JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 9 (2018) 340-344
Wysłane: 2018-09-20
Przyjęte: 2018-10-09

Określenie skuteczności zabezpieczeń aplikacji internetowych przed różnymi metodami ataków sieciowych

Mateusz Erbel*, Piotr Kopniak
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule poruszono kwestię bezpieczeństwa aplikacji internetowych. Omówiono najpopularniejsze rodzaje ataków oraz metody zabezpieczania przed nimi aplikacji internetowych. W pracy przeprowadzono badania skuteczności zabezpieczeń aplikacji internetowych. Metodykę badawczą oparto na autorskiej aplikacji, zaimplementowanej w technologii PHP. Wynikiem badań jest propozycja rozwiązań mających na celu poprawę bezpieczeństwa aplikacji.

Słowa kluczowe: Ataki sieciowe; Aplikacje internetowe; XSS; SQL Injection

*Autor do korespondencji.

Adres e-mail: mateuszerbel@gmail.com

Assessment of the web application security effectiveness against various methods of network attacks

Mateusz Erbel*, Piotr Kopniak
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article discusses the issue of the security of Internet applications. The most popular types of attacks and methods of securing web applications against them are discussed. The study conducted the effectiveness of security of web applications. The research methodology was based on the proprietary application implemented in PHP technology. The result of the research is a proposal of solutions aimed at improving application security.

Keywords: Network attacks; Internet applications; XSS; SQL Injection

*Corresponding author.

E-mail address: mateuszerbel@gmail.com

1. Wstęp

W obecnych czasach bezpieczeństwo niewątpliwie stało się jednym z najważniejszych aspektów aplikacji internetowych. Nie stworzono jeszcze aplikacji która byłaby całkowicie bezpieczna i niepodatna na żaden atak. Podczas tworzenia aplikacji programista powinien zaplanować je w taki sposób aby poziom ich ochrony był jak najwyższy, niestety często zdarza się, że aplikacje internetowe atakowane są kilka lat po ich wydaniu, przez co zamysł programistyczny musi tworzyć barierę, która pozwoli chronić dane i aplikację przez wiele lat.

Niniejszy artykuł pokazuje jak ważnym problemem jest bezpieczeństwo aplikacji internetowych, a także jak istotne jest to aby poziom zabezpieczeń był najwyższy. Temat bezpieczeństwa jest bardzo istotny dla aplikacji internetowych dlatego też został wybrany do analizy. Analiza polegała na przetestowaniu zabezpieczeń wprowadzonych do stworzonego w tym celu serwisu internetowego.

Istnieje wiele metod ataków na aplikacje internetowe. Przykładami takich ataków mogą być [1]:

- ataki odmowy dostępu DoS/DDoS,
- podsłuchanie nieszyfrowanego ruchu,
- ataki związane z wykonaniem wrogiego kodu,
- zagrożenia związanie z logowaniem i sesją użytkownika.

Aby uchronić się przed takimi zagrożeniami niezwykle ważne jest odpowiednie skonfigurowanie serwera, stworzenie funkcji i modułów, które zabezpieczą przed wykonaniem wrogiego kodu zawartego w danych wejściowych. Odpowiednie przechowywanie haseł w bazie danych oraz przeprowadzenie testów bezpieczeństwa w celu sprawdzenia czy zabezpieczenia zostały odpowiednio zaimplementowane.

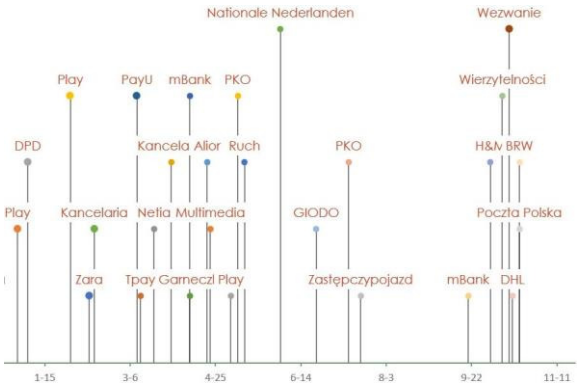
2. Wprowadzenie do bezpieczeństwa

Aplikacje internetowe mogą być atakowane z wielu powodów. Jedne z przyczyn mogą mieć podłoże finansowe, inne psychologiczne, polityczne, militarne a jeszcze inne mogą odbywać się automatycznie. Możemy wyróżnić kilka podstawowych czynników, które nakłaniają hakerów do ataków [2]:

