BIULETYN INSTYTUTU AUTOMATYKI I ROBOTYKI
NR 30, 2011

Podział klientów WWW w kontekście przekazywalności aplikacji WWW

Robert GUST-BARDON

Instytut Systemów Informatycznych WAT,
ul. Gen S. Kaliskiego 2, 00-908 Warszawa
robert@gust-bardon.org

STRESZCZENIE: World Wide Web (WWW) jest „przestrzenią informacyjną, w której każdy zasób identyfikowany jest przy pomocy Uniform Resource Identifier" (URI). Agentem WWW jest „osoba lub oprogramowanie, które działa w tej przestrzeni informacyjnej na rzecz osoby, bytu lub procesu". Klient WWW to agent WWW, który usiłuje uzyskać dostęp do zasobu identyfikowanego przy pomocy URI. Przekazywalność aplikacji WWW oznacza określenie tego, którą jej część stanowi kod przekazywalny – zbiór instrukcji i definicji danych przeznaczonych do wykonywania bezpośrednio przez klienta WWW lub przez proces zależny od tego klienta WWW. W niniejszej pracy podzielono klienty WWW na aktywne klienty WWW i pasywne klienty WWW, stosując jako kryterium gotowość wykonywania przez nie kodu przekazywalnego lub powierzenia wykonywania tego kodu procesom zależnym od nich. Następnie, posługując się nieskomplikowanym przykładem, pokazano metodę wytworzenia aplikacji WWW dostępnej zarówno dla aktywnego klienta WWW, jak i dla pasywnego klienta WWW, a ponadto wykorzystującej automatycznie możliwość wykonywania kodu przekazywalnego. Przykład ten dowodzi, iż aktywne klienty WWW stały się dostępne najpóźniej w marcu 1996 roku, a prezentowana metoda znalazłaby już wtedy zastosowanie.

SŁOWA KLUCZOWE: klient, przekazywalność, WWW

1. Wprowadzenie

W niniejszej pracy przedstawione zostanie kryterium zaliczenia klienta WWW do jednego z dwu rodzajów – aktywnych klientów WWW oraz pasywnych klientów WWW – poprzedzone podaniem niezbędnych w tym celu definicji. Wymienione zostaną znalezione w literaturze zalety i wady obydwu rodzajów klientów WWW. Objaśniona zostanie nieskomplikowana aplikacja WWW, która, będąc dostępną dla obydwu rodzajów klientów WWW,

będzie jednocześnie wykorzystywała możliwość wykonywania kodu przekazywalnego. Umożliwi to ustalenie najpóźniejszej możliwej daty dostępności klientów WWW obydwu rodzajów (marzec 1996 roku).

2. Definicje nieodzowne do przeprowadzenia tytułowego podziału

World Wide Web (WWW) jest „przestrzenią informacyjną, w której każdy zasób identyfikowany jest przy pomocy Uniform Resource Identifier[1]" [16]. Według cytowanego dokumentu agentem WWW (ang. *Web agent*) jest „osoba lub oprogramowanie, które działa w tej przestrzeni informacyjnej na rzecz osoby, bytu lub procesu", zasobem (ang. *resource*) – „cokolwiek co może być identyfikowane poprzez URI", a postacią zasobu (ang. *representation*) – „dane przedstawiające informacje o stanie zasobu".

Na wzór terminologii HTTP [10] można założyć, iż agenty WWW komunikują się między sobą, wymieniając wiadomości (ang. *messages*) – ciągi bajtów o strukturze zdefiniowanej przez wykorzystywany protokół. Klient WWW (ang. *Web client)* to agent WWW, który usiłuje uzyskać dostęp do zasobu identyfikowanego przy pomocy URI. Wiadomość, którą przekazuje w tym celu innemu agentowi WWW, to żądanie (ang. *request*). Serwerem WWW (ang. *Web server*) jest agent WWW odpowiadający na to żądanie. W tym wypadku wiadomość powstała w reakcji na żądanie nazywana jest odpowiedzią (ang. *response*).

Kod (ang. *code*) stanowią „instrukcje komputerowe i definicje danych wyrażone w języku programowania lub w formie utworzonej przez assembler, kompilator lub inny translator" [15]. Aplikacja WWW (ang. *Web application*) to kod, któremu serwer WWW może powierzyć przygotowanie odpowiedzi na wybrane żądania. Kod przekazywalny (ang. *transferable code*) to zawarty w odpowiedzi kod przeznaczony do wykonania bezpośrednio przez klienta WWW lub przez inny proces zależny od klienta WWW. Przekazywalność (ang. *transferability*) oznacza określenie tego, którą część aplikacji WWW stanowi kod przekazywalny. Przedstawiony zbiór definicji został zaproponowany i uzasadniony w będącej w recenzji pracy [13].

