Alicja i Bob BEZPIECZEŃSTWO APLIKACJI W PRAKTYCE

Tanya Janca

Helion

Tytuł oryginału: Alice and Bob Learn Application Security

Tłumaczenie: Krzysztof Bąbol

ISBN: 978-83-283-8283-1

Copyright © 2022 by John Wiley & Sons, Inc., Indianapolis, Indiana

All Rights Reserved. This translation published under license with the original publisher John Wiley & Sons, Inc.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, without either the prior written permission of the Publisher.

Translation copyright © 2022 by Helion S.A.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/alibob
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

Spis treści

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

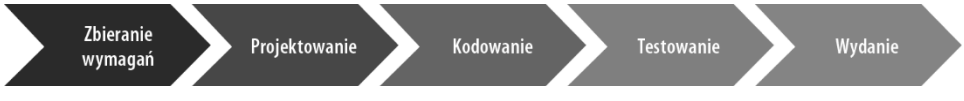
Rozdział

2

Wymagania związane z bezpieczeństwem

Przy tworzeniu aplikacji albo rozpoczynaniu jakiegoś projektu należy zebrać wymagania dla tego, co ma być zbudowane. Nie zależy to od stosowanej metodyki wytwarzania oprogramowania (czy jest to model kaskadowy, zwinny, czy DevOps), języka ani platformy programistycznej, w których pisany jest kod, ani od docelowego typu odbiorców — bez planu nie można zbudować niczego istotnego.

Osoby, które ukończyły studia z informatyki lub inżynierii komputerowej, zapewne dobrze pamiętają obraz z rysunku 2.1. Jest on powszechnie znany jako proces rozwoju systemu informatycznego (ang. *System Development Life Cycle* — SDLC) i składa się z pięciu faz: zbierania wymagań, projektowania, pisania kodu, jego testowania i wydania. W książce będziemy wracali do tego rysunku w celu wyjaśnienia, kiedy może i powinna mieć miejsce każda omawiana przez nas czynność. W tym rozdziale skoncentrujemy się na fazie zbierania wymagań.



Rysunek 2.1. Proces rozwoju systemu informatycznego (SDLC)

Wskazówka **Skrót SDLC jest czasem tłumaczony jako proces rozwoju *oprogramowania* (ang. *software development life cycle*), wtedy gdy nacisk kładziony jest raczej na oprogramowanie niż na system. Obie te definicje są używane wymiennie.**

Kup książkę Poleć książkę

Już podczas pierwszego spotkania projektowego (często zwanego spotkaniem inaugurującym projekt, ang. *project kickoff meeting*) powinna być obecna osoba z zespołu ds. bezpieczeństwa, by brać udział w projekcie od samego zarania. Choć nie będzie ona pracowała cały czas nad projektem, powinna być członkiem zespołu i pozostawać do dyspozycji, żeby wszystkie kwestie i problemy były na czas rozwiązane. Przydzielenie specjalisty ds. bezpieczeństwa do zespołu projektowego jest czasem nazywane *modelem partnerstwa* (ang. *partnership model*), a o nim samym można powiedzieć, że został włączony do zespołu w podejściu macierzowym (ang. *matrixed into a project*). Niezależnie od tego, jak to nazwiemy, osoba ta jest tam po to, by pilnować kwestii bezpieczeństwa (triady CIA) w ciągu całego projektu.

Uwaga **Nie wiem dokładnie, skąd się wziął model partnerstwa. Po raz pierwszy dowiedziałam się o nim od zespołu bezpieczeństwa aplikacji firmy Netflix. Z „włączaniem do zespołu w podejściu macierzowym" zetknęłam się po raz pierwszy w sekretariacie kanadyjskiej izby skarbowej (ang. *Treasury Board Secretariat of Canada*) i również nie znam pochodzenia tego wyrażenia.**

W tym rozdziale założono, że czytelnik ma podstawową wiedzę na temat tego, jak są prowadzone projekty informatyczne i jak wygląda proces wytwarzania oprogramowania.

Wskazówka **Zredaguj umowę o gwarantowanym poziomie wsparcia (ang. *Support Level Agreement* — SLA) między zespołem ds. bezpieczeństwa a innymi zespołami i zdefiniuj w niej rozsądne granice czasu oczekiwania na odpowiedź na zgłoszenia kierowane do zespołu ds. bezpieczeństwa. Zespół ten często powoduje opóźnienia w projektach, a gdy zacznie obowiązywać umowa SLA, będzie to mniej prawdopodobne. Aby osiągnąć najlepsze wyniki, docelowy poziom wsparcia ustal na początku ostrożnie, a z biegiem czasu staraj się o poprawę.**

Wymagania

Przy zbieraniu wymagań dla projektu należy zawsze stawiać sobie pytania o bezpieczeństwo. Specjaliści ds. bezpieczeństwa asystujący przy gromadzeniu wymagań i ich analizie powinni pytać o następujące rzeczy:

- Czy system będzie zawierał dane poufne, wrażliwe lub osobowe (ang. *Personally Identifiable Information* — PII) albo miał z nimi kontakt?
- Gdzie i jak będą przechowywane dane? Czy aplikacja będzie dostępna publicznie (w int*er*necie), czy tylko w sieci wewnętrznej (int*ra*necie)?
- Czy aplikacja będzie przeprowadzała poufne albo istotne zadania (takie jak transfery pieniężne, odryglowywanie drzwi albo dozowanie leków)?
- Czy aplikacja ta dokonuje ryzykownych operacji programowych (np. pozwala użytkownikom przekazywać pliki)?
- Jakiego poziomu dostępności (ang. *level of availability*) potrzeba?

Kup książkę Poleć książkę

- Czy wymagany jest 99,999-procentowy czas sprawności (ang. *up time*)? (Uwaga: podobnego poziomu sprawności nie wymaga tak naprawdę prawie żaden system).

Dobrze by było, gdyby podczas tworzenia listy wymagań przedstawiciel ds. bezpieczeństwa zadawał pytania, a potem dodał do listy wymagań projektu niezbędne wymogi związane z zabezpieczeniami. Może to wyglądać np. tak: „Czy aplikacja będzie pozwalać użytkownikom na przekazywanie (ang. *upload*) plików? Tak? W porządku, ale dodajmy do dokumentu założeń projektu pewne wymagania bezpieczeństwa, by przygotować ten projekt od początku w bezpieczny sposób".

Jeśli tworzysz oprogramowanie, to do Twoich obowiązków należy ochrona bezpieczeństwa i prywatności użytkowników. Zebranie wymagań związanych z bezpieczeństwem pomoże Ci się z tego wywiązać.

Te wymagania zostaną zdefiniowane i wyjaśnione w kolejnych punktach. Na końcu tego rozdziału podano listę kontrolną wymagań, które można dodać do specyfikacji każdego projektu aplikacji sieciowej.

Szyfrowanie

Kryptografia to typ działań matematycznych, których można użyć w stosunku do informacji po to, by ich wartość nie była już zrozumiała; dziedzina ta pozwala ukrywać tajemnice i zabezpieczać prywatność komunikacji. Szyfrowanie (ang. *encryption*) jest dwukierunkowe, tzn. można wymieszać informację, tak by powstał z niej niezrozumiały bałagan, a potem odszyfrować (ang. *decrypt*) ją z powrotem do formy oryginalnej. Wyznaczanie wartości skrótu (ang. *hashing*) to operacja jednokierunkowa; nie da się odzyskać pierwotnej wartości informacji.

Szyfrowanie stosowane jest dość często w celu ochrony tajnych informacji albo transmisji danych, bo system później potrzebuje tych danych z powrotem. To właśnie one mają wartość. Wyznaczanie wartości skrótu jest najczęściej stosowane w celu potwierdzenia tożsamości, uwierzytelniania w systemie, weryfikacji integralności danych albo w odpowiedzi na swego rodzaju wyzwania czy żądania weryfikacji; np. dla nikogo nie jest istotne, jakie naprawdę hasło ma użytkownik, ważne jest tylko to, czy należy dać mu dostęp do systemu, czy też nie. Wartości nie mają dane; ma je potwierdzenie tożsamości (użytkownik zna wartość pierwotną, czyli hasło, i przez to potwierdza swoją tożsamość). Tworzenie skrótów wartości gwarantuje też, że nawet gdyby kiedyś one wyciekły, będą bezużyteczne: ujawnione hasła (w formie zmodyfikowanej i skróconej) nic złodziejowi nie dadzą, bo po wpisaniu ich do systemu nie zostaną rozpoznane.

Uwaga **Choć istnieją obecnie obawy, że komputery kwantowe odeślą do lamusa obecne formy kryptografii i szyfrowania, w książce założono, że tak się jeszcze nie stało. Data, kiedy to może nastąpić, jest nieznana i nikt, łącznie ze mną, nie zna obecnie skutecznego algorytmu ani strategii szyfrowania odpornych na obliczenia kwantowe. Dlatego też zagadnienie to wykracza poza zakres tej książki.**

Aby zapewnić poufność danych, należy je szyfrować w trakcie przesyłania (po drodze od użytkownika, bazy danych, interfejsu API i z powrotem) oraz w spoczynku (w bazie danych). Warto zauważyć, że dzięki temu nikt nie będzie mógł poznać tajnych informacji; jeśli ktoś zdoła uzyskać nieautoryzowany dostęp do danych albo będzie nasłuchiwał ruchu sieciowego za pomocą narzędzia

Kup książkę Poleć książkę

przechwytywania pakietów, nie będzie w stanie zrozumieć tego, co znalazł. Nie da się jednak w ten sposób zapewnić dostępności danych ani ochronić ich integralności. Ktoś nadal będzie mógł usunąć dane z bazy lub je zmienić (co będzie od razu oczywiste, ale jeśli operacje tworzenia kopii zapasowych i przywracania z nich danych nie są zupełnie płynne, stanie się to dość niedogodne). Złośliwy podmiot może przechwycić ruch sieciowy oraz zmienić lub zablokować wiadomości, co nadal będzie powodowało problemy. Mimo to ochrona tajnych danych (C w akronimie CIA) ma istotne znaczenie, a więc niezależnie od tego, jaki jest rodzaj tworzonego systemu, dane powinny koniecznie znajdować się w postaci zaszyfrowanej (a nie skróconej) zarówno w czasie przesyłania, jak i w spoczynku.

Niektórzy mogą argumentować, że dane powinny być szyfrowane nawet wtedy, gdy są używane (w pamięci), jeśli jednak nie są one niezwykle wrażliwe, generalnie nie ma powodu, by traktować to jako wymaganie projektowe. W celu ochrony bardzo poufnych danych zaleca się opróżniać pamięć przy wychodzeniu z programu, wylogowywaniu użytkowników albo kończeniu pracy programu w inny sposób.

Nigdy nie ufaj danym wejściowym systemu

Każde dane trafiające do systemu mogły być potencjalnie zmienione przez osoby niepowołane albo mogą w inny sposób spowodować błędne działanie lub awarię aplikacji. Niezależnie od tego, czy dane wejściowe zostały celowo spreparowane, czy też nie, jeśli spowodują przejście aplikacji do nieprzewidzianego stanu (takiego, który nie był planowany ani nie zaprogramowano jego obsługi), może to być bardzo niebezpieczne. Jeśli aplikacja znajdzie się w nieprzewidzianym stanie, złośliwe podmioty będą w stanie zmusić ją do rzeczy, o których programiście nawet się nie śniło, łącznie z naruszeniem jednego lub kilku aspektów CIA. Program musi być w stanie *pomyślnie* obsłużyć każdy typ danych wyjściowych, nawet nieprawidłowych.

Dane wejściowe aplikacji oznaczają dosłownie wszystko, co nie jest jej częścią, a co mogło być poza nią zmanipulowane.

Uwaga **Jedno z największych zagrożeń dla oprogramowania komputerowego polega na tym, że dane (wartości zawarte w zmiennych albo pochodzące z interfejsu API lub bazy danych) mogą być wykonywane tak, jakby były częścią kodu aplikacji. Uruchamianie kodu, który nie powinien być częścią aplikacji, bywa określane jako podatność „wstrzykiwania" (ang. *injection*) i jest powszechnie uważane przez specjalistów ds. bezpieczeństwa od początku istnienia naszej branży za zagrożenie numer jeden dla bezpiecznego oprogramowania**

[1]. To ryzyko jest powodem sformułowania wielu opisanych w tym rozdziale wymagań projektowych, ale w szczególności właśnie tego, by nie ufać danym wejściowym systemu.

Oto przykłady danych wejściowych aplikacji:

- dane wprowadzane przez użytkownika na ekranie (np. wprowadzanie w polu fraz do wyszukiwania);
- informacje z bazy danych (nawet tej przeznaczonej dla aplikacji);

Kup książkę Poleć książkę

- informacje z interfejsu API (nawet własnoręcznie napisanego);
- informacje z innej aplikacji, z którą dana aplikacja się integruje albo z której przyjmuje dane wejściowe (dotyczy to również aplikacji bezserwerowych i skryptów);
- wartości parametrów URL oraz te zawarte w ciasteczkach i plikach konfiguracyjnych;
- dane lub polecenia z przepływów pracy w chmurze;
- obrazy dołączane z innych witryn (za zgodą ich właścicieli, jak również bez);
- wartości pobierane z internetowych przestrzeni dyskowych (ang. *online storage*).

Lista ta nie jest wyczerpująca. Warto pamiętać o tym, że *wszystko*, co pochodzi spoza programu, może potencjalnie być niszczące.

Uwaga ***Przepływy pracy w chmurze*** **(ang. *cloud workflows*) to wyzwalacze (ang. *triggers*), które są zwykle używane do wywoływania aplikacji bezserwerowych, ale mogą też być wykorzystane do uruchomienia operacji wewnątrz zwykłej aplikacji.**

Aplikacje bezserwerowe **(ang. *serverless applications*) to aplikacje lub skrypty uruchamiane w chmurze, bez potrzeby utrzymywania działającego cały czas serwera. Oznacza to, że jeśli nie są uruchomione, nie korzystają z zasobów infrastruktury. Po wywołaniu aplikacji bezserwerowej uruchamiany jest kontener, w którym wykonywana jest do końca aplikacja lub skrypt, po czym następuje jego destrukcja i zwolnienie zasobów infrastruktury.**

Oto przykłady elementów aplikacji, którymi można manipulować z zewnątrz:

- parametry URL (użytkownik może je zmienić);
- informacja zawarta w ciasteczku, które nie ma ustawionych flag `secure` (bezpieczne) i `HTTPS only` (tylko ruch HTTPS);
- ukryte pola (nie są one zabezpieczone przed atakującymi);
- nagłówki żądań HTTPS;
- wartości wprowadzone na ekranie, którymi da się manipulować nawet po tym, gdy przeszły weryfikację w skrypcie JavaScriptu, dzięki użyciu pośrednika WWW (ang. *web proxy*; więcej informacji na ten temat będzie w dalszej części rozdziału);
- front-endowe platformy programistyczne (ang. *front end frameworks*) niedołączone bezpośrednio do projektu, ale umieszczone w innych miejscach w internecie, do których aplikacja odwołuje się po uruchomieniu;
- kod innych podmiotów dołączany do aplikacji w czasie jej kompilowania (biblioteki, pliki dołączane, platformy programistyczne itd.);
- dołączone do aplikacji obrazy, które są umieszczone w innych miejscach w internecie;
- pliki konfiguracyjne niezarządzane przez programistę aplikacji;
- interfejsy API lub inne usługi wywoływane przez aplikację;
- skrypty, nad którymi twórca aplikacji nie ma kontroli.

Kup książkę Poleć książkę

Programiści zapominają czasem, że nawet te platformy programistyczne i usługi internetowe, które są zaufane, szanowane i mają wsparcie, wciąż mogą stanowić wektory ataku.