- Korzyści materialne – często jest to główny powód ataków, pozyskanie dóbr np. takich jak środki finansowe mogą posłużyć do przeprowadzenia kolejnych ataków;
- Wandalizm – umyślnie niszczenie czyjejś pracy;
- Szpiegostwo – pozyskanie danych w celach politycznych, przemysłowych lub na własny użytek;

- Sprawdzenie swoich możliwości – sprawdzanie, testowanie własnych umiejętności;
- Przypadek – ataki wykonywane przez boty zazwyczaj wybierające losowe zasoby Internetu w celach znalezienia i wykorzystania luk.

Konsekwencji ataków tak samo jak czynników do ich przeprowadzenia jest wiele. Od pewnego czasu dość popularne są ataki phishingowe. Rysunek 1. przedstawia większość znanych kampanii e-mailowych wymierzonych w klientów firmy, których one dotknęły [3].



Rys.1. Kampanie phishingowe w ostatnich latach

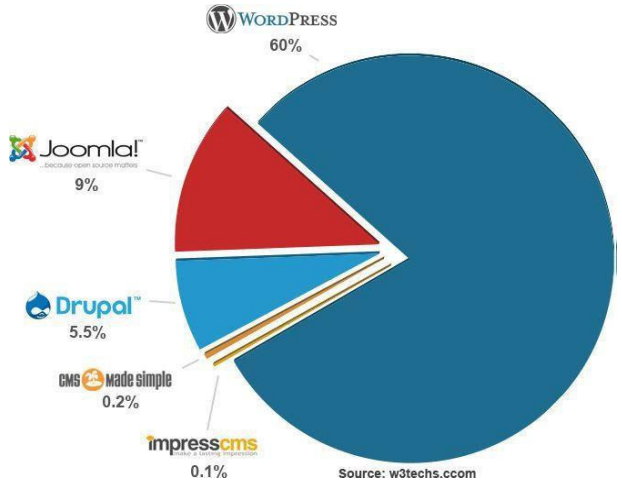
Za bezpieczeństwo aplikacji w głównej mierze odpowiada jej twórca. Niestety nie zawsze osoby tworzące oprogramowanie mają odpowiednie kwalifikacje. Częstą praktyką jest powierzanie ważnych mechanizmów programistom z małym doświadczeniem w tworzeniu aplikacji internetowych. Nie znają oni jeszcze znacznej liczby podatności przez co nie przykładają uwagi da zabezpieczenia aplikacji przed nimi.

Kolejny ważny problem wynika z typów tworzonych aplikacji internetowych. Aplikacje mogą być tworzone pod konkretnego klienta, często wynika to z potrzeby integracji z istniejącą już aplikacją. Dzięki indywidualnym rozwiązaniom dostęp do kodu źródłowego jest ograniczony przez co ograniczona jest też liczba osób znających mechanizmy wewnętrze i punkty krytyczne aplikacji. Niestety takie aplikacje przechodzą zazwyczaj podstawowe testy bezpieczeństwa przez co możliwe podatności mogą pojawić się dopiero po wdrożeniu aplikacji.

Drugim typem a zarazem najczęściej występującym są aplikacje powielane. Bazują one już na istniejących rozwiązaniach. Przykłady takich systemów przedstawia rys. 2 [9].

Inaczej niż w aplikacjach budowanych pod konkretnych klientów, ich kod jest jawny a ilość wdrożonych serwisów znaczna. Niestety w przypadku aplikacji powielanych czas od wykrycia podatności do jej załatania jest uzależniony znacząco od producenta oprogramowania. Zanim to nastąpi atakujący może dokonywać ataków na dane aplikacje powielane.