3. Proponowany podział klientów WWW

Mianem „aktywnego klienta WWW" (ang. *active Web client*) będzie określany klient WWW, który jest gotów wykonywać kod przekazywalny lub

[1] URI [2].

powierzyć wykonywanie tego kodu innemu procesowi zależnemu od niego. Mianem „pasywnego klienta WWW" (ang. *passive Web client*) będzie określany klient WWW, który nie jest aktywnym klientem WWW. Brak gotowości do wykonywania kodu przekazywalnego może być spowodowana albo nieposiadaniem możliwości jego wykonywania (np. brakiem interpretera języka programowania, w którym zapisany jest kod) lub powstrzymaniem się od wykonywania go z innych powodów (np. podjęciem takowej decyzji przez użytkownika).

Wykonywanie kodu przekazywalnego po stronie klienta WWW umożliwia odciążenie serwera WWW poprzez częściowe lub całkowite przetwarzanie danych i wykonywanie obliczeń przez klienta WWW. Kod przekazywalny może być na przykład odpowiedzialny za uaktualnianie dokumentu, którego postać została odwzorowana przez klienta WWW [19]. Uaktualnienie nie musi oznaczać całkowitego ponownego odwzorowania postaci zasobów; istnieje możliwość aktualizowania poszczególnych fragmentów dokumentu [33].

Kod przekazywalny może być również odpowiedzialny za weryfikację kompletności danych podanych przez użytkownika w formularzu, zanim zostaną one przekazane do serwera WWW [1], [18]. Morrison, Morrison i Keys [21] zaznaczyli, iż aktywny klient WWW jest w stanie uniknąć konieczności przesyłania niepełnych lub niepoprawnych danych do serwera WWW i przygotowania tam stosownego komunikatu o błędzie – wiadomości niezbędnej dla pasywnego klienta WWW. Jest to szczególnie istotne w przypadku, gdy klient WWW dysponuje pasmem o istotnie ograniczonej przepustowości [11].

Kod przekazywalny może w odpowiednim środowisku przesyłać żądania oraz przetwarzać przygotowane przez aplikację WWW odpowiedzi. Również w tym przypadku zbędne jest całkowite ponowne odwzorowanie postaci zasobów. Jeżeli po przesłaniu powstałego w ten sposób żądania, a przed otrzymaniem na nie odpowiedzi, aktywny klient WWW wykonuje kolejne instrukcje komputerowe kodu przekazywalnego, ma miejsce asynchroniczne przesyłanie wiadomości, któremu można przeciwstawić synchroniczne przesyłanie wiadomości.

Field, Marinescu i Stefansen [9] wyrazili pogląd, iż asynchroniczne przesyłanie wiadomości, w połączeniu ze wcześniej zasygnalizowaną możliwością uaktualniania fragmentów dokumentu, zmniejsza ilość danych, które muszą zostać przesłane pomiędzy serwerem WWW a klientem WWW, co z kolei może prowadzić do skrócenia czasu reakcji interfejsu użytkownika i zmniejszyć obciążenie serwera WWW. Wusteman i O'hIceadha [34] stwierdzili, że umożliwia to uniezależnienie tempa interakcji użytkownika z klientem WWW od wymiany danych pomiędzy klientem WWW a serwerem WWW. Przeprowadziwszy eksperyment dotyczący powstawania w świadomości pomysłów, Vul i Pashler [31] uznali, iż precyzja pomiaru czasu uzyskana z wykorzystaniem kodu przekazywalnego była nieosiągalna dla pasywnych klientów WWW.

Turing [30] zaproponował uniwersalną maszynę obliczeniową na potrzeby rozstrzygnięcia problemu decyzyjnego postawionego przez Hilberta (niem. *Entscheidungsproblem*), a następnie udowodnił równoważność swojego modelu obliczalności oraz funkcji rekurencyjnych i rachunku lambda [29]. Cohen [5] udowodnił, iż niemożliwe jest kompletne wykrywanie wirusów, jeżeli system wykonuje instrukcje zapisane w języku programowania pozwalającym na symulowanie wzmiankowanej uniwersalnej maszyny oraz umożliwia wykonywanemu programowi dostęp do innych programów.

Luki bezpieczeństwa aktywnych klientów WWW wykorzystało na przykład złośliwe oprogramowanie, które zainfekowało ponad 180 tysięcy maszyn i zgromadziło blisko 70 gigabajtów danych; było ono głównie nastawione na kradzież informacji o kontach bankowych i kartach kredytowych [28]. Callegati i Ramilli [4] zaprezentowali mechanizm umożliwiający wykonanie złośliwego kodu przez aktywnego klienta WWW celem przechwycenia informacji dołączanych do żądań i przemycania własnych żądań. Zdrnja [36] opisał atak polegający na osadzaniu złośliwego kodu w postaciach zasobów WWW poprzez wykorzystanie słabości Address Resolution Protocol [22].