W celu efektywnego zastosowania koncepcji „nieufania nigdy danym wejściowym systemu" należy zawsze sprawdzać poprawność wszystkich danych wejściowych (za każdym razem) przed ich użyciem. Dane wejściowe należy uważać za niezaufane do chwili ich zweryfikowania. Przez „weryfikację" należy rozumieć przeprowadzenie testów w celu upewnienia się, że dane wejściowe są odpowiednie i zgodne z oczekiwaniami, a jeśli takie nie są, trzeba je odrzucić. Istnieją przypadki specjalne, w których dane wejściowe należy oczyścić (usunąć wszystko, co może być złe). Zostaną one opisane później. W tym punkcie omawiamy sprawdzanie poprawności danych wejściowych aplikacji.

Oto przykłady weryfikacji wejściowej (ang. *input validation*):

- Oczekiwana jest data urodzenia, więc sprawdź, czy otrzymana wartość rzeczywiście ma format daty, lub/i skonwertuj ją na taki format. Upewnij się też, że dana osoba urodziła się w ciągu ostatnich stu lat (np. tak: `age > current year – 100 && age < current year`). Jeśli podano niewłaściwy format daty (np. „aaaaaaaa"), odrzuć to. Jeśli okaże się, że ta osoba ma 5000 lat, należy to odrzucić. Jeśli z wieku osoby wynika, że jeszcze się nie urodziła, odrzuć to. Wyemituj odpowiedni komunikat o błędzie, stwierdzający, że format daty jest nieprawidłowy, i wskaż, jaki powinien być. Jeśli zaś wartość wieku przekracza 100 lat, wyemituj komunikat o błędzie stwierdzający, że wiek jest niepoprawny.
- W polu ma być imię danej osoby, na które zarezerwowano 80 znaków. Zweryfikuj, czy otrzymane dane wejściowe mają 80 znaków albo mniej oraz czy znaki są odpowiednie dla imienia. Jeśli pole zawiera np. znak %, *[*, *{*, < lub |, nie jest to prawdziwe imię i trzeba odrzucić takie dane wejściowe z odpowiednim komunikatem o błędzie. Wiele nazwisk anglosaskich, np. O'Connor, zawiera apostrofy (') — takie dane wejściowe musisz przyjąć, ale obchodź się z nimi ostrożnie (zakoduj wartość; wkrótce w książce będzie więcej na ten temat). W przypadku imion i nazwisk nieanglosaskich musisz przyjmować znaki diakrytyczne (*ę*, *ą* itp.), litery z alfabetów innych niż łaciński używany do zapisu języka angielskiego, łączniki itp.
- W polu ma być adres poczty elektronicznej; w internecie dostępne są przykłady wyrażeń regularnych służących do sprawdzania takich adresów, a na platformach programistycznych — moduły sprawdzające (ang. *validators*). Proponuję skorzystać z wypróbowanych, przetestowanych i rzetelnych funkcji sprawdzających dostępnych na używanej platformie, które są dość skomplikowane, ale doskonale działają.
- Program wyszukuje dane w bazie i wyświetla na ekranie ciąg ze znalezionego rekordu. Zweryfikuj, czy te dane są zgodne z oczekiwaniem, tak samo jakby pochodziły od użytkownika. Upewnij się, że nie ma tam zapisanych skryptów międzywitrynowych (ang. *stored cross-site scripting* — XSS) lub czegoś innego, co może być potencjalnie szkodliwe. Zawsze przeprowadzaj kodowanie wyjściowe[2] ciągu przed jego wyświetleniem na ekranie.

Uwaga **Atak XSS polega na wstrzyknięciu do aplikacji przez złośliwy podmiot kodu JavaScriptu wykonywanego w przeglądarce na urządzeniu, na którym użytkownik korzysta z aplikacji sieciowej. Jeśli przeprowadzimy kodowanie wszystkich danych wyjściowych przed ich wyprowadzeniem na ekran, ataki XSS zostaną przedstawione jako tekst i nie zostaną uruchomione. Na ekranie pojawi się tylko tekst w rodzaju „`<script>...`", brzydki, ale niegroźny.**

Kup książkę Poleć książkę

- Wywołujesz interfejs API; przesyłasz do niego kod pocztowy, a on zwraca resztę adresu. Zweryfikuj, czy adres ma właściwy format i jest taki, jakiego oczekujesz. Jeśli zawiera 1000 lub więcej znaków, jest prawdopodobnie niepoprawny; przy wywoływaniu każdego interfejsu API powinno wystarczyć ich 500. Jeśli adres składa się z samych cyfr, jest prawdopodobnie niepoprawny. Jeśli zawiera znaki często spotykane w kodzie ([, {, < itp.), też jest zapewne niewłaściwy. Przyjmuj dane wejściowe do programu tylko wtedy, gdy przejdą weryfikację. Wreszcie dane przepływające między aplikacją a interfejsem API powinny być szyfrowane, by chronić prywatność użytkownika. Szyfrowanie można wymusić w aplikacji albo skorzystać w tym celu z siatki usług (ang. *service mesh*) lub dowolnego innego niezawodnego mechanizmu.
- Jakaś aplikacja wywołuje Twoją aplikację i przekazuje w parametrach adres URL. Z punktu widzenia bezpieczeństwa jest to groźne działanie, określane na ogół jako *otwarte przekierowanie* (ang. *open redirect*). Jeśli to tylko możliwe, Twoja aplikacja powinna przyjmować informacje z innych aplikacji bezpiecznymi kanałami (TLS) i należy wtedy sprawdzać poprawność odbieranych danych. Najlepiej poszukaj innego sposobu przekazywania takich wartości, bo złośliwy podmiot może zmienić adres URL i potencjalnie skierować użytkowników na niebezpieczną stronę. Lecz jeśli to jedyna dostępna opcja, przejdź do rozdziału 4., „Bezpieczny kod", w którym wyjaśniono, jak sobie z tym poradzić.
- Jeśli piszesz aplikację w języku, który nie jest bezpieczny pamięciowo (np. C/C++), musisz zadbać o tzw. *sprawdzanie zakresu* (ang. *bounds checking*)[3], by nie doszło do przepełnienia typów zmiennych przez dane wejściowe. W językach C/C++ możliwe jest wprowadzenie liczby przekraczającej maksymalną wielkość typu całkowitego i „przekręcenia" (ang. *rollover*)[4] jej znaku na ujemny, co oczywiście sprawia problemy. Możliwe jest też przepełnienie ciągów, które prowadzi do dobrze znanej podatności, zwanej *przepełnieniem bufora* (ang. *buffer overflow*)[5], pozwalającej atakującemu na nadpisanie części pamięci. W najlepszym razie w takiej sytuacji aplikacja ulegnie awarii, co wyzwoli alert, a zespół reagujący na incydenty powstrzyma atakującego. W najgorszym przypadku atakujący wykorzysta informacje o awarii do tego, by udoskonalać atak, aż uda mu się przejąć serwer WWW i zinfiltrować sieć. Nie sposób przecenić zagrożeń związanych z podatnością na przepełnienie; traktuj tę kategorię zagrożeń poważnie.

Istotne jest, by w aplikacji najpierw weryfikować dane wejściowe, a dopiero potem ich używać. Weryfikowanie danych *po* ich użyciu jest bezcelowe. Sprawdzanie poprawności danych wejściowych musi być pierwszą rzeczą wykonywaną po odebraniu ich w aplikacji.

Uwaga **Jeśli przy emitowaniu na ekran komunikatu o błędzie informującego o odrzuceniu danych wprowadzonych przez użytkownika zdecydujesz się pokazać te dane, miej świadomość, że mogą być szkodliwe i potencjalnie spowodować wadliwe działanie programu. *Zawsze* stosuj na wyjściu kodowanie HTML-a (funkcja taka jest dostępna na wszystkich nowoczesnych platformach programistycznych), o ile obowiązuje kontekst kodu HTML-a.**

Kup książkę Poleć książkę

Wskazówka **Typ kodowania danych wyjściowych zależy od kontekstu danych wyświetlanych na ekranie. Jeśli np. korzystasz z tekstów zawartych w ciągach JavaScriptu, musisz dodać sekwencje ucieczki do znaków Unicode; jeśli jednak osadzasz dane wprowadzone przez użytkownika w metodzie obsługi zdarzenia, koduj dane wyjściowe na dwóch poziomach (najpierw JavaScript, a potem HTML). W miarę możliwości unikaj tego typu sytuacji, a jeśli się nie da, zapoznaj się z dokumentem OWASP XSS Prevention Cheat Sheet (ściągawka dotycząca zapobiegania atakom XSS)[6].**

Uwaga **Jeśli piszesz lub przepisujesz od podstaw aplikację niskopoziomową, zawsze wybieraj język Rust zamiast C czy C++. Rust jest nowoczesnym językiem programowania, pozwalającym wykonywać zadania niskopoziomowe równie dobrze jak C i C++, ale w odróżnieniu od nich *Rust jest bezpieczny pamięciowo*. Oznacza to, że nie trzeba sprawdzać zakresów, bo nie jest możliwe przepełnienie zmiennych prowadzące do potencjalnych luk w zabezpieczeniach. Bezpieczeństwo pamięciowe to nie żart; przedstawiciele organizacji Mozilla odpowiadającej za przeglądarkę Firefox szacują, że 73% luk w komponencie stylów nigdy by się nie pojawiło, gdyby został on napisany w języku Rust, a nie C/C++[7]. Ta jedna decyzja projektowa może spowodować tak radykalne zmniejszenie powierzchni ataku, że pisanie nowych aplikacji w języku C, gdy dostępny jest Rust, nie ma racjonalnego uzasadnienia biznesowego. Wymówka, że „przecież wszyscy już wiemy, jak programować w C", nie może być wystarczającym powodem, by nie uczyć się i nie używać języka programowania Rust.**

Kodowanie i stosowanie znaków ucieczki

Najbardziej znana luka w zabezpieczeniach aplikacji sieciowych to wykonywanie *skryptów międzywitrynowych* (ang. *cross-site scripting* — XSS); w czasie pisania tej książki szacowano, że występuje w ponad $^2/_3$ aplikacji sieciowych[8]. Jest wiele sposobów na ograniczenie tego ryzyka: stosowanie nagłówka Content Security Policy (CSP), weryfikacja wejściowa (ang. *input validation*) i kodowanie danych wyjściowych (ang. *output encoding*, *niniejsze* wymaganie). Ta ostatnia operacja chroni tylko przed atakiem XSS, ale ponieważ zagrożenie nim jest tak powszechne i tak łatwo popełnić ten błąd, zdecydowanie warto dodać *wszystkie trzy* zabezpieczenia do *wszystkich* swoich aplikacji.

Wskazówka **Dodanie do aplikacji wszystkich trzech zabezpieczeń przed atakami XSS jest doskonałym przykładem zastosowania w niej zasady „dogłębnej ochrony".**

Dodawanie znaków ucieczki do znaków lub wartości pozwala wyeliminować ich szczególne właściwości, które mogłyby mieć, gdyby zostały potraktowane jako kod, a nie jako dane (czyli nie tak, jak powinny). Operacja ta polega zwykle na dodaniu przed każdym znakiem specjalnym odwrotnego ukośnika (\).

Kodowanie to zmiana istniejącego formatu wartości, nieważne jakich, na format docelowy (adres URL, Base64, HTML). Operację kodowania można z łatwością odwrócić i nie należy jej mylić z szyfrowaniem ani stosować zamiast niego. Kodowanie nie ma na celu ochrony danej wartości, ale zmianę

Kup książkę Poleć książkę

jej formatu na ten, w którym ma być używana. Jeśli np. coś ma być wyświetlone na ekranie, należy *zakodować dane wyjściowe* (ang. *output encode*) przy użyciu odpowiedniej funkcji dostępnej na używanej platformie programistycznej (ma ją każdy taki nowoczesny szkielet). Kodowanie danych wyjściowych gwarantuje, że jeśli wartość wyprowadzana na ekran zawiera kod (np. skrypt międzywitrynowy), to zostanie wyświetlona tylko wartość tekstowa i nie dojdzie do zinterpretowania (wykonania) tego kodu w przeglądarce jako skryptu JavaScriptu. Kodowanie danych wyjściowych sprawia, że atak XSS jest niegroźny.

Zalecenie będzie zapewne dla wszystkich oczywiste; należy kodować wszystkie dane wyjściowe (a w razie potrzeby dodawać do nich znaki specjalne).

Uwaga **Błąd XSS jest specjalnym typem podatności na wstrzyknięcie kodu, z tego względu, że gdy atak ten się powiedzie, kod (JavaScript) jest wykonywany po stronie klienta (w przeglądarce), podczas gdy w większości tego typu luk kod jest uruchamiany po stronie serwera (na poziomie interpretera, systemu operacyjnego itd.). Ochrona przed atakiem XSS polega na połączeniu weryfikacji wejściowej i kodowania lub dodawania znaków ucieczki do danych wyjściowych. Inne podatności na wstrzyknięcie kodu spowodowane są natomiast przede wszystkim niewłaściwą weryfikacją wejściową i nieprawidłowymi ustawieniami konfiguracyjnymi serwera. Błędy XSS występują też dużo częściej niż wszystkie inne poważne typy luk w zabezpieczeniach stron WWW. Dlatego pomimo faktu, że błędy XSS stanowią odmianę podatności na wstrzyknięcie kodu, zawsze traktuje się je jako odrębną klasę luk w zabezpieczeniach.**

Komponenty innych podmiotów

Jak wcześniej stwierdzono, każdy wiersz kodu dołączany z biblioteki, platformy programistycznej, wtyczki czy komponentu stworzonego przez inne podmioty stanowi czynnik ryzyka, które programista akceptuje w swojej aplikacji; jeśli jest niezabezpieczony, tak samo będzie z aplikacją, jeśli uzyskuje dostęp do tego składnika, używa go bądź wywołuje. Nawet gdy w aplikacji nie jest wykorzystywany niebezpieczny wiersz kodu z komponentu, to czasem, zależnie od sytuacji, i tak może być ona podatna na atak. Dzięki weryfikowaniu, czy wszystkie komponenty innych podmiotów nie mają znanych luk w zabezpieczeniach, można szybko i łatwo zdobyć wiedzę na temat tego, jak bezpieczna jest aplikacja. Naprawa znalezionych problemów może nie być szybka ani prosta, ale weryfikacja taka bardzo pomaga w zapewnieniu bezpieczeństwa aplikacji, dlatego zawsze zaleca się, by była ujęta w specyfikacji projektu i w wymaganiach związanych z jego utrzymywaniem.

Uwaga **Jeśli usłyszysz w podcaście albo przeczytasz w wiadomościach, że ktoś upublicznił błąd typu „zero-day", oznacza to, że jakaś osoba upubliczniła szczegóły luki w zabezpieczeniach, dla której nie ma znanej poprawki. Robi się to zwykle po to, żeby wytworzyć presję społeczną i wymusić na firmie wydanie poprawki, albo w celu popisania się umiejętnościami badacza bezpieczeństwa. Moja opinia i rada jest taka, że każdy, kto ma informacje o luce w zabezpieczeniach, powinien ją zawsze przed upublicznieniem zgłaszać twórcy oprogramowania — nie po to, by chronić producenta oprogramowania, ale jego użytkowników; nie są oni niczemu winni, a stawianie ich w sytuacji zagrożenia jest całkowicie sprzeczne z celami i zadaniami branży bezpieczeństwa informacji. </koniec tyrady>**

Kup książkę Poleć książkę

ZNANE LUKI

Co oznacza pojęcie „znanej luki”? Oprogramowanie tworzone na zlecenie jest z definicji unikatowe. Każdy fragment niestandardowego oprogramowania jest jak płatek śniegu, dlatego każda nowa luka, którą ktoś w nim znajdzie, jest nieznana ogółowi. Aplikacje, które nie są tworzone na zlecenie, często po prostu są określane mianem oprogramowania. Przykładami może być system operacyjny albo „oprogramowanie z półki” (COTS). [Skrót ten jest rozwijany po angielsku jako *Commercial/Customizable/Configurable Off The Shelf* (możliwe do kupienia/skonfigurowania/przystosowania prosto z półki). Takie oprogramowanie każdy może kupić, jest instalowane, ale ma też spore możliwości konfigurowania. Przykładami są SharePoint, WordPress, Microsoft Office lub Adobe Illustrator]. Jeśli tester penetracyjny, badacz bezpieczeństwa lub inny „haker” znajdzie lukę w oprogramowaniu (niezależnie od tego, czy jest standardowe, czy nie), powinien zgłosić to podmiotowi, który to oprogramowanie przygotował (nazywa się to skoordynowanym ujawnianiem informacji, ang. *coordinated disclosure*), a jeśli istnieje program nagród za błędy, może przesłać to w jego ramach i potencjalnie otrzymać wynagrodzenie. Jeśli oprogramowanie jest standardowe, to po zgłoszeniu błędu i wydaniu poprawki (z punktu widzenia użytkownika jest to zwykle aktualizacja oprogramowania) informacje o usterce są publikowane do publicznego wglądu w kilku miejscach w internecie (m.in. w bazie danych CVE organizacji Mitre[9]) i w ten sposób podatność uznaje się za „znaną”. Oprogramowanie takie jest odtąd uważane za wersję „ze znaną luką” i będzie zazwyczaj wychwytywane przez zautomatyzowane narzędzia, które skanują kod pod tym kątem.

