Ajax. Bezpieczne aplikacje internetowe

Autor: hristopher Wells
Tłumaczenie: Piotr Rajca
ISBN: 978-83-246-1373-1
Tytuł oryginału: Securing Ajax Applications: Ensuring the Safety of the Dynamic Web
Format: B5, stron: 248

Otwarte źródła danych a bezpieczeństwo aplikacji

- Jak zabezpieczyć przepływ danych klient-serwer?
- Jak strzec serwera aplikacji przed atakami?
- Jak przewidzieć i przeciwdziałać potencjalnym zagrożeniom w dynamicznych aplikacjach?

Otwartość i bezpieczeństwo, utopijne połączenie słów, a zarazem nieodwracalna przyszłość sieci internetowej. Współdzielenie zasobów niesie ze sobą szereg zagrożeń na różnych warstwach sieciowych. Efektywniej jest przewidzieć potencjalne zagrożenia na etapie tworzenia aplikacji i zapobiec im, niż później łatać dziury w oprogramowaniu. Każdy programista tworzący oprogramowanie sieciowe ostatecznie będzie musiał zmierzyć się z niepożądaną ingerencją mającą swoje źródło w sieci internetowej. Bądź na to przygotowany i nie daj się zaskoczyć, wykorzystaj wiedzę zawartą w tej książce.

Książka „Ajax. Bezpieczne aplikacje internetowe" traktuje o zagrożeniach i sposobach zabezpieczeń aplikacji sieciowych, a szczególnie dynamicznych interfejsów API. Przeznaczona jest zarówno dla programistów zaczynających przygodę z Ajaksem, jak i dla tych, którzy Ajaksa jeszcze nie znają. Przyda się każdemu, kto stoi na straży bezpieczeństwa aplikacji sieciowych, uczy bowiem, jak zapobiegać zagrożeniom w trakcie pisania aplikacji oraz jak przeciwdziałać im w już istniejącym oprogramowaniu sieciowym.

- Bezpieczeństwo aplikacji sieciowych
- Technologie zabezpieczeń komunikacji klient-serwer
- Zabezpieczenia na poziomie protokołów
- Serwer WWW i zagrożenia płynące z internetu
- Zabezpieczanie otwartych zasobów danych
- Bezpieczeństwo interfejsów API
- Zagrożenia bezpieczeństwa w aplikacjach typu mushup

Twórz rozległe aplikacje sieciowe i zadbaj o ich bezpieczeństwo?

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

ROZDZIAŁ 2.

Bezpieczeństwo aplikacji internetowych

W rozdziale 1. opisano, jak powstała Sieć oraz w jaki sposób działa. Ważne jest, by zapamiętać, że nowoczesna Sieć bazuje na grupie abstrakcji programowych oraz że tworzenie godnych zaufania i bezpiecznych aplikacji internetowych wciąż wymaga od programistów znajomości podstawowego protokołu i infrastruktury.

W niniejszym rozdziale dokładniej poznasz zasady działania mechanizmów zabezpieczeń oraz możliwości ich zastosowania w aplikacjach internetowych. Jeśli aplikacja działa w Internecie, to znajduje się w pierwszej linii Twojej firmowej lub domowej sieci. Można by ją porównać z drzwiami do świata zewnętrznego, przez które odwiedzający mogą wejść i sprawdzić, co masz do zaoferowania. Twoja aplikacja musi zatem być bezpieczna, a Ty sam musisz mieć świadomość zagrożeń, na jakie naraża ona Twoją sieć.

Zagadnienia podstawowe

Wyobraź sobie strażnika przechadzającego się nocą po słabo oświetlonych korytarzach jakiegoś biurowca. Kiedy strażnik wchodzi do kolejnych pomieszczeń, oświetla je latarką, szukając wszystkiego, co mogłoby się wydać podejrzane, po czym zamyka drzwi i idzie dalej. Strażnik wykonuje te czynności każdej nocy, zapewniając dzięki temu bezpieczeństwo budynku i znajdującym się w nim biur.

Cóż, aplikacje internetowe zazwyczaj nie mają takich strażników. Nie ma nikogo, kto mógłby zabezpieczyć je przed potencjalnymi włamywaczami.

Potrzeba wbudowania zabezpieczeń

A zatem co można zrobić? Pierwsze, co mogą zrobić programiści, to zauważyć konieczność wyposażenia aplikacji w mechanizmy zabezpieczeń. Oprócz tego należy się upewnić, co tak naprawdę wymaga ochrony. Gdzie aplikacja zaczyna się, a gdzie kończy? Jaka jest jej powierzchnia? Jeśli Twoja aplikacja przypomina typowe aplikacje internetowe, to składa się z trzech podstawowych elementów, które opiszę w dalszej części rozdziału.

Oczekuj niespodzianek

Alarm! Internetowi napastnicy próbują włamać się do naszej aplikacji. Używają aplikacji w nieprzewidziany sposób, by doprowadzić do wystąpienia błędów i innych sytuacji, które mogliby wykorzystać do swoich celów. Sytuacji, których nikt nie przewidział, niemal zawsze stanowią zagrożenie bezpieczeństwa.

Czy pamiętasz strażnika? Patroluje on budynek, szukając w nim wszystkiego, co odbiega od normy. Doskonale wie, że jeśli coś znajdzie się **nie** na swoim miejscu, to na pewno coś musiało do tego doprowadzić. Oczywiście inteligentny włamywacz czeka do momentu, gdy strażnik sprawdzi wszystkie pomieszczenia, i zaatakuje dopiero **później**.

Podmioty

Podmioty to wszyscy używający aplikacji. Oczywiście, najpopularniejszymi podmiotami będą zwyczajni użytkownicy (czyli osoby odwiedzające aplikację), jednak mogą to także być inne programy korzystające z usług sieciowych lub udostępnionego, zewnętrznego interfejsu programowania (API). Niezależnie od rodzaju podmioty zawsze są „bytami" zewnętrznymi korzystającymi z systemu.

Wyobraźmy sobie, że dysponujemy witryną WWW zajmującą się sprzedawaniem różnego typu kontrolek do aplikacji. Witryna implementuje zwyczajny koszyk zakupów oraz usługę sieciową.

Takiej aplikacji mogą używać dwa odrębne typy podmiotów:

Klienci:
: Czyli osoby, które zajrzały na witrynę, by kupić któryś z oferowanych produktów, używając przy tym koszyka do zakupów.

Partnerzy:
: Programy korzystające z usług sieciowych do zarządzania produktami i wchodzące w skład większej, złożonej aplikacji.

Gdyby coś miało pójść nie tak, chcielibyśmy wiedzieć, co się zdarzyło i kto doprowadził do wystąpienia problemów. Wyobraź sobie dowolny film kryminalny — podmiotami byliby wszyscy objęci dochodzeniem.

Obiekty

Obiektami nazywamy wszelkie zasoby aplikacji. Zazwyczaj będą to należące do niej dane, lecz mogą to być także pliki, połączenia, usługi oraz wszelkie inne rzeczy, które mają dla niej jakąś wartość lub które do niej należą.

W naszym przykładzie obiektami będą takie rzeczy, jak prywatne dane klientów, dane dostawców, firmowe dane zgromadzone w bazie danych oraz same sprzedawane kontrolki. Innymi słowy, obiekty te można by porównać do przeróżnych cennych drobiazgów przechowywanych przez ojca w kredensie.

Przyglądając się tym wszystkim obiektom i danym, należy zadać sobie pytanie, czy trzeba zachować ich poufność? Czy aplikacja powinna je w jakiś sposób zabezpieczać? Jakie będą konsekwencje dla Ciebie lub firmy, jeśli te dane pojawią się nagle wystawione na sprzedaż na jakiejś witrynie w byłym Związku Radzieckim?

Operacje

Operacje wiążą ze sobą podmioty i obiekty. To rzeczy, które aplikacja może wykonywać. Operacje zapewniają podmiotom dostęp do obiektów (a mówiąc precyzyjniej, podmioty używają operacji, by pobierać obiekty i manipulować nimi).

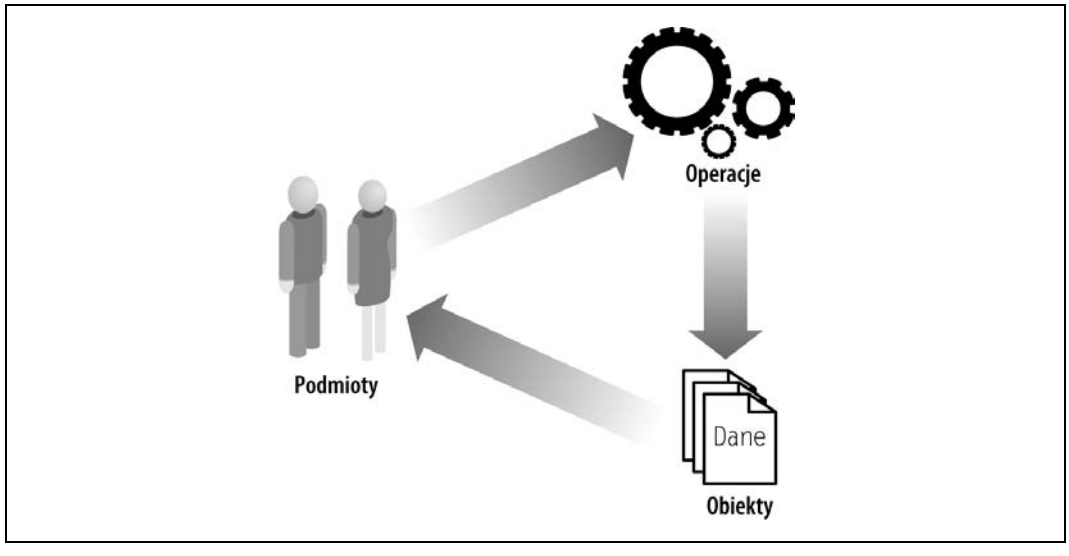
W naszej przykładowej witrynie operacjami dostępnymi dla klientów mogłyby być:

- dodanie wybranej kontrolki do koszyka z zakupami,
- usunięcie kontrolki z koszyka,
- zamówienie wybranych kontrolek,
- przeglądanie katalogu.

Podobnie usługa używana przez partnerów mogłaby udostępniać następujące operacje:

- wyszukiwanie kontrolek,
- zakup kontrolek.

Wszystkie trzy wymienione wcześniej podstawowe elementy aplikacji — podmioty, obiekty oraz operacje (przedstawione na rysunku 2.1) — określają wspólnie **powierzchnię** aplikacji.



Rysunek 2.1. Podmioty, obiekty oraz operacje

Powierzchnia

Każdy użytkownik dodany do systemu, każda operacja wykonywana przez aplikację oraz każdy używany przez nią zasób zwiększają powierzchnię aplikacji. A zatem, wziąwszy pod uwagę zagadnienia bezpieczeństwa, należy to sobie uświadomić i ograniczyć możliwości aplikacji jedynie do tych, które są jej absolutnie niezbędne.