Jedną z metod zapobieżenia tego rodzaju atakom jest korzystanie z pasywnych klientów WWW [8], [24]. Aktywne klienty WWW mogą być również niepożądane nie ze względu na szkodliwe, lecz irytujące skutki wykonywania kodu przekazywalnego [3], [12]. W przypadku urządzeń mobilnych wysyłanie dodatkowych żądań przy pomocy mechanizmu takiego jak XMLHttpRequest [17] wiąże się z istotnym zużyciem baterii oraz może narazić użytkownika na dodatkowe wydatki [6]. Innym powodem odstąpienia od aktywnych klientów WWW może być ingerowanie kodu przekazywalnego w prywatność [27].

Klient WWW powinien umożliwić użytkownikowi decydowanie o pełnieniu przez to oprogramowanie roli aktywnego klienta WWW [14]. Decyzja ta może zostać podjęta w stosunku do wybranych zasobów WWW lub rozciągać się na wszystkie przetwarzane postacie zasobów [25]. Dodatkowo istnieją klienty WWW mogące pełnić wyłącznie rolę pasywnych klientów WWW. Do tej grupy mogą należeć w szczególności klienty WWW indeksujące przestrzeń informacyjną [20], [32].

Podsumowując, wykonywanie kodu przekazywalnego po stronie aktywnego klienta WWW może – w porównaniu do przetwarzania wyłącznie po stronie serwera WWW – prowadzić do zmniejszenia obciążenia serwera WWW i skrócenia czasu reakcji interfejsu użytkownika poprzez częściowe lub całkowite przetwarzanie danych i wykonywanie obliczeń oraz opcjonalną możliwość przesyłania żądań i przetwarzania przygotowywanych przez aplikację WWW odpowiedzi; jest ono możliwe wyłącznie w przypadku aktywnych klientów WWW, może mieć złośliwe, irytujące i naruszającego prywatność skutki, a ponadto – w stosunku do urządzeń mobilnych – wiązać się z istotnym zużyciem baterii oraz narażeniem użytkownika na dodatkowe wydatki.

4. Aplikacja WWW przeznaczona dla obydwu rodzajów klientów

Opisana zostanie teraz nieskomplikowana aplikacja WWW z listingu 1 o następującej cesze: pozostając w pełni funkcjonalną dla pasywnych klientów WWW, aplikacja WWW będzie jednocześnie wykorzystywała sposobność wykonywania kodu przekazywalnego, jeżeli posłuży do przygotowywania odpowiedzi dla aktywnego klienta WWW. Dodatkowo przykład ten pokazuje, iż klient WWW spełniający definicję aktywnego klienta WWW istniał już w marcu 1996 roku (był nim Netscape Navigator 2.0[2]; rys. 1; cf. [35]).

Aplikacja WWW z wymienionego listingu działa poprawnie również we współczesnym aktywnym kliencie WWW – Mozilla Firefox 3.5.8[3] (wydanym po niespełna czternastu latach – 17 lutego 2010 roku; rys. 2) oraz w pasywnych klientach WWW – posiadającym graficzny interfejs użytkownika programie Netscape Navigator 1.2[4] (wydanym w lipcu 1995 roku; rys. 3) i działającym w trybie tekstowym programie ELinks 0.12pre5[5] (wydanym 8 lipca 2009 roku; rys. 4).

Aplikacja WWW z rzeczonego listingu jest w pełni funkcjonalna zarówno dla aktywnego klienta WWW, jak i dla pasywnego klienta WWW, a ponadto automatycznie wykorzystuje ona możliwość wykonywania kodu przekazywalnego przez tego pierwszego. Liczący 100 linii kod źródłowy napisany został w języku programowania Python 2.6.4[6] i wykorzystuje standardową bibliotekę tego języka, która zawiera moduł mogący pełnić funkcję prostego serwera HTTP. Jednakże 59 linii kodu źródłowego (linie: 9–22, 25–42, 45–52 oraz 55–73) zajmują szablony w HTML (*HyperText Markup Language*) 4.01[7] [23], z których 12 linii (linie 32–38 oraz 57–61) stanowi kod przekazywalny – osadzone instrukcje w pierwszej wersji języka programowania JavaScript (poddanego później standaryzacji zapoczątkowującej ECMAScript [7]).