Po znalezieniu błędu w oprogramowaniu, systemie operacyjnym bądź produkcie typu COTS, ale przed wydaniem poprawki, usterka jest nazywana luką „dnia zerowego” (ang. *zero day*, *0 day*), co daje do zrozumienia, że nie ma poprawki tego problemu. Zespoły ds. bezpieczeństwa często odwołują się do poprawek w kategoriach liczby dni od ich wydania, np. „ta luka ma już ponad 90 dni, a my wciąż nie zastosowaliśmy poprawki!”, dlatego wybrano termin „zero”.

Na rynku dostępnych jest bardzo wiele narzędzi do weryfikowania, czy stosowane komponenty innych podmiotów mają znane luki. Ta książka nie zawiera rekomendacji konkretnych narzędzi ani ich dostawców, a jedynie omawia ich strategiczne zastosowania. Pierwsza strategia polega na tym, że w miarę możliwości należy używać dwóch narzędzi, a nie jednego. Sprawdzają one w odmienny sposób różne bazy danych i przez to mogą nie wyłapać tych samych rzeczy. Druga strategia polega na tym, by skanować repozytorium kodu co pewien czas (codziennie, a przynajmniej co tydzień) oraz przed każdym produkcyjnym wydaniem kodu. Repozytorium należy skanować dlatego, że niektóre aplikacje mogą być rzadko wydawane, a ciągle znajdowane są nowe podatności w kodzie innych podmiotów.

Warto zauważyć, że w kroku tym ma miejsce skanowanie pod kątem *znanych* luk w kodzie innych podmiotów. Prawdopodobnie istnieje więcej podatności, które pozostają nieznane. Sprawdzanie kodu pod kątem znanych luk to absolutne minimum przy tworzeniu bezpiecznego oprogramowania. Jeśli budowane oprogramowanie wymaga wysokiego stopnia bezpieczeństwa, należy przetestować i przejrzeć każdy wiersz kodu, od którego zależy aplikacja. Najlepszym rozwiązaniem, jeśli

Kup książkę Poleć książkę

chodzi o komponenty innych podmiotów, jest dołączanie tylko tych naprawdę potrzebnych, a nie wszystkich nowych i atrakcyjnych rzeczy z internetu. Chociaż programistów trudno do tego przekonać (przyznajmy to otwarcie: nowe rozwiązania techniczne są ekscytujące i sprawiają frajdę), to jeśli postarasz się im wytłumaczyć związane z nimi ryzyko i zamiast zakazywać, zaproponujesz im alternatywę lub inne rozwiązanie, z większym prawdopodobieństwem odniesiesz pożądany skutek.

OSTRZEŻENIE DOTYCZĄCE PRYWATNOŚCI

Alicja chciała dołączyć swój internetowy zaszyfrowany kalendarz osobisty do oprogramowania kalendarzowego na swoim komputerze, aby mieć oba kalendarze w tym samym miejscu. Zapoznała się ze stroną pomocy komputerowego oprogramowania kalendarzowego, z której wynikało, że powinna zmienić ustawienia współdzielenia (*Share*) swojego prywatnego kalendarza na „publiczne" (*Public*). Alicja była zszokowana! Szyfrowała swój kalendarz, bo chciała, by jej informacje były *prywatne*. Wniosła skargę w sprawie strony pomocy, argumentując, że powinna ona ostrzegać użytkowników przed potencjalnymi problemami związanymi z prywatnością, które mogą wystąpić, jeśli zmienią w ten sposób konfigurację. Alicji zainstalowano na komputerze nowe oprogramowanie kalendarza.

Nagłówki bezpieczeństwa — pasy bezpieczeństwa dla aplikacji sieciowych

Nagłówki bezpieczeństwa to ustawienia, które informują przeglądarkę i serwer, jak obsługiwać różne sprawy związane z aplikacją sieciową; mają one zastosowanie tylko do zasobów sieciowych, z których korzysta się za pomocą przeglądarki (aplikacji internetowych i oprogramowania udostępnianego w formie usługi, ang. *software as a service* — SaaS, czyli programów dostępnych poprzez przeglądarkę). Nagłówków bezpieczeństwa nie stosuje się w oprogramowaniu instalowanym na komputerze, systemach operacyjnych ani systemach wbudowanych, np. w oprogramowaniu układowym (ang. *firmware*). Nagłówki te bardzo przypominają pasy bezpieczeństwa; nie są atrakcyjne ani ekscytujące, trudne ani czasochłonne w użyciu, a jeśli wyrobisz w sobie nawyk ich używania, mogą Cię ochronić w nagłych sytuacjach (wypadku samochodowego albo ataku na aplikację sieciową). Ustawia się je zwykle albo na serwerze WWW, albo we własnym kodzie. Aby je zastosować, wystarczy na ogół dodać jeden wiersz kodu albo zaznaczyć jedno pole wyboru w konfiguracji serwera WWW, co nie jest trudne. Wierzę, że temu podołasz.

Omówię teraz przeznaczenie każdego z nagłówków, powody ich stosowania albo niestosowania oraz sugerowane ustawienia. Jeśli uznasz te ustawienia za przydatne, możesz je skopiować bezpośrednio do swojej aplikacji, koniecznie jednak przetestuj konfigurację przed przeniesieniem do środowiska produkcyjnego.

Wskazówka **Aby dowiedzieć się więcej o nagłówkach bezpieczeństwa, odwiedź stronę *OWASP.org* albo *SecurityHeaders.com*!**

Kup książkę Poleć książkę

Nagłówki bezpieczeństwa w działaniu

W ramach projektu DevSlop organizacji OWASP moja przyjaciółka Franziska Bühler i ja przygotowałyśmy kilka nagrań wideo i postów blogowych na temat dodawania nagłówków bezpieczeństwa do witryny internetowej. Oto przykład kodu dla platformy ASP.NET:

```
<! Początek nagłówków bezpieczeństwa platformy ASP.NET ->
<httpProtocol>
<customHeaders>
<add name="X-XSS-Protection" value="1; mode=block"/>
<add name="Content-Security-Policy" value="default-src 'self'"/>
<add name="X-Frame-Options" value="SAMEORIGIN"/>
<add name="X-Content-Type-Options" value="nosniff"/>
<add name="Referrer-Policy" value="strict-origin-when-cross-origin"/>
<remove name="X-Powered-By"/>
</customHeaders>
</httpProtocol>
<! Koniec nagłówków bezpieczeństwa ->
```

X-XSS-Protection

Nagłówek ten uznaje się za przestarzały. Nie tylko nie jest obsługiwany przez nowoczesne przeglądarki, ale niektórzy eksperci z branży rekomendują wręcz, by wcale z niego nie korzystać, bo może powodować luki w zabezpieczeniach[10]. Choć przydaje się w przypadku bardzo przestarzałych przeglądarek, to jednak przynosi więcej szkody niż pożytku, więc już w czasie pisania tej książki odradzano jego użycie.

Content-Security-Policy (CSP)

Pierwszą rzeczą, którą robi złośliwy podmiot, gdy uświadomi sobie, że witryna jest podatna na atak XSS, jest odwołanie się do umieszczonego gdzieś w internecie własnego skryptu, który zazwyczaj jest znacznie dłuższy, niż pozwala na to zagrożona aplikacja, ale może być już przygotowany.

W nagłówku `Content-Security-Policy` wyszczególnia się listę wszystkich użytych na stronie źródeł treści (skryptów, obrazów, ramek, czcionek itd.), które znajdują się poza domeną. Nie pozwala to mającej lukę aplikacji odwołać się do drugiej części ataku i uruchomić go, co radykalnie zmniejsza ryzyko i potencjalne szkody. Jednak programistom kontrolowanie wszystkich używanych zasobów wydaje się czasochłonne, dlatego ten nagłówek bezpieczeństwa nie cieszy się wśród nich popularnością. Moim zdaniem używaliby go dużo częściej, gdyby byli świadomi zagrożeń i ochrony, jaką przed nimi zapewnia. Jeśli masz kłopot z przekonaniem kogoś do stosowania tego nagłówka, pomyśl o pożyczeniu mu tej książki; prawdopodobnie będzie Ci potem dziękował.

Uwaga ***Nigdy*** **nie włączaj nagłówka CSP bez zgody i asysty programisty. W ogóle nie należy uaktywniać funkcji bezpieczeństwa bez koordynacji z zespołami, których one dotyczą, ale zwłaszcza tego nagłówka. Włączenie go prawie na pewno popsuje wygląd i działanie witryny oraz niektóre z jej funkcji, a co gorsza, naruszy zaufanie zespołu programistów. Nie spiesz się z tym; przetestuj go dokładnie przed pierwszym wdrożeniem.**

Kup książkę Poleć książkę

Absolutnie najprostszym ustawieniem jest po prostu zablokowanie wszystkiego, co może być dobrym wyborem w przypadku statycznych i (lub) nudnych witryn (takich, które nie odwołują się do zewnętrznych treści). Oto taka konfiguracja:

```
Content-Security-Policy: default-src 'self'; block-all-mixed-content;
```

Postawmy jednak sprawę uczciwie: współcześnie niewiele witryn jest tak prostych. W porządku, jasne. Oto lista różnych źródeł, które można zdefiniować w ramach własnych zasad (dzięki uprzejmości organizacji OWASP — Open Web Application Security Project):

- **`default-src`** — jak można się spodziewać, jest to ustawienie domyślne, „obejmujące wszystko". Ma ono zastosowanie przy próbie załadowania czegoś, co nie jest jasno zdefiniowane w pozostałych zasadach. Często nadaje się mu wartość `self`, co oznacza, że jeśli nie dopuścimy czegoś jawnie w innych zasadach, odpowiedzią będzie „nie ładuj tego". Jeśli nie masz pewności, zawsze stosuj wartość `self`.

Uwaga **Wyjątkiem od reguły są zasady Frame Ancestors i Form Action; ustawienie `default-src` ich nie dotyczy.**

- **`script-src`** — lista domen (w których umieszczone są skrypty) albo dokładny adres URL skryptu, który może być uruchomiony w ramach witryny. Żaden skrypt z żadnego miejsca w internecie, oprócz tych znajdujących się w domenie (albo podanych tutaj), nie zostanie wykonany. Chroni to przed atakami XSS.

Ostrzeżenie **W celu pominięcia wszystkich blokad, które właśnie poczyniliśmy w zasadach Content Security Policy, można dodać do konfiguracji słowo kluczowe `unsafe-inline`; pozwala ono na uruchamianie każdego skryptu z dowolnego miejsca. To słowo kluczowe nie powinno pozostawać w aplikacjach na stałe; powinno być tymczasowe, na potrzeby testów. Należy je stosować w trakcie wypracowywania dojrzałej i kompletnej implementacji zasad CSP, ale nie powinno być rozwiązaniem docelowym ani pozostać na stałe. Należy dokonywać inspekcji tego ustawienia podczas oceny bezpieczeństwa w środowisku produkcyjnym.**

W każdym z wymienionych poniżej elementów stosuje się podobny wzorzec: jako część aplikacji sieciowej mogą być używane lub ładowane wyszczególnione zasoby danego typu (plus te, które znajdują się wewnątrz domeny). Jeśli stosowany jest nagłówek CSP, każdy inny niepodany tu typ zasobów zostanie zablokowany:

- ✓ `object-src` — wtyczki (właściwe źródła dla elementów `<object>`, `<embed>` i `<applet>`);
- ✓ `style-src` — style kaskadowe (ang. *Cascading Style Sheets* — CSS);
- ✓ `img-src` — obrazy;
- ✓ `media-src` — wideo i audio;
- ✓ `frame-src` — ramki;
- ✓ `font-src` — czcionki;
- ✓ `plugin-types` — ogranicza typy wtyczek, które można uruchomić.

Kup książkę Poleć książkę

- **`script-nonce`** — ta zasada jest skomplikowana, ale warta uwagi. Identyfikator jednorazowy (ang. *nonce*) to ciąg znaków tworzony i używany tylko jednokrotnie w celu potwierdzenia, że konkretny skrypt jest tym, który miał być wykonany. Jest to dodatkowy poziom zabezpieczeń w ramach nagłówka CSP. Korzystanie z tego ustawienia oznacza, że do uruchomienia skryptu konieczne jest podanie identyfikatora jednorazowego.

Wskazówka **Identyfikatory jednorazowe to skomplikowany temat, a implementacja tej funkcji nagłówka CSP z biegiem czasu się zmieniła. Mając to na uwadze, odsyłam do poświęconej temu zagadnieniu ściągawki OWASP Cheat Sheet w celu zapoznania się z nowymi informacjami i ze znacznie obszerniejszym omówieniem: *cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html.***

- **`report-uri`** — w ramach nagłówka CSP mogą być tworzone raporty informujące o tym, co zostało zablokowane, i o innych przydatnych rzeczach. Identyfikator URI informuje, gdzie ma być wysłany raport. W czasie pisania tej książki funkcje raportowania, które, mówiąc szczerze, są naprawdę wspaniałe, zapewniały tylko cztery nagłówki bezpieczeństwa. Posiadanie wskaźników (ang. *metrics*) i informacji o typach ataków kierowanych przeciwko witrynie to dobrodziejstwo.

Ostrzeżenie **Adres URL z raportami jest publicznie dostępny, co oznacza, że atakujący może je przejrzeć i w celu ukrycia swoich poczynań przeprowadzić jednocześnie atak typu „odmowa usługi" (ang. *denial-of-service* — DOS lub DDOS).**

Nagłówek CSP jest zdecydowanie najbardziej złożony ze wszystkich nagłówków bezpieczeństwa. Aby uzyskać więcej informacji na jego temat, odwiedź strony *csp-evaluator.withgoogle.com* (firmy Google) i *scotthelme.co.uk*.

Na przykład nagłówek:

```
Content-Security-Policy: default-src 'self'; img-src https://*.wehackpurple.com; media-src
https://*.wehackpurple.com;
```

pozwala przeglądarce ładować obrazy, klipy wideo i audio z adresów `*.wehackpurple.com`, a nagłówek:

```
Content-Security-Policy: default-src 'self'; style-src https://*.jquery.com; script-src
https://*.google.com;
```

umożliwia ładowanie stylów z domeny `jquery.com` i skryptów z `google.com`.

Wskazówka **Nagłówki bezpieczeństwa udostępniające raporty to: CSP, Expect-CT, Public-Key-Pins i X-XSS-Protection. Są bardzo przydatne!**

Uwaga **Akronimy DOS i DDOS pochodzą od angielskiego terminu *denial of service* (odmowa usługi), a dodatkowe D oznacza *distributed* (rozproszona). Celem ataku DOS jest obciążenie zasobów ofiary tak, by nikt nie mógł ich używać. Może to spowodować awarię witryny, wstrzymanie sprzedaży w sklepie internetowym albo inne formy zablokowania**

Kup książkę Poleć książkę

dostępu do zasobu w internecie. Kiedy pojawiły się ataki DOS, pochodziły one często tylko z jednego źródła, co oznaczało, że ten adres IP można było zablokować i udaremnić atak. Z czasem atakujący nauczyli się, że atak prowadzony z wielu (setek lub nawet tysięcy) różnych adresów IP powoduje znacznie większe szkody i trudno się przed nim obronić. Większość ataków DOS w czasie pisania tej książki miała charakter rozproszony (DDOS), często z wykorzystaniem urządzeń IoT o złamanych zabezpieczeniach.