Powyższe powody są główną przyczyna podatności aplikacji na ataki. Trzeba jednak pamiętać, że wszystkie techniki ataków na aplikacje webowe posiadają indywidualne warunki, w których mogą zajść.



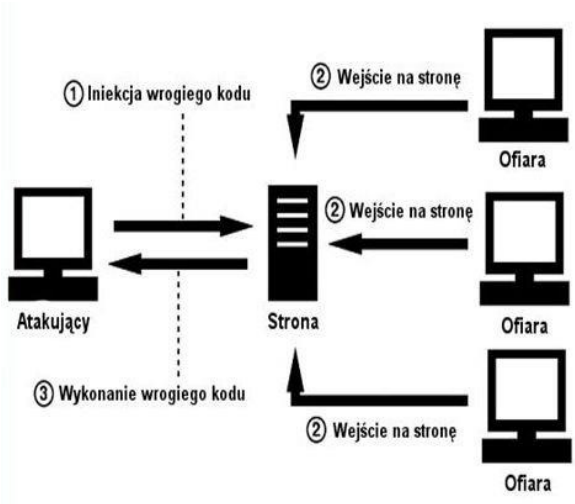
Rys.2. Udział systemów CMS na rynku w 2017 roku

3. Rodzaje ataków

W aplikacjach możemy zauważyć punkty wejścia czyli dane pobierane oraz punkty wyjścia np. takie jak dane i czynności generowane. Na pewną część danych wejściowych użytkownik ma pełny wpływ inne są dostępne tylko w konkretnych sytuacjach. Do danych wejściowych, na które użytkownik ma pełen wpływ zaliczają się wszelkie parametry przekazywane przez klienta w żądaniu HTTP. Natomiast do takich danych jak pliki konfiguracyjne, usługi sieciowe, dane innych źródeł klient ma dostęp w określonych przypadkach [4].

3.1. Ataki XSS

Jednym ze sposobów ataku na serwis WWW, są ataki typu XSS (Cross-site scripting) polegają na osadzeniu w treści atakowanej strony złośliwego kodu zazwyczaj jest to JavaScript , który wyświetlony innym użytkownikom może dążyć do wykonania przez nich niepożądanych akcji [5]. Przykładowy schemat ataku przedstawia rysunek 3.



Rys.3. Przykładowy schemat ataku XSS

Ataki XSS dzielą się na różne kategorie, główne z nich to Reflected i Stored. W pierwszym kod JavaScript zaszywany jest w linku, który następnie atakujący przesyła do ofiary. Po wejściu w link łączy się on z aplikacją w celu przekazania fragmentu HTML zawierającego kod JavaScript. Aplikacja zwraca ofierze wynik, w postaci wykonania wrogiego kodu w przeglądarce [6].

Drugim rodzajem ataku jest Stored inna spotykana nazwa to Presistent, najbardziej złośliwa odmiana, w odróżnieniu od Reflected polega na umieszczeniu kodu JavaScript po stronie serwerowej. Atakujący może podczas dodawania komentarza na stronie zamieścić w nim złośliwy kod. Dzięki odpowiedniemu filtrowaniu danych można uniknąć takich sytuacji.

3.2. Atak CSRF

Inną metodą ataków na serwisy internetowe jest atak CSRF, zmusza on zalogowaną przeglądarkę ofiary do wysłania sfałszowanego żądania http, między innymi pliku cookie sesji ofiary jak i wszelkich innych automatycznie włączonych informacji uwierzytelniających, do podatnej aplikacji internetowej. Atakujący zmusza przeglądarkę ofiary do generowania żądań, które aplikacja uważa za normalne żądania ofiary.