Wspomniana aplikacja WWW podaje jej użytkownikowi wynik zastosowania funkcji skrótu SHA-256 [26] w stosunku do podanego w formularzu łańcucha znaków. Zakłada się tu, iż obliczanie wyniku może odbywać się wyłącznie po stronie serwera WWW, chociaż nie ma przeszkód, aby wspomniany algorytm zaimplementować w języku programowania

---

2 Dostępny w World Wide Web: `http://browsers.evolt.org/?navigator/32bit/2.0`

3 Dostępny w World Wide Web: `ftp://ftp.mozilla.org/pub/firefox/releases/`

4 Dostępny w World Wide Web: `http://browsers.evolt.org/?navigator/32bit/1.22`

5 Dostępny w World Wide Web: `http://elinks.or.cz/download/`

6 Datowanym na 25 października 2009 roku. Dobór języka programowania stosowanego po stronie serwera WWW nie ma żadnego znaczenia dla zawartych tu rozważań, o ile możliwe jest napisanie w nim programu symulującego wspomnianą uniwersalną maszynę obliczeniową. Dostępny w World Wide Web: `http://www.python.org/ftp/python/2.6.4/`

7 Chociaż standard HTML 4.01 opublikowano 24 grudnia 1999 roku, wykorzystywany tutaj podzbiór tego języka jest w pełni obsługiwany przez wymienione klienty WWW.

ECMAScript po stronie aktywnego klienta WWW[8].

Po uruchomieniu programu z listingu 1. tworzony jest obiekt klasy `SocketServer.TCPServer` (linia 99), który stanowi serwer WWW dostępny pod domyślnym adresem IP (*Internet Protocol*) i portem 8080. W linii 100 wydawane jest polecenie ciągłego oczekiwania przez serwer WWW na nadchodzące od klientów WWW żądania. Jeżeli takowe nadejdą, odpowiedzi na nie zostaną przygotowane przez obiekt klasy `RequestHandler`. Klasa ta (linie 77–96) dziedziczy po klasie `BaseHTTPServer.BaseHTTPRequestHandler` i dostarcza metodę `do_GET` (linie 78–96) do obsługi metody `GET` żądania HTTP.

Opisywana aplikacja WWW obsługuje zatem tylko jedną metodę żądania HTTP (`GET`). Serwer WWW przekazuje żądania tego typu do metody `do_GET`. Tam następuje przetwarzanie użytego przez klienta WWW identyfikatora zasobu (linie 79–82). Istotne są tutaj dwie wartości: ścieżka (*path*) żądanego zasobu (`url.path`) oraz ciąg znaków odpowiadający zapytaniu (*query*). Jeżeli wspomniany ciąg znaków zawiera wartość dla klucza `string`, ciąg ten przypisywany do zmiennej `string` (w przeciwnym wypadku jej wartość stanowi pusty łańcuch znaków; linie 81–82). Wybierany jest jeden ze wskazanych wcześniej szablonów (łańcuchów znaków reprezentujących często niekompletne dokumenty zapisane w HTML 4.01); w zależności od ścieżki mogą to być: `checksum`, `interface` bądź `frameset` (linie 83–85).

Po podjęciu decyzji co do szablonu łańcuch znaków `${body}`, o ile występuje w dołączonym szablonie, zastępowany jest szablonem `body`. Jeżeli w wykorzystywanym szablonie znajduje się łańcuch znaków `${string}`, jest on zamieniany na taki, który można poprawnie umieścić w opatrzonym cudzysłowami atrybucie dokumentu HTML 4.01 (linie 88–89). Ponadto łańcuch znaków `${sum}` zamieniony zostaje na wartość funkcji skrótu dla argumentu znajdującego się w zmiennej `string` (linia 90). Następnie wysyłana jest odpowiedź posiadająca status „200 OK” (linia 91) oraz trzy nagłówki informujące o: typie MIME (*Multipurpose Internet Mail Extensions*) dokumentu i jego kodowaniu znaków (linia 92), typie MIME osadzonych w dokumencie skryptów (linia 93) oraz długości przeznaczonej do wysłania wiadomości (wyznaczana jest ona z pominięciem długości samych nagłówków; linia 94). Ostatnim poleceniem jest zwrócenie serwerowi WWW odpowiedzi zawierającej powstały dokument (linia 96).

Pierwszym z szablonów pojawiających się w kodzie źródłowym jest `frameset` (linie 9–22), którego zadaniem jest podział okna na dwie ramki – jedną niewidoczną dla użytkownika (jej wysokość wynosi zero) i jedną widoczną dla użytkownika, wypełniającą pozostałą wolną przestrzeń (linia 15). Do pierwszej ramki ma zostać wczytywany pusty dokument (o identyfikatorze

[8] Przykładowo: `http://www.movable-type.co.uk/scripts/sha256.js`

`about:blank`; linia 16), natomiast do drugiej z nich – dokument zawierający interfejs użytkownika (linia 17). Interfejs zostanie wyświetlony również wtedy, gdy klient WWW nie obsługuje podziału na ramki, co jest możliwe dzięki zastosowaniu bloku `NOFRAMES` (linie 18–20). Zawartość tego bloku – wartość zmiennej `$body` – jest identyczna z zawartością dokumentu otrzymanego w wyniku wykorzystania szablonu `interface` (linie 45–52).

