Zeszyty Naukowe Wydziału
Elektroniki i Informatyki
Politechniki Koszalińskiej
ISSN:1897-7421
Nr 13, Listopad 2018
Str: 27-36

Grzegorz Górski
Paweł Koziołko
Zakład Systemów Multimedialnych i Sztucznej Inteligencji
Wydział Elektroniki i Informatyki
Politechnika Koszalińska
ul. J.J. Śniadeckich 2
75-453 Koszalin

Semantyczne ataki na aplikacje internetowe wykorzystujące język HTML i arkusze CSS

Słowa kluczowe: podatności, aplikacje internetowe, zabezpieczenia, HTML, CSS, inwigilacja

1. Wprowadzenie

Istotnym wyzwaniem projektowym dla współczesnych aplikacji internetowych są ich podatności na wycieki informacji chronionych. Są to systemy dostępne w sieci publicznej, często dla nieautoryzowanych użytkowników. Efektywny poziom bezpieczeństwa informacji aplikacji internetowej zależy przede wszystkim od świadomości projektanta i programisty.

Artykuł przedstawia wybrane, istotne podatności języków HTML i CSS oraz prezentuje proste mechanizmy dzięki którym zwykły użytkownik może rozpoznać prawidłowo działającą aplikację, od jej wersji zainfekowanej tzn. przygotowanej do nieautoryzowanego pozyskania informacji. W artykule opisano także sposób zalecany postępowania w takim przypadku oraz zaproponowano rozwiązania pozwalające zwiększyć poziom bezpieczeństwa danych osobowych użytkownika.

Ten artykuł ma charakter wyłącznie edukacyjny i jego zadaniem jest zwiększenie świadomości programistów w zakresie różnego rodzaju podatności aplikacji internetowych.

2. HTML – podatności i metody zabezpieczeń

Zaprezentowane poniżej klasy ataków posiadają status krytycznych zagrożeń bezpieczeństwa wg metodologii OWASP (Open Web Application Security Project).

2.1. Ataki typu Future Proof

Język HTML w aktualnej wersji 5 jest zgodny z wcześniejszymi wersjami, co z jednej strony umożliwia powtórne użycie wcześniej stworzonego kodu, a z drugiej strony powoduje liczne zagrożenia bezpieczeństwa. Wybrane elementy składni odpowiadające za bezpieczeństwo wykonywanych operacji różnią się istotnie pomiędzy wersjami np. 4 i 5. Używanie niezaktualizowanych aplikacji z nowymi interpreterami tego języka skutkuje zgodnością na poziomie funkcjonalnym, lecz w odniesieniu do wymagań niefunkcjonalnych np. bezpieczeństwa ta zgodność już automatycznie nie występuje.

Jedną z podstawowych metod zabezpieczenia aplikacji internetowych przed tego typu atakami jest filtrowanie treści dodawanych przez użytkowników oparte na tzw. blacklistach, czyli listach tagów niedozwolonych.

W specyfikacji języka HTML5 wprowadzono wiele nowych atrybutów nie stosowanych w poprzednich wersjach. Jednym z nich jest „autofocus", który przenosi kursor użytkownika bezpośrednio do pola tekstowego skojarzonego z tym atrybutem. Może to być wykorzystane do przeprowadzenia ataku polegającego wprowadzeniu i wykonaniu obcego kodu do strony HTML. Łącząc ze sobą dwa atrybuty onfocus i autofocus można doprowadzić do potencjalnie niebezpiecznej sytuacji uruchomienia nadmiarowego kodu np. funkcji w języku JavaScript w momencie kiedy wybrane pole zostanie zaznaczone.

```
<input onfocus="alert(1)" autofocus>
```

Rys. 1. Przykład kodu ataku opartego o autofocus i onfocus

Kolejnym przykładem wykorzystującym atrybut autofocus jest dodanie co najmniej dwóch różnych pól z tym atrybutem z których pierwsze zawiera obsługę zdarzenia za pomocą atrybutu „**onblur**". Zdarzenie to zostanie uruchomione w momencie odznaczenia pola w którym jest określone, co nastąpi niezwłocznie, gdyż na drugim z pól został również ustawiony atrybut autofocus. Zaznaczenie drugiego pola w takim przypadku odbywa się bez ingerencji użytkownika.

```
<input onblur="alert(1)" autofocus><input autofocus>
```

