

VIII Seminarium PLOUG
Warszawa
Kwiecień 2003

Aplikacje internetowe – Przegląd zagrożeń

Wojciech Dworakowski
(*wojciech.dworakowski@securing.pl*)

SecuRing

Integracja baz danych z serwisami WWW oraz stosowanie dynamicznych metod generowania stron dają praktycznie nieograniczone możliwości prezentowania danych poprzez strony WWW. Współczesne portale i serwisy WWW to już nie statyczne strony, ale prawdziwe aplikacje, gdzie użytkownik może interaktywnie korzystać z zasobów udostępnionych przez twórców serwisu WWW. Coraz częściej zamiast określenia „strona WWW” używa się „aplikacja WWW”. Technologia ta, a właściwie – zbiór różnych technik i technologii, dają bardzo duże możliwości, ale jej niewłaściwe stosowanie może wprowadzić znaczne zagrożenia dla udostępnianych danych a nawet dla całości infrastruktury IT. Prezentacja ma na celu uświadomienie tych zagrożeń. Jest ona przeznaczona zarówno dla developerów, by dysponując wiedzą o zagrożeniach mogli ich unikać w tworzonych przez siebie aplikacjach, ale przede wszystkim dla osób odpowiedzialnych za utrzymanie bezpieczeństwa oraz administratorów serwerów aplikacyjnych i baz danych, by wiedzieli, jakim nowym zagrożeniom muszą stawić czoła.

Przedmiotem dalszych rozważań na temat bezpieczeństwa będą same aplikacje WWW. Przez pojęcie to rozumiem kod, który wykonuje się na serwerze aplikacji i komunikuje się z klientami za pomocą protokołu HTTP. W przypadku technologii Oracle będą to przede wszystkim aplikacje Java i PL/SQL udostępniane przez Oracle HTTP Server. Metody atakowania aplikacji opisane w dalszej części, będą przedstawione na przykładzie aplikacji webowych, jednakże warto zauważyć, że identyczne techniki można stosować dla praktycznie dowolnych aplikacji, niezależnie od platformy oraz architektury.

Zebrany tutaj materiał jest wynikiem doświadczeń zespołu firmy SecuRing zdobytych podczas testowania bezpieczeństwa aplikacji internetowych. Omawiając poszczególne sposoby ataków nie będę posługiwał się przykładami z życia wziętymi, gdyż mogłoby to narazić na szkodę serwisy podatne na poszczególne zagrożenia. Prezentowane przykłady są zaczerpnięte z powszechnie dostępnych materiałów na temat bezpieczeństwa aplikacji. Czytelników zachęcam do sięgnięcia do tych źródeł.:

- OWASP Guide (<http://www.owasp.org/guide/>)
- WebGoat (<http://www.owasp.org/webgoat/>) – przykładowa aplikacja J2EE, prezentująca w praktyce wiele różnych zagrożeń, stworzona dla celów edukacyjnych.
- NGSec Game (<http://quiz.ngsec.biz:8080/>) – „gra” polegająca na włamywaniu się do aplikacji webowej.

Trywialne zagrożenia

Część zagrożeń wynika z trywialnych błędów popełnionych przez projektantów i programistów. Jednym ze źródeł takich podstawowych zagrożeń mogą być informacje w komentarzach w generowanym przez aplikację kodzie HTML. Bywa, że komentarze takie stosuje się w fazie rozwoju aplikacji a potem zostają one również w wersji produkcyjnej. Trzeba pamiętać o tym, że źródło HTML jest dostępne dla klienta a więc również dla potencjalnego intruza.

Wydaje się, że zagrożenie to jest tak oczywiste dla wszystkich, że niemal niespotykane w rzeczywistości. Jednak informacje znajdowane przez nasz zespół w komentarzach HTML i znacznikach META różnych serwisów nie przestają wprawiać nas w zdziwienie.

Ukryte parametry

Równie trywialnym atakiem jest wykorzystanie ukrytych parametrów formularzy HTML (input type=hidden). Parametry te są z reguły w jakiś sposób wykorzystywane przez aplikację działającą na serwerze. Co prawda, nie są to parametry bezpośrednio edytowalne przez użytkownika, jednakże bardzo łatwo jest je modyfikować. Często zawierają one istotne zmienne. Podstawową metodą ingerowania w te parametry, jest zapisanie źródła strony, zmodyfikowanie parametru w źródle, uruchomienie zmodyfikowanej strony i wysłanie atakowanego formularza. Jednak w praktyce,

najczęściej stosuje się programy przechytujące po stronie przeglądarki wysyłane zlecenia i potrafiące je modyfikować w locie. Przykładami takich narzędzi są AtStake WebProxy (<http://www.webproxy.com>) oraz Odysseus (<http://www.wastelands.gen.nz/odysseus/>).