Należy ograniczać powierzchnię ataku. Ograniczenie liczby funkcji aplikacji jedynie do tych, które są niezbędne, ułatwi nam właściwe zajęcie się jej zabezpieczeniem.

Po ustaleniu powierzchni aplikacji — czyli określeniu, kim będą jej użytkownicy oraz jakie czynności będą mogli wykonywać — będziemy mogli zająć się wykorzystaniem najlepszych metod zabezpieczania w pozostałych obszarach aplikacji.

Poufność

Kluczowym elementem dynamicznych witryn WWW są dane. To właśnie one sprawiają, że witryny są dynamiczne. Niektóre dane są banalne, na przykład wartość pola wyboru lub przycisku opcji. Jednak są także dane specjalne, na przykład nazwiska i adresy klientów, daty ich urodzenia, numery ubezpieczeń, kart kredytowych i tak dalej. Takie specjalne dane muszą być chronione, a aplikacja powinna je ujawniać tylko w odpowiednich sytuacjach.

Ponieważ aplikacje internetowe obsługują wiele rodzajów takich wrażliwych informacji, zatem ich zadaniem jest także je chronić. To oznacza, że koniecznie należy upewnić się, iż w trakcie działania aplikacji informacje te w żaden niezaplanowany i przypadkowy sposób nie zostaną ujawnione.

Oprócz tego, jeśli aplikacja przesyła wrażliwe dane przez sieć, konieczne jest przedsięwzięcie odpowiednich środków mających na celu zapewnienie bezpieczeństwa informacji podczas transmisji, na przykład zaszyfrowanie ich.

Prywatność

Czym jest prywatność, albo mówiąc ściślej: czym są dane prywatne? Cóż, gdybym zdradził tę informację, to chyba nie można by ich już było nazywać prywatnymi, nieprawdaż? Problem z prywatnością polega na tym, że dla różnych osób oznacza ona zupełnie co innego. Jednak większość ludzi określa dane prywatne jako takie, które nie powinny stać się ogólnodostępne ani bezpańskie. Ich wartość polega na tym, że pomagają ustalić unikalną tożsamość ich właściciela.

Hackerzy uwielbiają dane prywatne. W ich poszukiwaniu będą grzebać w naszych śmieciach, segregować lepkie odpadki, rozmokłe kawałeczki papieru i wąchać resztki kości z kurczaka. Będą to robić chętnie, wiedząc, że kiedyś ich styl życia się zmieni. Wyobrażają sobie zupełnie nowe, inne życie — życie w którym będą kimś zupełnie innym, takim jak **Ty**.

Szyfrowanie

Dobrym sposobem zabezpieczenia danych jest ich szyfrowanie. Zalecanym sposobem szyfrowania jest wykorzystanie matematycznie mocnego algorytmu zatwierdzonego przez *National Institute of Standards and Technology* (w skrócie: NIST, Narodowy Instytut Standardów i Technologii), takiego jak AES.

Szyfrowanie wrażliwych danych — zabezpieczanie wrażliwych danych przy wykorzystaniu matematycznie mocnego algorytmu. Szyfrowanie zapewnia poufność oraz integralność danych.

Istnieje kilka ogólnie dostępnych, dobrych algorytmów szyfrowania, opracowanych przez godne zaufania instytucje. Wybierając jeden z nich, należy się także upewnić, że również wybrana implementacja algorytmu pochodzi od dostawcy godnego zaufania. Nie warto silić się na tworzenie własnej implementacji.

Zagadnienia związane z szyfrowaniem są trudne. Należy wybrać algorytm, który będzie stosowany, zarządzać kluczami szyfrującymi oraz zaimplementować specjalny kod służący do przechowywania danych. To naprawdę sporo zachodu. Dlatego też zamiast szyfrowania niektórzy programiści decydują się ukrywać, separować bądź w jakikolwiek inny sposób „zaciemniać" dane, utrudniając przez to ich analizę i zrozumienie.

Poniżej przedstawiono kilka sposobów takiego zaciemniania danych:

- kodowanie Base64,
- kodowanie stosowane w adresach URL,
- kodowanie UTF8,
- zaciemnianie przez przesuwanie bitów.

Jeszcze gorsi są programiści, którzy uważają, że są w stanie zaimplementować algorytmy szyfrowania lepiej, niż to zrobił Bruce Schneier[1]. Ci wkładają swoje klejnoty koronne do tekturowego pudełka i umieszczają je w Internecie, wprost przed wszelkimi hackerami.

Integralność i walidacja

Ponieważ obecnie tak wiele aplikacji internetowych bazuje na danych, dlatego tak ogromne znaczenie ma zapewnienie prawidłowości danych, przy czym w tym kontekście oznacza to, że stosowanie danych musi być bezpieczne, a same dane nie mogą być modyfikowane. Spełnienie tych warunków oznacza, że dane zostaną sprawdzone za każdym razem, gdy trafią do systemu, oraz że będą istnieć sposoby weryfikacji ich poprawności. Niestety, niezwykle łatwo jest założyć, że ktoś inny za nas zajął się zapewnieniem poprawności danych.

Dodatkowo należy się upewnić, że dane nie zmienią się ani nie uszkodzą. Jednym z dobrych rozwiązań tego problemu jest szyfrowanie danych, kiedy nie są używane. Dzięki temu można mieć pewność, że później, kiedy dane zostaną odszyfrowane, będą miały dokładnie taką samą postać jak w momencie szyfrowania.

Uwierzytelnianie

Od uwierzytelnienia powinna zaczynać się każda dobra sesja. Nic nie powinno się zdarzyć, zanim nie upewnimy się, z kim mamy do czynienia. **Uwierzytelnianie** jest procesem pozwalającym określić, czy osoba lub rzecz jest tym, za kogo lub za co się podaje.

[1] Bruce Schneier to amerykański kryptograf i specjalista z zakresu bezpieczeństwa teleinformatycznego — *przyp.tłum*.

Zarówno w prywatnych, jak i publicznych sieciach komputerowych (w tym w Internecie) uwierzytelnianie jest często wykonywane przy wykorzystaniu nazw użytkowników i haseł. Znajomość nazwy użytkownika oraz skojarzonego z nią hasła jest traktowana jako potwierdzenie autentyczności użytkownika. Każdy użytkownik początkowo rejestruje się (lub jest przez kogoś rejestrowany), podając jakieś przypisane mu lub samodzielnie określone hasło. Podczas następnych wizyt użytkownik musi znać to hasło i je podać.

Problem z hasłami polega na tym, że ludzie często je zapominają. To w końcu naturalne. Nie ma w tym nic dziwnego, zważywszy na narzucane reguły konstrukcji haseł, do których użytkownicy muszą się stosować.

Zobaczmy, czemu musimy stawić czoła. Otóż wymyślane hasła nie mogą być zwyczajnymi słowami, muszą zawierać dziwne znaki, duże i małe litery oraz cyfry. Któż byłby w stanie zapamiętać coś takiego? Dlatego też **ludzie zapominają hasła**. A to z kolei sprawia, że zapisują je na małych samoprzylepnych karteczkach i przyklejają pod klawiaturą, w szufladzie lub na monitorze.

Dlatego też coraz więcej firm wykorzystujących Internet w swojej działalności, na przykład banki, zaczyna wymagać stosowania dodatkowych mechanizmów uwierzytelniania. Niektóre używają specjalnych **żetonów** (ang. *token*), które użytkownicy mają przy sobie, inne stosują certyfikaty cyfrowe. Wraz z rozwojem prowadzonej działalności rośnie także potrzeba stosowania pewnych i wiarygodnych metod uwierzytelniania oraz zapewnienia niezaprzeczalności.

Jak już zaznaczono, wszelka interakcja z aplikacją powinna zaczynać się od uwierzytelnienia użytkownika. Zanim mu cokolwiek udostępnimy, użytkownik powinien zadeklarować, kim jest. Dotyczy to **wszystkich** żądań przesyłanych Internetem. Co więcej, jeśli nie uwierzytelniono użytkownika, nie należy tworzyć **sesji**.

Należy przeprowadzać uwierzytelnianie wszystkich żądań przesyłanych Internetem. Dotyczy to także żądań przesyłanych asynchronicznie. Nie zapominaj uwierzytelniać żądań przesyłanych przy użyciu obiektu `MXLHttpRequest` — i to każdego z nich.

Oczywiście, po uwierzytelnieniu należy przeprowadzić autoryzację (choć obie te operacje mogą się wydawać ze sobą powiązane).

Autoryzacja i kontrola dostępu

No dobrze, zatem użytkownik zalogował się… i co dalej? No cóż, samo to, że ktoś zalogował się do aplikacji, nie oznacza wcale, że powinien mieć uprawnienia do wykorzystania wszystkich jej możliwości. Zawsze należy określać, jakie możliwości będzie miał uwierzytelniony użytkownik.

Autoryzacja jest procesem przydzielania użytkownikom uprawnień do wykonywania pewnych czynności lub dostępu do określonych danych.

W systemach komputerowych obsługujących wielu użytkowników ich administratorzy określają, jacy użytkownicy mają prawo dostępu do systemu oraz jakie mają uprawnienia (na przykład do jakich katalogów będą mieć dostęp, w jakich godzinach będą mogli korzystać z systemu, jaki obszar dysku zostanie im przydzielony i tak dalej). Jeśli ktoś zalogował się do systemu operacyjnego lub aplikacji, system lub aplikacja może chcieć określić, do jakich zasobów dany użytkownik powinien mieć dostęp podczas sesji.

A zatem pod pojęciem autoryzacji rozumie się czasami zarówno wstępne przydzielenie uprawnień przez administratora systemu, jak i późniejsze, faktyczne sprawdzenie uprawnień przypisanych użytkownikowi, wykonywane w momencie, gdy żąda on dostępu do jakiegoś zasobu lub wykonania jakiejś operacji. Autoryzacja jest zazwyczaj wykonywana po uwierzytelnieniu. W końcu aby sprawdzić, co jakaś osoba może zrobić, trzeba najpierw określić, kim ona jest i jakie ma uprawnienia.

Rozdział obowiązków

Interfejsy i funkcje używane przez administratorów systemu powinny być oddzielone od funkcji dostępnych dla zwyczajnych użytkowników. Dlatego w każdym z tych obszarów aplikacji należy umieścić odpowiednie kontrolki. Poza tym kod wymagający uprawnień oraz kod obsługujący zwyczajnych użytkowników nie powinny być wdrażane i umieszczane razem.