Szablon `body` (linie 55–73) zawiera część dokumentu odpowiedzialną za interfejs użytkownika. Jest to formularz (linie 56–72), który składa się z dwóch pól tekstowych oraz jednego przycisku służącego do wydania polecenia przesłania żądania do serwera WWW (linia 66). Wartość podana przez użytkownika w polu tekstowym o nazwie `string` (linia 65) jest dołączana do tego żądania i oznacza argument dla funkcji skrótu. W polu tekstowym bez nazwy (linia 70) wstawiana jest wyznaczona wartość tej funkcji.

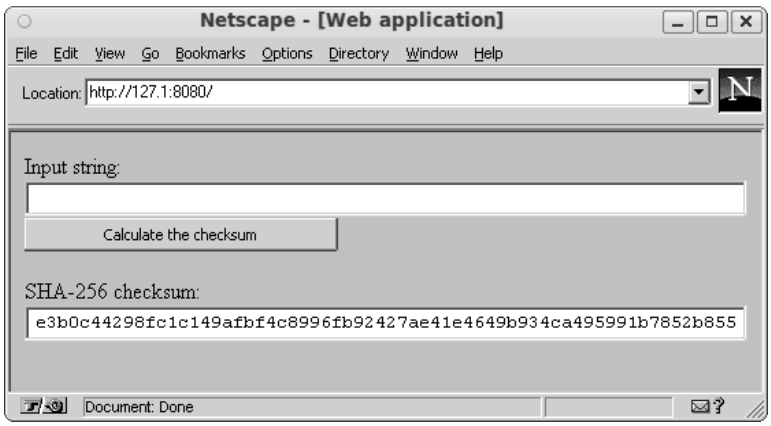
Pasywny klient WWW, wykorzystując identyfikator postaci `http://127.1:8080/`, otrzymuje postać zasobu o pustej ścieżce, której odwzorowanie stanowi interfejs użytkownika. Jeżeli klient WWW obsługuje podział na ramki, zaprezentowany zostanie formularz o identyfikatorze `http://127.1:8080/interface/`. Po wprowadzeniu łańcucha znaków do pola tekstowego o nazwie `string` i wydaniu polecenia przesłania zawie–rającego ten łańcuch żądania do serwera WWW, klient WWW zażąda zasobu o identyfikatorze `http://127.1:8080/interface/?string=`*łańcuch*, czego następstwem będzie otrzymanie dokumentu zawierającego w polu tekstowym o nazwie `string` podany łańcuch znaków, a w polu tekstowym bez nazwy wartość argumentu funkcji skrótu wyznaczonej dla tego łańcucha.

Gdy aktywny klient WWW (wykonujący ECMAScript oraz obsługujący podział na ramki) odwzoruje postać zasobu o pustej ścieżce, w momencie wydania polecenia przesłania żądania do serwera WWW wykona on instrukcje przypisane zdarzeniu `onSubmit` (linie 57–61), a to z kolei spowoduje otworzenie w niewidocznej ramce, generowanego z wykorzystaniem szablonu `checksum` (linie 25–42), niewykorzystywanego przez pasywnego klienta WWW zasobu `http://127.1:8080/checksum/?string=`*łańcuch* oraz powstrzymanie się klienta WWW od przesłania danych zawartych w formularzu. Wykonanie instrukcji zawartych w opisywanym dokumencie (linie 32–38), następujące wraz z jego odwzorowaniem (zdarzenie `onLoad`) spowoduje zastąpienia wartości pola tekstowego bez nazwy wartością funkcji skrótu dla przechwyconego argumentu (wartości pola tekstowego o nazwie `string`).