X-Frame-Options

Ten nagłówek pozwala na ochronę przeciw atakom przechwytywania kliknięć (ang. *clickjacking*)[11], polegającym na tym, że szkodliwa strona internetowa umieszcza w ramce prawowitą witrynę i jest w stanie wykradać informacje lub „kliknięcia". Chociaż mogą zachodzić specyficzne sytuacje, w których właściciel witryny chce, by była ona umieszczana w ramce przez jedną lub kilka konkretnych witryn, zezwalanie na to każdej stronie jest prawdopodobnie złym pomysłem, bo może prowadzić do sytuacji, gdy użytkownik myśli, że znajduje się na znanej mu witrynie, podczas gdy faktycznie jest w interakcji z niewidoczną stroną o najprawdopodobniej szkodliwym charakterze (nazywa się to przechwytywaniem kliknięć). Może to prowadzić do przechwytywania naciśniętych klawiszy (ang. *keylogging*) i wykradania poświadczeń użytkownika. Aby tego typu podatność mogła być wykorzystana, użytkownik musi współuczestniczyć przez kliknięcie łącza w wiadomości wyłudzającej informacje albo łącza do szkodliwej witryny, co oznacza, że atak ten nie zdarza się tak często jak inne, ale jeśli już się przydarzy, może wyrządzić duże szkody.

Ostrzeżenie **Nagłówek `X-Frame-Options` jest przestarzały, a w nowoczesnych przeglądarkach korzysta się zamiast niego z nagłówka `Content-Security-Policy` (CSP). Ten pierwszy jest stosowany w celu zapewnienia wstecznej zgodności ze starszymi przeglądarkami i, miejmy nadzieję, będzie powoli wycofywany z aktywnego użycia.**

Aby strona mogła znajdować się w ramkach pochodzących z tej samej domeny, nagłówkowi `X-Frame-Options` należy nadać wartość `sameorigin`:

```
X-Frame-Options: SAMEORIGIN
```

Aby ramkowanie było zupełnie niemożliwe, nagłówek `X-Frame-Options` powinien mieć wartość `deny`:

```
X-Frame-Options: DENY
```

X-Content-Type-Options

Piękno pisania oprogramowania polega m.in. na kreatywności i stosowaniu licencji poetyckiej w używaniu języka programowania; na znajdowaniu nowych i twórczych sposobów korzystania z języka i platformy programistycznej. Czasem prowadzi to jednak do wieloznaczności, a to z kolei może spowodować luki w zabezpieczeniach, jeśli nie wiadomo, jaka następna instrukcja aplikacji zostanie wykonana. Nazywamy to przejściem do nieprzewidzianego stanu i nie powinniśmy *nigdy* do tego dopuszczać. No chyba, że zajmujesz się testowaniem penetracyjnym albo badaniem bezpieczeństwa, wtedy owszem: w tym przypadku będzie to Twoje ulubione miejsce, w którym z pewnością znajdziesz luki w oprogramowaniu.

Kup książkę Poleć książkę

Ten nagłówek bezpieczeństwa instruuje przeglądarkę, by nie rozpoznawała (ang. *sniff* — wnioskowała/zgadywała) samodzielnie, jaki typ mediów został użyty w aplikacji sieciowej, lecz polegała wyłącznie na określeniu typu przez aplikację. Autorzy przeglądarek lubią myśleć, że mogą one przewidzieć serwowany im typ zawartości. Niestety przerodziło się to w znaną lukę w zabezpieczeniach, którą można wykorzystać na szkodę witryny. Ten nagłówek bezpieczeństwa ma tylko jedno możliwe ustawienie:

```
X-Content-Type-Options: nosniff
```

Referrer-Policy

Podczas surfowania po witrynach internetu każda z nich przesyła do następnej wartość o nazwie *referrer* (odsyłacz), która zawiera łącze do strony odwiedzonej wcześniej. Jest ona niezwykle przydatna dla osób analizujących ruch na swoich witrynach, bo dowiadują się, skąd przychodzą do nich internauci. Jeśli jednak ktoś odwiedza stronę o poufnym charakterze (np. aplikację do kredytów hipotecznych albo witrynę omawiającą konkretne schorzenie), może nie chcieć, by te szczegóły zostały wysłane do następnej odwiedzanej strony. W celu ochrony prywatności użytkowników twórca strony może ustalić, że w wartości odsyłacza będzie przekazywana tylko domena, a nie konkretna odwiedzana strona (`wehackpurple.com` zamiast `wehackpurple.com/embarrassing-blog-post-title`), albo że nic nie będzie w ogóle przekazywane. Wartość odsyłacza można też zmieniać na podstawie tego, czy dochodzi do „obniżenia kategorii" (ang. *downgrading*) połączenia z HTTPS na HTTP.

Aby przekazywać tylko informacje na temat protokołu i domeny, odsyłaczowi należy nadać wartość `origin`. Ustawienie to nie może być zmienione w żadnym innym kontekście.

```
Referrer-Policy: origin
```

Na przykład z dokumentu `https://wehackpurple.com/page.html` zostanie wysłany odsyłacz `https://wehackpurple.com/`.

To ustawienie spowoduje, że przy opuszczaniu domeny przez użytkownika do kolejnej strony przesłany zostanie tylko protokół i domena. Wewnątrz domeny przekazywana będzie cała ścieżka. Do obsługi większości sytuacji w witrynach niemających poufnego charakteru nadaje się następujące ustawienie:

```
Referrer-Policy: strict-origin-when-cross-origin
```

Brak wartości w polu odsyłacza niezależnie od kontekstu:

```
Referrer-Policy: no-referrer
```

Jeśli zajdzie taka potrzeba, to ustawienie może przybrać dość skomplikowaną formę, ale generalnie w większości sytuacji biznesowych wysyłanie tylko samej domeny źródłowej w przypadku jej opuszczania jest wystarczające i dostatecznie chroni prywatność użytkowników. Jeśli masz wątpliwości, zawsze możesz niczego nie wysyłać, co zapewni użytkownikom respektowanie ich prywatności.

Aby zdobyć więcej informacji, warto odwiedzić stronę wiodącej prym w tym zakresie organizacji Mozilla z doskonałymi poradami i wskazówkami technicznymi: *developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referrer-Policy*.

Dodatkowy zasób: doskonałym źródłem wiedzy na ten temat jest strona badacza bezpieczeństwa Scotta Helme'a, który udostępnia wiele informacji i narzędzi dotyczących nagłówków bezpieczeństwa: *scotthelme.co.uk*.

Kup książkę Poleć książkę

Strict-Transport-Security (HSTS)

Ten nagłówek bezpieczeństwa wymusza połączenie HTTPS (szyfrowane) nawet wtedy, gdy użytkownik próbuje połączyć się z witryną poprzez protokół HTTP. Oznacza, to że dane przesyłane tam i z powrotem będą szyfrowane w sposób wymuszony. Użytkownicy, a także atakujący nie będą mogli obniżyć kategorii połączenia na HTTP (nieszyfrowane); przeglądarka przed załadowaniem danych zostanie zmuszona do przejścia na protokół HTTPS.

DEFINICJE

Platforma jako usługa (ang. *platform as a service* — **PaaS)** — usługa przetwarzania w chmurze, polegająca na utrzymywaniu oprogramowania (zazwyczaj aplikacji sieciowych) w chmurze przez jej dostawcę. W usłudze PaaS nie trzeba instalować poprawek ani aktualizacji; te zadania wykonuje dostawca chmury.

Urząd certyfikacji (ang. *Certificate Authority* — **CA)** — ciesząca się zaufaniem firma lub organizacja, która weryfikuje tożsamość każdego, kto kupuje w niej certyfikat.

Electronic Frontier Foundation (EFF) — międzynarodowa organizacja typu non profit, pracująca nad ochroną prywatności i innych praw użytkowników internetu. EFF odpowiada za narzędzie certbot (*https://certbot.eff.org/*) wykorzystujące Let's Encrypt.

Internet Security Research Group (ISRG) — kalifornijska organizacja pożytku publicznego prowadząca urząd certyfikacji Let's Encrypt.

Let's Encrypt — udostępniający certyfikaty szyfrowania urząd certyfikacji prowadzony przez Internet Security Research Group (ISRG). W czasie pisania tej książki jako jedyny oferował publicznie darmowe certyfikaty dla użytkowników indywidualnych.

Certyfikat wieloznaczny (ang. *wild card certificate*) — obejmuje nie tylko domenę główną, ale też wszystkie jej poddomeny. Warto się o niego starać w razie ich posiadania. Poddomeny to wszystko to, co obejmuje symbol wieloznaczny * w formule `*.twojadomena.cokolwiek`. Na przykład zapis `*.wehackpurple.com` uwzględnia strony *newsletter.wehackpurple.com*, *store.wehackpurple.com* i *www.wehackpurple.com*.

Do poprawnego działania tego nagłówka bezpieczeństwa konieczne jest posiadanie certyfikatu wydanego przez urząd certyfikacji (ang. *certificate authority* — CA), zainstalowanego na serwerze WWW, w usłudze PaaS, w kontenerze albo w innym miejscu, w którym znajduje się aplikacja. Certyfikat ten używany jest w procesie szyfrowania i nie można bez niego włączyć nagłówka HSTS. Jeśli masz poddomeny, spróbuj się postarać o certyfikat wieloznaczny. Warto, aby certyfikat był ważny jak najdłużej (rok, a nie trzy miesiące), po to by nie tracić czasu na częstą wymianę certyfikatów.

Następnie trzeba obliczyć czas ważności certyfikatu w sekundach. Nie, nie żartuję: zdecydowano, że czas ma być mierzony w *sekundach*. Podpowiedź: jeden rok to 31 536 000 sekund:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

Wskazówka **Możesz zgłosić swoją domenę na stronie *hstspreload.org* i dodać do deklaracji nagłówka sufiks `preload`. Firma Google doda Twoją witrynę do listy wstępnego ładowania, zapewniając w ten sposób, że nikt nie będzie mógł nigdy połączyć się z Twoją stroną**

Kup książkę Poleć książkę

bez szyfrowania. Chociaż producenci głównych przeglądarek poinformowali, że zamierzają zaadaptować tę funkcję, nie jest ona częścią oficjalnej specyfikacji nagłówka HSTS[12].

Zmodyfikowana składnia poprzedniego przykładu będzie następująca: `Strict-Transport--Security: max-age=31536000; includeSubDomains; preload`.

Feature-Policy

Podczas pisania tej książki był to najnowszy
* nagłówek bezpieczeństwa obsługiwany przez współczesne przeglądarki. Z nadejściem standardu HTML 5 w bardziej nowoczesnych przeglądarkach pojawiło się wiele świetnych nowych funkcji. Ten nagłówek pozwala na stosowanie w aplikacji sieciowej tych funkcji nowego typu albo zabrania ich użycia.

Opcje ustawień są następujące:

- **`none`** — funkcja niedozwolona w ogóle;
- **`self`** — funkcja dozwolona, ale może być stosowana bądź wywoływana tylko we własnej domenie;
- **`src`** (tylko ramki `iframe`) — dokument załadowany do ramki `iframe` musi pochodzić z tego samego źródła co adres URL jej atrybutu `src`[13];
- **`*`** — funkcja może być używana lub wywoływana przez każdą domenę;
- ***<adres(y)>*** — funkcja jest dozwolona w przypadku konkretnych adresów URL[13].

W poniższym przykładzie dozwolone jest wyłącznie korzystanie z głośników (tylko we własnej domenie) i uruchamianie witryny w trybie pełnoekranowym:

```
Feature-Policy: camera 'none'; microphone 'none'; speaker 'self';
vibrate 'none'; geolocation 'none'; accelerometer 'none';
ambient-light-sensor 'none'; autoplay 'none'; encrypted-media 'none';
gyroscope 'none'; magnetometer 'none'; midi 'none'; payment 'none';
picture-in-picture 'none'; usb 'none'; vr 'none'; fullscreen *
```

Takie ustawienia zostały użyte na stronie projektu OWASP DevSlop. Zabraniamy korzystania z prawie wszystkich funkcji; pozwalamy tylko na użycie głośników wywołane w obrębie naszej własnej strony. Pozwalamy też w każdej domenie na pełnoekranowy widok przeglądarki. Jeśli masz wątpliwości, zastosuj bardziej, a nie mniej restrykcyjne ustawienia. Użytkownicy będą Ci za to wdzięczni.

X-Permitted-Cross-Domain-Policies

Ten nagłówek bezpieczeństwa ma zastosowanie tylko do wchodzących w skład aplikacji sieciowej produktów firmy Adobe (programów Reader i Flash). Wtyczka Adobe Flash jest niezwykle niebezpieczna i nie ma już wsparcia ze strony Adobe, dlatego nie powinna być używana w nowoczesnych witrynach ani aplikacjach.

Zadaniem tego nagłówka jest określanie, czy dostęp do plików z Twojej domeny dla produktów firmy Adobe używanych na innych stronach jest dozwolony, czy też nie. Jeśli zamierzasz pozwalać

* Obecnie zastąpiony przez Permissions-Policy — *przyp. red.*

Kup książkę Poleć książkę

plikom programu Adobe Reader utrzymywanym w miejscach innych niż Twoja domena na dostęp do dokumentów w Twojej witrynie, musisz podać tutaj nazwy tych domen. W przeciwnym razie nadaj mu wartość `none`, przez co osoby używające produktów Adobe w innych niż Twoja domenach nie będą miały dostępu do Twoich dokumentów ani zasobów:

```
X-Permitted-Cross-Domain-Policies: none
```

Expect-CT

CT odnosi się do Certification Transparency, otwartej platformy, która umożliwia nadzór nad urzędami certyfikacji (ang. *Certificate Authorities* — CA). Od czasu do czasu urzędy te przypadkowo wydają certyfikaty witrynom niezbyt wzorowym, a czasem nawet wręcz szkodliwym. Cały system urzędów certyfikacji został zaprojektowany po to, by budować zaufanie i sprawdzać, czy witryna mająca certyfikat jest bezpieczna dla przeglądarek i wiarygodna dla użytkowników. Wydawanie certyfikatów szkodliwym witrynom, nieważne, czy w wyniku błędu, zaniedbania, czy współudziału w tym procederze, jest niedopuszczalne. Certificate Transparency Framework rejestruje szczegóły różnych transakcji, co pozwala na sprawdzenie, kiedy urząd certyfikacji wydał niewłaściwie certyfikat.

Jeśli urząd certyfikujący podejmie kilka „złych decyzji", producent przeglądarki albo organizacja może zdecydować, że nie będzie już „ufać" wydawanym przez ten urząd certyfikatom.

Być może zastanawiasz się, co programista, specjalista ds. wsparcia aplikacji albo bezpieczeństwa może mieć z tym wspólnego. W celu zachowania integralności całego systemu urzędów certyfikacji musimy zgłaszać certyfikaty do internetowego rejestru CT. Jeśli certyfikat jakiejś witryny nie znajduje się w rejestrze, współczesne przeglądarki będą ostrzegały użytkowników, że nie jest ona godna zaufania. Nikt w branży nie chce, by przeglądarki informowały, że jego witryna jest niebezpieczna.

Gdy włączysz ten nagłówek bezpieczeństwa:

1. Przeglądarki użytkowników będą — z lepszym lub gorszym skutkiem — sprawdzały, czy w dziennikach CT znajduje się Twój certyfikat.
2. Jeśli zostanie mu nadana wartość `enforce`, przeglądarka użytkownika będzie wymuszała przejrzystość certyfikatu. Oznacza to, że jeśli Twojego certyfikatu nie będzie w rejestrze albo wystąpi inny problem z jego przejrzystością, przeglądarka przerwie połączenie między Twoją stroną a użytkownikiem. Jeśli w nagłówku bezpieczeństwa nie ma trybu wymuszania, na określony adres URL wysyłany jest raport.