Także samo środowisko aplikacji powinno być dzielone na warstwy, na podstawie których będzie następnie przeprowadzany proces tworzenia, testowania i wdrażania aplikacji. Taka organizacja zapewnia, że do użytkownika końcowego trafi sprawdzony kod o odpowiedniej, produkcyjnej jakości.

Oto typowe warstwy, na które jest dzielone środowisko aplikacji:

Warstwa programowania
: Wykorzystywana do celów tworzenia, wykrywania i usuwania błędów w aplikacji.

Warstwa testowania
: Używana do przeprowadzania testów modułów, testowania wydajności i jakości.

Warstwa produkcyjna
: Gotowa i działająca witryna WWW.

Przez zastosowanie strategii polegającej na zwiększaniu ograniczeń wraz ze zbliżaniem się do momentu wdrożenia aplikacji, można dokładniej kontrolować dostęp w miejscach, gdzie autoryzacja ma największe znaczenie.

Niezaprzeczalność

Kiedy użytkownik zostanie już uwierzytelniony, należy śledzić wykonywane przez niego operacje i rejestrować te spośród nich, które mają kluczowe znaczenie dla bezpieczeństwa systemu. Dzięki temu, jeśli stanie się coś złego, będzie można uzyskać informację o tym, co się stało oraz kto może być za to odpowiedzialny.

Rejestracja zdarzeń związanych z bezpieczeństwem systemu — rejestrowanie takich zdarzeń, jak próby uwierzytelniania, dostępu, edycji i usuwania danych, tworzy ich fizyczny ślad, który może pomóc w uzyskaniu niepodważalności.

Poniżej podano kilka przykładów ważnych zdarzeń związanych z bezpieczeństwem systemu:

- inicjalizacja lub utworzenie sesji;
- udane, jak i nieudane próby zalogowania się do systemu;
- wylogowanie się użytkownika;

- próby logowania, w których podano nieprawidłowe hasło;
- operacje utworzenia, odczytu, aktualizacji i usunięcia konta użytkownika;
- zmiany konfiguracji;
- uruchomienie i zatrzymanie serwera;
- nieprzewidziane zdarzenia systemowe;
- próby wykonania operacji, do których użytkownik nie ma uprawnień;
- zmiany hasła;
- wykonanie czynności wymagającej większych uprawnień;
- transakcje;
- zastosowanie metody `GET` zamiast `POST`.

Podczas rejestrowania powyższych zdarzeń koniecznie należy zapisywać także następujące dane:

- kto wykonał operację;
- skąd pochodziło żądanie;
- na jakich zasobach żądanie operowało;
- na jakiej stronie została wykonana operacja;
- data oraz godzina zajścia zdarzenia

… oraz wszelkie inne informacje, które mogą później przydać się w prowadzeniu dochodzenia.

Dostępność

To, czy aplikacja będzie dobra czy zła, nie będzie miało większego znaczenia, jeśli nie zapewnimy użytkownikom możliwości korzystania z niej. Ponieważ aplikacje zależą od dostępności zasobów i danych, należy przedsięwziąć odpowiednie kroki, by także te systemy były dostępne.

Jednym ze sposobów mierzenia dostępności jest wyrażenie jej przy wykorzystaniu **mitu dziewiątek**. Stwierdzenie „nasz system jest dostępny przez 99,99% czasu" można rozumieć w ten sposób, że system nie działa przez 52,6 minuty w roku lub przez 1,01 minuty dziennie (odpowiednie przeliczenia można znaleźć w tabeli 2.1).

Tabela 2.1. Tabela dostępności

% dostępności	Czas przestoju w roku	Czas przestoju w miesiącu	Czas przestoju w tygodniu
98%	7,30 dnia	7,30 dnia	3,36 godziny
99%	3,65 dnia	3,65 dnia	1,68 godziny
99,5%	1,83 dnia	3,60 godziny	50,4 minuty
99,9%	8,76 godziny	43,2 minuty	10,1 minuty
99,99%	52,6 minuty	4,32 minuty	1,01 minuty
99,999%	5,26 minuty	25,9 sekundy	6,05 sekundy
99,9999%	31,5 sekundy	2,59 sekundy	0,605 sekundy

W przeważającej większości przypadków ten sposób pomiaru dostępności jest stosowany w dokumentach marketingowych — zapewne dlatego, że wygląda dosyć imponująco. Niemniej jednak takie informacje o dostępności często są zamieszczane w oficjalnych kontraktach i umowach o świadczeniu usług, dlatego warto je zapamiętać.

Zaufanie

Zaufanie to pewność co do integralności danej osoby lub rzeczy. W kontekście aplikacji internetowych zaufanie w przeważającej większości przypadków odnosi się do użytkowników. Aby uzyskać zaufanie użytkowników, musimy:

- zapewnić odpowiednie uwierzytelnianie;
- upewnić się, że użytkownik wykonuje jedynie te operacje, które może wykonywać;
- sprawdzać i przeprowadzać weryfikację wszystkich pobieranych danych;
- rejestrować i generować raporty z informacjami o wszystkich ważnych operacjach.

Problem z zaufaniem polega na tym, że zawsze jest ono aktem wiary. Nigdy do końca nie wiadomo, czy danej osobie bądź rzeczy można w pełni zaufać.

Analiza ryzyka

A co, jeśli coś pójdzie źle? Trzeba mieć plan na takie sytuacje. Musimy wiedzieć, co należy zrobić, kiedy nasza aplikacja zostanie zaatakowana. Doskonałym sposobem znalezienia rozwiązania tego problemu jest opracowanie modelu zagrożenia dla aplikacji.

W jaki sposób można oszacować bezpieczeństwo aplikacji? Cóż, w pierwszej kolejności musimy odpowiedzieć na pytanie, czym jest aplikacja internetowa.

Anatomia aplikacji internetowych

Aplikacje internetowe zapewniają użytkownikom dostęp do bazy danych z dowolnego miejsca na świecie. Z jednej strony aplikacje te działają w Internecie, obsługują nadsyłane żądania HTTP i wysyłają odpowiedzi. Z drugiej strony operują na znajomych elementach: plikach, zasobach systemowych i danych. Ponieważ zapewniają one dostęp do zasobów na serwerze, trzeba je sprawdzić bardzo uważnie i krytycznie.

Punkty wejścia

Punktami wejścia są te miejsca aplikacji, w których dane są wprowadzane do systemu. Takie dane muszą być zweryfikowane. Jeśli nie przeprowadzimy walidacji (czy też kontroli) danych przed ich zastosowaniem, to należy je uznać za potencjalnie skażone.

Do poprawnego działania aplikacje wymagają prawidłowych danych. Jeśli do systemu trafią skażone dane, to aplikacja mogłaby je nieumyślnie wyświetlić. Oprócz tego stosowanie skażonych danych mogłoby doprowadzić do zgłoszenia wyjątku, który stanowiłby źródło informacji o systemie. Internetowi napastnicy poszukują takich sytuacji i wykorzystują je do swoich celów.

Informacje mogą trafiać do aplikacji z wielu różnych źródeł:

- od użytkowników,
- z plików,
- z gniazd sieciowych,
- z właściwości systemowych,
- z potoków,
- z interfejsów programistycznych,
- z rejestru systemowego,
- z poczty elektronicznej,
- z argumentów wiersza poleceń,
- z parametrów inicjalizacyjnych,
- z zmiennych środowiskowych,
- z bazy danych.

Ważne, by przeanalizować każdy z tych punktów wejścia i określić typy przekazywanych przez niego danych oraz sposób, w jaki są one używane przez aplikację.

Poziom zaufania

Poziom zaufania to zaufanie, jakim obdarzany jest zewnętrzny byt przez przydzielenie mu roli określającej możliwości dostępu do punktu wejścia aplikacji. Na przykład rola Administrator jest rolą uprzywilejowaną, mającą znacznie więcej uprawnień niż zwyczajni użytkownicy.

Użytkownikom aplikacji należy przypisywać role, na podstawie których system będzie określać, czy mają oni uprawnienia niezbędne do wykonywania poszczególnych operacji. Przez rozdzielenie operacji, które można wykonywać w aplikacji, pomiędzy kilka różnych ról można utrudnić jednemu użytkownikowi uzyskanie zbyt dużej kontroli nad systemem.

Aktywa

Osoby atakujące aplikację zazwyczaj czegoś chcą. Tym „czymś" są aktywa aplikacji. Mogą to być dane albo użytkownicy. Może to być tajny przepis na pieczonego kurczaka. Czymkolwiek by te aktywa były, atakujący ich chcą, a zadaniem twórców aplikacji jest ich ochrona.

Zagrożenia i ścieżki ataku

Napastnicy nie mają żadnego powodu do atakowania aplikacji, o ile nie zawiera czegoś, co ich interesuje. Zanim zaczniemy chronić wszystko, co tylko można zobaczyć w aplikacji, musimy odpowiedzieć na pytanie, czy dany punkt wejścia bądź operacja stwarzają jakieś zagrożenie dla bezpieczeństwa aplikacji. Czy jest w niej coś interesującego? Czy w efekcie ataku system może przestać prawidłowo działać?

Jeśli weźmiemy pod uwagę punkt wejścia, połączymy go z poziomem zaufania użytkownika, a następnie z aktywami aplikacji, uzyskamy **ścieżkę ataku**. Analizując przepływ danych wzdłuż ścieżki ataku, można określić wszystkie możliwe zagrożenia czyhające na dane, użytkownika oraz system.

Trzeba myśleć jak włamywacz

A zatem w jaki sposób można się włamać do aplikacji? W jaki sposób można wykorzystać punkt wejścia lub dane? W jaki sposób można skazić dane trafiające do systemu? W jaki sposób można by przeprowadzić atak? Nadszedł czas, by pomyśleć jak internetowy włamywacz. Co najgorszego może się zdarzyć?

Zadaj sobie następujące pytania:

- Jak przejąć kontrolę nad systemem?
- Jak uzyskać dostęp do informacji?
- Jak manipulować informacjami?
- Jak spowodować awarię systemu?
- Jak uzyskać dodatkowe uprawnienia?

Doskonale! A skoro system został już zaatakowany, co mógłby zrobić napastnik:

- bez zostania zarejestrowanym?
- przez pominięcie kroków kontroli dostępu?
- przez podszycie się pod innego użytkownika?

Napastnicy w swoich atakach wcale nie muszą silić się na oryginalność. Okazuje się, że w praktyce nowe rodzaje ataków pojawiają się stosunkowo rzadko. Zazwyczaj atakujący wykorzystują dobrze znane słabości, gdyż jest to znacznie prostsze od poszukiwania nowych sposobów. **Internetowi napastnicy nie lubią utrudniać sobie życia.**

Najczęściej spotykane typy ataków

Analizując punkt wejścia w poszukiwaniu potencjalnych słabych punktów, należy sprawdzić, czy atakujący może:

- dokonać trwałych modyfikacji;
- bezpośrednio przeglądać zasoby;
- modyfikować lub wprowadzać fałszywe dane.