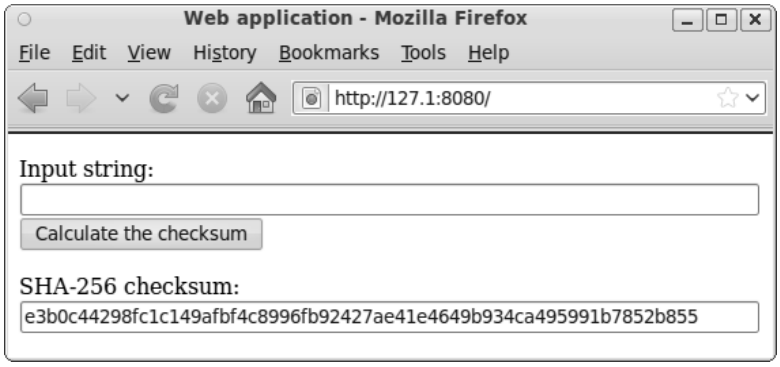
Listing 1. Autorska, przykładowa, obliczająca wartość funkcji skrótu SHA-256 aplikacja WWW przeznaczona dla pasywnych klientów WWW i aktywnych klientów WWW

```
 1. #!/usr/bin/env python
 2. import SocketServer
 3. import BaseHTTPServer
 4. import urlparse
 5. import hashlib
 6.
 7. templates = {
 8.   'frameset': """
 9. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
10.   "http://www.w3.org/TR/html4/frameset.dtd">
11. <HTML>
12.   <HEAD>
13.     <TITLE>Web application</TITLE>
14.   </HEAD>
15.   <FRAMESET rows="0,*">
16.     <FRAME src="about:blank">
17.     <FRAME src="/interface/">
18.     <NOFRAMES>
19. ${body}
20.     </NOFRAMES>
21.   </FRAMESET>
22. </HTML>
23. """,
24.   'checksum': """
25. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
26.   "http://www.w3.org/TR/html4/strict.dtd">
27. <HTML>
28.   <HEAD>
29.     <TITLE>The update of the checksum</TITLE>
30.   </HEAD>
31.   <BODY onLoad="
32. if (parent.frames) {
33.   if (parent.frames.length != 0) {
34.     if (parent.frames[1].document.forms.length != 0) {
35.       parent.frames[1].document.forms[0].elements[2].value = '${sum}';
36.     }
37.   }
38. }
39.   ">
40.     <SCRIPT type="text/javascript"></SCRIPT>
41.   </BODY>
42. </HTML>
43. """,
44.   'interface': """
45. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
46.   "http://www.w3.org/TR/html4/strict.dtd">
47. <HTML>
48.   <HEAD>
49.     <TITLE>Web application</TITLE>
50.   </HEAD>
51. ${body}
52. </HTML>
53. """,
54.   'body': """
55.   <BODY>
56.     <FORM method="GET" action="/interface/" onSubmit="
57. if (parent.frames) {
58.   parent.frames[0].location = 'http://127.1:8080/checksum/?string=' +
59.     escape(this.elements[0].value);
60.   return false;
61. }
62.     ">
63.       <P>
64.         Input string:<BR>
```

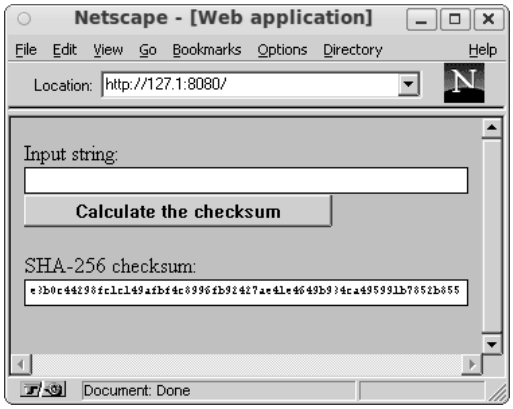
```
 65.         <INPUT name="string" size="65" value="${string}"><BR>
 66.         <INPUT type="submit" value="Calculate the checksum">
 67.       </P>
 68.       <P>
 69.         SHA-256 checksum:<BR>
 70.         <INPUT size="65" value="${sum}">
 71.       </P>
 72.     </FORM>
 73.   </BODY>
 74. """
 75. }
 76.
 77. class RequestHandler(BaseHTTPServer.BaseHTTPRequestHandler):
 78.     def do_GET(self):
 79.         url = urlparse.urlparse(self.path)
 80.         query, string = urlparse.parse_qs(url.query), ''
 81.         if 'string' in query:
 82.             string = query['string'][0]
 83.         msg = templates['checksum'] if url.path == '/checksum/' \
 84.          else templates['interface'] if url.path == '/interface/' \
 85.          else templates['frameset']
 86.         if url.path != '/checksum/':
 87.             msg = msg.replace('${body}', templates['body'])
 88.         msg = msg.replace('${string}', string.replace('&', '&#38;').\
 89.               replace('>', '&#62;').replace('"', '&#34;'), 1).\
 90.               replace('${sum}', hashlib.sha256(string).hexdigest(), 1)
 91.         self.send_response(200)
 92.         self.send_header('Content-Type', 'text/html;charset=us-ascii')
 93.         self.send_header('Content-Script-Type', 'application/ecmascript')
 94.         self.send_header('Content-Length', len(msg))
 95.         self.end_headers()
 96.         self.wfile.write(msg)
 97.
 98. if __name__ == '__main__':
 99.     server = SocketServer.TCPServer(('', 8080), RequestHandler)
100.     server.serve_forever()
```



