Podatność w obsłudze sesyjności

Opis podatności:

W badanej aplikacji zidentyfikowano złą implementację obsługi sesyjności.

Przypadek testowy:

Podszycie się pod sesję innego użytkownika

Identyfikator sesji jest przekazywany w kodzie źródłowym aktualnie wyświetlanej strony oraz jako parametr POST wysyłany do serwera. Przez to, że jest to połączenie HTTP, możliwy jest atak typu Man In The Middle, który umożliwiłby atakującemu przechwycenie tego identyfikatora.

Skutki wykorzystania podatności:

W wyniku wykorzystania podatności atakujący może uzyskać dostęp do sesji, a co za tym idzie, prywatnych wiadomości oraz uprawnień związanych z kontem.

Wpływ na ryzyko:

Duże.

Sposób usunięcia:

Aplikacja powinna zostać przeniesiona na szyfrowane połączenie HTTPS, co uniemożliwiłoby atakującemu przechwycenie tokenu naszej sesji. Dodatkowo, w każdej ramce powinna zawarta zostać flaga wymuszająca szyfrowane połączenie.

Brak kontroli uprawnień dostępu do strony

Opis podatności:

W badanej aplikacji nie ma żadnego mechanizmu weryfikującego uprawnienia dostępu do strony.

Przypadek testowy:

Dostęp do dowolnej strony

Strona żądana od serwera jest podawana jako parametr POST. Manipulując zapytanie wysyłane do serwera, atakujący może je zmienić.

Skutki wykorzystania podatności:

W wyniku wykorzystania podatności, atakujący może uzyskać dostęp do dowolnej strony dostępnej w naszej aplikacji, takiej jak panel administratorski, mając nawet najniższe uprawnienia.

Wpływ na ryzyko:

Duże.

Sposób usunięcia:

W aplikacji powinno zostać wprowadzona identyfikacja poziomu uprawnień użytkownika. Mogłaby zostać wprowadzona poprzez użycie unikalnego identyfikatora sesji, który nie zmieniał by się w trakcie nawigacji po aplikacji. Użytkownik mógłby też za każdym razem legitymować się hasłem i nazwą konta wysyłanym jako hash do serwera. Byłby on porównywany z listą istniejących kont, i sprawdzany pod kątem uprawnień.

Podatność typu SQL Injection

Opis podatności:

W badanej aplikacji znaleziono miejsca posiadające podatność typu SQL Injection.

Przypadek testowy:

1. **Nieautoryzowany dostęp do wiadomości**

   W panelu z wiadomościami, maile ściągane są z serwera poprzez zapytanie o rząd w bazie danych. Numer rzędu/wiadomości przekazywany jest w URLu.

   Skutki wykorzystania podatności

   Napastnik może bez autoryzacji odczytać wiadomości nienależące do niego, poprzez wklejenie dowolnego numeru wiadomości do zapytania wysłanego do serwera.

   Wpływ na ryzyko:

   Zależne od treści wiadomości. Zdefraudowane w ten sposób dane, można wykorzystać do przeprowadzenia ataku phishingowego, co znacząco zwiększa negatywne skutki wykorzystania tej podatności.

   Sposób usunięcia:

   Tak jak w przypadku podatności związanej z brakiem kontroli uprawnień, konieczne jest sprawdzenie poziomu dostępu przed wyświetleniem żądanej wiadomości.

2. Zdefraudowanie danych

W panelu administratora, podczas wysyłania zapytania o historię logowań, możemy je zmodyfikować wklejając swój kod. Poniżej przedstawione zostało przykładowe zapytanie przy wykorzystaniu programu sqlmap.

sqlmap -u 'http://rekrutacja.securing.pl/?fromDate=2017/06/21&toDate=2017/06/21*' --data='task=admin&SessionState=8e57b395ad2df17fa658236e8a2f8ba81802d341%40eJxNjUsOgzAMRO%2FiA6AkDvmY06TB0EiFViRdVdy9BhbtyqP3NONElj6VPEHeODUeYSikbQxKYYz9UCkStLJwbWl5%2FaR1AUUagiK0CiUIvjMYO2NMp407qMB35Q2GdL3pCR7PuayHk5zG5cqOYCvzvdVzigCPi1Iu4z%2BRyvpcM5%2FTSqBCH7Bnn9g6DMEbvE3JK50dJp1D4DwZb2Vj379r2UAi' --headers='Host: rekrutacja.securing.pl\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\nAccept-Language: en-US,en;q=0.5\nReferer: http://rekrutacja.securing.pl/?698\nContent-Type: application/x-www-form-urlencoded\nAuthorization: Basic dGNobWllbGVja2k6aVNoZWk2dXhlaTFh' -o*

Skutki wykorzystania podatności:

Tworząc specjalnie przygotowane polecenie do bazy danych, napastnik może uzyskać dostęp do wszystkich informacji zawartych w bazie danych.

Wpływ na ryzyko:

Wysokie.

Sposób usunięcia:

Zapytania do bazy danych nie powinny być tworzone przez sklejanie statycznej części zapytania z wartością parametru pobieranego z przeglądarki, lecz przez wywołania procedur składowanych w bazie lub użycie Dynamic SQL. Ponadto zalecana jest szczegółowa walidacja wszystkich danych trafiających z zewnątrz do aplikacji, pod względem typu, długości, formatu i zakresu.

Dodatkowe uwagi

- W ramce do serwera, przesyłany jest również login i hasło do profilu testera aplikacji, w formacie base64. Jest to w pełni odwracalny sposób kodowania, a że połączenie jest HTTP, czyli nieszyfrowane, dane te są proste do przechwycenia. Rozwiązaniem może być przejście na HTTPS, lub wysyłanie tej informacji jako sumę kontrolną, która porównywana będzie z tą trzymaną na serwerze.

- Token generowany przez serwer, i służący do wykonania następnej akcji na danej podstronie, jest generowany w formacie <hash_sha-1@token>, oprócz możliwości jego wielokrotnego użycia, hash jest też po prostu suma kontrolną tokenu za znakiem '@'. Jeśli atakujący znalazłby sposób generowania tej części tokenu, mógłby tworzyć zapytania generowane w 100% przez siebie i wykonywać dowolne akcje na serwerze, nie potrzebując wykorzystywać żadnych podatności wymienionych wyżej w raporcie.

- W przypadku drugiego miejsca w którym występuje SQL Injection, chciałbym przetestować jeszcze scenariusz w którym do bazy danych, wstrzeliłbym trwały XSS, włączający się w trakcie przeglądania wiadomości lub w panelu administratora. Niestety wszystkie moje próby wylistowania zawartości bazy danych, kończyły się crashem aplikacji, co uniemożliwiło mi narazie poznanie jej struktury.

Planowanie, wykonanie i raportowanie

- Zaczynając testować aplikację, na początku przejrzałem wszystkie dostępne podstrony, zaznaczając sobie potencjalnie podatne miejsca (pola z danymi do wpisania, zapytania do bazy danych). Następnie używając proxy (Burp) sprawdziłem zawartość ramek i tam również zaznaczyłem możliwe dziury. Potem zmieniałem treści ramek i wprowadzałem nieoczekiwane dane wejściowe. Doprowadzając w ten sposób do crashów identyfikowałem podatności. Dodatkowo w miejscach gdzie wysyłane były zapytania do bazy danych użyłem programu sqlmap. Podczas pisania raportu, opierałem się na tym dostanym w mailu. Po napisaniu pierwszego szkicu, dodawałem dodatkowe informacje zdobyte w trakcie pentestu.