Oprócz tego należy sprawdzić, czy aplikacja jest w stanie prawidłowo:

- weryfikować dane wejściowe;
- stosować najlepsze praktyki pozytywnej weryfikacji;
- uwierzytelniać użytkowników;
- autoryzować role;
- zarządzać konfiguracją;
- prawidłowo obsługiwać wyjątki;
- uwierzytelniać i autoryzować systemy serwerowe;
- rejestrować zdarzenia w dziennikach;
- szyfrować obecnie nieużywane dane.

Analiza zagrożeń

W praktyce analiza zagrożeń polega na zrozumieniu, w jaki sposób atakujący postrzega naszą aplikację. Co chce osiągnąć? Musimy określić, w jaki sposób atakujący będzie chciał wykorzystać aplikację. Z jakich ról oraz operacji atakujący skorzysta, by zagrozić jej bezpieczeństwu?

Znalezienie odpowiedzi na te wszystkie pytania wymaga pewnych założeń co do możliwości zręcznego napastnika. Zatem przy jakich założeniach można mówić o zagrożeniu?

Otóż często zakłada się, że atakujący może na przykład chcieć:

- czegoś (na przykład danych);
- coś uszkodzić (atak typu odmowa dostępu);
- uniemożliwić komuś zrealizowanie pewnego celu;
- coś zmienić;
- coś ukryć.

Nie należy jednak zgadywać na ślepo, lecz kierować się przemyśleniami bazującymi na wiedzy dotyczącej ataków.

Oto co jeszcze można założyć:

- w jakim stanie musi się znajdować aplikacja;
- jaką rolę ma atakujący;
- w którym miejscu systemu zostanie przeprowadzony atak;
- czy atak nie zostanie wykryty?

Modelowanie zagrożeń pozwala także spojrzeć na aplikację strukturalnie i zbadać zagrożenia faktyczne, a nie jedynie reagować na ogóle słabości zabezpieczeń. Nie można zabezpieczyć aplikacji bez wcześniejszego określenia zagrożeń.

Pionierem badań dotyczących modelowania zagrożeń jest firma Microsoft, a zgromadzone przez nią wnioski stanowią doskonały punkt wyjściowy. Według Microsoftu proces modelowania zagrożeń składa się z sześciu następujących etapów:

1. Identyfikacji aktywów — określenia typów danych lub informacji, które mogą interesować napastników, oraz przeanalizowania ich obecnych zabezpieczeń.
2. Stworzenia ogólnego opisu architektury — przeanalizowania komponentów systemu w celu wskazania punktów wejścia aplikacji oraz udokumentowania ścieżek przepływu danych w systemie.
3. Dekompozycji aplikacji — przeanalizowania każdej z funkcji aplikacji oddzielnie i określenia ścieżek przepływu danych oraz komponentów, które biorą w nim udział.
4. Określenia zagrożeń — wskazania miejsc, w których mogą się pojawić usterki w aplikacji, oraz potencjalnych miejsc ataku.
5. Dokumentacji zagrożeń — zapisania wszystkich możliwych zagrożeń.
6. Oszacowania zagrożeń — określenia możliwości wystąpienia i wykrycia poszczególnych zagrożeń.

Często spotykane słabe punkty aplikacji internetowych

Czasami najprostszym sposobem znalezienia słabych punktów aplikacji jest przeanalizowanie tego, co się zdarzyło w przeszłości. Przez przeanalizowanie słabych punktów wykrytych w innych aplikacjach możemy uczyć się na cudzych błędach.

OWASP

OWASP — *Open Web Application Security Project* (Otwarty projekt do spraw bezpieczeństwa aplikacji internetowych) — to otwarta społeczność osób, których celem jest umożliwienie organizacjom i firmom tworzenia, kupowania oraz utrzymywania godnych zaufania aplikacji.

OWASP posiada narzędzia, dokumenty, fora dyskusyjne i lokalne oddziały zajmujące się rozwojem bezpieczeństwa aplikacji internetowych. Wszystkie te zasoby są darmowe i dostępne bez ograniczeń dla wszystkich osób zainteresowanych poprawą bezpieczeństwa aplikacji.

OWASP sugeruje rozwiązywanie problemów bezpieczeństwa aplikacji poprzez analizę zagadnień związanych z ludźmi, procesami oraz technologią. Okazuje się bowiem, że najbardziej efektywne sposoby poprawiania bezpieczeństwa aplikacji internetowych ściśle wiążą się z usprawnieniami w tych trzech dziedzinach.

Jeśli jeszcze nie trafiłeś na stronę OWASP, to koniecznie powinieneś na nią zajrzeć: *http://www.owasp.org*.

10 najczęściej spotykanych słabych punktów według OWASP

OWASP określiła listę 10 najczęściej występujących słabych punktów, stanowiących plagę aplikacji internetowych. Oto ona:

Brak weryfikacji danych wejściowych
: Informacje przekazywane w żądaniach nie są weryfikowane przed ich zastosowaniem w aplikacji. Napastnicy mogą wykorzystać tę usterkę, by za pośrednictwem aplikacji internetowej przeprowadzić atak na komponenty serwerowe.

Wadliwa kontrola dostępu
: Ten problem polega na niewłaściwym określaniu lub stosowaniu ograniczeń dotyczących możliwości, jakie ma uwierzytelniony użytkownik systemu. Napastnicy mogą wykorzystać usterki tego typu do uzyskania dostępu do kont innych użytkowników, przeglądania ważnych plików bądź wykonywania operacji, do których nie mają prawa.

Wadliwe uwierzytelnianie lub zarządzanie sesjami
: Ten problem polega na niewłaściwej ochronie informacji stosowanych do uwierzytelniania użytkowników lub danych sesji. Napastnicy, którzy mogą przejąć hasła, klucze, cookies używane do obsługi sesji oraz inne dane, mogą przełamać system uwierzytelniania i podszywać się pod innych użytkowników.

Ataki typu cross-site scripting
: Aplikację internetową można wykorzystać jako środek ataku na przeglądarkę użytkownika. Udany atak tego typu może ujawnić napastnikowi informacje o sesji użytkownika, umożliwić zaatakowanie lokalnego komputera lub zmodyfikowanie treści strony w celu wprowadzenia użytkownika w błąd.

Przepełnienie bufora
W przypadku nieprawidłowej weryfikacji danych komponenty aplikacji internetowych pisanych w niektórych językach programowania mogą działać nieprawidłowo, a niekiedy można je wykorzystać do przejęcia kontroli nad procesem. Te komponenty to między innymi skrypty CGI, biblioteki, sterowniki oraz wykonywane na serwerze komponenty aplikacji internetowych.

Usterki związane ze wstawianiem kodu
Kiedy aplikacje internetowe korzystają z systemów zewnętrznych lub lokalnego systemu operacyjnego, przekazują do nich parametry. Jeśli atakujący będzie w stanie przekazać w nich odpowiednio spreparowane polecenia, to zewnętrzny system będzie mógł je wykonać.

Nieprawidłowa obsługa błędów
Ten problem polega na niewłaściwej obsłudze błędów pojawiających się podczas normalnego działania aplikacji. Jeśli napastnikowi uda się doprowadzić do wystąpienia błędu, którego aplikacja nie obsługuje prawidłowo, to będzie mógł uzyskać szczegółowe informacje o systemie, sprawić, że aplikacja nie będzie odpowiadać na żądania, przełamać mechanizmy zabezpieczeń, lub nawet doprowadzić do awarii serwera.

Mechanizmy przechowywania danych, które nie zapewniają bezpieczeństwa
Aplikacje internetowe często wykorzystują funkcje kryptograficzne do zabezpieczania informacji i danych uwierzytelniających. Zastosowanie tych funkcji jest trudne, a kod integrujący je z resztą aplikacji jest trudny do napisania, co często powoduje, że ochrona danych jest słaba.

Odmowa działania aplikacji
Ten problem polega na tym, że atakującym uda się wykorzystać zasoby aplikacji do tego stopnia, iż jej pełnoprawni użytkownicy nie będą w stanie z niej korzystać. Atakujący mogą także odciąć użytkowników aplikacji od ich kont, a nawet doprowadzić do całkowitej awarii aplikacji.

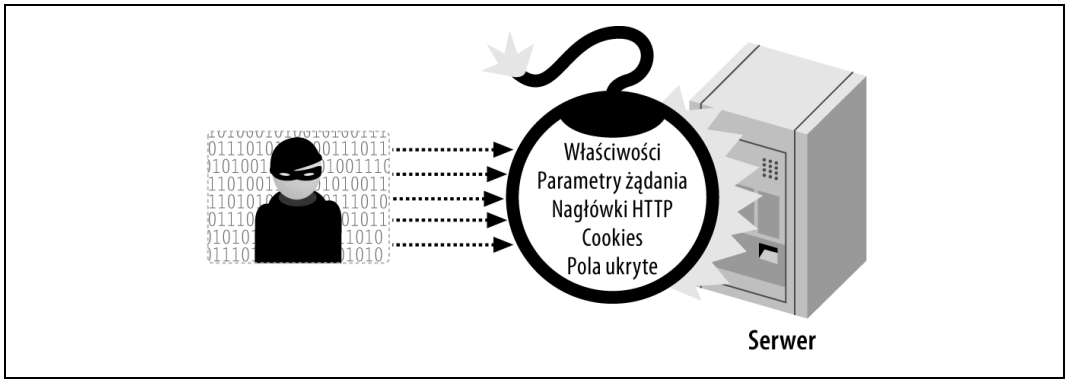
Niebezpieczne zarządzanie konfiguracją
Zastosowanie pewnego standardu konfiguracji serwera ma kluczowe znaczenie dla bezpieczeństwa aplikacji internetowych. Wiele aspektów konfiguracji serwerów ma wpływ na bezpieczeństwo, a ich domyślne ustawienia nie zawsze są optymalne.

Oczywiście istnieje znacznie więcej słabych punktów, które mogą stanowić zagrożenie dla bezpieczeństwa aplikacji, niemniej jednak powyższa lista obejmuje te najważniejsze i najczęściej występujące.

Teraz, kiedy poznałeś już 10 najczęściej spotykanych słabych punktów aplikacji internetowych, przyjrzyjmy się każdemu z nich nieco dokładniej.

Niezweryfikowane dane wejściowe

Jeśli z niniejszej książki miałbyś zapamiętać tylko jedną informację, to niech nią będzie ta: **nie można ufać żadnym informacjom pochodzącym z klienta lub przeglądarki**. Rysunek 2.2 pokazuje, gdzie często mogą się pojawiać niezweryfikowane dane wejściowe.



