Koncepcja zwiększenia bezpieczeństwa transakcyjnych aplikacji internetowych

Łukasz STRZELECKI, Andrzej LUDEW

Instytut Teleinformatyki i Automatyki, Wydział Cybernetyki WAT,
ul. Gen. W. Urbanowicza 2, 00-908 Warszawa 46
lukasz.strzelecki@wat.edu.pl, andrzej.ludew@wat.edu.pl

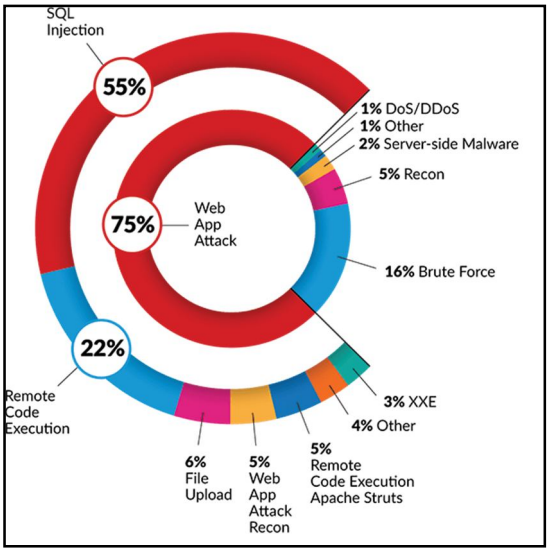
STRESZCZENIE: W artykule przedstawiono nowatorską koncepcję podniesienia bezpieczeństwa istotnych aplikacji internetowych wykorzystywanych np. podczas realizacji operacji finansowych, poprzez niestandardowe wykorzystanie mechanizmów języka HTML5 oraz tzw. *przetwarzania po stronie serwera*. Omówiono zalety proponowanego podejścia oraz wskazano jego potencjalne wady.

SŁOWA KLUCZOWE: bezpieczeństwo, witryny WWW, aplikacje internetowe, HTML5, Canvas

1. Wprowadzenie

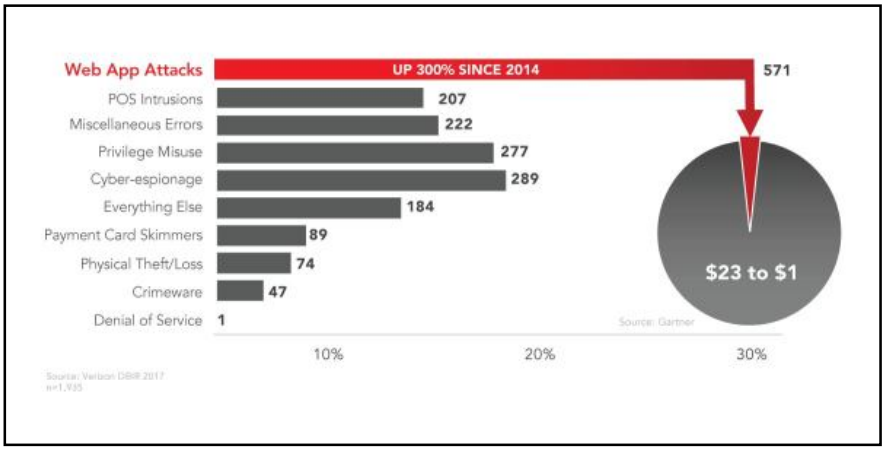
Kwestie związane z bezpieczeństwem teleinformatycznym oraz stale wzrastającą liczbą odnotowywanych incydentów teleinformatycznych są szeroko poruszane w literaturze przedmiotu. Warto jednakże zwrócić uwagę, że zdecydowana większość spośród nich związana jest z tzw. *aplikacjami internetowymi*, rozumianymi jako interaktywne oprogramowanie działające po stronie serwera, komunikujące się z użytkownikami przy wykorzystaniu przeglądarek internetowych. Według danych firmy *Alert Logic* oferującej rozwiązanie typu SOC, ataki na aplikacje internetowe stanowiły 75% wszystkich incydentów odnotowanych przez oferowane przez nią rozwiązania[1].

[1] Źródło: https://www.alertlogic.com/solutions/activewatch-managed-detection-and-response/



Rys. 1 Typy ataków wykryte przez rozwiązania firmy *Alert Logic* (źródło: *https://alertlogic.com*)

Warto dodać, iż wspomniana firma powołuje się również na raport *„2017 Data Breach Investigations Report*[2]*”* opracowany przez firmę *Verizon*. Zgodnie z informacjami zawartymi we wspomnianym dokumencie, udział ataków na aplikacje internetowe stale wzrasta, a od 2014 roku zwiększył się ponad 3-krotnie.



Rys. 2 Zestawienie typów ataków odnotowanych przez firmę Verizon (źródło: *https://alertlogic.com*)

[2] Źródła: https://www.ictsecuritymagazine.com/wp-content/uploads/2017-Data-Breach-Investigations-Report.pdf
https://www.alertlogic.com/solutions/why-alert-logic/full-stack-security/

Powyższe informacje potwierdzają również raporty „*Krajobraz bezpieczeństwa Polskiego Internetu”* opublikowany w 2016 roku przez *CERT Polska*[3] oraz „*Raport CERT Orange Polska za rok 2017”*[4], w których opisane są liczne incydenty związane z atakami na popularne aplikacje internetowe.