Rys. 1. Odwzorowanie aplikacji WWW z listingu 1. w aktywnym kliencie WWW z marca 1996 roku (Netscape Navigator 2.0)



Rys. 2. Odwzorowanie aplikacji WWW z listingu 1. w aktywnym kliencie WWW z lutego 2010 roku (Mozilla Firefox 3.5.8)



Rys. 3. Odwzorowanie aplikacji WWW z listingu 1. w pasywnym kliencie WWW z sierpnia 1995 roku (Netscape Navigator 1.22)

Rys. 4. Odwzorowanie aplikacji WWW z listingu 1. w działającym w trybie tekstowym, pasywnym kliencie WWW z lipca 2009 roku (ELinks 0.12pre5)

4. Podsumowanie

W artykule zaproponowano podział klientów WWW na pasywne klienty WWW i aktywne klienty WWW, a następnie dokonano przeglądu literatury, w której sygnalizowano zalety i wady stosowania aktywnych klientów WWW. Na autorskim przykładzie pokazano, w jaki sposób aplikacja WWW, pozostając dostępną dla pasywnych klientów WWW, może wykorzystywać możliwość wykonywania kodu przekazywalnego, gdy korzysta z niej aktywny klient WWW. Działa ona zgodnie z oczekiwaniami zarówno we współczesnych klientach WWW, jak i w klientach WWW sprzed ponad piętnastu lat.

Wprowadzony podział jest niezbędny do sformułowania założeń projektowych realizowanej pracy badawczej, polegającej na zaproponowaniu, opisaniu i wdrożeniu stylu architektonicznego, którego zastosowanie pozwoli aplikacji WWW na decydowanie o swojej przekazywalności. W zawartym w niniejszej pracy przykładzie aplikacja WWW nie decydowała o swojej przekazywalności, gdyż przekazywalność pozostawała niezmienna (kod przekazywalny stanowiły kod z linii 32–38 oraz 57–61), jednakże pokazano, iż wykorzystywanie sposobności wykonywania kodu przekazywalnego możliwe jest bez wykluczenia pasywnych klientów WWW.

Literatura

[1] AVIDAN A., WEISSMAN C., SPRUNG C., *An Internet Web Site as a Data Collection Platform for Multicenter Research*, Anesthesia & Analgesia, 2 (100), 2005, pp. 506 – 511.

[2] BERNERS-LEE T., FIELDING R., MASINTER L., *Uniform Resource Identifier (URI): Generic Syntax*, IETF, 2005.

[3] BURKEY J., KUECHLER W. L., *Web-Based Surveys for Corporate Information Gathering: A Bias-Reducing Design Framework*, IEEE Transactions on Professional Communication, 2 (46), 2003, pp. 81 – 93.

[4] CALLEGATI F., RAMILLI M., *Frightened by Links*, IEEE Security and Privacy, 6 (7), 2009, pp. 72 – 76.

[5] COHEN F., *Computer viruses: Theory and experiments*, Computers & Security, 1 (6), 1987, pp. 22 – 35.

[6] CONNORS A., SULLIVAN B., *Mobile Web Application Best Practices*, W3C, 2010.

[7] *ECMAScript Language Specification*, Ecma International, Geneva, 2009.

[8] ELAHI G., YU E., ZANNONE N., *A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities*, Requirements Engineering, 1 (15), 2010, pp. 41 – 62.

[9] FIELD J., MARINESCU M., STEFANSEN C., *Reactors: A data–oriented synchronous/asynchronous programming model for distributed applications*, Theoretical Computer Science, 2–3 (410), 2009, pp. 168 – 201.

[10] FIELDING R., GETTYS J., MOGUL J., FRYSTYK H., MASINTER L., LEACH P., BERNERS-LEE T., *Hypertext Transfer Protocol – HTTP/1.1*, IETF, 1999.

[11] GHOSH A. K., SWAMINATHA T. M., *Software security and privacy risks in mobile e-commerce*, Commun. ACM, 2 (44), 2001, pp. 51 – 57.

[12] GUPTA S., KAISER G. E., GRIMM P., CHIANG M. F., STARREN J., *Automating Content Extraction of HTML Documents*, World Wide Web: Internet and Web Information Systems, 2 (8), 2005, pp. 179 – 224.

[13] GUST-BARDON R., *The Notion of Transferability of a Web Application*, (maszynopis).

[14] HANSEN E., JACOBS I., GUNDERSON J., *User Agent Accessibility Guidelines 1.0*, W3C, 2002.

[15] *IEEE STD 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology*, IEEE, New York, 1990.

[16] JACOBS I., WALSH N., *Architecture of the World Wide Web, Volume One*, W3C, 2004.