Przykład:

Za przykład niech posłuży hipotetyczna aplikacja obsługująca sklep internetowy. Użytkownik dokonuje zakupów, wybierając określone towary. Towary te są dokładane do wirtualnego „koszyka”, jednocześnie jest zwiększana o odpowiednią kwotę zmienna przechowująca wartość zakupionych towarów. Zagrożenia pojawia się jeśli zmienna ta jest przechowywana po stronie klienta np. w ukrytym polu formularza HTML. Gdy klient jest gotowy do zapłacenia, aplikacja generuje odpowiednią stronę z podsumowaniem transakcji:

Confirm purchase: **46 inch HDTV (model KTV-551)**

Klient jest proszony o potwierdzenie. Po kliknięciu klawisza „Purchase”, do serwera są przekazywane parametry transakcji – w tym wartość!

Formularz jest generowany przez aplikację działającą po stronie serwera. Przykładowy kod Java odpowiedzialny za wygenerowanie tego formularza:

```
ec.addElement(new P().addElement("Confirm purchase: "));
ec.addElement(new B("46 inch HDTV (model KTV-551)"));
Input input = new Input(Input.HIDDEN, PRICE, "4999.99");
ec.addElement(input); ec.addElement(new BR());
Element b = ECSFactory.makeButton("Purchase");
ec.addElement(b);
```

Oraz wygenerowany przez niego kod HTML obsługujący formularz:

```
Confirm purchase: <b> 46 inch HDTV (model KTV-551)</b>
<input name="Price" type="HIDDEN" value="4999.99"><br>
<input type="SUBMIT" value="Purchase">
```

W momencie naciśnięcia przycisku „Purchase”, ukryty parametr „Price” jest przekazywany do aplikacji metodą POST. Wystarczy zmodyfikować w locie (za pomocą WebProxy czy Odysseus-a) wartość tego parametru w zleceniu POST na dużo niższą (np. 1) i gotowe:

Your total price is: **\$1**

This amount will be charged to your credit card immediately.

Taka technika nazywa się wirtualnym „przeklejeniem ceny”, gdyż sama istota ataku jest podobna do prostych oszustw stosowanych kiedyś w zwykłych sklepach.

Innym wariantem tego zagrożenia, jest pobieranie istotnych wartości z cookie użytkownika. Programista nigdy nie powinien ufać wartościom pobranym ze środowiska użytkownika. Nawet, gdy je sam tam wcześniej umieścił.

Doklejanie parametru

Podobnym w swojej istocie typem zagrożenia jest brak inicjowania wartości zmiennych i brak sprawdzania ilości pobieranych parametrów.. W niektórych językach szablonowych, działających po stronie serwera aplikacji (np. PHP) zmienne pobierane od użytkownika są automatycznie rejestrowane jako zmienne globalne. Jeżeli intruz dopisze do zlecenia kolejny parametr i jeśli zmienna o takiej samej nazwie jest używana wewnętrznie przez daną procedurę, to może to spowodować zmianę logiki działania procedury.

Przykład:

Za uwierzytelnienie użytkownika odpowiada następujący kod:
(pseudokod ilustrujący algorytm)

```
if (sprawdz_uzytkownika_w_bazie($login, $password)) {
    $state="OK";
}
...
if ($state=="OK") {
    UWIERZYTELNIENIE OK
} else { BŁĄD ! }
```

Jak widać za pomyślnie uwierzytelnienie odpowiada ostatecznie wartość flagi „state”. Flaga ta jest wcześniej ustawiana przez procedurę wyszukującą dane logowania w bazie. Jeśli zmienna „state” nie jest inicjowana przed użyciem, to jest możliwe arbitralne ustawienie jej wartości w zleceniu przesyłanym przez klienta:

http://serwer_aplikacji/login.php?login=admin&password=xxx&state=OK

Niezależnie od wartości zmiennych login i password, intruz zostanie pomyślnie uwierzytelniony.

