010011000

1000110101

0110001101

ZABEZPIECZANIE APLIKACJI INTERNETOWYCH W .NET - PODSTAWY

Jak nie dać się podejść hackerom

Adam Jachocki

Nerdolando
ul. Rybna 18/18
91-051 Łódź
e-mail: books@nerdolando.pl

ISBN: 978-83-947977-0-6

Wydanie 1 poprawione

Spis treści

O autorze

Cześć, nazywam się Adam Jachocki. Programuję od siódmego roku życia (1991 r), odkąd dorwałem się do książki „Atari BASIC dla dzieci”. Już wtedy wiedziałem, co będę w życiu robił.

Zawodowo zajmuję się programowaniem od 2005 roku. Przeszedłem przez wiele technologii, kilka języków i różne podejścia.

Jak każdy szanujący się programista, wciąż się rozwijam i uczę. Backend to jest to, co kręci mnie najbardziej.

Od jakiegoś czasu tworzę również bloga https://www.masterbranch.pl, na którym przedstawiam wiedzę głównie z zakresu .NET. Znajdziesz tam artykuły o wbudowanych mechanizmach, jak również bardziej ogólne porady (np. jak tworzyć aplikacje wielojęzyczne). Zapraszam.

Tworzyłem lub współtworzyłem systemy magazynowo-sprzedażowo-lojalnościowe, systemy ERP, RCP i kontroli dostępu. Współtworzyłem też oprogramowanie do obsługi targów, m.in. Camerimage.

Najciekawszym projektem, w którym brałem udział do tej pory był wg mnie multibiometryczny system identyfikacji osób do przeciwdziałania zagrożeniom terrorystycznym, który współtworzyłem razem z Wojskową Akademią Techniczną.

Wstęp

Dla kogo jest ta książka

Ta książka jest kierowana dla osób programujących w .NET Core i .NET. Przeznaczona głównie dla juniorów i „midów”. Ale wierzę, że niejeden senior też znajdzie w niej coś ciekawego.

Książka zdecydowanie NIE JEST dla osób zajmujących się bezpieczeństwem aplikacji, administratorów ani hakerów.

Przykłady

Tworząc tę pracę, napisałem kilka projektów w .NET6, które ilustrują niektóre ataki i pokazują jak się przed nimi uchronić. Powinieneś pobrać ten kod i przeglądać go razem z książką. Solucja została stworzona w Visual Studio 2022. Kody znajdują się na GitHubie: https://github.com/AdamJachocki/Attacks

Słowniczek

Klient – aplikacja, która wysyła żądanie do serwera. To może być np. przeglądarka internetowa.

Żądanie – informacja wysyłana przez klienta do serwera, w celu uzyskania odpowiedzi lub jakiegoś zasobu – np. pliku lub kodu HTML.

Odpowiedź – informacja wysyłana z serwera do klienta na podstawie otrzymanego żądania.

Frontend – część aplikacji internetowej widoczna dla użytkownika.

Backend – część aplikacji internetowej niewidoczna dla użytkownika. Dostępna jedynie na serwerze.

Dziura – inaczej podatność – kod programu (lub błędna konfiguracja), który sprawia, że aplikacja jest podatna na atak hakerski.

Ciasteczko, Cookie – dane przechowywane na komputerze użytkownika. Są wysyłane z żądaniem do serwera. Serwer też może wysłać ciastko do przeglądarki – wtedy przeglądarka powinna je zapisać. Ciastka są wykorzystywane m.in. w mechanizmie zapamiętywania zalogowanego użytkownika.

RESTful API – rodzaj komunikacji pomiędzy klientem a serwerem. Klient za pomocą odpowiednich żądań może działać na zasobach serwera (pobierać je, dodawać, usuwać). W tym kontekście klient jest rzadziej przeglądarką internetową, a zdecydowanie częściej specjalnym oprogramowaniem (zwanym również *api consumer*).

Hosting – usługa polegająca na utrzymywaniu strony na serwerze.

Hostingodawca – firma oferująca usługi hostingowe.