Istnieje cały szereg metod, które utrudniają przeprowadzenie skutecznego ataku CSRF [7]:

- Dodawanie do każdego formularz ukrytych pól, zawierających liczbę pseudolosową, która przekazywana jest wraz z żądaniem wykonania akcji. Następnie wszystkie akcje bez ukrytej wartości lub z niepokrywającą się z liczbą zachowaną po stronie serwera są ignorowane.
- Im ważniejsze informacje przechowuje strona, tym krótszy powinien być dopuszczalny czas bezczynności i okres ważności zalogowania.
- Wprowadzenie haseł jednorazowych, logowania dwuskładnikowego w znacznym stopniu uniemożliwia to osobie niepowołanej spreparowanie poprawnego żądania do serwera aplikacji.
- Wykonywanie ważnych żądań powinno wymagać ponownej autoryzacji.
- Zamiast dodawania do formularza ukrytych pól z liczbą pseudolosową, można porównać zawartość cookie służącego do uwierzytelnienie z wartością przesyłaną w żądaniu HTTP oraz z wartością zapisaną po stronie serwera.

3.3. Atak SQL Injection

Ostatnim atakiem związanym z wykonaniem wrogiego kodu, przedstawionym w pracy będzie SQL Injection. Jest to dość częsta i jednocześnie jedna z najniebezpieczniejszych podatności aplikacji webowych. Polega na wstrzyknięciu do aplikacji fragmentu zapytania SQL. Często przez brak odpowiedniej walidacji parametrów przekazywanych przez użytkownika możliwe jest wstrzyknięcie dodatkowego kodu SQL.

Wykorzystując podatność SQL Injection mamy bardzo wiele możliwości, co jakiś czas w Internecie pojawiają się bazy danych z loginami, hasłami do różnych stron czy portali. W większości przypadków są to dane pozyskane właśnie za pomocą tego ataku. Należy pamiętać, że ataki SQL Injection są drugimi najczęściej występującymi atakami na aplikacje webowe. W praktyce atakujący korzysta ze spreparowanego zapytania UNION SELECT aby pobrać interesujące go informację, które zawiera atakowana baza danych. Nieodłączną częścią udanego ataku na aplikację internetową jest kompromitacja jego twórcy. Niestety zazwyczaj narażona zostaje reputacja instytucji lub całego serwisu dlatego niezwykle ważnym jest odpowiednie zabezpieczenie się przed atakami SQL Injection. W celu zminimalizowania powodzenia ataku SQL Injection na aplikację należy zastosować się do kilku zasad [8]:

- Odpowiednia walidacja danych - jeżeli w formularzu oczekujemy nazwiska, sprawdzamy czy dany ciąg znaków zawiera wyłącznie litery (preg_match()), jeżeli oczekujemy liczby sprawdźmy czy na pewno tylko liczby zostały przekazane (is_numeric()).
- Używanie parametryzowanych zapytań do bazy danych.
- Odcinanie danych, które z punktu widzenia danej akcji są nieistotne .
- Ustawianie mniejszych uprawnień dla użytkowników bazy danych.
- Uniemożliwienie wyświetlanie błędów w aplikacji.

4. Badania

W trakcie przeprowadzanych badań strona jest atakowana a następnie zabezpieczana przed wykonanymi wcześniej atakami. Stworzona aplikacja umożliwia dodawanie aktualności, nowych użytkowników, dostęp do ukrytych podstron. Wymienione moduły będą główną częścią wykonywanych badań.

Podczas dodawania aktualności sprawdzono czy możliwa jest edycja wyświetlanych aktualności za pomocą znaczników co przedstawia rysunek 4.

Rys.4. Dodanie nowej aktualności wraz ze znacznikami HTML

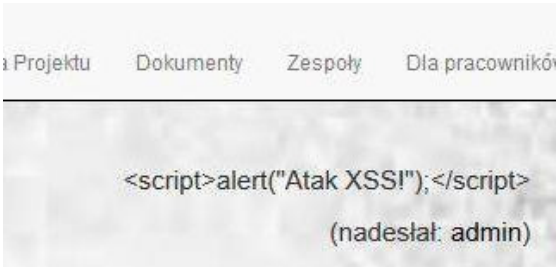
W aktualnościach został opublikowany wpis „Próba ataku XSS" a dzięki znacznikom <h1> i <b> został pogrubiony i przedstawiony jako nagłówek. Jak widać

aplikacja jest podatna na ataki XSS, kolejnym krokiem będzie próba osadzenia w aplikacji kodu JavaScript co przedstawia rysunek 5.