Uwzględniając powyższe, w niniejszym artykule zaproponowano nowatorską metodę budowy i zabezpieczania szczególnie wrażliwych aplikacji internetowych, za które należy uznać m.in. serwisy transakcyjne banków (tzw. *bankowość internetowa*) dostępne dla standardowych użytkowników. Koncentracja na tym konkretnym typie aplikacji internetowych wynika bezpośrednio z ich specyfiki (konieczność logowania się użytkowników) oraz istotności, która przekłada się bezpośrednio na środki, które poszczególne instytucje są w stanie poświęcić na zwiększenie ich bezpieczeństwa (np. poprzez ich przebudowę do proponowanego modelu działania). W dalszej części artykułu wspomniany typ aplikacji internetowych będzie nazywany *transakcyjnymi aplikacjami internetowymi,* natomiast zawartość renderowana i wyświetlana bezpośrednio w przeglądarce użytkownika będzie nazywana *witryną internetową* (niezależnie od tego w jaki sposób jej kod został wytworzony[5]).

2. Ogólna budowa tradycyjnych aplikacji internetowych

W pewnym uproszczeniu, abstrahując od technologii wykorzystywanych zarówno po stronie serwera, jak i klienta można przyjąć, że:

a) przeglądarka internetowa wysyła do serwera WWW asynchroniczne żądanie[6],
b) serwer odpowiadając na otrzymane żądanie dostarcza przeglądarce internetowej użytkownika treść, która jest renderowana i wyświetlana

[3] Raport CERT Polska jest dostępny pod adresem URL: https://www.cert.pl/PDF/Raport_CP_2016.pdf

[4] Raport CERT Orange Polska jest dostępny pod adresem URL: https://cert.orange.pl/opracowania

[5] Tzn. niezależnie od tego, czy bazuje ona na statycznych plikach HTML zamieszczonych na serwer, czy jej kod HTML jest dynamicznie generowany przez (transakcyjną) aplikację internetową.

[6] Protokół HTTP nie przewiduje utrzymywania sesji użytkownika ani rozróżniania sekwencji żądań i stąd wszystkie żądania wykonywane z jego wykorzystaniem mają charakter asynchroniczny. Warto natomiast nadmienić, że istnieją rozwiązania, które zapewniają utrzymywanie stanu sesji/widoku użytkownika, np. *ViewState*.

w przeglądarce klienta lub w inny sposób wpływa na działanie witryny wyświetlanej w oknie przeglądarki internetowej użytkownika[7].

Wysłanie żądania od klienta (przeglądarki internetowej) do serwera zazwyczaj spowodowane jest jednym z trzech podstawowych zdarzeń:

a) użytkownik odwołał się do adresu URL zamieszczonego na wyświetlanej mu witrynie;
b) skrypt języka *JavaScript* funkcjonujący w ramach witryny wyświetlanej użytkownikowi automatycznie wykonał żądanie (np. w celu zaktualizowania danych wyświetlanych użytkownikowi);
c) użytkownik wypełnił i zatwierdził formularz HTML znajdujący się na wyświetlanej mu witrynie.

W każdym z powyższych przypadków żądanie wysyłane jest do serwera (przez przeglądarkę użytkownika) z wykorzystaniem odpowiedniej metody HTTP (m.in. GET, POST itp.). Ponadto, jeśli do przesłanego żądania dołączone były parametry (których oczekiwała aplikacja działająca po stronie serwera), to są one dodatkowo przetwarzane. Warto zwrócić uwagę, że to właśnie na tym etapie popełnianych jest przez programistów aplikacji internetowych najwięcej błędów[8] i w związku z tym należy się spodziewać występowania tutaj największej liczby podatności na ataki. Ze względu na powyższe duża część testów penetracyjnych aplikacji internetowych koncentruje się na weryfikacji poprawności przetwarzania danych przesyłanych do serwera (i osadzonej na nim aplikacji internetowej). Należy też zauważyć, że sposób działania zdecydowanej większości tzw. *automatycznych skanerów bezpieczeństwa* przeznaczonych dla aplikacji internetowych sprowadza się głównie do symulowania działania przeglądarki internetowej użytkownika i przesyłania (podszywając się pod nią) do serwera różnego typu ciągów formatujących, które mają na celu zweryfikowanie, czy aplikacja działająca po stronie serwera poprawnie je obsłużyła (albo zwróciła komunikat pozwalający wnioskować o występowaniu luki bezpieczeństwa).