WebApplication – aplikacja internetowa stworzona w VisualStudio z szablonu RazorMVC lub RazorPages. Taka aplikacja domyślnie składa się ze stron prezentowanych użytkownikowi w przeglądarce. Może być to też aplikacja typu SPA (Single Page Application).

WebApi – aplikacja internetowa stworzona w VisualStudio z szablonu WebApi. Taka aplikacja domyślnie jest RESTowym API.

Potok – middleware pipeline – kolejność obsługi żądania przez komponenty. Potok konfiguruje się domyślnie w metodzie *Configure* klasy *Startup* w .NetCore.

ORM – Object Relational Mapper – to mechanizm, który mapuje rekordy bazodanowe na modele w Twojej aplikacji i na odwrót. Przykładami ORMów są: Entity Framework (Core), nHibernate, Dapper. Są to darmowe biblioteki, z którymi warto się zapoznać tworząc aplikacje bazodanowe.

Motywacja

Często jest tak, że to jakiś szczegół decyduje o tym, czy aplikacja jest podatna na pewne ataki, czy nie. Najczęściej błędy popełniają juniorzy, ale u seniorów też zdarzyło mi się pewne rzeczy wychwycić podczas *code review*.

Sam kiedyś szukałem przystępnej wiedzy na temat podstawowego zabezpieczania aplikacji. Niestety nie udało mi się znaleźć żadnego dobrego opracowania ani konkretnych przykładów z opisami.

Gdy wchodzi się w świat aplikacji internetowych, na każdym rogu można spodziewać się ataku. I tu rodzą się pytania – czy moja aplikacja jest zabezpieczona? Czy rozumiem zagrożenia? Czy piszę bezpieczny kod? Czy tworzę głupie dziury?

Ta książka oczywiście nie wyczerpuje tematu, bo to studnia bez dna. Natomiast odpowiada na podstawowe pytania. Sam .NET ma sporo zabezpieczeń wbudowanych w siebie. Do tego stopnia, że musiałem pominąć kilka ataków, ponieważ w tej wersji (.NET 6) nie udało się ich przeprowadzić (np. *Cross-Site-Scripting*). Niemniej jednak .NET6 nie uchroni Cię przed wszystkim. Możesz sam wpakować się w dość poważne kłopoty.

Mam nadzieję, że ta książka poszerzy wiedzę w temacie i sprawi, że moje jak i Twoje aplikacje będą z założenia bardziej bezpieczne niż kiedykolwiek.

Po lekturze proponuję zrobić *code review* wybranych aplikacji, które utrzymujesz.

Open Redirect

To kolejny ciekawy atak. Jeśli strona jest napisana niepoprawnie, to pozwala na wykradanie danych logowania w bardzo prosty sposób. A wszystko przez próbę ułatwienia życia.

Przykładowe projekty:

- 8-OpenRedirect-Fake
- 8-OpenRedirect-Genuine

Geneza

Wyobraź sobie, że trafiłeś na jakiś bardzo ciekawy artykuł w wyszukiwarce. Klikasz i widzisz komunikat mówiący o tym, że musisz się zalogować, żeby artykuł przeczytać.

Logujesz się do serwisu, bo akurat masz tam konto. I teraz mogą zadziać się dwie rzeczy:

- Zostaniesz przekierowany na stronę główną, co spowoduje zapewne atak szału, ponieważ będziesz musiał znowu wyszukać ten artykuł w czeluściach Internetu.
- Zostaniesz przekierowany do artykułu – super, tego chcemy.

Ja zdecydowanie wolę opcję drugą. Jest wygodniejsza. Ale daje podatność.

Jak działa open redirect?

Żeby to zrozumieć, zastanówmy się najpierw jak działa przekierowanie po logowaniu.

Załóżmy, że wchodzisz na stronę *www.example.com/super-artykul*. Strona *example.com* widzi, że nie jesteś zalogowany, więc automatycznie przekierowuje Cię na stronę logowania, dodając do adresu parametr, np.:

```
www.example.com/login?return_url=/super-artykul
```

Dzięki temu serwis wie, że po poprawnym zalogowaniu ma Cię przekierować na stronę */super-artykul*.