Rys.5. Osadzenie kodu JavaScript w aplikacji

Poprzez osadzenie kodu JavaScript w aplikacji, alert będzie pokazywał się każdemu użytkownikowi który wejdzie w zakładkę aktualności. Możemy używać też znaczników <img> po to aby na atakowanej stronie osadzić dowolny obrazek. W celu zabezpieczenia aplikacji została włączona funkcję magic_quotes_sybase, która powoduje dodawanie znaków „\" w tablicach Get, Post oraz Cookie przed znacznikami specjalnymi.

Kolejną włączoną funkcją jest htmlspecialchars($string), przeszukuje ona tekst w poszukiwaniu znaczników PHP i HTML następnie zamienia znaki specjalne (<,>,',",&), na ich bezpieczne odpowiedniki. Dzięki zastosowaniu tej funkcji wyświetla się nieaktywny kod HTML, co przedstawia rysunek 6.



Rys.6. Dodanie kodu JavaScript po uruchomieniu funkcji

Podczas zmiany hasła użytkownika w systemie jest ono przesyłane w adresie URL, za pomocą ataku CSRF można próbować zmienić hasło dla zalogowanego użytkownika modyfikując adres.

Przykład 1. Zmodyfikowany adres URL

```
http://localhost/PD/admin/users/?password_new=123456&password_conf=12456Change=Change#
```

Dla zwiększenia powodzenia ataku można skorzystać ze stron zmieniających linki wtedy ofiara nie będzie widzieć co jest zapisane w prawdziwym linku. Następnie użytkownik z zalogowaną sesją musi kliknąć w przesłany link. Pomóc w tym mogą różne rodzaje phishingu. Po wejściu ofiary w podany link automatycznie hasło zostanie zmienione na to wpisane w adresie URL.

Istnieje szereg metod, które w znacznym stopniu mogą utrudnić atak typu CSRF, w aplikacji wprowadzono sesje użytkownika. Ich głównym zadaniem jest sprawdzanie czy użytkownik jest zalogowany lub wylogowanie go po ustalonym czasie bezczynności co przedstawia listing 1.

Przykład 2. Wylogowanie po czasie bezczynności

```
<?php
$intTimeoutSeconds = 600;
session_start();
if(isset($_SESSION['intLastRefreshTime']))
{
if(($_SESSION['intLastRefreshTime']+$intTimeoutSeconds)<
time())
{
session_destroy();
session_start();
} }
$_SESSION['intLastRefreshTime'] = time();
?>
```

Zmienna $intTimeoutSeconds została ustawiona na 600 dzięki temu użytkownik po dziesięciu minutach nieaktywności zostanie automatycznie wylogowany.

Kolejnym zabezpieczeniem jest wymuszenie na użytkowniku ponownego wpisania hasła dla krytycznych operacji

W ostatniej części badań sprawdzono podatność aplikacji na ataki typu SQL Injection. Wykorzystując zakładkę aktualności i modyfikując jej adres URL wydostano istotne informacje z bazy danych. Podczas wyświetlania aktualności z bazy aplikacja wykonuje zapytanie.

Przykład 3. Wyświetlanie aktualności

```
$sql = 'SELECT id, text, authorid FROM aktualnosci WHERE id = :id';
```

Jeżeli aplikacja jest podatna na atak, można modyfikując adres URL wykonywać polecenia w bazie danych.

Przykład 4. Zmodyfikowany adres URL

```
http://localhost/pd/aktualnosci.php?id=-1' union select (dalsza część zapytania); --
```

Znakiem cudzysłów zamknięto wykonywane polecenie aby następnie można dopisać własne, na końcu użyto średnika i dwóch myślników, które w dialekcie języka SQL stosowanego w systemie zarządzania bazami danych MySQL służą do komentowania, dzięki temu dalsza część zapytania aplikacji jest traktowana jako komentarz i się nie wykona. W celu lepszego poznania struktury bazy danych wykonano atak z poleceniem:

Przykład 5. Atak w celu poznania struktury bazy

```
union select tabelname, 2 information_schema.tabels;--
```

tabela ta zawiera nazwy wszystkich tabel występujących w bazie danych. Wynik takiego zapytania przedstawia rysunek 7.