Analizując sposób działania tradycyjnych aplikacji internetowych warto także zwrócić uwagę, że zazwyczaj po stronie serwera działa jeden program omawianego typu i samodzielnie obsługuje wszystkie żądania użytkowników do niego kierowane[9]. W celu rozróżnienia żądań kierowanych od poszczególnych

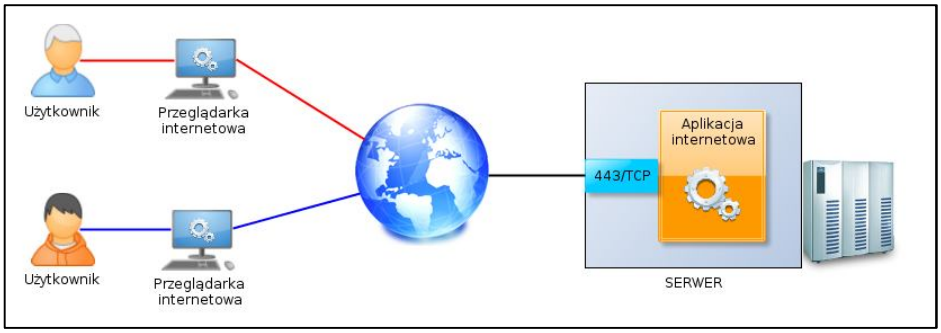
[7] Opisany sposób działania znajduje zastosowanie zarówno w przypadku statycznych stron WWW, jak i w przypadku aplikacji internetowych korzystających np. z technologii *Asynchronous JavaScript and XML* (AJAX).

[8] Do najczęstszych błędów należy brak odpowiedniego, bezpiecznego przetwarzania danych pochodzących od użytkownika, tj. przyjmowane jest często założenie, że dane przesłane do serwera są poprawne.

[9] W przypadku aplikacji uruchamianych oddzielnie dla każdego żądania należy zwrócić uwagę, że w rzeczywistości sprowadza się to do działania tego samego kodu w tym samym środowisku, a zatem można tę sytuację interpretować, jako działanie pojedynczej aplikacji.

użytkowników wykorzystywany jest mechanizm tzw. *ciasteczek*, czyli niewielkich, unikalnych ciągów znaków zapisywanych w pamięci przeglądarki internetowej użytkownika. Wspomniany ciąg jest dołączany do każdego żądania wysyłanego przez przeglądarkę do serwera i na tej podstawie konkretny użytkownik może zostać zidentyfikowany. Dla porządku należy zaznaczyć, że wspomniane *ciasteczka* również są częstym celem ataków, gdyż ich przechwycenie umożliwia podszycie się pod (zalogowanego do danej witryny) użytkownika.

W celach poglądowych ogólny schemat sposobu działania tradycyjnych aplikacji internetowych został przedstawiony na rysunku 3.



Rys. 3 Schemat poglądowy sposobu działania tradycyjnych aplikacji internetowych (źródło: opracowanie własne)[10]

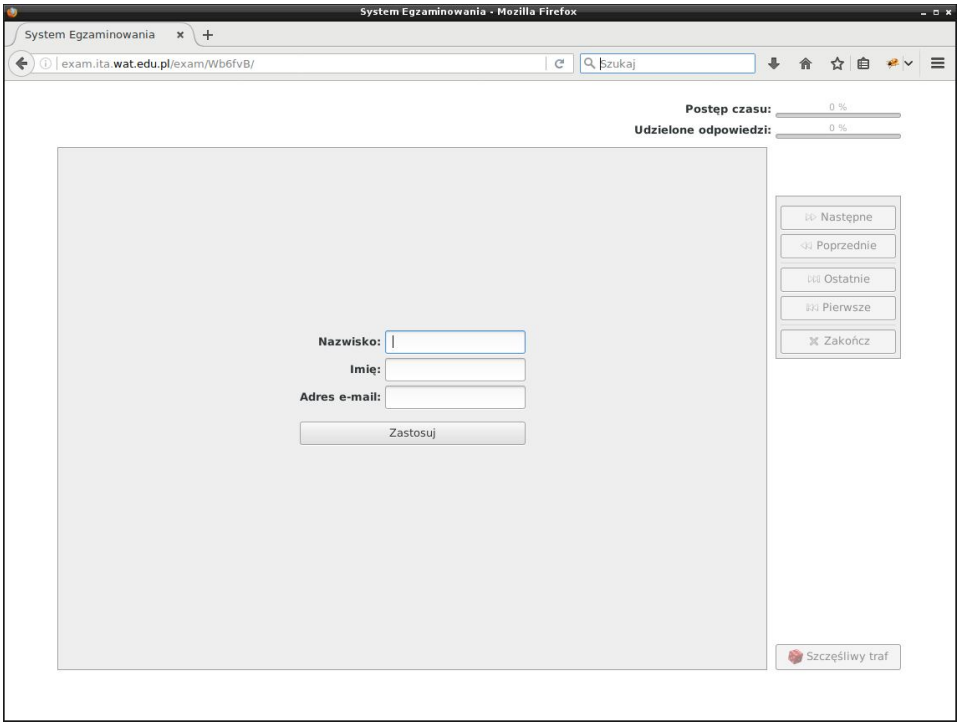
3. Koncepcja zwiększenia bezpieczeństwa transakcyjnych aplikacji internetowych

Wraz z wprowadzeniem w 2014 roku standardu HTML 5 zestaw elementów możliwych do wykorzystania w ramach witryn internetowych został rozbudowany m.in. o znacznik `<canvas>`. Jego podstawowym zastosowaniem – zgodnie z założeniami projektantów – było umożliwienie rysowania oraz tworzenia interaktywnych wykresów bezpośrednio na stronie internetowej. Jednakże praktyka pokazuje, że zakres jego zastosowań jest istotnie większy, gdyż umożliwia on również:

- wyświetlanie filmów/animacji,
- wyświetlanie obrazu interaktywnych gier,

[10] Na rysunku przedstawiono (w celach poglądowych) sytuację, w której serwer nasłuchuje na porcie 443/TCP z tego względu, iż port ten zazwyczaj jest wykorzystywany przy połączeniach szyfrowanych, a te z kolei są standardem w przypadku *transakcyjnych aplikacji internetowych.*

co z kolei umożliwia wykorzystanie go także jako ekranu dla stacjonarnej aplikacji (tj. takiej, która nie została przewidziana do pracy w formie aplikacji internetowej). Należy podkreślić, że w takim przypadku aplikacja pozostanie funkcjonalna, a użytkownik będzie mógł w niej wykonywać wszystkie operacje, jednakże z wykorzystaniem przeglądarki internetowej, w ramach której będzie wyświetlane jej okno. Taki sposób działania aplikacji graficznych dopuszcza między innymi biblioteka GTK3 – na rysunku 4 przedstawiony został przykład okna programu stacjonarnego wyświetlonego w ramach przeglądarki internetowej *Firefox.*



Rys. 4 Widok aplikacji stacjonarnej *System Egzaminowania* uruchomionej w ramach przeglądarki internetowej *Firefox* (źródło: opracowanie własne)

Należy w tym miejscu nadmienić, że zmiana sposobu wyświetlania głównego okna programu (tj. wyświetlanie go w ramach przeglądarki internetowej) nie powoduje jego automatycznej transformacji w typową aplikację internetową. Jest tak m.in. dlatego, że do jego interakcji z użytkownikiem (np. przy wprowadzaniu i zatwierdzaniu danych) nie są stosowane formularze HTML jak w tradycyjnych aplikacjach internetowych, lecz połączenia wykorzystujące

mechanizm *WebSocket*[11], za pośrednictwem którego przesyłane są dane binarne informujące aplikację m.in. o tym, że w ramach jej okna użytkownik poruszył kursorem myszy, wcisnął przycisk, czy też wprowadził ciąg znaków. Warto podkreślić, że zmiana sposobu przesyłania danych do aplikacji stanowi główny mechanizm zwiększający jej bezpieczeństwo, co zostanie rozwinięte w kolejnym punkcie.

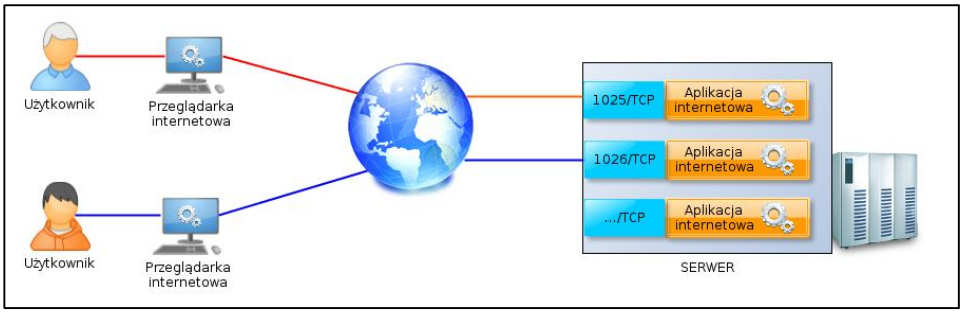
Analizując sposób działania aplikacji stacjonarnej, której okno główne wyświetlane jest w ramach przeglądarki internetowej koniecznie trzeba zwrócić uwagę, iż ze względu na specyfikę mechanizmu *WebSocket*:

a) tylko jeden klient w danym momencie (poprzez przeglądarkę internetową) może korzystać z konkretnej instancji aplikacji stacjonarnej – każda próba dołączenia nowego klienta do już wykorzystanego gniazda *WebSocket* spowoduje rozłączenie klienta wcześniej połączonego;

b) liczba aplikacji obsługujących klientów musi być nie mniejsza niż ich liczba,

c) każde połączenie wymaga wykorzystania innego portu TCP po stronie serwera.

Powyższe ograniczenia powodują, że w ogólności schemat połączenia klientów (za pomocą przeglądarek sieci Internet) do aplikacji stacjonarnych udostępnianych z wykorzystaniem elementu *Canvas* języka HTML5 musi być analogiczny do przedstawionego na rysunku 5.