A teraz przemyślmy taki scenariusz. Otrzymujesz maila, który wygląda zupełnie jak wiadomość z banku o treści:

Dzień dobry,

w związku z aktualizacją naszej polityki prywatności (albo super tajnego prawa bankowego)*, prosimy Cię o zweryfikowanie i ewentualną aktualizację Twoich danych osobowych. Zaloguj się i sprawdź poprawność danych: https://www.prawdziwy-bank.pl?return_url=https://prawdziwy-bankk.pl*

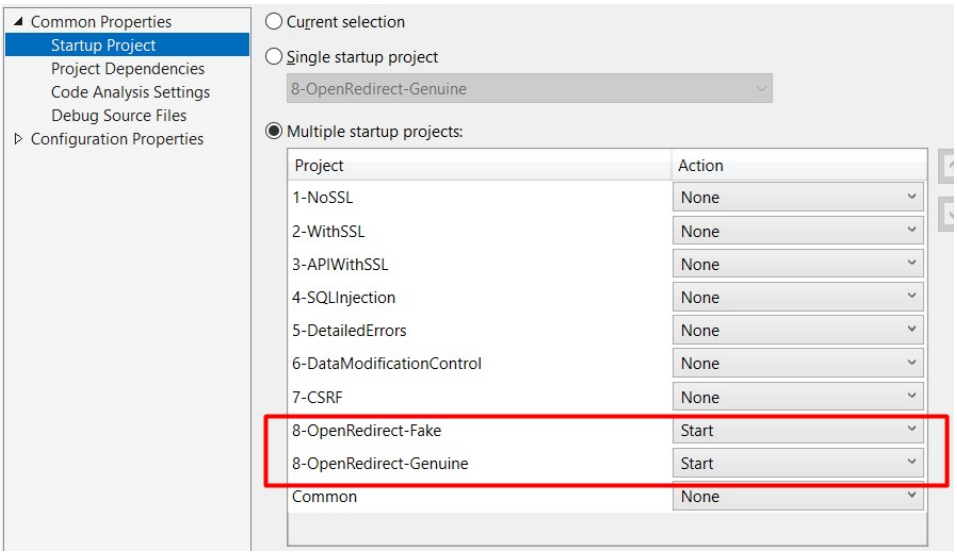
Zwróć uwagę na adres, na który ma być przekierowanie po logowaniu: *prawdziwy-bankk ->* (podwójne 'k' na końcu) wygląda jak domena banku, ale nią nie jest.

Logujesz się. Teraz następuje przekierowanie do złej strony (zamiast do strony serwisu), która wygląda dokładnie tak samo jak strona prawdziwego serwisu i widzisz komunikat o niepoprawnym logowaniu. Logujesz się drugi raz – wracasz na stronę prawdziwego serwisu.

O tym ataku możesz się nigdy nie dowiedzieć. Możesz się dowiedzieć też dopiero po długim czasie. A atakujący posiada dane do Twojego konta. Jak to zrobić?

Do ataku!

Tym razem musisz uruchomić dwa projekty jednocześnie. Aby to zrobić, w Visual Studio w okienku *SolutionExplorer*, kliknij prawym klawiszem myszy na **solucję** i z menu wybierz *Properties*. W okienku, które się pojawi, wybierz zakładkę *CommonProperties -> Startup Project*. Teraz możesz wybrać dwa projekty startowe – to będzie *8-OpenRedirect-Fake* i *8-OpenRedirect-Genuine*. Ustaw ich akcję na Start:



Projekty są już skonfigurowane w odpowiedni sposób.

Strona przykładowego banku otworzy się pod adresem: *https://localhost:8000*, natomiast strona kradnąca dane pod: *https://localhost:9000*. Potraktuj to jako dwa zupełnie inne adresy, jak np. *https://prawdziwy-bank.pl* i *https://prawdziwy-bankk.pl* (podwójne „k" na końcu)

Teraz po prostu wciśnij F5, żeby uruchomić te aplikacje (aplikacja oszukanego banku uruchomi się w tle, żeby wszystko odbyło się w jednym oknie przeglądarki).