Rysunek 2.2. Niezweryfikowane dane wejściowe

Trzeba pamiętać, że aplikacje internetowe są bezstanowe, co oznacza, że poszczególne żądania są od siebie całkowicie odseparowane, a pomiędzy nimi występują przerwy. W trakcie jednej z takich przerw, przed przesłaniem żądania na serwer, atakujący może zmodyfikować dowolną jego część. Wartości zapisane w nagłówkach, cookies, polach formularzy, polach ukrytych, parametrach łańcucha zapytania, informacjach o kliencie — wszystkie te dane są zagrożone. Oznacza to, że bez przeprowadzenia odpowiedniej weryfikacji nie można ufać żadnym danych przesyłanym z przeglądarki.

Zawsze należy weryfikować dane pochodzące ze źródeł zewnętrznych. Dane trafiające do aplikacji ze źródeł zewnętrznych, takich jak użytkownicy, kanały informacyjne i inne aplikacje, zawsze powinny być weryfikowane.

Weryfikacja pozytywna oraz negatywna

Jednym z popularnych rozwiązań stosowanych przez programistów jest poszukiwanie konkretnych danych wykorzystywanych do przeprowadzania ataków. Kiedy takie dane uda się odnaleźć, zostają one usunięte. Rozwiązania tego typu określane są mianem **weryfikacji negatywnej**.

Weryfikacja negatywna ma jednak pewną wadę: nie można znać wszystkich potencjalnych metod ataku i zabezpieczyć aplikacji przed nimi.

Poza tym jest tyle różnych metod pozwalających na zakodowanie lub inną zmianę postaci danych, iż uzyskanie 100-procentowej pewności, że żadne spreparowane i wrogie dane nie przedostaną się do systemu, jest praktycznie niemożliwe.

Znacznie lepszym rozwiązaniem jest zastosowanie **weryfikacji pozytywnej**. Polega ona na określeniu, jak powinny wyglądać prawidłowe dane, i podejmowaniu odpowiednich działań, jeśli dane, które dotarły do aplikacji, nie są zgodne z tymi prawidłowymi wzorcami. Na przykład, jeśli dane wpisane przez użytkownika w polu *Nazwisko* będą się zaczynały od znaku apostrofu, a nie od jakiejś litery, to będzie można uznać, że podane nazwisko nie jest prawidłowe.

Weryfikacja po stronie klienta

Zadziwiająco dużo aplikacji wykorzystuje kod pisany w języku JavaScript do przeglądania formularzy przed ich przesłaniem na serwer i sprawdzeniem, czy podane w nich informacje są prawidłowe.

W przeszłości powszechna była opinia, że przeprowadzanie weryfikacji pól formularza po stronie klienta przy wykorzystaniu do tego celu mocy obliczeniowej przeglądarki jest chytrym rozwiązaniem.

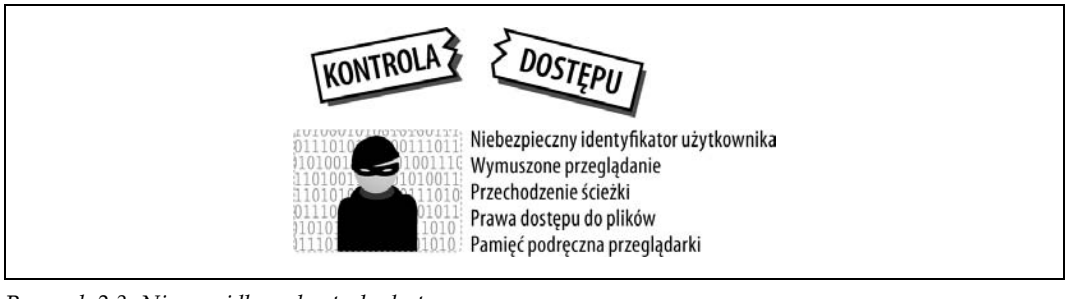
Niestety, ponieważ cały ten kod ma działać po stronie przeglądarki, nie ma żadnej gwarancji, że faktycznie zostanie wykonany. Atakujący może przejrzeć ten kod i ominąć go samodzielnie, przesyłając formularz z dowolnie spreparowanymi danymi.

Rozmywanie

Modyfikowanie danych oraz wprowadzanie fałszywych danych w celu doprowadzenia do awarii działających aplikacji internetowych określane jest jako **rozmywanie** (ang. *fuzzing*). Jest to najprawdopodobniej najpopularniejszy sposób przeprowadzania ataków na aplikacje internetowe. Skrypty i narzędzia używane przez internetowych napastników zaczynają powoli automatyzować przeprowadzanie ataków tego typu, a zatem można przypuszczać, że w przyszłości staną się one jeszcze bardziej popularne.

Nieprawidłowa kontrola dostępu

Kontrola dostępu to proces polegający na ograniczaniu możliwości dostępu do funkcji i danych systemu. Nie każdy powinien mieć dostęp do wszystkiego. Model uwierzytelniania aplikacji internetowych jest ściśle powiązany z wykorzystywanymi w nich rolami oraz funkcjonalnością aplikacji, a użytkownik powinien mieć możliwość wykonywania wyłącznie tych operacji, które dopuszcza przydzielona mu rola. Rysunek 2.3 przedstawia kilka potencjalnych słabych punktów, związanych z nieprawidłowo działającymi mechanizmami kontroli dostępu.



Rysunek 2.3. Nieprawidłowa kontrola dostępu

Prawidłowe zaimplementowanie mechanizmów kontroli dostępu jest trudne. Programiści stosunkowo często nie doceniają złożoności i trudności tego zagadnienia. Niektórzy próbują implementować własne metody autoryzacji, co stwarza zagrożenie dla bezpieczeństwa aplikacji. Zastosowanie wypróbowanego modelu autoryzacji jest znacznie lepszym rozwiązaniem niż tworzenie własnego z nadzieją, że uda się obsłużyć wszystkie miejsca i sytuacje wymagające kontroli dostępu.

Zasada najniższych uprawnień: użytkownik powinien mieć możliwość wykonywania tylko absolutnie niezbędnych czynności.

Interfejsy administracyjne

Kolejny problem związany z autoryzacją pojawia się, kiedy użytkownicy korzystający z aplikacji internetowej mają dostęp do interfejsów lub funkcji administracyjnych. Interfejsy te stanowią łakomy kąsek dla napastników atakujących aplikację, gdyż jeśli uda się do nich dostać, napastnik będzie mógł dowolnie zwiększyć swoje uprawnienia.

Rozdział obowiązków — należy przypisywać użytkownikom role i przydzielać różne poziomy uprawnień. Należy kontrolować sposób, w jaki aplikacja jest tworzona, testowana i wdrażana, oraz to, kto ma dostęp do jej danych.

Interfejsy administracyjne ze względu na swoje ogromne możliwości powinny być wdrażane niezależnie od podstawowej aplikacji internetowej; dzięki temu będzie można monitorować i określać prawa dostępu bardziej dokładnie.

Wadliwe uwierzytelnianie i zarządzanie sesjami

Stój! Kto idzie? Kim są osoby logujące się do aplikacji? Skąd to wiemy? Jak można się przekonać, czy osoby korzystające z aplikacji faktycznie są tymi, za które się podają?

Przed frontowymi drzwiami do naszej internetowej aplikacji stoją systemy uwierzytelniania. Wymagają one, by użytkownicy przed „wejściem" do aplikacji przeszli rodzaj testu. Każdy taki test jest uważany za czynnik uwierzytelniania i użytkownik musi go przejść, aby uzyskać dostęp do systemu. Uwierzytelnianie zostało symbolicznie przedstawione na rysunku 2.4.

Rysunek 2.4. Wadliwe uwierzytelnianie

Czym jest czynnik uwierzytelniania?

Czynnik uwierzytelniania to pewna informacja wykorzystywana do potwierdzenia tożsamości osoby. Czterema najlepiej znanymi i najczęściej stosowanymi czynnikami uwierzytelniania są:

- coś, co użytkownik zna, na przykład hasło lub kod;
- coś, co użytkownik ma, na przykład karta kredytowa lub specjalne urządzenie sprzętowe (tak zwany *token*);
- coś związanego z samą osobą użytkownika, na przykład odcisk palca, wzór siatkówki oka lub inne cechy biometryczne;
- miejsce przebywania, na przykład fizyczna lokalizacja użytkownika.

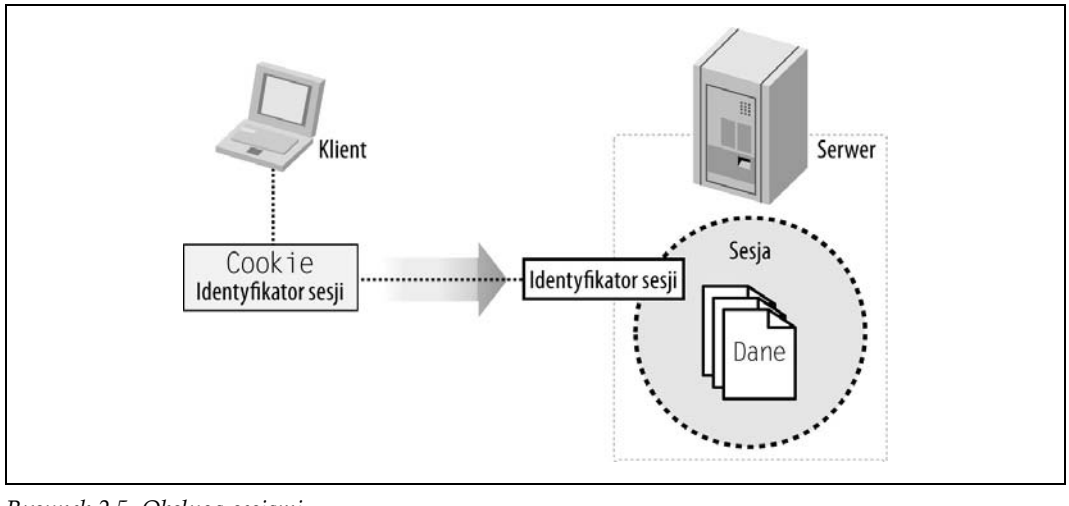
Informacje uwierzytelniające

W przypadku aplikacji internetowych najczęściej stosowanymi informacjami uwierzytelniającymi są nazwa użytkownika i hasło. Istnieją pewniejsze metody uwierzytelniania, takie jak cechy biometryczne lub certyfikaty cyfrowe, niemniej jednak ich koszt jest zbyt duży, by stosować je w aplikacjach internetowych.

Zdecydowanie najważniejsze jest, by system uwierzytelniania był bezpieczny. Nawet dobre mechanizmy uwierzytelniania mogą zostać przełamane w efekcie wystąpienia błędu, zastosowania niewłaściwej konfiguracji, braku dostępu (na przykład do bazy danych z informacjami uwierzytelniającymi), nieodpowiedniego zarządzania nazwą użytkownika i hasłem dostępu i tak dalej.