Rys. 2. Przykład kodu ataku opartego o autofocus i onblur

Nieco inną odmianą podatności polegającej na wykorzystaniu atrybutu autofocus dla kilku pól jest atak z wykorzystaniem atrybutu onscroll zamiast onblur. W takim przypadku uruchomienie nieautoryzowanego kodu następuje w wyniku przesunięcia strony przez użytkownika lub samoistnie przez odpowiednio spreparowany kod. Oba powyższe ataki zakładają modyfikację aplikacji internetowej, która jest niewidoczna dla zwykłego użytkownika ani dla okresowo uruchamianych automatów testów regresywnych.

Analizując podatności pól tekstowych języka HTML można przedstawić kolejne potencjalne zagrożenia związane z zalecanym do stosowania kontenerem **from.** Kontener ten powinien być używany jako otoczenie dla fragmentu kodu, który zawiera pól tekstowych.

Standardowy sposób działania formularza zakłada, że wszystkie zmiany (np. zmiana wartości w polach tekstowych, wybór opcji z listy rozwijanej, wysłanie samego formularza) są aktywne tylko w obrębie danego kontenera (tagu) **form**. Możliwe jest przeprowadzenie ataku polegające na odwołanie się do formularza spoza kontenera z użyciem identyfikatora tego formularza. Przykład prezentujący odwołanie poza standardowo określonymi granicami kontenera zawarto na rys. 3.

```
<form id="testForm" onforminput="alert(1)">
  <input>
</form>
<button form="testForm" onformchange="alert(2)">x</button>
```

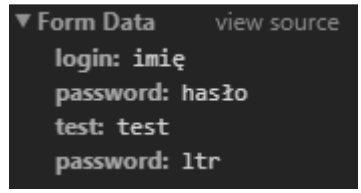
Rys. 3. Przykład inwigilacji formularza z atrybutami onformchange, onforminput

Obrona przed takimi atakami wymaga świadomości programisty polegającej na unikaniu ustawiania identyfikatora formularzy i blokowania stosowania atrybutów „form", „onformchange" i „onforminput". Zasięg zaprezentowanego powyżej przykładu jest ograniczony do użytkowników używających przeglądarki Opera.

Zarówno przeglądarki Opera i Chrome obsługują kolejny potencjalnie niebezpieczny atrybut „dirname", który może być wykorzystany przez atakującego. W takim przypadku przeglądarka udostępni informację na temat kierunku pisania tekstu innego pola w formularzu dodając go do treści żądania wysłanego do serwera. Poprzez umieszczenie atrybutu „dirname", następuje nadpisanie danych wprowadzonych przez użytkownika. Zmodyfikowane pole przyjmie wartość "ltr" lub "rtl" - w zależności od faktycznego kierunku przepływu tekstu. Na rysunkach 4 i 5 zaprezentowano tego typu atak w celu uniemożliwienia wprowadzenia poprawnego hasła. Użytkownik pomimo wprowadzania poprawnego hasła nie zostanie uwierzytelniony przez system.

```
<form method="POST">
    <input name="login"/>
    <input name="password" type="password"/>
    <input name="test" dirname="password" />
    <input type="submit" >
</form>
```

Rys. 4. Przykład kodu wykorzystującego atrybut dirname



Rys. 5. Przykład żądania wysłanego do serwera zmodyfikowanego przez atrybut dirname

Atak z wykorzystaniem atrybutu dirname nie dotyczy przeglądarek Internet Explorer oraz Firefox, gdyż nie jest on jeszcze przez nie obsługiwany.

Kolejna podatność dotyczy trzech najbardziej popularnych przeglądarek tzn. Opera, Chrome i Firefox, które umożliwiają wykorzystanie do obsługi błędu mechanizmu przetwarzającego plik źródłowy ograniczony znacznikami „**video**" i „**audio**" poprzez atrybut „**onerror**". Taka podatność może zostać wykorzystana w przypadku żądania obsługi przez aplikację internetową nieistniejącego lub uszkodzonego pliku wideo. W takim przypadku może zostać uruchomiony nadmiarowy kod zawarty pomiędzy ww. znacznikami.

```
<video><source onerror="alert(1)">
```

Rys. 6. Przykład kodu wykorzystującego atrybut onerror

Jedyną metodą ochrony przed atakiem za pośrednictwem atrybutu onerror jest usunięcie tego atrybutu z listy dopuszczalnych ciągów znaków (whitelist).