Zaleca się, by na początku wdrożyć tryb „wyłącznie raportowania", a potem, gdy będziesz mieć pewność, że certyfikaty są przyjmowane i poprawnie zarejestrowane, możesz przejść do trybu „wymuszania".

Pole `max-age` (definiowane w sekundach) to okres, przez który przeglądarka stosuje to ustawienie (przez tyle czasu jest buforowane w przeglądarce).

Ostrzeżenie **Podobnie jak w przypadku raportów nagłówka CSP, adresy URL raportów nagłówka Expect-CT nie są prywatne. Do znajdujących się tam informacji dostęp mają wszystkie osoby, również te o niezbyt czystych intencjach.**

Kup książkę Poleć książkę

Oto dwa przykłady implementacji nagłówka Expect-CT:

Tylko raportowanie:

```
Expect-CT: max-age=86400, report-uri="https://wehackpurple.com/report"
```

Raportowanie i blokowanie:

```
Expect-CT: max-age=86400, enforce, report- uri="https://wehackpurple.com/report"
```

A oto kolejny przykład z projektu OWASP DevSlop, tym razem dla aplikacji .NET CORE. Wymaga on dodania do projektu pakietu Nuget („Nwebsec.AspNetCore.Middleware”)[14], tak jak w pierwszym wierszu tego przykładu:

```
<PackageReference Include="Nwebsec.AspNetCore.Middleware" Version="2.0.0"/>

//Nagłówki bezpieczeństwa .NET CORE (to nie to samo co ASP.NET)
app.UseHsts(hsts => hsts.MaxAge(365).IncludeSubdomains());
app.UseXContentTypeOptions();
app.UseReferrerPolicy(opts => opts.NoReferrer());
app.UseXXssProtection(options => options.EnabledWithBlockMode());
app.UseXfo(options => options.Deny());
app.UseCsp(opts => opts
.BlockAllMixedContent()
.StyleSources(s => s.Self())
.StyleSources(s => s.UnsafeInline())
.FontSources(s => s.Self())
.FormActions(s => s.Self())
.FrameAncestors(s => s.Self()).
ImageSources(s => s.Self()).
ScriptSources(s => s.Self())
);
//Koniec nagłówków bezpieczeństwa
```

Rozszerzenie przypinania klucza publicznego dla protokołu HTTP (HPKP)

Przypinanie klucza publicznego (ang. *Public Key Pinning*) to system stworzony w celu ochrony przed oszukańczymi certyfikatami. Założeniem tego systemu było to, że każdy adres URL bądź witryna będą miały tylko jeden albo dwa zaufane urzędy certyfikacji. Na początku zaimplementowano go przy użyciu statycznych powiązań (wbudowanych bezpośrednio i ręcznie w przeglądarkę, a konkretnie Chrome i Firefox), ale ostatecznie właściciele innych witryn również chcieli „przypinać” do nich certyfikaty, więc pozwolono robić to dynamicznie. Certyfikat jest przypinany na pewien czas (zwykle rok) do konkretnej tożsamości kryptograficznej (klucza). Utrata klucza oznaczała jednak utratę kontroli nad witryną przez okres nawet jednego roku, bo bez niego nie mogła ona działać. Przez ten czas adres URL witryny nie nadawał się w zasadzie do użytku i w ten sposób funkcja bezpieczeństwa mogła spowodować katastrofalne szkody w działalności biznesowej. Sytuację tę porównywano do samobójczego gola.

Chociaż ten nagłówek bezpieczeństwa daje znaczne korzyści, związane z nim ryzyko sprawia, że korzystanie z tej funkcji jest niezalecane, chyba że ktoś musi zapewnić wyjątkowo wysoki poziom bezpieczeństwa, ma zespół, który dysponuje ogromną wiedzą na ten temat, i chce podjąć ryzyko biznesowe.

Kup książkę Poleć książkę

Ostrzeżenie **Ten nagłówek bezpieczeństwa jest uważany za przestarzały i przestał być obsługiwany.**

Zabezpieczanie ciasteczek

Protokół HTTP nigdy nie był projektowany pod kątem obsługi sesji użytkownika (śledzenia, czy ktoś jest zalogowany albo ma coś w koszyku na zakupy); obserwowanie czegokolwiek podczas przechodzenia użytkownika ze strony na stronę w obrębie witryny jest określane jako śledzenie „stanu" (ang. *state*). Do przekazywania informacji o sesji użytkownika z przeglądarki do serwera i z powrotem służą ciasteczka (ang. *cookie*). Można je też zachować na przyszłość, aby zapewnić użytkownikowi większy stopień personalizacji (np. zapamiętać wybrany przez niego język). Chociaż w wielu witrynach ciasteczka służą do przechowywania informacji w celach marketingowych i do śledzenia użytkownika, a nawet sprzedawania zebranych informacji innym firmom, nie będziemy się tu zajmować aspektami etycznymi związanymi z prywatnością danych przechowywanych w ciasteczkach.

Jeśli w aplikacji stosowane są ciasteczka, to aby zapewnić bezpieczeństwo zawartych w nich danych, należy użyć pewnych ustawień, które zostaną omówione w kolejnych podpunktach.

Uwaga: czasami programiści decydują się na użycie zamiast ciasteczek magazynu lokalnego (ang. *local storage*). Aby ochronić ten magazyn, należy podjąć zupełnie inne środki ostrożności, które nie zostaną opisane w tym punkcie. Poza tym informację o sesji należy *zawsze* przekazywać w ciasteczku sesji (a nie w trwałym ciasteczku) i nigdy nie wolno zapisywać jej w magazynie lokalnym. *Na potrzeby sesji zawsze używaj ciasteczka sesji* (ang. *session cookie*).

Nie zapominaj też zawsze sprawdzać poprawności danych wejściowych, które umieszczasz w ciasteczkach. Jeśli pobrane dane nie mają sensu, odrzuć je i spróbuj ponownie. Dane zawarte w ciasteczku są bardzo cenne i należy zadbać, by zawsze były chronione (więcej informacji na ten temat będzie w następnych rozdziałach).

Flaga Secure

Flaga `Secure` zapewnia, że ciasteczko będzie wysyłane tylko kanałami szyfrowanymi (HTTPS). Jeśli atakujący spróbuje obniżyć poziom bezpieczeństwa sesji na HTTP, aplikacja sieciowa odmówi wysłania ciasteczka. To ustawienie powinno zawsze być włączone:

```
Set-Cookie: Secure; (plus pozostałe ustawienia)
```

Flaga HttpOnly

Nazwa tej flagi nie jest intuicyjna, bo nie ma nic wspólnego z wymuszaniem nieszyfrowanego połączenia (HTTP), i może być dla programistów myląca. Ustawienie w ciasteczku tej flagi oznacza, że nie jest ono dostępne dla kodu JavaScriptu; może być zmienione tylko po stronie serwera. To ustawienie jest stosowane po to, by chronić przed atakami XSS mającymi na celu uzyskanie dostępu do danych ciasteczka. Wartość tę powinno się ustawiać zawsze, we wszystkich ciasteczkach, jako kolejną warstwę ochrony przeciw atakom XSS wymierzonym w poufność danych zawartych w ciasteczku:

```
Set-Cookie: HttpOnly; (plus pozostałe ustawienia)
```

Kup książkę Poleć książkę

Trwałość

Jeśli zbierasz poufne dane użytkowników lub zarządzasz sesją, ciasteczko nie powinno być trwałe; w celu ochrony tych danych powinno ulegać destrukcji na końcu sesji. Ciasteczko niszczone na końcu sesji jest nazywane ciasteczkiem sesji. Jeśli tak nie jest, nazywa się je ciasteczkiem trwałym (ang. *persistent cookie*) lub śledzącym (ang. *tracking cookie*). Aby ustalić, czy ciasteczka śledzące są potrzebne, porozmawiaj z zespołem ds. prywatności i z analitykiem biznesowym.

Datę ważności trwałych ciasteczek ustawia się w atrybucie `expires`, a maksymalny czas ich istnienia — w ustawieniu `max-age`.

Wygasa 1 stycznia 2021 r.

```
Set-Cookie: Expires=Mon, 1st Jan 2021 00:00:00 GMT; (plus pozostałe ustawienia)
```

Maksymalny czas: 1 godzina

```
Set-Cookie: Max-Age=3600; (plus pozostałe ustawienia)
```

Domain

Jeśli chcesz, by do ciasteczka miały dostęp domeny inne niż Twoja, musisz je jawnie wyszczególnić jako zaufane przy użyciu atrybutu `Domain`. W przeciwnym razie przeglądarki przyjmą, że ciasteczka mają ograniczenie tylko do hosta (ang. *host-only*), co oznacza, że dostęp do nich ma tylko Twoja domena, i zablokują próby dostępu z innych miejsc. Taki typ wbudowanej ochrony jest uważany za „domyślnie bezpieczny"; oby tak działały wszystkie ustawienia oprogramowania!

```
Set-Cookie: Domain=app.NOTwehackpurple.com; (plus pozostałe ustawienia)
```

Ostrzeżenie **Jeśli nie ustawiłeś poddomeny (`NOTwehackpurple.com` zamiast `app.NOTwehackpurple.com`), to dostęp do ciasteczka będzie miała każda aplikacja i strona znajdująca się w tej domenie. Prawdopodobnie tego nie chcesz.**

path

Wiele adresów URL w sieci WWW składa się tak naprawdę z wielu oddzielnych aplikacji znajdujących się pod innymi ścieżkami i w różnych poddomenach. Dla użytkownika wyglądają jak jedna ogromna strona, ale w rzeczywistości mogą to być tysiące różnych aplikacji. Jeśli ktoś ma do czynienia

z taką sytuacją, zapewne powinien ograniczyć dostęp do ciasteczka tylko do specyficznego obszaru lokalizacji, w której znajduje się aplikacja — jej „ścieżki" (ang. *path*). Atrybut `path` nadaje się po to, by ograniczyć zakres ciasteczka:

```
Set-Cookie: path=/SciezkaTwojejAplikacji; (plus pozostałe ustawienia)
```

same-site

Atrybut `same-site` został utworzony przez firmę Google w celu walki z fałszerstwami żądań międzywitrynowych (ang. *cross-site request forgery* — CSRF). Akronim CSRF, wymawiany potocznie po angielsku jako „sea surf" (surfowanie w morzu), oznacza atak na użytkowników zalogowanych

Kup książkę Poleć książkę

w witrynie i polega na tym, że atakujący usiłuje wykonać działanie w ich imieniu, ale bez ich wiedzy i zgody. Zwykle robi to za pośrednictwem wiadomości e-mail wyłudzającej informacje (ang. *phishing email*).

Wyobraź sobie, że Alicja ma wystąpić w programie telewizyjnym, by porozmawiać o firmie, w której pracuje. Chciałaby tam świetnie wyglądać, więc postanowiła kupić w internecie nowy kostium. Loguje się na ulubionej stronie z ubraniami i przegląda wszystkie nowe rzeczy. Zauważa, że otrzymała e-mail. Jest to spreparowana wiadomość z łączem do strony, na której Alicja jest aktualnie zalogowana. Znajduje się tam kod instruujący witrynę, że pewien produkt ma być kupiony i wysłany, ale nie Alicji, tylko osobie atakującej. Gdyby Alicja kliknęła to łącze, a witryna nie miała odpowiednich środków ochrony, cały atak mógłby odbyć się tak, że Alicja nawet nie zdałaby sobie z niego sprawy. Taki atak może się powieść tylko wtedy, gdy użytkownik będzie zalogowany do witryny użytej w wiadomości e-mail wyłudzającej informacje.

Może się to wydawać nieprawdopodobne, ale przypomnij sobie, jak się logujesz do zwykle używanych przez siebie witryn. Czy zawsze klikasz przycisk wylogowania? Czy nigdy nie zostawiasz strony otwartej przez kilka dni? Nikt nie jest doskonały, a naszym obowiązkiem jest ochrona użytkowników, nawet jeśli pomylą się i klikną łącze wyłudzające informacje.

Wracając do tematu: wspomniany atrybut wymusza przestrzeganie reguły określającej, że ciasteczka mogą pochodzić tylko z tej samej witryny. Ciasteczka nie powinny mieć źródła skrośnego (ang. *cross-site* — spoza danej witryny). Dostępne opcje to `None` (ciasteczka mogą być przesyłane z dowolnego miejsca), `Strict` (tylko z Twojej domeny) i `Lax` (jeśli chcesz, by ciasteczka były przesyłane wtedy, gdy użytkownik wchodzi przez łącze albo z innej strony, będą w nich tylko jawne informacje). Jeśli nie chcesz ustawiać tego atrybutu w ciasteczkach, w nowoczesnych przeglądarkach wartością domyślną będzie `lax`, a w starszych `none`[15].

Wartość `lax` oznacza, że użytkownicy mogą pozostać zalogowani i przejść do innych witryn, a po powrocie ciasteczka wciąż będą działały. Ustawienie to jest zwykle stosowane na potrzeby nawigacji oraz dla innych funkcji dostępnych dla użytkownika w aplikacji, *zanim* się w niej zaloguje. Blokowane będą ewidentne ataki CSRF (żądania `POST`). Ustawienie to nie jest niezawodne, ale stanowi dobry kompromis, jeśli nie można przekonać kogoś do stosowania wartości `strict`.

```
Set-Cookie: same-site=Strict; (plus pozostałe ustawienia)
Set-Cookie: same-site=Lax; // Blokuje tylko żądania POST, zezwala na łącza.
```

Prefiksy ciasteczek

Prefiksy ciasteczek to nowość nieprzyjęta we wszystkich przeglądarkach, stanowią one jednak środek dogłębnej ochrony (dodatkową warstwę zabezpieczeń) na wypadek przypadkowej niewłaściwej obsługi ciasteczek w aplikacji. Jeśli np. doszło do złamania zabezpieczeń poddomeny, a ustawienie `path` dla ciasteczka obejmuje całą domenę, to poddomena o naruszonym bezpieczeństwie może próbować uzyskać do niego dostęp. Prefiksy poprzedzają nazwę ciasteczka, więc są widoczne dla serwera niezależnie od ich szyfrowania. Aby ciasteczko było dostępne tylko w konkretnej poddomenie, stosuje się prefiks `host`. Więcej informacji na ten temat dostępnych jest na stronach organizacji Mozilla i przeglądarki Chrome.

Kup książkę Poleć książkę

Prywatność danych

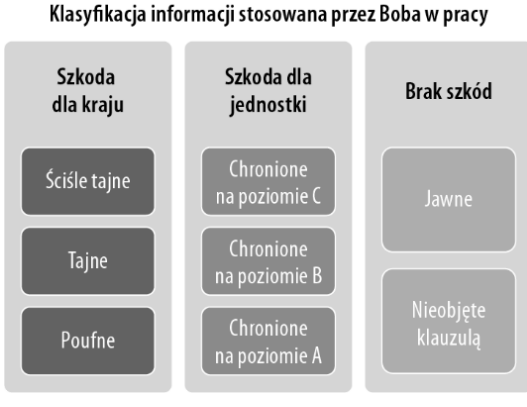
Większość witryn posiada obecnie politykę prywatności. Zgodnie z wymogami prawnymi Unii Europejskiej witryna musi informować, jakie dane gromadzi i w jaki sposób ich używa. Jeśli w Twoim miejscu pracy istnieje taka polityka, koniecznie się do niej stosuj. Jeśli nie, być może jej potrzebujesz. To sprawa do przemyślenia.

Klasyfikowanie informacji

Wszystkie dane zbierane, używane albo tworzone w aplikacji muszą być klasyfikowane i oznaczane, by każdy, kto pracuje nad projektem, wiedział, jak się z nimi obchodzić. W zależności od tego, w jakim kraju mieszkasz i gdzie pracujesz (w prywatnej firmie czy w agencji rządowej), stosowany będzie inny system klasyfikacji. Jeśli potrzebujesz wskazań, zapytaj w zespole ds. prywatności albo bezpieczeństwa.

Przyjrzyjmy się kilku przykładom obrazującym, dlaczego musimy klasyfikować informacje i jak to robić.