Interfejsy administracyjne

Ponieważ interfejsy administracyjne mają duże możliwości i mogą wykonywać operacje wymagające wyższych uprawnień, należy je lepiej ochraniać. Na przykład można zastosować dodatkowe czynniki uwierzytelniania. Optymalnym rozwiązaniem jest zastosowanie co najmniej dwóch czynników. Rysunek 2.5 pokazuje sposób obsługi sesji przy wykorzystaniu identyfikatora sesji oraz cookie.



Rysunek 2.5. Obsługa sesjami

Obsługa sesji

Czy pamiętasz, że protokół HTTP jest bezstanowy? Oznacza to, że **nie** jest on wyposażony w mechanizmy obsługi sesji. Obsługa sesji została dodana do aplikacji internetowych oraz serwerów aplikacji jako sposób pozwalający na zachowanie stanu. Często się jednak zdarza, że programiści sami obsługują zarządzanie stanem.

Serwery udostępniają ograniczone mechanizmy obsługi sesji, które zazwyczaj bazują na wykorzystaniu nagłówków HTTP oraz cookies. Sesje należy jednak traktować jako obiekty chronione, a otrzymanie sesji przez użytkownika powinno być poprzedzone jego uwierzytelnieniem.

A zatem początek sesji powinien być wyznaczany przez uwierzytelnienie. W rzeczywistości w ogóle nie należy tworzyć sesji, jeśli użytkownik nie został uwierzytelniony.

Po przeprowadzeniu uwierzytelnienia system musi zapewnić, że gdy użytkownik ponownie odwiedzi aplikację, zostanie rozpoznany bez konieczności ponownego uwierzytelniania. Niestety, większość systemów internetowych wykorzystuje do tego celu cookies. Aplikacja tworzy cookie zawierające identyfikator użytkownika. Identyfikator ten pozwala, by w przyszłości, w momencie nadesłania nowego żądania, zostało ono powiązane z sesją użytkownika.

Koniecznie należy zrozumieć i zapamiętać, że po udanym przeprowadzeniu uwierzytelniania wykorzystującego dwa czynniki (takie, jak identyfikator użytkownika i hasło) kolejne żądania będą obsługiwane na podstawie tylko jednego czynnika uwierzytelniania — identyfikatora użytkownika.

Należy zauważyć, że ze względu na bezstanowy charakter protokołu HTTP cała komunikacja z klientem (bądź przeglądarką) powinna być realizowana przy wykorzystaniu bezpiecznego kanału, takiego jak HTTPS (SSL/TLS). Jest to konieczny warunek zachowania integralności cookie przechowującego identyfikator użytkownika oraz wszystkich przesyłanych danych.

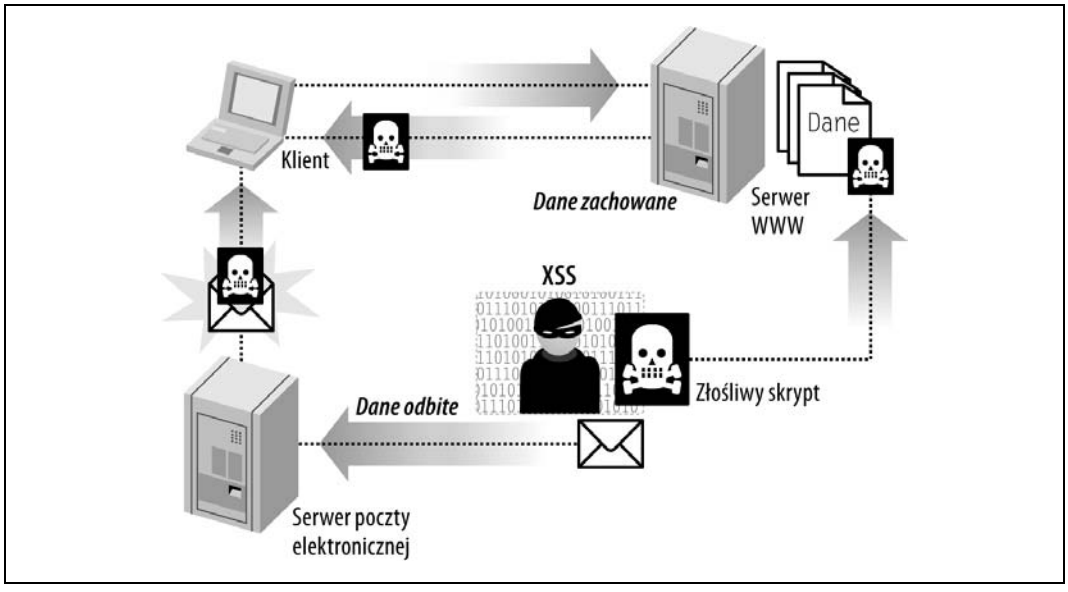
Nie pozwalaj na ponowne wizyty starych gości

Często systemy uwierzytelniania aplikacji są tworzone w taki sposób, by rozpoznawały użytkowników odwiedzających aplikację po dłuższym czasie. Nie należy jednak wpuszczać użytkownika do aplikacji tylko i wyłącznie dlatego, że kiedyś się do niej zalogował.

Jeśli sesja użytkownika wygasła lub jeśli od ostatniej wizyty użytkownika minął dłuższy okres, należy ponownie go uwierzytelnić.

Ataki typu cross-site scripting (XSS)

Ataki typu **cross-site scripting** (w skrócie: XSS) wykorzystują słabe punkty aplikacji internetowych, używając przy tym danych dostarczonych przez atakującego, które w dynamiczny sposób zostają wyświetlone w przeglądarce użytkownika. Dane dostarczane przez napastników przybierają zazwyczaj formę skryptów i są wykonywane w przeglądarce użytkownika. Sposób przeprowadzania ataków typu cross-site scripting z wykorzystaniem odbitych oraz zachowanych danych przedstawiono na rysunku 2.6.



Rysunek 2.6. Atak typu cross-site scripting

Zazwyczaj ataki XSS przybierają jedną z dwóch form: ataku z wykorzystaniem danych zachowanych lub odbitych. W przypadku ataku z wykorzystaniem danych zachowanych atakujący zapisuje swój skrypt na serwerze (na przykład w bazie danych). Później, gdy ofiara odwiedzi witrynę WWW, w dynamiczny sposób wyświetli ona zachowany złośliwy kod, co spowoduje przeprowadzenie ataku. Z kolei atak z wykorzystaniem danych odbitych polega na tym, że atakujący wstawia swój skrypt do zmiennej żądania lub parametru łańcucha zapytania i przekazuje łącze do ofiary.

Ataki XSS pozwalają napastnikom także na wyświetlanie w przeglądarce wybranej zawartości HTML, wykonywanie w przeglądarce dowolnych skryptów (napisanych w języku JavaScript lub VB Script) oraz umieszczanie na stronach złośliwego kodu (takiego, jak aplety, kontrolki ActiveX lub Flash). Ataki tego typu mogą powodować niezamierzone ujawnienie danych, doprowadzić do zniszczenia witryny WWW, pozwalać na przechwytywanie sesji, przejmowanie tożsamości i zasobów konta, podszywanie się oraz uniemożliwianie obsługi innych żądań.

Jeszcze innym sposobem przeprowadzania ataków XSS jest wykorzystanie serwera WWW oraz jego domyślnych stron i mechanizmów obsługi błędów. Często zdarza się, że serwery WWW oraz serwery aplikacji wyświetlają na stronach informujących o błędach dane przekazane w żądaniu. Jeśli atakującemu uda się umieścić kod XSS w żądaniu, a serwer umieści go na stronie informacyjnej i zwróci do użytkownika, to taka strona informacyjna może stać się narzędziem do przeprowadzania ataków.

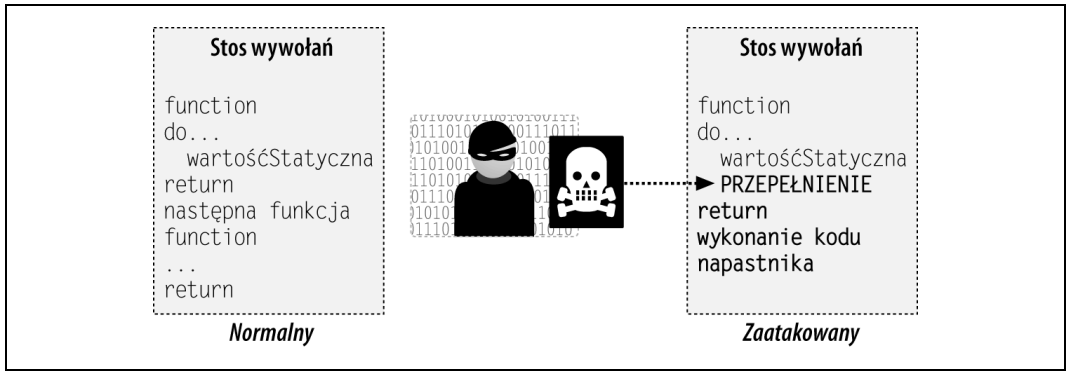
Wiele witryn ma słabe punkty pozwalające na przeprowadzanie ataków XSS. Zapewne dlatego jest to obecnie najczęściej spotykany typ ataków na aplikacje internetowe. Usterki i słabe punkty pozwalające na stosowanie ataków XSS można wykorzystywać na wiele różnych sposobów. Programiści, którzy próbują filtrować i odrzucać złośliwe części żądania, łatwo mogą przegapić potencjalne ataki, które na przykład wykorzystują nowe sposoby kodowania.

Popularność ataków XSS ma jednak także swoją pozytywną stronę. Otóż powstało wiele narzędzi ułatwiających wykrywanie słabych punktów aplikacji, które mogłyby zostać wykorzystane do przeprowadzenia ataku.

Najlepszym i najbezpieczniejszym rozwiązaniem, jakie można zastosować, jest kodowanie wszystkich dynamicznych danych przed ich przesłaniem do przeglądarki. W takim przypadku cały kod skryptowy zostanie potraktowany jako tekst, a przeglądarka, zamiast go wykonać, wyświetli go.

Przepełnienie bufora

Przepełnienie bufora jest zapewne najlepiej znanym problemem występującym we wszelkiego typu aplikacjach komputerowych. Najczęściej występuje ono w aplikacjach, które samodzielnie zarządzają pamięcią oraz zasobami i wykorzystują dane wejściowe, których długość i typ nie zostały sprawdzone. Atak polegający na przepełnieniu bufora został przedstawiony na rysunku 2.7.



Rysunek 2.7. Przepełnienie bufora

Usterki umożliwiające przeprowadzenie ataków tego typu jest trudno wykryć, gdyż zazwyczaj występują one wyłącznie w sytuacji, gdy aplikacja znajduje się w ściśle określonym stanie.