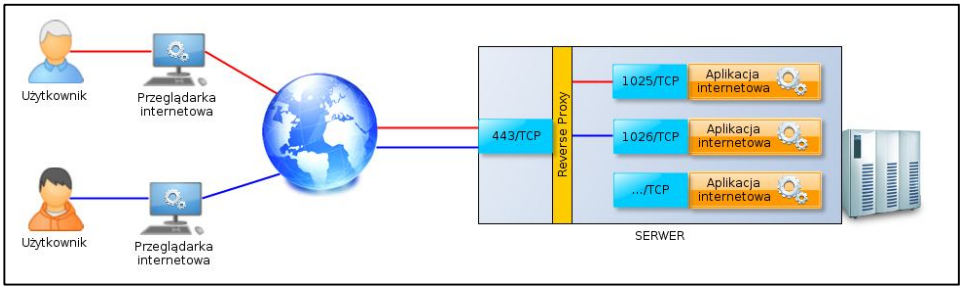


Rys. 5 Schemat poglądowy proponowanego sposobu działania transakcyjnych aplikacji internetowych (źródło: opracowanie własne)

Łatwo zauważyć, że powyżej przedstawiony schemat działania jest niedopuszczalny w zdecydowanej większości przypadków, gdyż zazwyczaj połączenia do aplikacji internetowych są dozwolone jedynie przy wykorzystaniu portów serwera 80/TCP i 443/TCP (w przypadku połączeń szyfrowanych). Ze względów bezpieczeństwa połączenia do innych portów są blokowane bezpośrednio na zaporach sieciowych (ochraniających serwery udostępniające

[11] Mechanizm *WebSocket* jest zdefiniowany w dokumencie normatywnym RFC 6455 i od roku 2011 jest obsługiwany przez wszystkie popularne przeglądarki sieci Internet.

aplikacje internetowe). Jednakże analiza technicznych możliwości ominięcia opisanego problemu wskazuje, że rozwiązaniem może być zastosowanie mechanizmu tzw. *Reverse Proxy*. Umożliwia on mianowicie udostępnianie wielu aplikacji internetowych poprzez ten sam port, jednakże przy wykorzystaniu odmiennych adresów URL. Ogólny schemat połączenia użytkowników do w ten sposób udostępnianych aplikacji stacjonarnych (przy wykorzystaniu mechanizmów *WebSocket* oraz *Reverse Proxy*) został przedstawiony na rysunku 6.

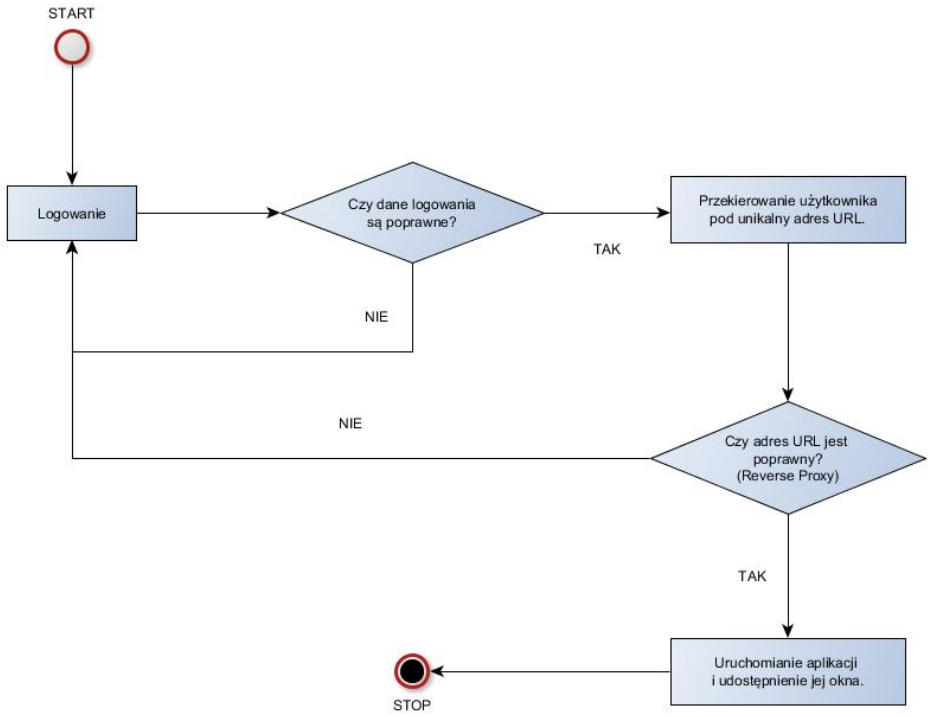


Rys. 6 Schemat poglądowy proponowanego sposobu działania transakcyjnych aplikacji internetowych (źródło: opracowanie własne)

Reasumując jako praktycznie realizowalny sposób zwiększenia bezpieczeństwa *transakcyjnych aplikacji internetowych* można przyjąć ich przygotowanie w formie aplikacji stacjonarnych (z założenia odpornych na zdecydowaną większość ataków ukierunkowanych na tradycyjne aplikacje internetowe), których główne okna będą udostępniane użytkownikom przy wykorzystaniu mechanizmu *WebSocket* ukrytego za mechanizmem *Reverse Proxy*. W tej sytuacji standardowy sposób obsługi każdego użytkownika powinien sprowadzać się do:

a) odwiedzenia przez niego strony głównej wybranej instytucji;
b) skierowania go do aplikacji internetowej, która – po jego poprawnym uwierzytelnieniu – przekieruje go pod specjalnie dla niego przygotowany, unikalny adres URL, np. postaci:
`https://example.com/4006A84DAE5B7DD7F13F5A9477…`
c) pod którym będzie (z wykorzystaniem mechanizmu *Reverse Proxy*) dostępne dla niego (udostępnione z wykorzystaniem mechanizmu *WebSocket*) okno obsługującej go (i tylko jego) aplikacji stacjonarnej.