Bob pracuje w organie rządowym, w którym obowiązuje państwowy system klasyfikowania informacji (rysunek 2.2). Na obecnym stanowisku ma do czynienia z informacjami poufnymi (ang. *classified*), tajnymi (ang. *secret*) i ściśle tajnymi (ang. *top secret*). Istnieje rygorystyczny system identyfikowania plików i typów danych, którego Bob zawsze przestrzega. Poufność, tajność i ścisła tajność informacji oznaczają, że ich ujawnienie spowodowałoby zagrożenie dla całego państwa, a nie tylko dla pewnych osób. Na poprzednim stanowisku Bob miał do czynienia z danymi dużo mniej wrażliwymi. Informacje klasyfikowano tam jako publiczne (dostępne dla każdego) oraz chronione na poziomach A (ang. *Protected A*, mogące przynieść komuś szkodę lub narazić go na kłopot), B (ang. *Protected B*, szkodliwe dla jednej lub większej liczby osób) i C (ang. *Protected C*, mogące doprowadzić do śmierci lub innej nienaprawialnej krzywdy jednej lub większej liczby osób).



Rysunek 2.2. Klasyfikacja informacji stosowana przez Boba w pracy

Bob zawsze klasyfikuje (decyduje o klauzuli tajności) i oznacza zbierane przez siebie informacje, dzięki czemu każdy członek zespołu wie, jak się z nimi właściwie obchodzić. Gdyby były nieoznaczone, ktoś mógłby popełnić błąd i niechcący ujawnić lub zostawić bez ochrony bardzo wartościowe i (lub) wrażliwe informacje. Wiele systemów baz danych (m.in. Microsoft SQL Server) pozwala dodawać do pól danych lub tabel poziom klasyfikacji. Jeśli baza danych nie zapewnia natywnie

Kup książkę Poleć książkę

takich funkcji, można wprowadzić do tabeli dodatkowe pole i w ten sposób samodzielnie oznaczyć dane (albo dodać nowe pole dla każdego pola, jeśli ich poziomy poufności są zróżnicowane). Oznaczać należy nawet dane nieobjęte klauzulą i jawne.

Zawsze przestrzegaj zasad i regulacji dotyczących klasyfikowania informacji. Jeśli ich nie znasz, pytaj. Jeśli u Ciebie w pracy nie ma żadnych reguł, możesz przyjąć i stosować standardy rządowe albo postępować zgodnie z dokumentem organizacji NIST „Guide for Mapping Types of Information and Information Systems to Security Categories".

Uwaga **Choć nie należy tego traktować jako wymagania projektowego, to jednak powinna istnieć dokumentacja techniczna wyjaśniająca, jak obsługiwać wszystkie poziomy poufności danych, połączona z gotowym do użycia kodem lub bibliotekami do obsługi takich danych. Najlepiej, gdyby w tym kodzie lub bibliotece scentralizować zarządzanie danymi i ich przetwarzanie, co pozwoli uzyskać jednolite rezultaty.**

Hasła, przechowywanie danych i inne ważne ustalenia

Alicja ma kilka różnych kont: firmowe, domowe oraz na potrzeby hobby. Sytuacja Boba jest taka sama — w pracy i w życiu prywatnym ma do zapamiętania dosłownie setki haseł. Alicja, aby je wszystkie zapamiętać, używa menedżera haseł, a Bob po prostu ciągle korzysta z tych samych kilku (co przez specjalistów ds. bezpieczeństwa jest nazywane wielokrotnym używaniem haseł, ang. *password reuse*).

Wskazówka **Punkt ten zawiera wymagania, takie jak korzystanie z menedżera haseł, które niekoniecznie muszą być odpowiednie dla projektu, ale powinny raczej stanowić standard dla firmy informatycznej jako całości. Rób to, co ma sens w przypadku Twojej organizacji.**

Menedżer haseł (ang. *password manager*) to program, który pomaga użytkownikom tworzyć unikatowe, długie i złożone hasła oraz bezpiecznie je przechowuje. Ludzie używają takich programów po to, by nie musieć pamiętać wielu haseł; wystarczy, że znają jedno hasło dostępu do menedżera (oraz hasła, które nie mogą być w nim zapamiętane, np. do telefonu albo komputera). Tworzenie haseł przez menedżer gwarantuje, że są one losowe (nieprzewidywalne tak z punktu widzenia komputera, jak i człowieka) i unikatowe (użytkownik nie korzysta wielokrotnie z tych samych haseł, tak jak Bob).

Większość menedżerów haseł zawiera wtyczki do przeglądarek, więc po zalogowaniu się w menedżerze wystarczy nacisnąć przycisk, by zalogować się do każdej odwiedzanej strony. Nazwa użytkownika i hasło są kopiowane do przeglądarki, co chroni przed ich błędnym wpisaniem. Naprawdę dobre menedżery haseł informują, że to samo hasło zostało użyte w kilku miejscach, że nazwa i hasło użytkownika zostały ujawnione w wyniku naruszenia ochrony danych oraz że użytkownik wybrał słabe hasło (przy korzystaniu z menedżera wciąż można tworzyć własne hasła). Jeśli witryna, na której użytkownik ma konto, oferuje uwierzytelnianie MFA (wieloskładnikowe), dobry menedżer haseł powiadomi, że można włączyć tę funkcję.

W menedżerach haseł zwykle jest też miejsce na wpisywanie bezpiecznych notatek (można tam zapisać numer PESEL, dane kart kredytowych itp.). Doradzam korzystanie z menedżerów haseł wszystkim pasjonatom komputerów, nie tylko ze względu na dodatkowe zabezpieczenie życiowych

Kup książkę Poleć książkę

spraw, ale również w celu oszczędzenia czasu poświęconego na resetowanie zapomnianych haseł. Menedżery haseł są przeznaczone nie dla komputerów, ale dla ich użytkowników.

Jedną z największych zalet korzystania z menedżera i stosowania wszędzie unikatowych haseł jest to, że jeśli w jednej z witryn dojdzie do naruszenia ochrony danych i nazwy użytkowników wraz z hasłami (tzw. poświadczenia, ang. *credentials*) wyciekną do internetu lub dostaną się w ręce złośliwych podmiotów, będzie to miało wpływ tylko na to jedno konto, a nie na wszystkie. Jeśli przy włamaniu dochodzi do kradzieży danych uwierzytelniających, dość często są one wykorzystywane w *atakach zapychania poświadczeniami* (ang. *credential stuffing attack*), co oznacza, że atakujący używa skradzionych poświadczeń w zautomatyzowanym ataku na jedną lub więcej witryn, by zobaczyć, czy któreś z nich nie były wielokrotnie używane, dzięki czemu mógłby uzyskać dostęp do kont na innych stronach. Niektórzy atakujący posuwają się nawet do „dopasowywania" haseł w swoich skryptach, by wypróbowywać warianty haseł, np. zmieniają „Hasło1" na „Hasło2", „Hasło3" itd. To postępowanie nosi nazwę *ataku zapychania poświadczeniami z użyciem tablic tęczowych* (ang. *rainbow credential stuffing attack*).

Gdyby Alicja i Bob korzystali z tej samej witryny, w której doszło do naruszenia ochrony danych, Alicja, która wszędzie używa unikatowych haseł, musiałaby zresetować tylko to jedno hasło i martwiłaby się jedynie o to konto. Z kolei Bob miałby mnóstwo pracy i zmartwień! Inną metodą, jaką Alicja i Bob mogliby się zabezpieczyć przed atakami tego typu, jest włączenie uwierzytelniania MFA na każdym koncie, które jest dla nich wartościowe albo potencjalnie może przynieść szkodę (konta związane z pracą, bankowe, poczty elektronicznej, konta, do których przypięto karty kredytowe, konta bardzo osobiste, związane ze zdrowiem itp.).

menedżer haseł + wszędzie unikatowe hasła + MFA = dogłębna ochrona użytkownika

Programistom i informatykom, od których wymaga się zapamiętywania nieprawdopodobnej liczby haseł, powinno się zapewnić w miejscu pracy dostęp do menedżera haseł.

Magazyny wpisów tajnych (ang. *secret stores*) są podobne do menedżerów haseł, ale tworzy się je dla systemów komputerowych, a nie ludzi. Magazyn wpisów tajnych to programowy magazyn, który szyfruje wpisy tajne, a potem pozwala aplikacji na dostęp do nich w sposób programowy (poprzez kod zawarty w aplikacji albo w procesie budowania aplikacji w potoku ciągłej integracji lub ciągłego dostarczania). Można w nim przechowywać certyfikaty, poświadczenia (nazwy użytkowników i hasła), ciągi połączenia, wartości skrótu i wszystko to, co traktuje się jako „wpis tajny" (ang. *secret*), z którego korzysta aplikacja.

Konta usług (ang. *service accounts*) to konta używane przez komputery, nie przez ludzi. Takie konto reprezentuje tożsamość systemu komputerowego w sieci, a nie osobę. Jeśli tworzysz aplikację, która potrzebuje dostępu do bazy danych, lepiej utwórz dla niej konto usługi, zamiast łączyć się z bazą danych z własnego konta. Jest wiele powodów, dla których należy tak robić, a najbardziej oczywistym jest ten, że jeśli odejdziesz i zaczniesz pracować gdzie indziej, oprogramowanie nadal będzie musiało działać. Inne powody to monitorowanie (jeśli każda aplikacja będzie miała własne konto, wówczas zespół zajmujący się monitorowaniem będzie mógł zobaczyć, co robi aplikacja, a nie programista), kwestia dochodzenia w sprawie incydentów (będziemy chcieli się dowiedzieć, kto co zrobił, a jeśli wszędzie będzie Twoja nazwa użytkownika, może wyglądać na to, że to Twoja sprawka), najmniejsze uprzywilejowanie (Twoje konto prawdopodobnie ma mnóstwo uprawnień do różnych rzeczy i jeśli ktoś się do niego dostanie, może przejąć też wszystkie aplikacje) itd. Przy tworzeniu i włączaniu kont usług stosuj zawsze zasadę najmniejszego uprzywilejowania.

Kup książkę Poleć książkę

Wskazówka **Hasła do kont usług nie powinny nigdy wygasać, chyba że doszło do naruszenia zabezpieczeń (albo się to podejrzewa). Powinny też być niezwykle złożone i tak długie, jak to tylko możliwe. Konta usług nie powinny mieć też możliwości (przywilejów) logowania do stacji roboczych ani dostępu do żadnego innego zasobu poza tymi, które były dla nich dostępne zaraz po utworzeniu.**

A co z hasłami użytkowników aplikacji? Gdzie mają być przechowywane? Powinny być zapisane w bazie danych (lub innym centralnym miejscu zarządzania, takim jak dostawca tożsamości, ang. *identity provider*) w formacie posolonym (ang. *salted*) i skróconym (ang. *hashed*). Jak być może pamiętasz z wcześniejszej części rozdziału, tworzenie skrótu to jednokierunkowy proces kryptograficzny, którego nie można cofnąć. Sól to unikatowa, długa wartość dodawana do hasła przed jego skróceniem w celu zwiększenia entropii i utrudnienia potencjalnemu atakującemu złamania lub odgadnięcia hasła. Oznacza to, że przy logowaniu użytkownika do aplikacji na wprowadzonej przez niego wartości wykonywane są funkcje skracania i solenia, a wynik jest porównywany z tym, co jest przechowywane w bazie danych. Gdyby doszło do kradzieży nazw użytkowników i ich haseł, dla atakującego będą one bezużyteczne, bo jeśli wprowadzi skróconą i posoloną wartość do aplikacji, dokonane zostaną na niej ponownie te same operacje, co spowoduje zmianę tej wartości, przez co nie będzie ona odpowiadać danym z bazy.

Uwaga ***Łamanie* (ang. *cracking*) oznacza wielokrotne zgadywanie hasła za pomocą zautomatyzowanego narzędzia przy użyciu listy słów aż do skutku. Proces ten nazywany jest też *metodą siłową* (ang. *brute forcing*) — jest to ciągłe odgadywanie czegoś, aż do chwili uzyskania dostępu, z wykorzystaniem automatyzacji.**

Sól powinna mieć długość co najmniej 28 znaków (najlepiej, by była dużo dłuższa), być wytwarzana przez bezpieczny generator liczb losowych i unikatowa dla każdego użytkownika aplikacji. Wartość soli dla każdego użytkownika przechowuje się w bazie danych wraz ze skrótem wartości pary sól + hasło. Wartość soli nie jest tajna, w odróżnieniu od pieprzu (który zostanie omówiony za moment).

Do nowszych technik sprawiających, że hasła są wyjątkowo trudne do złamania, należą zwiększanie nakładu pracy i dodawanie pieprzu.

Zwiększanie *nakładu pracy* (ang. *work factor*) oznacza, że algorytm wyznaczania wartości skrótu powtarza się *X* razy, gdzie *X* oznacza nakład pracy[16]. Wartość nakładu pracy powinna wynosić co najmniej 2, ale można ją w razie potrzeby zwiększyć (stosownie do ulepszeń sprzętu w naszej branży, poziomu wrażliwości danych i kont albo innych zmian na płaszczyźnie zagrożeń aplikacji).

Kryptograficzny „pieprz" (ang. *pepper*) jest podobny do soli pod tym względem, że jest dodawany do hasła przed jego skróceniem i powinien być tworzony w bezpiecznym generatorze liczb losowych. Wartość pieprzu jest jednak tajna i nie należy jej przechowywać w bazie danych, tak jak sól (tylko w magazynie wpisów tajnych). Powinna być dość długa (mieć co najmniej 32 znaki, a najlepiej 128). Pieprz jest unikatowy dla każdej aplikacji, ale taki sam dla wszystkich jej użytkowników.

Możliwe jest jednoczesne dodawanie do haseł soli i pieprzu, jednak generalnie w większości systemów konieczne jest tylko solenie. Podejmij decyzję razem z zespołem ds. bezpieczeństwa.

Kup książkę Poleć książkę

Wskazówka **Obsługę użytkowników Twojego systemu najlepiej byłoby prowadzić w ramach rozwiązania do zarządzania tożsamością. Trudna praca związana z zabezpieczeniami zostałaby wtedy przeniesiona do systemu stworzonego specjalnie w tym celu.**

Ostrzeżenie **Okresowa zmiana wartości pieprzu, o ile jest konieczna, może doprowadzić do unieważnienia wszystkich haseł wszystkich użytkowników aplikacji. Musisz zachować ostrożność przy jego rotacji. Istnieje spore ryzyko, że konieczność resetowania haseł zdenerwuje użytkowników i klientów, więc jeśli zdecydujesz się korzystać z pieprzu, zaprojektuj aplikację z uwzględnieniem jego rotacji.**

Dopilnuj, by hasła użytkowników Twojej aplikacji były długie, ale niekoniecznie skomplikowane; jeśli ma być w nich mała i duża litera, cyfra i znak specjalny, to dobrze, ale określanie liczby znaków z każdej kategorii jest dla użytkowników frustrujące. W celu spełnienia wymogu „złożoności" wystarczy zażądać tylko po jednym. Pozwól wprowadzać hasła o długości nawet do 64 znaków. Im dłuższe, tym lepsze; zachęcaj do stosowania fraz hasłowych. Nie zmuszaj użytkowników do zmiany haseł po upływie określonego czasu, chyba że podejrzewasz, że doszło do naruszenia zabezpieczeń. Weryfikuj, czy hasła nowych użytkowników nie zostały wcześniej złamane, poprzez porównywanie fragmentarycznych skrótów SHA1 przy użyciu usługi takiej jak np. HaveIBeenPwned.

