korelacji logów, takiego jak Elasticsearch, Logstash, które oferują niestandardowe pulpity i systemy alarmowania.

2.10 Server-Side Request Forgery (SSRF)

2.10.1 Omówienie Ataku

Luki w SSRF pojawiają się, gdy aplikacja internetowa pobiera zasób zdalny bez weryfikacji podanego przez użytkownika adresu URL. Pozwala to atakującemu zmusić aplikację do wysłania sformułowanego żądania do nieoczekiwanego miejsca docelowego, nawet jeśli jest chroniona przez firewall, VPN lub inną listę kontroli dostępu do sieci.

Wraz z rozwojem nowoczesnych aplikacji internetowych, które oferują użytkownikom wygodne funkcje, pobieranie adresu URL staje się powszechnością. W rezultacie SSRF jest coraz powszechniejszym atakiem. Dodatkowo, powaga ataków SSRF wzrasta ze względu na usługi chmurowe i złożoność architektur.

2.10.2 Przykład Ataku

Przykładowy atak wykorzystujący **Server-Side Request Forgery** może zostać przeprowadzony w kilku krokach:

1. **Wykrycie podatności na SSRF** - Atakujący identyfikuje aplikację webową, która jest podatna na ataki SSRF. Podatność ta umożliwia atakującemu wysyłanie zapytań z serwera aplikacji do innych systemów, które są z nim połączone.
2. **Analiza architektury sieciowej** - Atakujący zdaje sobie sprawę, że architektura sieciowa organizacji nie jest segmentowana, co oznacza, że wewnętrzne serwery są dostępne z poziomu serwera aplikacji. Brak segmentacji umożliwia atakującemu swobodne skanowanie sieci wewnętrznej.
3. **Przygotowanie i wysyłanie żądań SSRF** - Atakujący tworzy i wysyła żądania SSRF z serwera aplikacji do różnych adresów IP i portów w sieci wewnętrznej. Żądania te mają na celu ustalenie, które porty są otwarte na wewnętrznych serwerach.
4. **Analiza wyników skanowania** - Na podstawie wyników żądań SSRF, atakujący jest w stanie zmapować sieć wewnętrzną organizacji. Analizuje, które porty na wewnętrznych serwerach są otwarte, co może ujawnić informacje o działających usługach i potencjalnych punktach wejścia dla dalszych ataków.
5. **Wykorzystanie znalezionych słabości** - Korzystając ze zdobytych informacji, atakujący może planować i przeprowadzać dalsze ataki na wewnętrzne serwery, wykorzystując otwarte porty i znajdujące się tam usługi. Może to obejmować ataki typu **Injection**, **Broken Access Control**, czy nawet przejęcie kontroli nad wewnętrznymi systemami.

2.10.3 Implementacja Podatności

W naszym projekcie zaimplementowaliśmy **Server-Side Request Forgery (SSRF)** poprzez brak weryfikacji adresu URL podawanego przez użytkownika.

OWASP Top 10 - Podatność 10: Server-Side Request Forgery (SSRF)

Omówienie Ataku

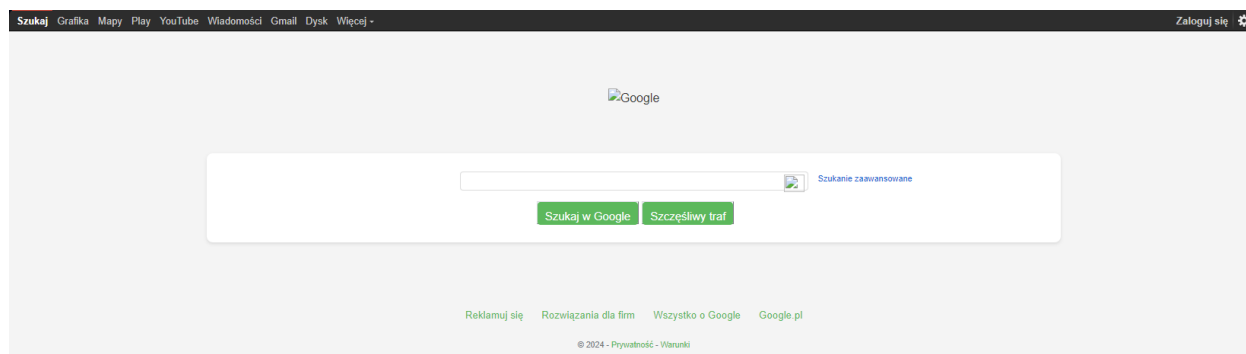
Luki w SSRF pojawiają się, gdy aplikacja internetowa pobiera zasób zdalny bez weryfikacji podanego przez użytkownika adresu URL. Pozwala to atakującemu zmusić aplikację do wysłania spreparowanego żądania do nieoczekiwanego miejsca docelowego, nawet jeśli jest chroniona przez firewall, VPN lub inną listę kontroli dostępu do sieci. Wraz z rozwojem nowoczesnych aplikacji internetowych, które oferują użytkownikom wygodne funkcje, pobieranie adresu URL staje się powszechnością. W rezultacie SSRF jest coraz powszechniejszym atakiem. Dodatkowo, powaga ataków SSRF wzrasta ze względu na usługi chmurowe i złożoność architektur.

URL:

Fetch content

Rysunek 13: Domyślny wygląd strony

Podstrona normalnie wygląda tak jak przedstawiono powyżej. Jeśli użytkownik w miejscu na wpisanie danych poda adres innej strony np. `https://www.google.com` to uzyska dostęp do tej strony.



Rysunek 14: Nielegalny dostęp do innej strony

Użytkownik może w ten sposób dostać dostęp do innych podstron serwisu, do których normalnie nie miałby dostępu lub łączyć się ze stronami zewnętrznymi.

2.10.4 Jak temu zapobiec?

Aby zapobiegać usterkom SSRF, deweloperzy mogą wdrożyć niektóre lub wszystkie z poniższych kontroli w ramach strategii obrony wielopoziomowej:

Z poziomu sieci:

- Oddzielić funkcjonalność dostępu do zasobów zdalnych w osobnych sieciach, aby zmniejszyć wpływ SSRF.

- Wprowadzić polityki firewalla lub reguły kontroli dostępu do sieci oparte na zasadzie „deny by default”, blokując cały ruch intranetowy, poza tym niezbędnym.

Z poziomu aplikacji:

- Oczyszczać i weryfikować wszystkie dane wejściowe dostarczone przez klienta.
- Wymuszać schemat URL, port i miejsce docelowe za pomocą pozytywnej listy dozwolonych.
- Wyłączyć przekierowania HTTP.
- Być świadomym spójności URL, aby unikać ataków takich jak DNS rebinding i warunki wyścigu „time of check, time of use” (TOCTOU).
- Nie wysyłać surowych odpowiedzi do klientów.

Dodatkowo należałoby rozważyć szyfrowanie sieci (np. VPN) na niezależnych systemach, aby zapewnić wysoki poziom ochrony.