3. Rodzina ataków bezpośrednio związanych historią przeglądania stron

Do specyfikacji HTML5 wprowadzono dodatkowy mechanizm **history API,** który pozwala uzyskać dostęp do historii przeglądania stron zarejestrowanych w przeglądarce. Jest to realizowane z wykorzystaniem obiektu „**window**" drzewa DOM. Za pośrednictwem obiektu historii uzyskujemy dostęp do historii zapisanej przez przeglądarkę. Udostępnione metody zwykle stosowane do przeglądania historii użytkownika umożliwiają także przeprowadzenie ataku polegającego na modyfikacji stosu historii.

Do przygotowania ataku mogą zostać wykorzystane poniższe dwie funkcje udostępnione przez History API:

- **history.pushState** (data, title, url)
- **history.replaceState** (data, title, url)

gdzie:

data – dane, które mogą być odczytane w skrypcie,

title – tytuł strony, widoczny w historii przeglądarki (obecnie parametr ignorowany przez Firefox)

url – adres wynikowy, widoczny w pasku przeglądarki

Metoda **history.pushState()** pozwala zmienić adres URL w pasku przeglądarki na inny, zgodny z zasadą Same Origin Policy, co najczęściej jest ograniczone do dokładnie takiej samej nazwy domeny. Zmiana adresu w pasku przeglądarki może być wykorzystana do podania innego adresu (linku) zawartego w nagłówku HTTP podczas komunikacji z innym serwerem.

Metoda **history.replaceState()** działa w sposób zbliżony do history.pushState(), z tą jednak różnicą, że replaceState() modyfikuje ona bieżącą historię zamiast tworzyć nową. Należy jednak nadmienić, że funkcja: history.pushState() uniemożliwia utworzenie nowego wpisu w globalnej historii przeglądarki, co skutkuje brakiem możliwości powrotu do poprzedniej strony.

3.1. Rodzina podatności Reflected XSS

Ataki typu Reflected XSS wykorzystują mechanizm pozwalający na umieszczenie w parametrze zapytania http dodatkowego kodu, który zostanie wyświetlony w wygenerowanej aplikacji. Jednym z najczęściej stosowanych funkcjonalności używającej tego mechanizmu jest obsługa wyszukiwania w obrębie wyświetlanej strony.

```
http://domena.com/szukaj?q=<script>alert(1)</script>
```

Rys. 7. Przykład adresu www z zaszytym atakiem reflected XSS

W przypadku gdy aplikacja wyświetli stronę z wyrażeniem q wstawionym do jej kodu HTML, będzie to atak XSS typu odwzorowanego. Po wykonaniu takiego ataku, adres www wraz z niebezpieczną zawartością zostaje wyświetlony przez przeglądarkę w pasku adresu URL. W celu ukrycia takiej informacji tzn. ukrycia ataku XSS można użyć metody history.pushState() i zmienić aktualny adres www.

```
http://domena.com/szukaj?q=<script>alert(1); history.pushState({}, "",
location.href.split("?").shift())</script>
```

Rys. 8. Przykład adresu www z zaszytym atakiem reflected XSS i maskowaniem adresu www

Jedyną formą obrony przed tego typu atakiem jest zapewnienie aktualnych list filtracyjnych serwera aplikacji. Ataki i metody ich przeciwdziałania po stronie serwerowej wychodzą poza zakres niniejszego artykułu.

4. Wybrane ataki na arkusze styli CSS

CSS jest to język programowania służący do modyfikowania wyglądu aplikacji webowych, który może zostać wykorzystany do kradzieży wrażliwych danych z aplikacji internetowej użytkownika.

Szczególnie podatnym na ataki procesem jest obsługa przez CSS komunikacji z serwerami zewnętrznymi.