Diagram obrazujący opisany powyżej sposób obsługi użytkowników został przedstawiony na rysunku 7.



Rys. 7 Diagram obrazujący sugerowany sposób obsługi użytkowników (źródło: opracowanie własne)

Warto zauważyć, że opisany sposób działania posiada dodatkową własność, mianowicie nie wykorzystuje *ciasteczek* do utrzymywania sesji użytkownika i identyfikowania żądań kierowanych do serwera przez jego przeglądarkę internetową. Jest to możliwe dlatego, że każdy użytkownik łączy się z innym adresem URL, który jednoznacznie go identyfikuje. Brak użycia *ciasteczek* implikuje niewystępowanie wszystkich (potencjalnych) podatności z nimi związanych w aplikacji internetowej, co jest dodatkową zaletą.

4. Analiza eliminowanych podatności aplikacji internetowych na ataki

W celu potwierdzenia skuteczności proponowanego rozwiązania konieczne jest poddanie analizie potencjalnych podatności (aplikacji internetowych), które mogą zostać wyeliminowane poprzez jego zastosowanie. Jakkolwiek nie istnieje jedna klasyfikacja luk bezpieczeństwa, to uzasadnionym wydaje się odwołanie do zestawienia OWASP Top 10, które podaje najczęściej wykorzystywane

podatności podczas ataków na aplikacje internetowe. W celach poglądowych poniżej zostało przytoczone zestawienie z roku 2017 (aktualne w czasie powstawania niniejszego artykułu).

- *A1:2017 - Injection*
- *A2:2017 - Broken Authentication*
- *A3:2017 - Sensitive Data Exposure*
- *A4:2017 - XML External Entities (XXE)*
- *A5:2017 - Broken Access Control*
- *A6:2017 - Security Misconfiguration*
- *A7:2017 - Cross-Site Scripting (XSS)*
- *A8:2017 - Insecure Deserialization*
- *A9:2017 - Using Components with Known Vulnerabilities*
- *A10:2017 - Insufficient Logging & Monitoring*

Łatwo zauważyć, że niektóre spośród przytoczonych typów podatności nie mogą zostać wyeliminowane przez proponowane rozwiązanie, gdyż są one całkowicie niezależne od użytej technologii, języków programowania itp. (mogą wystąpić w każdym przypadku) – zostały one oznaczone na czerwono. Jako przykład można podać „*A10:2017 - Insufficient Logging & Monitoring*", gdyż poziom szczegółowości dzienników zdarzeń utrzymywanych po stronie serwera jest ściśle zależny od jego konfiguracji i założeń przyjętych przez jego administratora. Błędy na tym poziomie mogą wystąpić niezależnie od zastosowanego oprogramowania.

Kolorem pomarańczowym zostały oznaczone typy podatności, które jedynie w pewnym (często istotnym) zakresie są eliminowane przez proponowane rozwiązanie.

Powodem występowania większości podatności typu *wstrzyknięcia* (*A1:2017 - Injection*) jest nieprawidłowa obsługa przez aplikację danych pozyskanych od użytkownika – często przyjmowane jest, że są one poprawne i nie są poddawane dodatkowej weryfikacji. Jakkolwiek nie jest możliwe całkowite wyeliminowanie tego typu błędów, to w przypadku aplikacji stacjonarnych sposób przetwarzania wprowadzanych do nich danych zazwyczaj istotnie ogranicza możliwość występowania takich podatności[12]. Dodatkowym aspektem

[12] Między innymi ze względu na wykorzystywane biblioteki i środowiska programowe, które zazwyczaj bazują na kodzie binarnym/bajtowym i bezpośrednio nie interpretują kodu źródłowego oraz danych przekazywanych przez użytkowników.

jest również to, że w przypadku aplikacji internetowych duża część omawianych podatności może zostać wykryta jedynie przez tzw. *automaty programowe*, które wykorzystują mechanizm jawnego (w żądaniach HTTP) przekazywania parametrów i ich wartości do serwera w celu modyfikacji przesyłanych danych. Przykładem może być luka bezpieczeństwa znana jako *Time Based SQL Injection,* gdyż jej użycie przez agresora bez wykorzystania specjalistycznych narzędzi jest niemal niemożliwe. Natomiast wykorzystanie obiektu *Canvas* języka HTML5 do interakcji z użytkownikiem powoduje, że wspomniane narzędzia specjalistyczne nie znajdują zastosowania, gdyż dane do aplikacji można przekazać jedynie manualnie, a do serwera są one przesyłane w postaci binarnej (a nie jawnej).

Warto zauważyć, że podobna sytuacja ma miejsce w przypadku podatności typu „*A7:2017 - Cross-Site Scripting (XSS)”.* Ich występowanie związane jest przede wszystkim z metodą prezentacji danych stosowaną w tradycyjnych aplikacjach internetowych – z wykorzystaniem języka HTML. Dla przykładu – jeśli dane pozyskane od użytkownika zostaną bezpośrednio dołączona do generowanej zawartości witryny, a zawierały one złośliwe sekwencje HTML, to mogą one wpłynąć na wyświetlaną treść. Natomiast w przypadku (istotnej większości) aplikacji stacjonarnych takie zagrożenie w ogóle nie występuje, gdyż ich *Graficzny Interfejs Użytkownika* (GUI) jest tworzony przy wykorzystaniu systemowych/bibliotecznych funkcji graficznych, a nie języka opisu (HTML). Implikuje to brak podatności na ten rodzaj nadużyć.