Oczywiście zastosowanie tej metody nie ogranicza się tylko do procedur uwierzytelniających. Jedynym problemem dla atakującego jest odgadnięcie nazw zmiennych.

Problem ten może również dotyczyć JSP! Jest to możliwe, jeżeli aplikacja korzysta z komponentu JavaBean i inicjuje wszystkie wartości parametrów komponentu ze zlecenia HTTP, przy użyciu metaznaku * . Np:

```
<jsp:useBean id="myBasket" class="BasketBean">
    <jsp:setProperty name="myBasket" property="*" />
</jsp:useBean>
```

Tego typu komponent będzie podatny na ataki polegające na dopisaniu wartości dodatkowego parametru. Można to wykorzystać nie tylko do zmieniania logiki działania aplikacji, ale również do ataków typu „przeklejanie ceny”.

Brak obsługi błędów (fail open)

Kolejnym często spotykanym zagrożeniem w aplikacjach internetowych jest brak obsługi wyjątków. Programiści często zapominają o obsłudze błędów. Wynika to z tego, że zakładają, iż wszystkie parametry zostaną wprowadzone poprawnie, bo przecież robi to zaprojektowany przez nich formularz. **Pamiętajmy – użytkownik to potencjalny intruz!** – więc należy brać pod uwagę również sytuacje specjalne, wynikające z prób modyfikacji parametrów poza przeglądarką.

Przykładem wykorzystania tego zagrożenia może być atak na mechanizm uwierzytelniający. Typowy mechanizm uwierzytelniania do aplikacji WWW, pobiera od użytkownika dwa parametry: nazwę użytkownika i hasło i przesyła je metodą GET lub POST do aplikacji po stronie serwera. Tam parametry te są obrabiane. Na podstawie wprowadzonych parametrów (nazwa użytkownika, hasło) jest wykonywane odpowiednie zapytanie SQL do bazy użytkowników i decyzja o uwierzytelnieniu jest podejmowana na podstawie liczby zwróconych rekordów.

Atak może polegać np. na usunięciu jednego z parametrów (np. hasła). Program spodziewa się, że klient przekaże dwa parametry: nazwę użytkownika i hasło. Tymczasem dostanie tylko jeden (np. nazwę użytkownika). Jeżeli nie jest obsłużony wyjątek przy pobieraniu parametrów ze zlecenia, to dalsze zachowanie się aplikacji zależy od wielu czynników, np. od wartości początkowej zmiennej „password” oraz przede wszystkim od postaci warunku sprawdzającego rezultaty zapytania i od sposobu konstrukcji samego zapytania do bazy. W szczególnym wypadku może zdarzyć się tak, że taka sytuacja specjalna spowoduje spełnienie odpowiednich warunków „if” i w rezultacie – pomyślnie uwierzytelnienie intruza.

Parametry w cookies

Równie skutecznie można atakować aplikacje manipulując parametrami przekazywanymi w cookies.

Zdarza się że w cookies są przechowywane bardzo istotne dane. Tak jak parametry GET/POST mogą być one modyfikowane przez użytkownika.

Przykład:

Po uwierzytelnieniu, aplikacja ustawia w cookie użytkownika określony parametr, np: „authorized=yes”.

Jeśli aplikacja podejmuje decyzje o autoryzacji do danego zasobu tylko i wyłącznie na podstawie tej flagi, to jest możliwy trywialny atak polegający na ustawieniu tego parametru zgodnie z wymaganiami uwierzytelnionego użytkownika.

```
lang=en-us; AUTHORIZED=yes; y=1 ; time=12:30GMT;
```

Warto zauważyć że niezależnie od tego czy cookie jest przesyłane przez SSL, czy też normalne, daje się je modyfikować w momencie przesyłania przez przeglądarkę, przy pomocy programów typu proxy.

Path traversal, dostęp do plików

Innym skutkiem wynikającym z niewłaściwego walidowania parametrów pobieranych od klienta, jest nieuprawniony dostęp do plików na serwerze.

Często zdarza się że aplikacja wykorzystuje różne parametry pobierane ze środowiska użytkownika, do konstrukcji nazw wykorzystywanych plików. Jeśli te parametry nie są właściwie sprawdzane, to otwiera to drogę do odczytania dowolnych plików na serwerze.