Ostrzeżenie **Podczas weryfikowania, czy hasło użytkownika nie zostało już wcześniej złamane, zadbaj o ochronę anonimowości swoich użytkowników. Istnieje kilka modeli takich operacji, np. K-anonymity; więcej informacji uzyskasz na stronie *haveibeenpwned.com/API/v3#SearchingPwnedPasswordsByRange*.**

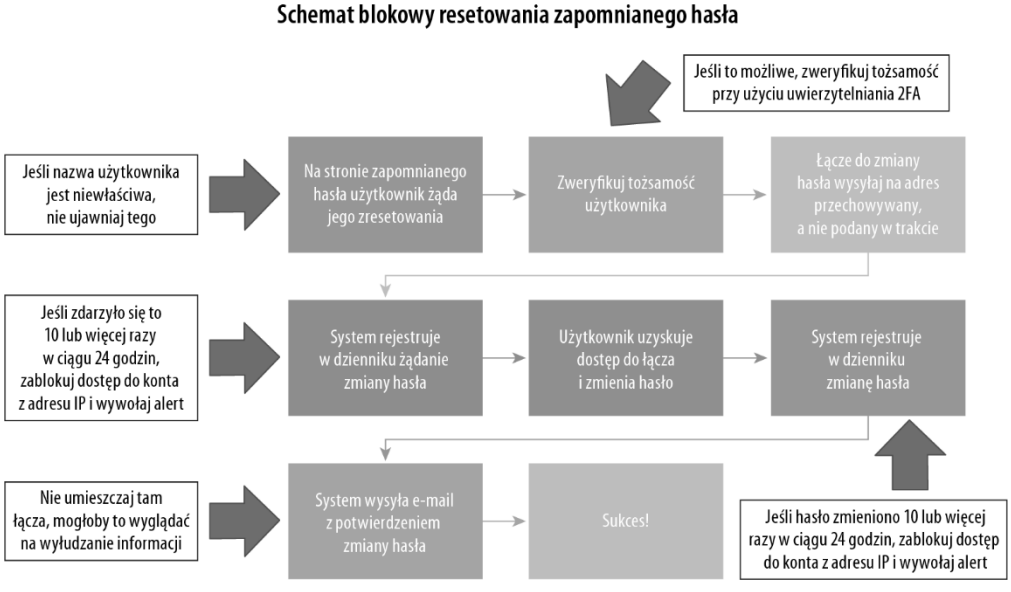
Powiadamiając o nieudanym logowaniu, nigdy nie pozwalaj zweryfikować, czy niepoprawna była nazwa użytkownika, czy jego hasło. Ujawnianie takich informacji pozwala potencjalnym atakującym zbierać nazwy użytkowników (sprawdzać, czy dany użytkownik korzysta, czy nie korzysta z Twojego systemu).

Wskazówka **Pytania zabezpieczające nie są nowoczesną metodą weryfikacji, bo większość użytkowników wybiera pytania, na które odpowiedzi są publicznie dostępne (np. w mediach społecznościowych), dlatego w miarę możliwości pytań zabezpieczających należy unikać.**

W razie *zapomnienia hasła* przed wysłaniem łącza do jego zresetowania w wiadomości e-mail lub SMS należy zweryfikować tożsamość użytkownika poprzez użycie innej formy uwierzytelnienia albo przez zadanie mu pytań zabezpieczających. Jeśli uwierzytelnienie użytkownika się nie powiedzie, nigdy nie ujawniaj, dlaczego tak się stało. Nigdy nie pozwalaj użytkownikowi resetować hasła bezpośrednio w Twoim systemie; zawsze stosuj komunikację „poza pasmem" (przez łącze w wiadomości e-mail lub SMS) jako drugą formę uwierzytelniania. Łącze do resetowania powinno być jednorazowe, a nieużyte wygasać w ciągu godziny. Zawsze rejestruj zdarzenie resetowania hasła (lub podjętą próbę, jeśli się nie powiodła) i wysyłaj wiadomość e-mail na adres poczty albo SMS na numer telefonu podany na tym koncie z potwierdzeniem zresetowania hasła. Jeśli użytkownik będzie bez powodzenia próbował resetować hasło 10 razy w ciągu 24 godzin, zablokuj dostęp do tego konta

Kup książkę Poleć książkę

z adresu IP, z którego pochodziły żądania, zarejestruj to zdarzenie i wywołaj alert. Jeśli z jakiegoś adresu IP podejmowane są próby zresetowania haseł dla konta, które nie istnieje, potraktuj ten adres tak, jak gdyby konto istniało, by nie pozwalać atakującym na wyliczanie (zbieranie) poprawnych nazw użytkowników. Przejrzyj rysunek 2.3 przedstawiający schemat blokowy bezpiecznego resetowania zapomnianego hasła.



Rysunek 2.3. Schemat blokowy resetowania zapomnianego hasła

Zasady dotyczące haseł

- Skracaj i sól wszystkie hasła użytkowników. Niech sól ma co najmniej 28 znaków.
- Wszystkie wpisy tajne aplikacji muszą być przechowywane w magazynie wpisów tajnych.
- Wszystkie używane w aplikacji konta muszą być kontami usług (a nie osób). Muszą być unikatowe dla każdej aplikacji i powinny być zgodne z koncepcją najmniejszego uprzywilejowania.
- Dopilnuj, by wszyscy członkowie zespołu korzystali z zarządzanych przez organizację menedżerów haseł, i ustal zasadę, że nie wolno wielokrotnie używać haseł ani wariantów tego samego hasła.
- Włącz uwierzytelnianie MFA we wszystkich ważnych kontach, zarówno w pracy, jak i w domu. Jeśli to możliwe, jako drugi składnik uwierzytelniania zastosuj aplikację albo urządzenie uwierzytelniające zamiast wiadomości SMS.
- Nie wymuszaj zmiany haseł według harmonogramu, ale wyłącznie po naruszeniu zabezpieczeń albo stwierdzeniu podejrzanej aktywności.
- Zawsze używaj *nowoczesnego* algorytmu wyznaczania wartości skrótu.

Kup książkę Poleć książkę

- W razie wątpliwości postępuj zgodnie z ustalonymi przez rząd zasadami przechowywania haseł, a jeśli takich nie ma, stosuj się do reguł organizacji NIST. Zostały one stworzone przez zespół ekspertów i sprawdzone przez ludzi z branży, więc są to porady najlepsze z możliwych.

Wskazówka **Informacje o nowoczesnych algorytmach wyznaczania wartości skrótu i dodatkowe porady na temat przechowywania haseł można znaleźć na stronie *cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html*.**

Wszędzie powinien być ruch HTTPS

Gdy powstawał internet, nikt nie wyobrażał sobie, że rozwinie się on do obecnego stanu: miliardów użytkowników codziennie i bilionów witryn, ani tego, że stanie się główną metodą komunikacji dla większości ludzi na świecie. Nie zaplanowano go więc pod tym kątem. Zabezpieczenia, prywatność, sposób transferu funduszy itd. nie zostały wbudowane w protokoły i cały czas tworzymy rozwiązania zastępcze. Pierwotnie wszystko w internecie było nieszyfrowane (przesyłane jawnym tekstem), bo projektantom nawet się nie śniło, że będziemy mieli dzisiejsze narzędzia, pozwalające obserwować i odczytywać ruch w sieciach i internecie (tzw. przechwytywanie pakietów, ang. *sniffing*), a nawet go zatrzymywać, zmieniać i wysyłać do celu zupełnie niezauważalnie dla odbiorcy. Właśnie dlatego musimy szyfrować cały ruch w internecie. Nie każdy jednak akceptuje tę koncepcję, więc zanim opowiem, *jak* to zrobić, wyjaśnię, *dlaczego* jest to konieczne.

Gdy użytkownik odwiedza witrynę internetową, nie wiadomo, skąd przychodzi. Może korzystać z otwartej sieci wi-fi w kawiarni, być na konferencji w wielkim hotelu, przebywać w mieszkaniu w sieci współdzielonej z sąsiadem albo w bardzo zabezpieczonym środowisku wojskowym. Aplikację niestety trzeba projektować pod kątem najgorszego, a nie najlepszego z wymienionych scenariuszy, co oznacza, że trzeba koniecznie chronić użytkowników, którzy korzystają z niezabezpieczonej sieci.

Sytuacje, kiedy ktoś może nasłuchiwać ruch, to takie, w których ludzie znajdują się w niezabezpieczonej sieci, np. w kawiarni, albo gdy sąsiedzi korzystają razem z sieci wi-fi. Nie wiadomo, czy nie ma tam jeszcze kogoś, kto obserwuje osoby odwiedzające daną witrynę.

Istnieje też wiele sytuacji, w których ruch jest obserwowany ze stuprocentową pewnością, np. w hotelu, w pracy albo po połączeniu z siecią, w której przeprowadzana jest „głęboka inspekcja pakietów". Zdarza się to dużo częściej, niż ludziom się wydaje.

Uwaga ***Głęboka inspekcja pakietów*** **(ang. *deep packet inspection*) oznacza takie przetwarzanie danych, podczas którego badane są pakiety (dane) przesyłane przez sieć, co pozwala m.in. na blokowanie, przekierowywanie i (lub) logowanie ruchu[17].**

Jeśli ktoś przechwytuje ruch sieciowy, może nie tylko sprawdzić, co ogląda użytkownik (przez co łamie jego prywatność), ale też podmienić przesyłane do niego informacje. Obecnie w dużych sieciach hoteli często spotykaną praktyką jest zastępowanie wszystkich reklam na nieszyfrowanych witrynach własnymi; to właśnie jeden z przykładów zmiany widzianych przez użytkownika informacji. Innym przykładem może być wprowadzanie do odwiedzanych witryn szkodliwego oprogramowania, skryptów lub nieprawdziwych informacji. Szkody ponoszone przez użytkowników mogą

Kup książkę Poleć książkę

być zaskakująco duże, chociaż zagrożenie poważnym atakiem jest stosunkowo niewielkie (chyba że dotyczy on naprawdę bardzo dużej lub popularnej witryny). Jeśli nie są to wystarczające argumenty za tym, by używać tylko protokołu HTTPS, warto wspomnieć, że współczesne przeglądarki ostrzegają teraz użytkowników przed tym, że odwiedzają niezabezpieczoną stronę (w formie graficznej lub tekstowej), kiedy ruch odbywa się protokołem HTTP zamiast HTTPS. Nie warto mówić, jak negatywnie wpływa to na prowadzoną działalność!

Skoro przekonaliśmy się, że zawsze musimy używać protokołu HTTPS, zdefiniujmy *regułę*.

Udostępniaj stronę tylko w protokole HTTPS. Przekierowuj ruch HTTP na HTTPS. Jeśli ktoś będzie próbował obniżyć poziom bezpieczeństwa połączenia, przekieruj to połączenie. Można tego dokonać za pomocą nagłówków bezpieczeństwa w kodzie albo w ustawieniach serwera. O tym, jak to zrobić, wspomniano nieco wcześniej w punkcie „Nagłówki bezpieczeństwa — pasy bezpieczeństwa dla aplikacji sieciowych".

Ustawienia protokołu TLS

Upewnij się, że przy szyfrowaniu korzystasz z ostatniej wersji protokołu TLS (obecnie 1.3, ale w chwili pisania tej książki, jeśli dostawca nie zapewniał obsługi wydania 1.3, wersja 1.2 wciąż nadawała się do użycia). Ponieważ zmiany następują bardzo szybko, warto poszukać w internecie najlepszych aktualnie rozwiązań.

Inne zalecane rozwiązania w tym obszarze są omawiane w rozdziale 3., „Projektowanie pod kątem bezpieczeństwa".

WSKAZÓWKI DOTYCZĄCE PROTOKOŁU TLS

Z rekomendowanymi ustawieniami protokołu TLS można zapoznać się na stronach: *bettercrypto.org/#webservers* i *https://ssl-config.mozilla.org.*

W celu uzyskania pomocy w zakresie konfiguracji zgodności zapoznaj się ze stroną: *wiki.mozilla.org/Security/Server_Side_TLS.*

Aby przetestować konfigurację, uruchom skrypt *testssl.sh*.

Komentarze

Programiści piszą komentarze (uwagi w kodzie, które nie są wykonywane, stanowią po prostu tekst) po to, by oni lub ich następcy mogli łatwiej zrozumieć, co się dzieje w programie. Czasem jednak umieszczają w nich rzeczy, których nie powinni, np. hasła do bazy danych lub inne wpisy tajne, informacje o firmie albo użytkownikach, które nie powinny być udostępniane, lub jeszcze inne poufne treści. Komentarze umieszczone w kodzie front-endu (JavaScript, HTML, CSS) nie zawsze są usuwane w czasie tworzenia paczek i osoba przeprowadzająca inżynierię wsteczną, jeśli będzie ich szukała, często może je znaleźć. Dlatego nigdy, przenigdy nie powinniśmy umieszczać w komentarzach poufnych informacji.

Wskazówka **Istnieje wiele narzędzi poszukujących wpisów tajnych w kodzie, zarówno dla potoków CI/CD, jak i do skanowania repozytorium. Zawsze z korzystaj z takiego narzędzia!**

Kup książkę Poleć książkę

Tworzenie kopii zapasowych i przywracanie danych

Należy regularnie (zgodnie z obowiązującymi w organizacji zasadami klasyfikowania danych) tworzyć kopie zapasowe wszystkich danych używanych i tworzonych przez aplikację lub w niej przechowywanych. Cotygodniowe kopie zapasowe powinny być tworzone w innej, odmiennej geograficznie lokalizacji. System musi być zdolny do przywracania danych w razie napotkania szkodliwego oprogramowania, incydentu naruszenia bezpieczeństwa czy próby wyłudzenia okupu. Procedury tworzenia kopii zapasowych i przywracania z nich danych muszą być przetestowane i przećwiczone.

Funkcje bezpieczeństwa na platformach programistycznych

Utrzymaniem platform programistycznych (ang. *programming frameworks*) zajmują się zespoły ekspertów, a regularnie testują je setki, a nawet tysiące programistów. Funkcje bezpieczeństwa platform sprawdzają hakerzy, nie tylko ci etyczni. Oznacza to, że za tymi funkcjami, w porównaniu z napisanymi samodzielnie, stoi dużo więcej osób, doświadczenia, czasu i testów, więc niemal pewne jest, że będą one bezpieczniejsze niż napisany własnoręcznie kod zabezpieczeń. Właśnie dlatego należy korzystać z funkcji bezpieczeństwa wbudowanych w platformę, zamiast pisać własne.

Każdy doświadczony tester penetracyjny aplikacji sieciowych czy inżynier bezpieczeństwa zna co najmniej jeden przypadek programisty, który zdecydowawszy się na funkcję kryptograficzną „własnej roboty" zamiast tych zawartych we frameworku, koduje wartość algorytmem Base64 (czasem wielokrotnie) albo pisze funkcję mieszającą wartości. Na nieszczęście dla tych programistów najprostsze wyzwania w konkursach Capture the Flag (CTF — zdobądź flagę) często dotyczą znajdowania takich prostych i nieadekwatnych metod zabezpieczeń, co oznacza, że te „środki ochronne" są na ogół łatwo odkrywane i obchodzone.

Inne przykłady to pisanie własnej funkcji kodującej, usiłowanie samodzielnego zarządzania sesjami, zapisywanie danych sesji do magazynu lokalnego zamiast do bezpiecznych ciasteczek itp. Jeśli Twoja platforma programistyczna oferuje daną funkcję bezpieczeństwa, korzystaj z niej. Jeśli nie, zastanów się nad dodaniem do aplikacji innej platformy lub komponentu, który zawiera potrzebne Ci funkcje, a własne rozwiązania twórz tylko w razie absolutnej konieczności.

Dług techniczny = dług bezpieczeństwa

> *Dług techniczny to pojęcie z dziedziny programowania odzwierciedlające domniemany koszt dodatkowej pracy spowodowanej wybraniem w pewnej chwili łatwego (ograniczonego) rozwiązania zamiast lepszego, które zajęłoby więcej czasu.*
>
> — Wikipedia[18]

Dług techniczny może obejmować wpisywanie wartości bezpośrednio w kod (ang. *hard coding*), nieaktualizowanie serwerów lub platform przez dłuższe okresy, niestosowanie poprawek, wybieranie dróg na skróty w celu „szybkiego załatwienia sprawy" itd. Dług techniczny powoduje często, że organizacja wolno reaguje na zmiany, tzn. nie może odpowiedzieć w adekwatnym czasie na incydent naruszenia bezpieczeństwa lub inne jego zagrożenie. Obie te sytuacje prowadzą do osłabienia bezpieczeństwa i niezdolności organizacji do efektywnej samoobrony.