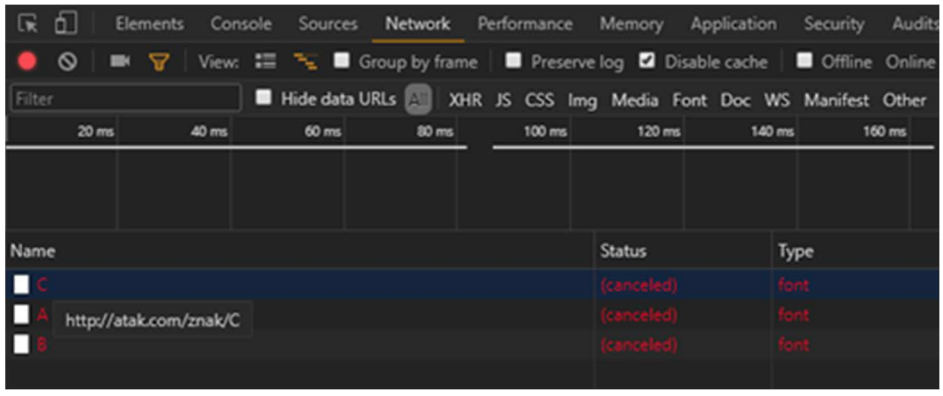
4.1. Ataki za pomocą font-face

Jednym z ataków na arkusz styli CSS polega na wykorzystaniu właściwości **unicode-range.** Scpecyfikacja CSS pozwala na ustawienie specjalnego kroju czcionki **@font-face. P**osiada on właściwość określenia zakresu (unicode-range), która to definiuje jakie znaki mogą występować w pliku pobieranym z zewnętrznego serwera. Atakujący posiadający taką informację może przygotować zestaw czcionek dla każdego znaku (unikodu), która będzie wysyłała żądanie do serwera hackera o przesłanie czcionki z konkretną literą. W kolejnym kroku użytkownik wpisując dane w polu tekstowym, wykorzystującym zhackowany krój czcionki, nieświadomie wysyła je bezpośrednio do serwera atakującego. Mechanizm działania przeglądarek zakłada, że raz wczytana czcionka dla konkretnego znaku nie będzie ponownie pobierana. Oznacza to, że każde kolejne wprowadzenie wybranego znaku spowoduje odwołanie do pamięci przeglądarki. Należy jednak zauważyć, że

atak tego typu pozwoli na stworzenie kompletnego alfabetu znaków w przypadku gdy każdy z nich zostanie chociaż jeden raz wprowadzony przez użytkownika.

```
@font-face {
    font-family: 'atackfont';
    src: url(//atak.com/znak/A);
    unicode-range: U+0041;
}
@font-face {
    font-family: 'atackfont';
    src: url(//atak.com/znak/B);
    unicode-range: U+0042;
}
.unsecured-field {
    font-family: 'atackfont';
}
```

Rys. 9. Deklaracja czcionki



Rys. 10. Wykaz żądań http po wpisaniu znaków CAB w polu tekstowym z klasą unsecred-field

4.2. Ataki poprzez selektor

Istnieje możliwość zaatakowania aplikacji internetowej przy użyciu selektorów CSS, które określają elementy drzewa DOM wraz z przypisaną do nich regułą.

Selektor value wskazuje tylko na te elementy drzewa DOM, których atrybut value przyjmuje zadana wartość. W przypadku gdy w arkuszu styli zostanie umieszczone polecenie input[value=A] to przeglądarka wskaże na pole tekstowe, które posiada dokładnie wartość A w atrybucie value.

Porównując atak z wykorzystaniem selektora do ataku za pomocą font-face należy zauważyć, że atak zostanie zainicjowany tylko w przypadku, kiedy atrybut value będzie miał wcześniej ustawioną wartość, a nie podczas wpisywania tekstu przez użytkownika, tak jak to miało miejsce przy ataku **font-face**.