Zgodnie z definicją podawaną przez *OWASP Top 10* podatności typu „*A2:2017 - Broken Authentication*” związane są z ogólnie pojętym logowaniem i zarządzaniem sesją użytkownika. W tym przypadku należy przyjąć, że samo uwierzytelnienie będzie realizowane przez tradycyjną aplikację internetową (która będzie realizowała tylko tę funkcję – patrz punkt 3) i na tym etapie mogą wystąpić ewentualne niedostatki – w zależności od implementacji. Jednakże należy podkreślić, że już po przekierowaniu użytkownika na witrynę z przeznaczoną mu aplikacją nie występują żadne elementy, które mogłyby być podatne na omawiany rodzaj nadużyć (np. *ciasteczka*, co zostało już wskazane w punkcie numer 3).

W przypadku luk bezpieczeństwa typu „*A3:2017 - Sensitive Data Exposure*” należy zwrócić uwagę, że w zdecydowanej większości przypadków odnoszą się one do udostępniania danych wrażliwych w tradycyjnych aplikacjach internetowych, w których mogą one (dane wrażliwe) zostać zidentyfikowane i pobrane przez agresora przy wykorzystaniu metod typu *Web Scrapping*. Metody te bazują na zautomatyzowanym pozyskiwaniu treści z witryn internetowych oraz na ich maszynowym przetwarzaniu. Jednakże w proponowanej metodzie dane prezentowane są użytkownikowi przy wykorzystaniu obiektu *Canvas* języka HTML5, który nie jest podatny na żadną odmianę metod typu *Web Scrapping.* W tej sytuacji możliwość eksploatacji omawianego typu podatności jest istotnie

ograniczona. Pomimo warto to pamiętać, że nie istnieje żadne zabezpieczenie przed nieumyślnym prezentowaniem użytkownikowi danych wrażliwych, nie dla niego przeznaczonych (np. wyświetlenie danych z konta innego użytkownika bankowości internetowej). Jednakże tego typu sytuacje należą do rzadkości i są zazwyczaj spowodowane istotnymi błędami w logice aplikacji, które nie zostały wykryte na etapie jej testowania przedwdrożeniowego.

Podatności typów „*A4:2017 - XML External Entities (XXE)"* oraz „*A8:2017 - Insecure Deserialization*" nie występują, gdyż w rozpatrywanym przypadku nie są do serwera przekazywane dane poddane serializacji, ani zapisane w formie dokumentu XML.

Uwzględniając, że zastosowanie proponowanego rozwiązania oznacza, że w praktyce każdy użytkownik pracuje w ramach wyizolowanego środowiska uruchomionego specjalnie dla niego po stronie serwera, można stwierdzić, że podatności typu „*A5:2017 - Broken Access Control"* nie znajdują tutaj zastosowania. Jest to zasadne, gdyż z zasady użytkownik ma dostęp jedynie do danych mu przypisanych.

Dla porządku warto zauważyć, że typy podatności eliminowane przez zastosowanie zaproponowanego rozwiązania nie ograniczają się jedynie do tych, które zostały bezpośrednio wymienione w zestawieniu *OWASP Top 10*. Zauważmy, że jedną z częstszych luk bezpieczeństwa jest umożliwianie przeglądarce internetowej zapamiętywania danych wprowadzanych w różnego typu formularzach HTML przez użytkowników[13]. W przypadku zaproponowanego rozwiązania opisany problem w ogóle nie występuje, gdyż wspomniane formularze HTML w ogóle nie są wykorzystywane.

5. Zalety i wady proponowanego rozwiązania

Przedstawiona w artykule koncepcja zwiększenia bezpieczeństwa transakcyjnych aplikacji internetowych bazuje na modyfikacji zarówno ich sposobu działania (wykorzystanie obiektu *Canvas* języka HTML5 do wprowadzania danych), jak i budowy (wykorzystanie dodatkowego mechanizmu *Reverse Proxy*; uruchamianie aplikacji dla każdego użytkownika oddzielnie). Zaproponowane rozwiązanie skutecznie zmniejsza zakres zastosowania dużej części podatności (spotykanych w aplikacjach internetowych) lub w ogóle je eliminuje – co zostało wykazane w punkcie 4 na podstawie luk bezpieczeństwa wymienianych w zestawieniu *OWASP Top 10*. W przypadku aplikacji

[13] Działanie to jest często wykorzystywane przez użytkowników ze względu na ich wygodę, jednakże ze względów bezpieczeństwa nie jest ono zalecane i powinno być uznawane za podatność.

przetwarzających dane wrażliwe jest to niewątpliwa zaleta, jednakże trzeba mieć także świadomość ograniczeń/wad omawianego podejścia do budowy tego typu programów.