Przykład:

W nagłówku zlecenia HTTP wysyłanego przez przeglądarkę, znajduje się parametr „Accept-Language” określający preferencje językowe klienta. Aplikacja webowa obsługująca różne języki może wykorzystywać ten parametr nagłówka, do zbudowania nazwy pliku z którego należy pobrać tekst we właściwym języku.

Intruz odpowiednio modyfikując parametr „Accept-Language” może doprowadzić do pobrania arbitralnie wybranego pliku z serwera.

Ten typ ataku jest zwykle stosowany w połączeniu z innymi technikami, np. SQL-injection czy też (omówione wcześniej) dopisywanie dodatkowych parametrów.

Parameter injection

Zdarza się, że aplikacja webowa wykorzystuje funkcje systemu operacyjnego do konstruowania stron HTML. Funkcje te są uruchamiane po stronie serwera. Często aplikacja przekazuje do tych funkcji czy poleceń systemowych parametry pobierane od użytkownika. Jeśli parametry te nie są szczegółowo sprawdzane to może to prowadzić do manipulacji funkcjami i komendami systemu operacyjnego.

Przykład:

Aplikacja pobiera od użytkownika nazwę katalogu, a następnie wyświetla jego zawartość. Parametr pobrany od użytkownika jest doklejany do komendy:

```
cmd.exe /c dir /b .....
```

Jeżeli intruz dopisze do parametru znak & to będzie mógł wykonać dowolną komendę systemową. Znak & dla Windows2000 (tak jak znak ; dla Unixów) oddziela komendy systemowe.

SQL injection

Bardzo groźną w skutkach techniką ataku jest SQL-injection. Zagrożenie, które umożliwia tą technikę ataku, wynika ze sposobu konstruowania przez aplikacje zapytań SQL do bazy.

Parametry pobierane ze środowiska użytkownika przez aplikacje WWW najczęściej służą do skonstruowania odpowiedniego zapytania SQL. Rezultaty tego zapytania służą do wygenerowania odpowiedzi w formie dokumentu HTML (lub coraz częściej - XML). Jeżeli zapytanie to jest tworzone w sposób dynamiczny, przez sklejenie parametrów pobranych od użytkownika z odpowiednim zapytaniem SELECT, to manipulowanie parametrami pobieranymi z zewnątrz może doprowadzić do zmodyfikowania wykonywanego zapytania.

Przykład:

Procedura odpowiada za zmianę hasła użytkownika. Nowe hasło jest pobierane z formularza i bezpośrednio wstawiane do wyrażenia:

```
String query = "UPDATE users SET password = '" + haslo + "'  
                WHERE username = '" + uzytkownik + "'";
```

Zwykły użytkownik takiej aplikacji może wejść w posiadanie konta administratora, przez odpowiednie manipulowanie parametrami wpisywanymi w formularz zmiany hasła. Jeśli poda takie wartości parametrów:

Hasło: oops!

Użytkownik: x` OR username LIKE 'admin

(parametr "użytkownik" prawdopodobnie będzie pobierany z ukrytego pola formularza, albo z cookie)

W rezultacie wykona się zapytanie:

```
UPDATE users SET password = 'oops!' WHERE username = 'x' OR username  
LIKE 'admin'
```

Przejęcie sesji

Protokół HTTP odpowiadający za komunikację przeglądarki z aplikacją internetową, nie zna pojęcia sesji (jest protokołem bezstanowym). Każde odwołanie jest traktowane jako osobna sesja HTTP. Aplikacja internetowa musi operować pojęciem sesji, gdyż jej interakcja z użytkownikiem

składa się z wielu odwołań HTTP. W związku z tym to aplikacja musi implementować kontrolę sesji¹.

Na początku sesji użytkownika (po uwierzytelnieniu się do aplikacji, lub przy pierwszym odwołaniu) jest tworzony identyfikator sesji (token), który jest przekazywany do klienta. Później ten identyfikator jest przedstawiany przez klienta dla serwera za każdym razem. W ten sposób serwer wie że dane odwołanie należy do określonej sesji. Najczęściej wykorzystywaną metodą jest przekazywanie tokena sesji w cookie, ale spotyka się również przekazywanie tokena jako parametr GET w URL czy też metodą POST, w ukrytym polu formularza.