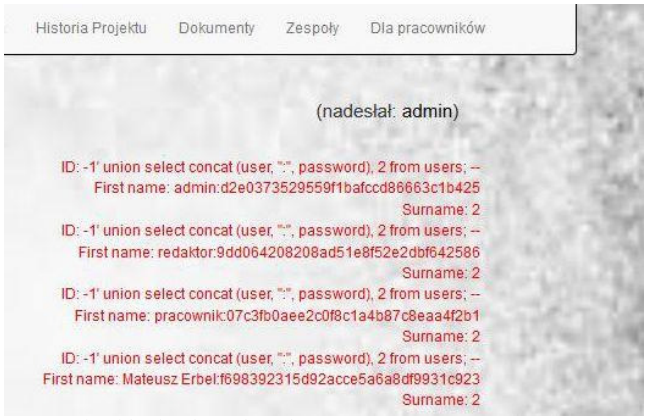
Rys.7. Wynik zapytania wyświetlającego tabele w bazie danych.

Następnie można zaatakować dowolnie wybraną tabelę, w tym przypadku wybrano tabelę users, metodą prób i błędów zmodyfikowano adres URL do następującej postaci:

Przykład 6. Zmodyfikowany adres URL

```
http://localhost/pd/aktualnosci.php?id=-1' union select concat (user, ":", password), 2 from users; --
```

Dzięki funkcji concat() loginy i hasła będą obok siebie rozdzielone znakiem dwukropka, wynik zapytania przedstawia rysunek 8.



Rys.8. Loginy i hasła z bazy danych

Głównym zabezpieczeniem przed SQL Injection jest odpowiednie filtrowanie danych wprowadzanych przez użytkownika.

5. Wnioski

Dzięki przeprowadzonym badaniom, określono zagrożenia, którym podlegają aplikacje internetowe oraz zaproponowano możliwości zabezpieczeń przed najpopularniejszymi metodami ataków. Zanim zaimplementowano wszystkie zabezpieczenia z powodzeniem udało się wykonać szereg ataków. Dzięki wykorzystaniu luk XSS agresor mógł osadzać na stronie dowolne znaczniki HTML, zdjęcia czy nawet kod JavaScript. Najbardziej niebezpieczny okazał się atak SQL Injection, który skutkował wyciekiem loginów i zahashowanych haseł użytkowników. Aplikacja została zabezpieczona przed wcześniej wymienionymi atakami, niestety przez ich złożoność do tej pory może pozostać niezabezpieczona. Kolejne badania powinny skupić się na atakach SQL Injection, które mogą przynieść największe szkody dla twórcy.

Literatura

[1] Zagrożenia aplikacji internetowych http://tadek.pietraszek.org/publications/kasprowski03_zagrozenia.pdf, luty 2018.

[2] Ziaja A.: Practical break-in analysis, PWN, 2017.

[3] Thomas, najbardziej uciążliwy cyberprzestępca https://zaufanatrzeciastrona.pl/post/thomas-najbardziej-uciazliwy-polski-cyberprzestepca-zatrzymany-przez-policje/, marzec 2018.

[4] Mueller J.: Security for Web Developers. O'Reilly Media, 2015.

[5] Hope P, Walther B.: Web Security Testing Cookbook, O'Reilly Media, 2012.

[6] Agarwal M, Singh A.: Metasploit. Receptury pentestera. Helion 2014.

[7] Ataak CSRF, https://haker.edu.pl/2016/04/23/atak-csrf-xsrf-i hasla-wep-9/, czerwiec 2018.

[8] Prasad P.: Testy penetracyjne nowoczesnych serwisów. Helion 2017.

[9] W3Techs – extensive and reliable web technology surveys, https://w3techs.com/, marzec 2018.