Zobaczysz przykładową stronę jakiegoś banku z formularzem do logowania. Zwróć uwagę na to, co jest w pasku adresu przeglądarki. Zobaczysz tam adres: *https://localhost:8000/?return_url=https%3A%2F%2Flocalhost%3A9000*

Adres zawiera parametr *return_url* – to jest url, na który zostaniesz przekierowany po poprawnym logowaniu. Ten adres jest symulacją, jakbyś kliknął na oszukany link.

Jak widzisz, w tym przypadku mamy pełny zewnętrzny adres http. Jeśli zdekodujesz sobie znaki specjalne, zobaczysz jako wartość tego parametru: *https://localhost:9000*

To jest adres drugiej strony, którą uruchomiłeś – strony kradnącej dane.

Spójrzmy teraz w kod logowania na stronie banku:

```
public async Task<IActionResult> OnPost()
{
    if (!ModelState.IsValid)
        return Page();

    //symulacja logowania
    bool loginResult = await LoginUser(LoginData);
    if (loginResult)
        return RedirectUserAfterValidLogin(ReturnUrl);
    else
        return RedirectToPage();
}
```

Logujemy użytkownika. Następnie, po prawidłowym logowaniu (co w przypadku tego testu zawsze jest prawdą), następuje przekierowanie na stronę wskazaną w parametrze *ReturnUrl*:

```
IActionResult RedirectUserAfterValidLogin(string returnUrl)
{
    if (string.IsNullOrEmpty(returnUrl))
        return RedirectToPage("/Index");
    else
        return Redirect(returnUrl);
}
```

Wpisz teraz do formularza jakikolwiek login i hasło i wciśnij przycisk zaloguj.

Logowanie powiodło się, zostałeś przekierowany do strony kradnącej hasła. A jak ta strona wygląda? Dokładnie tak, jak strona *Twojego Banku*.

Widzisz komunikat o niepoprawnym logowaniu, więc automatycznie logujesz się jeszcze raz – tyle że tym razem przekazujesz swoje dane logowania atakującemu (projekt *8-OpenRedirect-Fake*) – bo jesteś w formularzu na stronie hakera:

```
public async Task<IActionResult> OnPost()
```

```
{
    Console.WriteLine("User: " + LoginData.UserName);
    Console.WriteLine("Hasło: " + LoginData.Password);

    return Redirect("https://localhost:8000");
}
```

Atakujący zapisuje sobie Twoje dane logowania i przekierowuje Cię z powrotem na stronę banku – na której już jesteś zalogowany (bo zalogowałeś się chwilę wcześniej).

Sprytne, prawda?

Jak uniknąć podatności?

Spójrz jeszcze raz, jak jest zrobione przekierowanie na prawdziwej stronie banku:

```
IActionResult RedirectUserAfterValidLogin(string returnUrl)
{
    if (string.IsNullOrEmpty(returnUrl))
        return RedirectToPage("/Index");
    else
        return Redirect(returnUrl);
}
```

Problemem tutaj jest metoda *Redirect*, która pozwala na przekierowanie na **dowolny** adres – w tym przekierowanie do zewnętrznego serwisu.

Przy logowaniu musisz się upewnić, że nie przekierowujesz użytkownika do zewnętrznego serwisu. Możesz to zrobić na dwa sposoby:

- Nie stosuj metody *Redirect*, chyba że chcesz przekierować jawnie użytkownika na adres zewnętrzny
- Użyj metody *RedirectToPage* (lub *RedirectToView* w MVC) zamiast *Redirect* – Metoda *RedirectToPage* przekieruje Cię tylko na stronę wewnętrzną. W takim przypadku spróbowałaby przekierować na: *https://localhost:8000/https://localhost:9000* i dostałbyś błąd 404.

- Upewnij się, że adres na który przekierowujesz jest wewnętrznym:

```
if (Url.IsLocalUrl(ReturnUrl))
    return Redirect(ReturnUrl);
else
    return Redirect("/Index");
```

Ja w swoich rozwiązaniach stosuję zawsze *RedirectToPage* (*RedirectToView*). To jest po prostu wygodne.