Niezależnie od wykorzystywanej metody przekazywania, należy pamiętać o tym że identyfikator sesji jest jednym z parametrów przekazywanym przez użytkownika. W związku z tym jest on narażony na różne ataki, np:

- manipulacje identyfikatorem przez klienta
- podsłuch
- podszywanie się przy użyciu podsłuchanego lub przewidywalnego identyfikatora sesji.

Identyfikator sesji ściśle określa użytkownika w aplikacji. W związku z tym podsłuchanie lub odgadnięcie tego identyfikatora może prowadzić do omińnięcia mechanizmów uwierzytelniających lub przechwycenia sesji użytkownika.

Dobre praktyki dotyczące bezpieczeństwa identyfikatorów sesji mówią o kilku podstawowych zasadach bezpieczeństwa:

1. Identyfikator powinien być: unikalny dla każdej sesji i nieprzewidywalny
Przewidzenie identyfikatora sesji pozwala na omińnięcie mechanizmów uwierzytelniania w aplikacji oraz podszywanie się pod innego użytkownika. Do generowania identyfikatorów sesji powinien zostać użyty silny algorytm losowy. Przy czym przestrzeń generowanych identyfikatorów powinna być bardzo duża, by zapobiec zgadywaniu prawidłowych i aktywnych identyfikatorów.
2. Identyfikator powinien być związany z daną instancją klienta
Takie powiązanie dodatkowo zabezpiecza przed przechwytywaniem sesji oraz atakami przez powtórzenie uprzednio podsłuchanej sesji (np. Sesji w której klient dokonuje zakupów przez Internet).
3. Identyfikator powinien być przekazywany w sposób bezpieczny, utrudniający jego podsłuchanie. Cookies mogą być „bezpieczne” – wtedy każdorazowo są przekazywane przy użyciu szyfrowania SSL.

Cross-Site Scripting

Dotychczasowe przykłady ataków skupiały się na atakowaniu serwera (lub raczej aplikacji internetowej). Istnieją również możliwości atakowania klienta aplikacji internetowej. Po co? Jak już wcześniej wyjaśniłem, w przeglądarce klienta są przechowywane istotne dane, na przykład cookies zawierające identyfikatory sesji, hasła zapamiętane w przeglądarce, itp. Poza tym można spróbować zmusić przeglądarkę do załadowania z serwera należącego do intruza, kodu wykorzystującego różne słabości przeglądarek.

Tego typu ataki na stronę klienta mogą być wykonane przy użyciu techniki cross-site scripting. Ogólnie – ten typ ataku polega na zmuszeniu klienta (przeglądarki) do załadowania ustalonego przez intruza adresu URL, z tym że ofiara nie powinna się dowiedzieć, że jej przeglądarka załadowała ten URL. Przy tego typu atakach występują trzy strony: atakujący, ofiara oraz nosiciel. Nosiciel to medium, którego w sposób świadomy używa ofiara, a które jest wykorzystywane do nieświadomego załadowania URL-a wybranego przez intruza. Nosicielem może być np:

- Odpowiednio sformatowany list e-mail zawierający skrypt ładujący wyznaczony URL
- Strona WWW, do której odwiedzenia należy przekonać ofiarę

¹ Zwykle za kontrolę sesji odpowiada warstwa pośrednicząca wchodząca w skład danego serwera aplikacji.

- Serwis internetowy prezentujący treść w formie HTML, do którego intruz jest w stanie dopisać nowe wiadomości i kod HTML (np. chat-room).

Przykład:

Aplikacja (nosiciel) obsługuje forum internetowe. Użytkownicy przez formularz WWW, mogą umieszczać swoje wiadomości. Jeżeli tekst wpisywany przez użytkownika jest potem wprost umieszczany na stronie WWW przez aplikację obsługującą forum, to można w nich umieścić tagi HTML. Jeżeli intruz umieści we wpisywanej wiadomości kod JavaScript, to kod ten zostanie wykonany na przeglądarce każdego z użytkowników przeglądających tą wiadomość. Kod ten może wykonywać wrogie operacje.

Przykłady kodu będącego elementem ataków cross-site scripting:

```
<script>document.write('')</script>
```

Załadowanie dowolnego URL (w szczególności skryptu). Dzięki ukryciu w tagu , akcja pozostaje niewidoczna dla ofiary.

```
<script>document.write('
```

Wykradnięcie cookie sesji. Dane z cookie zostaną zapisane w logach serwera "zly.com".