Na przykład klasyczny scenariusz przeprowadzenia ataku wykorzystującego przepełnienie bufora polega na tym, że atakujący znajduje obszar, w którym aplikacja nie sprawdza długości zapisanych danych. Następnie atakujący umieszcza w nim wartość, której długość przekracza wielkość obszaru przydzielonego przez aplikację. W efekcie dane umieszczone na stosie wywołań zostają nadpisane przez dane podane przez napastnika, który zyskuje w ten sposób możliwość określenia, gdzie zostanie przekazane sterowanie.

W ten sposób atakujący może wykonywać dowolne operacje w systemie, wykorzystując przy tym uprawnienia, jakie miał zaatakowany program.

Czasami usterki umożliwiające dokonanie ataku tego typu są ukryte głęboko w kodzie. Zazwyczaj występują one dlatego, że programista dokonał pewnych założeń co do długości danych i z jakiegoś względu zrezygnował z ich weryfikacji.

Przepełnienie bufora w aplikacjach internetowych

Aplikacje internetowe nie są odporne. Atakujący mogą w nich przesyłać „złośliwą" zawartość przy wykorzystaniu formularzy, a jeśli aplikacja nie będzie sprawdzać typów i długości odbieranych danych, to także może paść ofiarą ataku typu przepełnienie bufora.

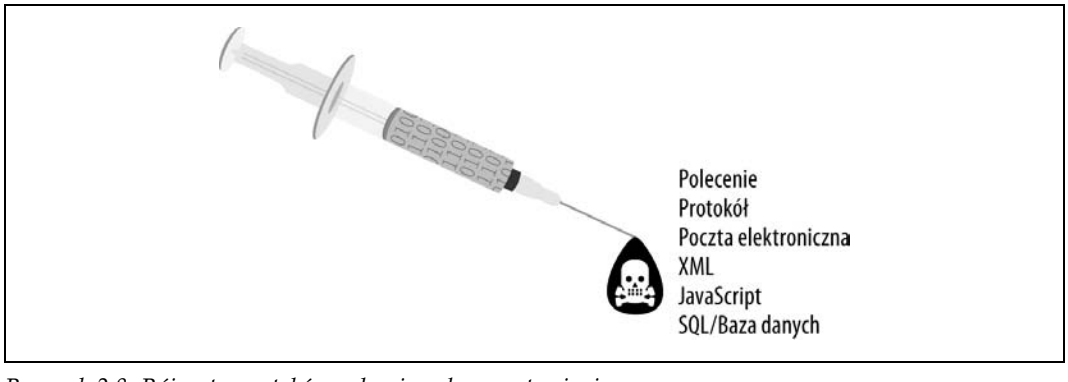
Usterki umożliwiające przeprowadzenie ataku tego typu mogą być umiejscowione w samej aplikacji, serwerze WWW bądź serwerze aplikacji. W końcu serwery wykonują oprogramowanie pisane przez programistów, którzy zawsze robią jakieś założenia.

Może się okazać, że usterka nie jest nawet zlokalizowana w kodzie, do którego ma dostęp twórca aplikacji — może się ona znajdować w którejś z używanych bibliotek lub komponentów.

Dlatego tak ważne jest sprawdzanie typów oraz długości wszystkich odbieranych danych przed ich przekazaniem do innego kodu wykonywanego na serwerze.

Usterki związane ze wstawianiem

Usterki związane ze wstawianiem występują w sytuacji, gdy aplikacja pobiera niezweryfikowane dane wejściowe i zapisuje je w zmiennych, używanych następnie do uzyskania dostępu i korzystania z innych zasobów systemowych, takich jak bazy danych SQL, polecenia systemowe oraz inne programy. Różne rodzaje ataków polegających na wstawianiu zostały przedstawione na rysunku 2.8.



Rysunek 2.8. Różne typy ataków polegających na wstawianiu

Ponieważ do zmiennych dodawane są niezweryfikowane dane wejściowe, atakujący może zmienić te dane w dowolny sposób, określając w ten sposób informacje, które zostaną później przekazane do systemu operacyjnego bądź innego programu. Za każdym razem, gdy aplikacja używa dowolnego zasobu interpretera lub systemu operacyjnego, jest narażona na przeprowadzenie ataku polegającego na wstawianiu.

Szczególnie narażone na takie ataki są funkcje zewnętrzne, takie jak wysyłanie poczty elektronicznej lub korzystanie z baz danych przy wykorzystaniu zapytań SQL. Ataki polegające na wstawianiu można wykrywać w prosty sposób. W Internecie można znaleźć wiele skryptów i programów, które robią to automatycznie.

Niewłaściwa obsługa błędów

Nawet w przypadku, gdy w naszej aplikacji wydarzy się jakaś awaria, musimy zwrócić uwagę, by nie zdradzić zbyt wielu informacji o używanym środowisku. Każda informacja, jaką zdobędzie napastnik, może zwiększyć możliwość przeprowadzenia pomyślnego ataku.

Podczas pracy w środowisku roboczym, przed wdrożeniem aplikacji w środowisku produkcyjnym, każdy chce wiedzieć, gdzie występują problemy i usterki; dlatego też tworzone są wszelkiego rodzaju dzienniki i kod testujący, ułatwiający wykrywanie problemów. Kiedy jednak kod aplikacji będzie już gotów do umieszczenia w środowisku produkcyjnym, wszelki kod testujący powinien być usunięty.

Jeśli programista zapomni usunąć z aplikacji kod testujący, w przeglądarce mogą być wyświetlane informacje o błędach oraz zrzuty stosu wywołań. Zarówno komunikaty o błędach, jak i zrzuty stosu wywołań mogą dostarczyć napastnikom bardzo cennych informacji. Dzięki nim napastnik może poznać kluczowe szczegóły dotyczące stosowanej infrastruktury: nazwy zmiennych i metod, schemat przepływu danych i tak dalej.

Komunikaty o błędach należy skonfigurować w taki sposób, by prezentowały jedynie te informacje, które powinny dotrzeć do użytkownika. Większość serwerów WWW oraz serwerów aplikacji zawiera grupę domyślnych stron z informacjami o błędach oraz narzędzi pomocnych przy testowaniu aplikacji. Zarówno te domyślne strony, jak i narzędzia mogą jednak ujawnić szczegółowe informacje o środowisku, w którym aplikacja jest wykonywana. Informacje tego typu są nieocenione dla osób przygotowujących ataki. Jeśli użytkownik planujący atak na aplikację uzyska możliwość przeglądania komunikatów o błędach, tworzonych, by ułatwić jej testowanie (takich, jak informacje o wyjątkach i postać stosu wywołań), to będzie mógł wykorzystać te informacje do dokładniejszego wybrania miejsca ataku oraz określenia sposobu, w jaki należy go przeprowadzić.

Kolejnym źródłem problemów są sytuacje, w których atakujący jest w stanie doprowadzić do wystąpienia błędu w aplikacji, który pozwoli mu ominąć pewne jej kluczowe mechanizmy (na przykład uwierzytelnianie lub kontrolę dostępu). Sytuacja taka występuje, gdy aplikacja reaguje pozytywnie na wystąpienie problemu. W przypadku implementacji mechanizmów zabezpieczeń, takich jak uwierzytelnianie lub kontrola dostępu, znacznie lepszym rozwiązaniem jest tworzenie kodu, który w razie jakichkolwiek problemów nie będzie zezwalał użytkownikowi na dostęp do aplikacji. Wszystkie mechanizmy związane z zabezpieczeniami powinny odmawiać dostępu aż do momentu pomyślnego uwierzytelnienia użytkownika oraz określenia jego uprawnień.

Poniższy listing demonstruje, w jaki sposób nieprawidłowo napisany kod obsługi błędów może doprowadzić do błędnego działania mechanizmu uwierzytelniania:

```
authenticated = true;
try {
  if (authenticateUser(user, password) {
    // użytkownik został uwierzytelniony
    authenticated = true;
  } else {
    // użytkownik nie został uwierzytelniony
    authenticated = false;
  }
} catch (Exception e) {
  System.out.print("Błąd uwierzytelniania: " + e.message());
}
```

Jeśli w metodzie `authenticateUser()` zostanie zgłoszony wyjątek, będzie można kontynuować uwierzytelnianie, gdyż sam test określający, czy użytkownik powinien być uwierzytelniony, czy nie, zostanie pominięty.

Bezpiecznie obsługuj sytuacje awaryjne. Jeśli jakiś warunek może doprowadzić do awarii aplikacji lub sprawić, że jakaś jej część nie zostanie wykonana prawidłowo, upewnij się, że domyślny stan, w jakim znajdzie się aplikacja, będzie bezpieczny.

Kod obsługi błędów implementowany w mechanizmach związanych z zabezpieczeniami aplikacji powinien być także znacznie bardziej podejrzliwy. To właśnie nim hackerzy będą najbardziej zainteresowani. Ze względu na znaczenie funkcjonalności związanej z uwierzytelnianiem i kontrolą dostępu można przypuszczać, że napastnicy znacznie częściej będą próbowali wywoływać błędy właśnie w tych, a nie innych fragmentach aplikacji. Z tych samych powodów mechanizmy zabezpieczeń powinny znacznie częściej korzystać z dzienników. W ich przypadku należy rejestrować absolutnie wszystko! Rejestracja błędów w kodzie związanym z zabezpieczeniami aplikacji ma kluczowe znaczenie, jeśli zależy nam na tworzeniu dobrych i przydatnych dzienników. W końcu po udanym ataku dziennik może być wszystkim, co nam pozostanie.

Niebezpieczne mechanizmy przechowywania

Aplikacje zazwyczaj muszą przechowywać wrażliwe dane lub informacje. W niektórych przypadkach przechowywane dane (które w danej chwili nie są używane) wymagają dodatkowego zabezpieczenia. Takie informacje, jak hasła, numery ubezpieczeń społecznych czy kart kredytowych, wymagają takich zabezpieczeń, by nie można ich było wykorzystać, jeśli zostaną przypadkowo znalezione w systemie. W takich przypadkach najczęściej stosowanym rozwiązaniem jest użycie szyfrowania.

Chociaż mechanizmy kontroli dostępu stały się stosunkowo łatwe do zaimplementowania i użycia, programiści i tak zazwyczaj popełniają błędy podczas stosowania ich w aplikacjach internetowych. Programiści mogą przeceniać poziom ochrony, jaki zapewnia szyfrowanie, i nie przykładać odpowiedniej wagi do wykorzystania innych mechanizmów zabezpieczeń.