Analizując przedstawioną koncepcję zwiększania bezpieczeństwa aplikacji internetowych można zauważyć, że jej praktyczne zastosowanie wymaga istotnych zmian w architekturze obecnie funkcjonujących systemów (np. bankowości internetowej). Ich modyfikacja, co oczywiste, wiąże się z wysokimi kosztami, które mogą zostać zaakceptowane tylko przez niektóre instytucje i jedynie w przypadku stwierdzenia, że poczynione inwestycje w przyszłości zwrócą się (co może mieć miejsce po uwzględnieniu nakładów związanych z zabezpieczaniem aplikacji internetowych, wykonywaniem odpowiednich testów penetracyjnych, zakupem specjalizowanych rozwiązań typu *Web Application Firewall* itp.). W związku z powyższym należy przyjąć, że omawiana koncepcja budowy transakcyjnych aplikacji internetowych może znaleźć zastosowanie głównie w nowo projektowanych i implementowanych aplikacjach, co istotnie ogranicza zakres jej zastosowania.

Kolejną wadą, której nie można pominąć, jest kwestia zasobów wykorzystywanych po stronie serwera w przypadku uruchomienia dla każdego użytkownika oddzielnej instancji aplikacji internetowej. Przeprowadzone testy empiryczne dla aplikacji posiadającej około 50. widoków wprowadzania danych z 20. aktywnymi polami każdy (co można traktować jako odpowiednik tradycyjnego formularza HTML) konieczne było przydzielenie (zaalokowanie) około 50MB pamięci RAM. Jest to wielkość istotnie przekraczająca wartości osiągane przez tradycyjne aplikacje internetowe (często jest to około 5-10MB dla użytkownika). Należy także zaznaczyć, że w omawianym przypadku pamięć RAM była alokowana na całą sesję użytkownika, zaś przy tradycyjnych aplikacjach internetowych pamięć RAM jest alokowana jedynie podczas generowania odpowiedzi na żądanie użytkownika. W tej sytuacji jednoczesna obsługa wielu sesji użytkowników wymaga zaangażowania sprzętu o istotnie wyższych parametrach niż w przypadku tradycyjnych aplikacji internetowych. Jednakże należy pamiętać, że obecnie efektywność serwerów rośnie, zaś koszty ich nabycia istotnie spadają i często są pomijalne w porównaniu do strat, które instytucja może ponieść w przypadku zakończonego sukcesem ataku teleinformatycznego na jej zasoby.

6. Podsumowanie

Celem artykułu było przedstawienie nowatorskiej koncepcji zwiększania bezpieczeństwa transakcyjnych aplikacji internetowych, co wydaje się zagadnieniem istotnym ze względu na stale rosnącą liczbę ataków na ten rodzaj oprogramowania (co zostało dokładniej omówione w punkcie pierwszym).

W punkcie drugim niniejszego artykułu przytoczono podstawowe informacje na temat sposobu działania tradycyjnych aplikacji internetowych, zaś w rozdziale trzecim przedstawiono tytułową koncepcję oraz omówiono elementy techniczne, które można wykorzystać do jej wdrożenia. Punkt czwarty poświęcono ogólnej analizie podatności, które mogą zostać wyeliminowane lub, których poziom istotności może zostać istotnie zmniejszony poprzez zastosowanie zaproponowanego rozwiązania. Warto nadmienić, że w celach badawczych autorzy przygotowali aplikację internetową zaprojektowaną i zaimplementowaną zgodnie z wytycznymi przedstawionymi w punkcie trzecim. Umożliwiło to praktyczną analizę sposobu działania tego typu oprogramowania oraz wskazanie jego istotnych wad oraz zalet – rezultaty te zostały omówione w ramach punktu piątego.

Literatura

[1] BALOCH R., *Ethical Hacking and Penetration Testing Guide*, CRC Press, Boca Raton, 2015.

[2] CHAUCHAN S., Kumar N., *Hacking Web Intelligence: Open Source Intelligence and Web Reconnaissance Concepts and Techniques*, Syngress, Waltham, 2015.

[3] CIAMPA M., *Security Awareness: Applying Practical Security in Your World*, Cengage Learning, Boston, 2015.

[4] COLLINS M., *Network Security Through Data Analysis*, O'Reilly Media, Sebastopol, 2017.

[5] PRASAD P., *Mastering Modern Web Penetration Testing*, Packt Publishing, Birmingham, 2016.

[6] Open Web Application Security Project, *Penetration testing methodologies*, dostępny w: https://www.owasp.org/index.php/Penetration_testing_methodologies

[7] Open Web Application Security Project, *OWASP Top 10 Application Security Risks – 2017,* dostępny w: https://www.owasp.org/index.php/Top_10-2017_Top_10

The concept for increasing transactional internet applications security

ABSTRACT: The paper presents an innovative concept for increasing the security of crucial Internet applications, for instance ones used in financial operations processing, through non-standard use of HTML5 language mechanisms and server-side processing. Advantages of the proposed approach were discussed, and potential disadvantages of this concept were also pointed out.

KEYWORDS: security, websites, web applications, HTML5, Canvas

Praca wpłynęła do redakcji: 25.05.2018 r.