Atak z selektorem value będzie wykorzystywał specjalnie przygotowany słownik wartości. Jego tworzenie można usprawnić wstawiając do kodu arkusza CSS znak ^ przed wczytywaną wartością co sprawi, że selektor będzie sprawdzał czy wartość w polu tekstowym zaczyna się od wartości zadanej w selektorze. W takim przypadku jeśli wprowadzana wartość będzie dłuższa od zadanej, ale początek będzie zawierał wartość podaną w selektorze to atakujący otrzyma szczątkową informację o pożądanej treści, np. może to być część hasła użytkownika. Przesłanie tej szczątkowej informacji do hakera może być związane z wykonaniem procedury komunikacji z serwerem atakującego np. przy pobraniu pliku graficznego jako tła elementu określonego selektorem.

```
input[value=A] {
    background-image: url(//atak.com/obraz/A.jpg);
}
input[value=B] {
    background-image: url(//atak.com/obraz/B.jpg);
}
input[value=C] {
    background-image: url(//atak.com/obraz/C.jpg);
}
```

Rys. 11. Przykład ataku poprzez selektor

5. Podsumowanie

Zaprezentowane w artykule podatności języka HTML i arkuszy styli CSS mogą powodować ataki na aplikacje internetowe po stronie klienta. Zaprezentowane metody ochrony oprócz świadomości programisty tworzącego kod aplikacji internetowej mogą wymagać także uruchomienia dostępnych mechanizmów filtracji po stronie serwera.

Bibliografia

1. Hague, Matthew, Lin, Anthony W., Ong, C. -H. Luke, 2015, *Detecting Redundant CSS Rules in HTML5 Applications: A Tree Rewriting Approach*
2. Mazinanian, Davood, Tsantalis, Nikolaos, 2016, An empirical study on the use of CSS preprocessors
3. Hale, Matthew L., Hanson, Seth, 2015, *A Testbed and Process for Analyzing Attack Vectors and Vulnerabilities in Hybrid Mobile Apps Connected to RESTful Web Services*
4. Mohammadi, Mahmoud, Chu, Bill, Lipford, Heather Richter, 2017, *Detecting Cross-Site Scripting Vulnerabilities through Automated Unit Testing*
5. Jan, Sadeeq, Nguyen, Cu D., Arcuri, Andrea, Briand, Lionel, 2017, A Search-based Testing Approach for XML Injection Vulnerabilities in Web Applications
6. Choi, Su Yeon, Lee, Hae Young, 2016, *Toward Automated Scanning for Code Injection Vulnerabilities in HTML5-Based Mobile Apps*
7. Dayal, Mohit, Singh, Nanhay, Raw, Ram Shringar, 2016, *A Comprehensive Inspection Of Cross Site Scripting Attack*
8. Gupta, Shashank, Gupta, B. B., 2016, XSS-SAFE: A Server-Side Approach to Detect and Mitigate Cross-Site Scripting (XSS) Attacks in JavaScript Code
9. Pan, Jinkun, Mao, Xiaoguang, 2017, *Detecting DOM-Sourced Cross-Site Scripting in Browser Extensions*
10. Liu, Shukai, Yan, Xuexiong, Wang, Qingxian, Zhao, Xu, Chai, Chuansen, (Sun, Yajing ,2016, *A Protection Mechanism against Malicious HTML and JavaScript Code in Vulnerable Web Applications*
11. https://developer.mozilla.org/en-US/docs/Web/API/History_API
12. Wu, Da-Chun, Su, Hsiu-Yang, 2013, *Information Hiding in EPUB Files by Rearranging the Contents of CSS Files*

Streszczenie

Języki programowania HTML i CSS to najczęściej wykorzystywane technologie do tworzenia aplikacji internetowych, których semantyka w powszechnym przekonaniu nie jest zbyt skomplikowana. Prosta składnia języka skutkuje efektywnością implementacji co jednak stoi w sprzeczności z koniecznością zapewnienia ochrony informacji i minimalizacji prawdopodobieństwa wycieku poufnych informacji. W artykule dokonano przeglądu najnowszych ataków na aplikacje internetowe oraz zaprezentowano skuteczne metody ochrony.

Abstract

The programming languages HTML and CSS are the most commonly used technologies for creating internet applications, the semantics of which are generally not too complicated. The simple language syntax results in implementation efficiency, that is contrary to ensure information protection and minimize the likelihood of confidential information leakage. The article reviews the latest attacks on Internet applications and presents effective protection methods.

Keywords: Vulnerabilities, web applications, security, HTML, CSS, surveillance.