Możliwości ataków przy wykorzystaniu techniki cross-site scripting jest bardzo wiele. Często celem ataku jest administrator danej aplikacji. Np. Znany jest atak cross-site scripting na serwer WWW iPlanet. Program zarządzający tym serwerem jest aplikacją WWW. Gdy administrator przegląda logi serwera, czyni to przez odpowiedni interfejs WWW. Wystarczy że atakujący wykona zlecenie HTTP, w którym będzie zaszyty odpowiedni skrypt. Serwer cytuje całość zleceń w swoich logach. Gdy administrator będzie przeglądał logi, kod intruza wykona się w jego przeglądarce.

Buffer Overflow

Buffer overflow jest jedną z najczęściej obecnie wykorzystywanych technik ataku na różnego rodzaju oprogramowanie (w tym Oracle). Zagrożenie, które jest wykorzystywane przez tą technikę wiąże się z brakiem sprawdzania przez aplikacje długości wprowadzanych danych.

Np. Jeżeli programista zadeklarował w swojej aplikacji na zmienną „nazwisko” 50 bajtów, a użytkownik wpisze więcej znaków, to nastąpi przepełnienie bufora. W rezultacie, dane które są dalej w pamięci zostaną nadpisane. Z reguły prowadzi to do wystąpienia błędów w aplikacji, jednak jeśli intruz przewidzi strukturę danych, które są w pamięci po podanej zmiennej, to przepełniając bufor podanej zmiennej jest w stanie ingerować w wartości następnych zmiennych.

W praktyce tą technikę używa się zwykle do przeprowadzenia ataku, którego skutkiem jest wykonanie kodu intruza na atakowanym serwerze. Jedną z danych umieszczanych w pamięci (na stosie) za zmiennymi procedury, jest adres powrotu z procedury. Intruz nadpisując dane w pamięci zmienia ten adres tak, żeby wskazywał adres jego procedury, którą umieszcza na początku atakowanego bufora.

Ataki buffer-overflow można przeprowadzać również przeciw aplikacjom webowym. Jednakże większość platform, na których są budowane aplikacje WWW jest odporna na tego typu ataki. W szczególności Java nie jest podatna na ataki związane z manipulacją pamięcią² (w tym buffer-overflow). Można natomiast atakować w ten sposób aplikacje oparte np. o programy CGI.

² Więcej informacji o zabezpieczeniach platform Java: <http://www.app-serv.com/security.html>

Ukrywanie tożsamości

Wydawać by się mogło, że atakowanie aplikacji WWW jest dość skomplikowane. Wymaga sporo eksperymentowania. Aplikacje internetowe są zreguły unikalne i jedynym miejscem gdzie intruz może eksperymentować jest atakowany serwis. Ponadto aplikacje internetowe są oparte o serwery WWW, w związku z tym każde odwołanie jest zapisywane w logach. Wydawać by się mogło, że atakowanie aplikacji internetowej pozostawia w logach mnóstwo informacji, która może pomóc w schwyтaniu intruza.

W związku z powyższym, intruzi używają metod ukrywania swojej tożsamości. Przede wszystkim nie dokonują ataków bezpośrednich, lecz korzystają ze źle skonfigurowanych serwerów proxy obecnych w Internecie, które umożliwiają korzystanie z proxy dla każdego komputera³. Atakowany serwer aplikacji zanotuje ruch od takiego serwera proxy, a nie od prawdziwego źródła ataku.

Podsumowanie

Aplikacje internetowe to niewątpliwie bardzo nowoczesne i użyteczne narzędzie. Jednak z korzystaniem z ich dobrodziejstw wiąże się spore ryzyko. Na dodatek tradycyjne środki ochrony, takie jak firewalle i systemy wykrywania włamań (IDS) w tym wypadku zawodzą. Wykrywanie potencjalnych zagrożeń w istniejących już aplikacjach jest procesem pracochłonnym i wymagającym bardzo dużego doświadczenia. W zasadzie jedyną metodą zabezpieczania się jest w tym wypadku ... niepopelnianie błędów. Wiąże się to z edukowaniem developerów aplikacji internetowych w technikach bezpiecznego kodowania aplikacji i utrzymywaniem tej wiedzy na wysokim poziomie aktualności.

³ Jeden z wielu spisów otwartych serwerów proxy znajduje się pod adresem: <http://www.samair.ru/proxy/>