[17] KESTEREN A. VAN, *XMLHttpRequest*, W3C, 2010.

[18] MAGRABI F., LOVELL N. H., CELLER B. G., *A web-based approach for electrocardiogram monitoring in the home*, International Journal of Medical Informatics, 2 (54), 1999, pp. 145 – 153.

[19] MANOLA F., *Technologies for a Web Object Model*, IEEE Internet Computing, 1 (3), 1999, pp. 38 – 47.

[20] MCCOWN F., MARSHALL C. C., NELSON M. L., *Why Web Sites Are Lost (and How They're Sometimes Found)*, Commun. ACM, 11 (52), 2009, pp. 141 – 145.

[21] MORRISON M., MORRISON J., KEYS A., *Integrating Web Sites and Databases*, Commun. ACM, 9 (45), 2002, pp. 81 – 86.

[22] PLUMMER D. C., *An Ethernet Address Resolution Protocol–or–Converting Network Protocol Addresses*, IETF, 1982.

[23] RAGGETT D., LE HORS A., JACOBS I., *HTML 4.01 Specification*, W3C, 1999.

[24] RAY H. T., VEMURI R., KANTUBHUKTA H. R., *Toward an Automated Attack Model for Red Teams*, IEEE Security and Privacy, 4 (3), 2005, pp. 18 – 25.

[25] SADAN Z., SCHWARTZ D., *WhiteScript: Using social network analysis parameters to balance between browser usability and malware exposure*, Computers & Security, 1 (30), 2011, pp. 4 – 12.

[26] *Secure Hash Standard (SHS)*, NIST, Gaithersburg, 2008.

[27] STIEGER S., REIPS U., *What are participants doing while filling in an online questionnaire: A paradata collection tool and an empirical study*, Computers in Human Behavior, 6 (26), 2010, pp. 1488 – 1495.

[28] STONE-GROSS B., COVA M., GILBERT B., KEMMERER R., KRUEGEL C., VIGNA G., *Analysis of a Botnet Takeover*, IEEE Security and Privacy, 1 (9), 2011, pp. 64 – 72.

[29] TURING A. M., *Computability and λ-Definability*, The Journal of Symbolic Logic, 4 (2), 1937, pp. 153–163.

[30] TURING A. M., *On Computable Numbers, with an Application to the Entscheidungsproblem*, Proceedings of the London Mathematical Society, 1(s2□42), 1937, pp. 230 – 265.

[31] VUL E., PASHLER H., *Incubation benefits only after people have been misdirected*, Memory & Cognition, 4 (35), 2007, pp. 701 – 710.

[32] WEIDEMAN M., SCHWENKE F., *The influence that JavaScript has on the visibility of a Website to search engines–a pilot study*, Information Research–An International Electronic Journal, 4 (11), 2006.

[33] WU Q., ZHU M., GU Y., RAO N. S. V., *System Design and Algorithmic Development for Computational Steering in Distributed Environments*, IEEE Transactions on Parallel and Distributed Systems, 4 (21), 2010, pp. 438 – 451.

[34] WUSTEMAN J., O'HICEADHA P., *Using Ajax to Empower Dynamic Searching, Information Technology and Libraries,* 2 (25), 2006, pp. 57 – 64.

[35] ZAGEL J. B., *Eolas Technologies, Inc. and The Regents of the University of California, Plaintiffs, V. Microsoft Corporation, Defendant. Ruling on the Defense of Inequitable Conduct*, United States District Court Northern District of Illinois Eastern Division, Chicago.

[36] ZDRNJA B., *Malicious JavaScript Insertion through ARP Poisoning Attacks*, IEEE Security and Privacy, 3 (7), 2009, pp. 72 – 74.

The Classification of Web Clients with regard to the Transferability of a Web Application

ABSTRACT: The World Wide Web (WWW) is “an information space in which the items of interest, referred to as resources, are identified by global identifiers called Uniform Resource Identifiers” (URIs). A Web agent is “a person or a piece of software acting on the information space on behalf of a person, entity, or process.” A Web client is a Web agent that attempts to access a resource referenced by a URI. Transferability is the extent to which a Web application constitutes of transferable code, which is computer instructions and data definitions intended to be executed by a Web client or by another process on behalf of a Web client. In this paper, we define two types of Web clients–the active ones and the passive ones–with regard to their readiness to execute transferable code or to entrust another process to execute it on their behalf. We then proceed to show on a simple example that it is possible to write a Web application that is not only accessible to both an active Web client and a passive Web client, but also automatically takes advantage of the capability to execute a transferable code. This example proves that active Web clients were available in March 1996 at the latest and the presented method was applicable back then.

KEYWORDS: client, transferability, WWW

Praca wpłynęła do redakcji: 26.02.2011