OWASP wskazuje często popełniane błędy:

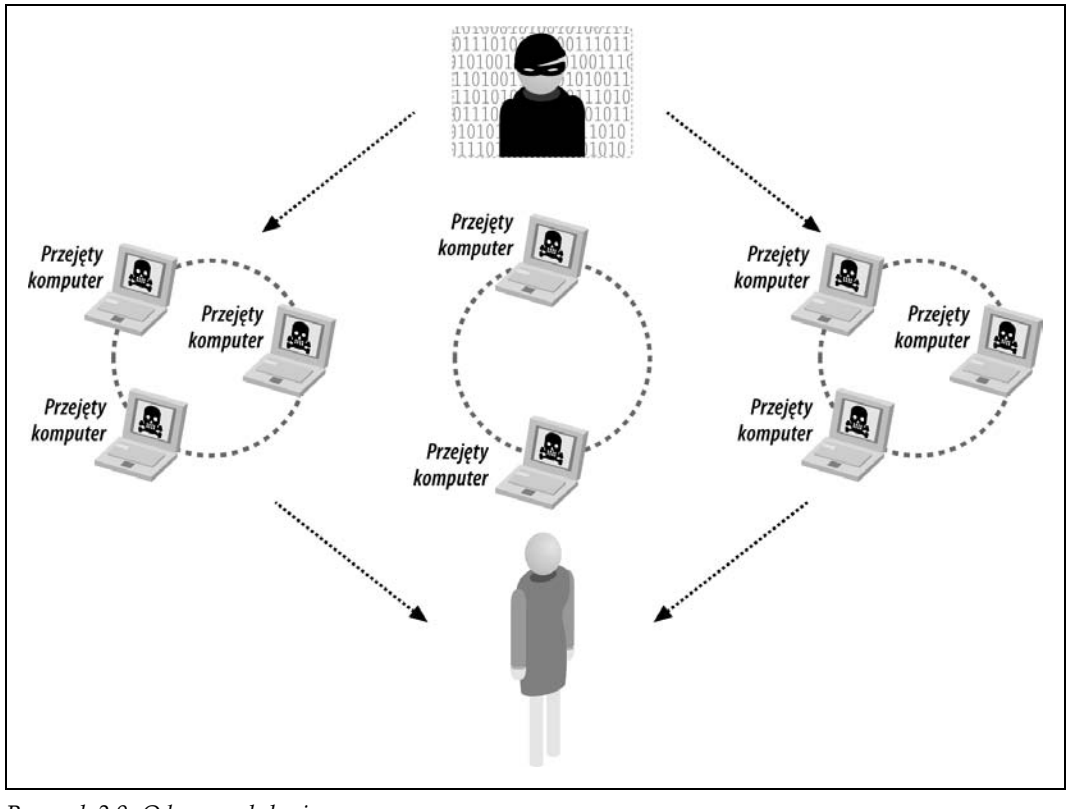
- całkowitą rezygnację z szyfrowania krytycznych danych;
- zastosowanie niewłaściwych sposobów przechowywania kluczy, certyfikatów i haseł;
- niewłaściwą konfigurację;
- zastosowanie niewłaściwych sposobów przechowywania tajnych danych w pamięci;
- słabe generatory liczb pseudolosowych;
- niewłaściwy dobór algorytmów;
- próby wymyślania nowych algorytmów szyfrujących;
- rezygnację z implementacji mechanizmów zmiany kluczy szyfrujących oraz innych wymaganych procedur pielęgnacyjnych.

Jeśli zastosowane mechanizmy szyfrowania zostaną przełamane, bezpieczeństwo danych będzie zagrożone. Dlatego też poprawna konfiguracja oraz efektywne zarządzanie mechanizmami szyfrowania mają kluczowe znaczenie dla bezpieczeństwa aplikacji.

Odmowa działania aplikacji

Czasami nie mamy kontroli nad niektórymi zdarzeniami. Doskonale sobie radzimy. Prowadzimy fantastyczną, nową witrynę WWW. Jest ona na tyle interesująca, że ktoś zamieścił informację o niej w popularnym serwisie agregującym, takim jak Digg. I nagle BUM! Nasz niewielki serwer umieszczony w piwnicy zawiesza się z powodu nawału żądań od osób ze wszystkich stron świata, które nagle chcą obejrzeć naszą witrynę.

Innym razem jakiś napastnik może znaleźć sposób na to, by aplikacja korzystała z coraz to większej ilości zasobów systemowych — pamięci, dysku itp. — i zużywała je aż do momentu, gdy serwer nie będzie w stanie dalej obsługiwać nadsyłanych żądań. Niestety, czasami będziemy musieli zastosować efektywne metody rozróżniania, co jest atakiem, a co zwyczajnym ruchem na witrynie. Atak poprzez odmowę obsługi został przedstawiony na rysunku 2.9.



Rysunek 2.9. Odmowa obsługi

Problem ten jest dodatkowo komplikowany przez wiele innych czynników, takich jak bezstanowość protokołu HTTP oraz brak pewności, że danym przesyłanym w żądaniu można zaufać. Czynniki te powodują, że znacznie trudniej jest odnajdywać niebezpieczne żądania, na przykład na podstawie przesyłanych w nich informacji, takich jak adres IP, i odrzucać je.

Najpopularniejsze zasoby, które można wykorzystać do przeprowadzenia ataku tego typu, to:

- pamięć,
- przepustowość,
- uchwyty plików,
- połączenia z bazą danych,
- wątki,
- mechanizmy rejestracji danych w dziennikach,
- pojemność obszaru przeznaczonego do przechowywania plików lub danych.

Kiedy atakujący znajdzie sposób, by wykorzystać wszystkie lub tylko niektóre spośród wymaganych zasobów, będzie w stanie uniemożliwić korzystanie z systemu normalnym użytkownikom.

Na przykład, jeśli witryna pozwala nieuwierzytelnionym użytkownikom pobierać informacje o ruchu na forach dyskusyjnych, to dla jednego żądania HTTP może być tworzonych wiele połączeń z bazą danych. Atakujący bez większych problemów może wygenerować na tyle wiele żądań, by w całości wyczerpać dostępną pulę połączeń z bazą danych. W takim przypadku zabraknie połączeń do obsługi żądań normalnych użytkowników witryny.

Inny typ ataku może polegać na celowym generowaniu wyjątku w celu zapisywania informacji o nim w pliku dziennika systemowego. W ten sposób plik dziennika może rosnąć aż do wyczerpania się całego wolnego miejsca na dysku.

Istnieją setki różnych sposobów przeprowadzania ataków polegających na odmowie obsługi, a większość z nich można bez trudu przeprowadzić przy wykorzystaniu kilku wierszy kodu. Choć nie ma żadnej metody pozwalającej pewnie zabezpieczyć się przed tymi atakami, to jednak można utrudnić ich przeprowadzanie i zmniejszyć ich skuteczność.

Niebezpieczne zarządzanie konfiguracją

Właściwa konfiguracja ma kluczowe znaczenie dla bezpieczeństwa działającej aplikacji internetowej.

Konfiguracja aplikacji, jak również konfiguracja serwera i zasobów systemowych, musi być odpowiednio przygotowana. Bardzo często programiści składają odpowiedzialność za przygotowanie konfiguracji środowiska serwera na barki administratorów. Choć może się wydawać, że takie rozwiązanie jest prawidłowe, to jednak programiści nie mogą całkowicie uniknąć zajmowania się konfiguracją.

Aplikacje internetowe są w tak wielkim stopniu zależne od warstwy internetowej oraz warstwy serwera aplikacji, że trzeba przeanalizować ich konfigurację, aby uzyskać pewność, iż środowisko aplikacji faktycznie jest bezpieczne.

Zabezpieczenie ustawień domyślnych — należy się upewnić, że domyślna konfiguracja systemu jest bezpieczna.

OWASP wskazuje na następujące problemy związane z konfiguracją, które mogą występować w aplikacjach internetowych:

- nieuaktualnione błędy związane z bezpieczeństwem oprogramowania serwera;
- błędy oprogramowania serwera lub usterki w jego konfiguracji, które umożliwiają generowanie listingu zawartości katalogu lub ataki metodą przechodzenia katalogów;
- niepotrzebne pliki domyślne, pliki kopii zapasowych, pliki przykładowe, w tym skrypty, aplikacje, pliki konfiguracyjne oraz strony WWW;
- niewłaściwie ustawione uprawnienia do plików i katalogów;
- udostępnienie niepotrzebnych usług, w tym usług zarządzania zawartością i zdalnej administracji;
- domyślne konta użytkowników wraz z ich domyślnymi hasłami;
- włączone i dostępne funkcje administracyjne i testujące;
- komunikaty o błędach zawierające niepotrzebnie dużo informacji (zagadnienie to zostało rozwinięte w punkcie poświęconym obsłudze błędów);
- nieprawidłowo skonfigurowane certyfikaty SSL oraz ustawienia szyfrowania;
- zastosowanie samodzielnie podpisywanych certyfikatów cyfrowych do uwierzytelniania oraz zabezpieczenia się przed atakami typu „człowiek pomiędzy" (ang. *man-in-the-middle*);
- zastosowanie domyślnych certyfikatów;
- nieodpowiednie uwierzytelnianie podczas korzystania z systemów zewnętrznych.

Niektóre z tych problemów można łatwo wykryć przy wykorzystaniu ogólnie dostępnego oprogramowania analizującego bezpieczeństwo aplikacji. Po wykryciu problemu można go łatwo wykorzystać i doprowadzić do całkowitego przełamania zabezpieczeń witryny. Udane ataki mogą także doprowadzić do przełamania zabezpieczeń innych systemów działających na serwerze, takich jak bazy danych oraz sieci korporacyjne. Oznacza to, że do skutecznego zabezpieczenia witryny konieczne jest wykorzystanie bezpiecznego oprogramowania oraz odpowiedniej, bezpiecznej konfiguracji.

Inne słabe punkty

Lista 10 najczęściej występujących problemów bezpieczeństwa opracowana przez OWASP jest doskonałym miejscem, od którego należy zacząć zabezpieczanie aplikacji. Niemniej jednak są także inne obszary aplikacji, którym warto się przyjrzeć. Należą do nich:

- niepotrzebny lub złośliwy kod;
- niewłaściwe zastosowanie wątków i innych mechanizmów programowania współbieżnego;
- gromadzenie informacji, do których nie ma się praw;
- nieodpowiednie wykorzystanie odpowiedzialności oraz niewystarczająca rejestracja operacji w dziennikach systemowych;

- przekłamywanie danych;
- niewłaściwe wykorzystanie pamięci podręcznej i pul oraz nieprawidłowe wielokrotne korzystanie z zasobów.

Dodatkowe źródła informacji

Microsoft, artykuł „Improving Web Applications Security", *http://msdn2.microsoft.com/en-us/library/aa302419.aspx*.

OWASP.org, „Open Web Applications Security Project (OWASP)", *http://www.owasp.org/* (sprawdzono 26 września 2007 roku).