Kup książkę Poleć książkę

Bob musiał pracować z dużą ilością długu technicznego w pewnym urzędzie państwowym. Było tam tak wiele procesów i pozwoleń, że czasem nawet nie wiedział, jak przeprowadzić daną zmianę. Konieczna była akceptacja zespołów CAB (Change Approval Board — rada ds. zmian) i TAG (Technical Architecture Group — grupa architektury technicznej), ale najpierw zmianę zatwierdzał szef szefa szefa jego szefa. Konieczność zatwierdzania zmian przez kogoś, kto stoi cztery szczeble wyżej w hierarchii, dla Boba po prostu nie miała sensu. Skąd dyrektor generalny, bo o nim mowa, może mieć pojęcie, czy zmiana proponowana przez Boba nie powoduje konfliktu z innymi rozwiązaniami technicznymi? Taka osoba nie pisze przecież kodu. Bob był często sfrustrowany i czuł, że kierownictwo po prostu mu nie ufa, co spowalnia wiele spraw, a nie przynosi żadnych korzyści.

Gdy mężczyzna pracował w tym urzędzie, miał tam miejsce bardzo poważny incydent naruszenia bezpieczeństwa. Bob podsłuchał, jak programiści mówili osobie z zespołu ds. bezpieczeństwa, że mają 16-miesięczny cykl wydawniczy dla nowych funkcji, a na wydanie awaryjne potrzeba co najmniej 4 miesięcy. Twarz specjalisty od bezpieczeństwa mówiła wszystko: dla urzędu nadchodziły bardzo złe dni. Gdy Bobowi zaproponowano pracę w innym wydziale, z nowocześniejszym podejściem do rozwoju oprogramowania, skorzystał z tej szansy; krępujący dług techniczny w poprzednim wydziale był dla niego sporym obciążeniem w codziennych obowiązkach, poza tym mężczyzna chciał pracować w miejscu, w którym kierownictwo ma do niego zaufanie.

Gdy organizacje nie są zdolne do dokonywania zmian bez długich okresów wstępnych czy herkulesowych wysiłków, ich efektywność niweczy właśnie dług techniczny. Jeśli ich pracownicy spędzają większość czasu na pilnowaniu tego, by świeciło się światło, i na ciągłym gaszeniu pożarów, najpewniej będą mieli również problemy z bezpieczeństwem.

Przekazywanie plików

Jeśli konieczne jest przekazywanie plików (ang. *file upload*) przez użytkowników, to zamiast pisać kod samodzielnie, znajdź, jeśli to możliwe, dedykowany, uznany komponent innej firmy i użyj go w aplikacji. Ryzyko będzie mniejsze, bo był on już na szeroką skalę testowany (tylko takie komponenty wybieraj). Umożliwienie osobom postronnym (w przeciwieństwie do uwierzytelnionych użytkowników z własnej organizacji) przekazywania plików jest najbardziej ryzykowną funkcją typowej aplikacji. Zazwyczaj wprowadzanie danych przez użytkowników jest uważane za najbardziej groźny element aplikacji, lecz umożliwienie im przekazywania plików (które potencjalnie mogą być szkodliwe) podnosi zagrożenie danymi wejściowymi użytkowników na zupełnie nowy poziom.

Alicja pamięta, jak przypadkiem przyniosła do pracy wirusa; było to bardzo żenujące. W domu ma maca, ale w pracy korzysta z systemu Windows. Przyniosła pamięć USB z muzyką, którą pobrała w domu i od dawien dawna słuchała. Gdy podłączyła ją w pracy do swojego komputera z Windowsem, niezauważony przez nią plik EXE uruchomił się i… przy jej biurku natychmiast pojawili się ludzie z zespołu ds. bezpieczeństwa.

Alicja jest niezwykle inteligentna, ale nie jest specjalistką od bezpieczeństwa, zresztą każdy może się pomylić. Przy opracowywaniu programów należy brać pod uwagę zarówno złośliwe podmioty, jak i ludzi, którzy przez pomyłkę przekazują do aplikacji pliki niewłaściwego typu.

Podczas przyjmowania pliku przekazanego przez użytkownika przygotuj się na najgorsze. Zweryfikuj jego typ i rozmiar, zmień jego nazwę, nie pozwól użytkownikowi na ustalenie miejsca, w którym będzie zapisany, i przechowuj go w bezpiecznym miejscu, z dala od reszty aplikacji i serwera WWW.

Kup książkę Poleć książkę

Po przyjęciu pliku przeskanuj go co najmniej jednym narzędziem w celu zweryfikowania, czy plik nie budzi jakichś wątpliwości. Jeśli dział biznesowy na to pozwoli, ogranicz przekazywanie plików tylko do pewnych ich typów, które są mniej niebezpieczne, np. przyjmuj pliki typu JPG, TXT i PNG, ale nie PDF ani EXE.

Jedna ze ściągawek OWASP Cheat Sheet, projektu o otwartych źródłach rozwijanego pod egidą organizacji Open Web Application Security Project, zawiera doskonałą, obszerną listę środków ostrożności, które należy stosować w razie pozwolenia na przekazywanie plików. Zapoznaj się z nią, jeśli musisz napisać te funkcje od podstaw: *https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html*.

Złośliwe przekazywanie plików jest tak poważną i istotną kwestią, że jednostka rządu Kanady zajmująca się cyberbezpieczeństwem, Communications Security Establishment of Canada (CSE), stworzyła i udostępniła na otwartej licencji darmowe narzędzie do sprawdzania przekazywanych plików; nosi ono nazwę AssemblyLine (*cyber.gc.ca/en/assemblyline*). Nie udostępnia ono informacji rządowi kanadyjskiemu, jak wiele innych narzędzi dostępnych za darmo w internecie.

Błędy i rejestrowanie zdarzeń

Wszystkie błędy powinny być poprawnie wychwytywane i obsługiwane; na ekranie nie powinno być śladów stosu ani błędów bazy danych. Nie tylko dlatego, że aplikacja powinna wyglądać profesjonalnie i oferować użytkownikom przyjemne doznania, ale również po to, by nie udzielać atakującym dodatkowych informacji, które mogliby wykorzystać podczas przygotowywania ataku na aplikację. W razie błędu aplikacja nie powinna nigdy przechodzić w nieoczekiwany stan. Każda przeprowadzana transakcja powinna być zawsze wycofywana, a wszystko, co mogło być otwarte — „zamykane" (ang. *fail-closed*). Taką sytuację należy też zawsze rejestrować, by w razie wystąpienia incydentu osoby zajmujące się reagowaniem miały materiały do śledztwa, a audytorzy mogli zweryfikować, czy system działa i działał poprawnie.

Twój zespół ds. bezpieczeństwa zapewne dysponuje rozwiązaniem o nazwie SIEM (ang. *Security Incident and Event Management* — system do zarządzania incydentami i przypadkami naruszeń bezpieczeństwa), które pozyskuje wszystkie pliki dzienników (ang. *log files*) wszystkich narzędzi sieciowych. Może też przyjmować dzienniki aplikacji. Porozmawiaj z zespołem, by dowiedzieć się, w jakim formacie chcieliby otrzymywać dzienniki aplikacji, i dopilnuj, by rzeczywiście miały taki format, oraz upewnij się, że system SIEM będzie miał dostęp do dzienników po przeniesieniu aplikacji do środowiska produkcyjnego.

Nigdy nie rejestruj poufnych informacji, takich jak hasła, numery PESEL, całe numery kart kredytowych (zapisywanie ostatnich czterech cyfr jest w porządku), daty urodzenia wraz z imionami i nazwiskami itd. Nie należy logować żadnych kombinacji informacji, które łącznie stanowiłyby dane osobowe (ang. *personally identifiable information* — PII). Jeśli masz wątpliwości, poradź się w zespole ds. prywatności i u analityka biznesowego.

Jeśli w Twojej aplikacji przytrafi się coś, co wygląda na incydent naruszenia bezpieczeństwa, aplikacja powinna nie tylko zarejestrować to zdarzenie, ale również wywołać alert. Może to być wiadomość do zespołu ds. bezpieczeństwa, dyżurnego administratora albo coś innego, co w Twojej organizacji zostanie uznane za właściwe. Zadecyduj o tym z wyprzedzeniem, w fazie ustalania wymagań. Nie wysyłaj wiadomości na osobiste adresy e-mail, ale na konto zespołu lub służbowe, żeby

Kup książkę Poleć książkę

w razie odejścia danej osoby alerty nie trafiały donikąd. Okresowo sprawdzaj tę funkcję, by upewnić się, że nadal działa, bo niewyzwalane lub niedostarczane alerty nikomu nie pomogą.

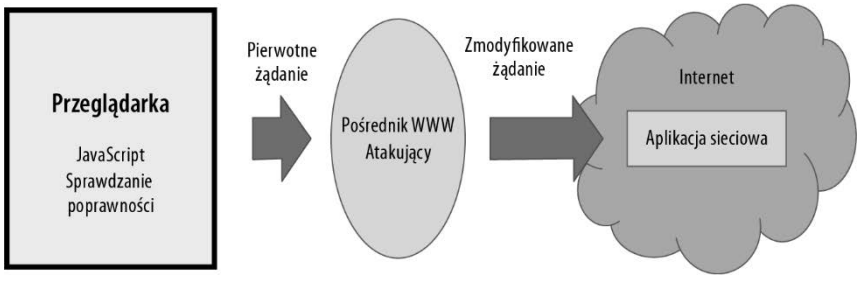
Zdarzenia, które potencjalnie powinny wyzwalać alert bezpieczeństwa, to np.: przejście aplikacji w nieoczekiwany stan, odmowa dostępu do funkcji lub dokumentu, próba obejścia przepływu pracy, przekroczenie przez użytkownika limitów przydziału użycia (np. 10 prób logowania w ciągu sekundy albo 100 w ciągu godziny — ludzie nie zachowują się w ten sposób), awaria aplikacji, wykorzystanie przez użytkownika pewnej (dużej) przepustowości, zasobów CPU lub miejsca przechowywania, wywołanie zablokowanych przez Ciebie zleceń HTTP (ang. *HTTP verbs*). O tym, co będzie odpowiednie dla Twojej organizacji i aplikacji, zadecyduj razem z analitykami biznesowymi i zespołem ds. bezpieczeństwa.

Wskazówka **Aby uzyskać więcej informacji na temat alertów i rejestrowania zdarzeń, odwiedź stronę *cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html*.**

Weryfikowanie i oczyszczanie danych wejściowych

JavaScript to szczególnego typu język programowania, który działa w przeglądarce. Jest ona często zwana stroną klienta, ponieważ ludzie (klienci) muszą korzystać z aplikacji sieciowych przy użyciu przeglądarki, znajdującej się na *ich* komputerze, a nie na *Twoich* serwerach. Wszystkie inne języki programowania są wykonywane po stronie serwera: na serwerze WWW, w rozwiązaniu PaaS (ang. *Platform as a Service* — platforma jako usługa, od dostawcy chmury obliczeniowej) lub w kontenerze (więcej na ten temat w dalszej części książki).

Podczas korzystania z aplikacji sieciowej przez pośrednik WWW (ang. *web proxy*) zaczyna on działanie *po* wykonaniu kodu JavaScriptu aplikacji, ale *przed* wysłaniem żądania przez sieć do serwera WWW. Osoba korzystająca z pośrednika WWW może z łatwością zmienić żądanie HTTP po tym, jak wyjdzie z komputera, i w ten sposób obejść weryfikację czy oczyszczanie danych wejściowych zapisane w kodzie JavaScriptu. Na rysunku 2.4 pokazano wyraźnie, dlaczego weryfikacja pod kątem bezpieczeństwa i oczyszczanie muszą być przeprowadzane po stronie serwera — obchodzenie zabezpieczeń kodu JavaScriptu jest aż nazbyt łatwe.



Rysunek 2.4. Ilustracja przechwytywania ruchu internetowego przez pośrednik WWW

Drugą rzeczą, o której należy pamiętać przy przeprowadzaniu weryfikacji, jest korzystanie z tzw. listy wyrażeń „przyjmowanych" (ang. *accepted list*) czy „aprobowanych" (ang. *approved list*),

Kup książkę Poleć książkę

a nie „blokowanych". Okazuje się, że złośliwe podmioty i testerzy penetracyjni potrafią relatywnie łatwo i systematycznie obchodzić listy blokowanych wyrażeń dzięki kreatywnemu użyciu opcji wejściowych, które nie zostały zablokowane. Liczba sposobów, na jakie można wprowadzić pojedynczy cudzysłów (znak najczęściej używany w atakach polegających na wstrzyknięciu kodu SQL-a, ang. *SQL injection*) w pole danych wejściowych, jest doprawdy zadziwiająca. Zamiast próbować blokować długą listę różnego typu ataków, utwórz listę tego, co *jest* dozwolone, a potem odrzucaj resztę. Twoja „biała lista" powinna być zapisana za pomocą wyrażeń regularnych (ang. *regular expressions*, *regex*), a wszystko, co nie będzie pasowało, jest niedozwolone. Jeśli np. chcesz zezwolić na użycie w nazwach użytkowników tylko znaków a – z i A – Z, możesz użyć następującego wyrażenia: `^[a-zA-Z]{1,10}$`.

Ostrzeżenie **Wyrażenia regularne bywają celem ataków typu „odmowa usługi" (ang. *denial-of-service* — DOS), ponieważ ich wykonywanie zużywa wiele zasobów. Z tego powodu w aplikacji powinny być wyzwalane alerty w razie wielokrotnego wywoływania tej funkcji.**

Autoryzacja i uwierzytelnianie

Jeśli w aplikacji mają być różne poziomy dostępu dla różnych typów użytkowników, co generalnie określa się jako kontrolę dostępu opartą na rolach (ang. *Role-Based Access Control* — RBAC), wszystko to należy zdefiniować w fazie zbierania wymagań. Z pewnością nikt nie chce, by narzucano mu zmiany w czasie opracowywania systemu, bo przez to praca staje się trudniejsza, niż to konieczne. Wszystko to ma związek z autoryzacją (ang. *authorization* — AuthZ).

W tym samym momencie należy też określić stosowane metody albo systemy uwierzytelniania (ang. *authentication* — AuthN) i ustalania tożsamości (ang. *identity*): w jaki sposób system będzie weryfikował tożsamość użytkowników? Jak będziesz ją śledzić? Czy musisz to robić w obrębie kilku systemów, czy tylko w swojej witrynie? Najlepiej by było podjąć decyzję o tych wszystkich systemach zawczasu, ale można to zrobić jeszcze w fazie projektowania.

Wskazówka **Zaimplementuj zautomatyzowany zestaw testów do weryfikacji, czy Twoja implementacja autoryzacji jest faktycznie taka, jaka powinna być. Zautomatyzowanie tych testów pozwoli je często powtarzać w celu sprawdzenia, czy coś się przypadkiem nie zmieniło.**

Zapytania parametryczne

Gdy aplikacja odwołuje się do bazy danych, instruuje ją, by wykonała działanie w jej imieniu. Może to być żądanie odczytu, zapisu, aktualizacji lub usunięcia danych, ale chodzi o to, że aplikacja komunikuje się z bazą danych *bezpośrednio*. Żadna osoba ani aplikacja nie powinna mieć nigdy możliwości bezpośredniej komunikacji z *Waszą* bazą danych, oprócz *Waszej* aplikacji, *Waszego* zespołu wytwarzającego oprogramowanie czy *Waszego* administratora bazy danych. Jeśli atakujący przeprowadza na aplikację atak SQL lub NoSQL, próbuje komunikować się bezpośrednio z bazą danych i wysyłać jej polecenia zgodne z własnymi życzeniami, a nie instrukcje zaprogramowane dla niej do wykonania. Określa się to jako atak polegający na wstrzyknięciu kodu (ang. *injection attack*) i jest on

Kup książkę Poleć książkę

zagrożeniem numer jeden aplikacji sieciowych (według listy OWASP Top Ten, która zostanie omówiona w rozdziale 5.).

Gdy stosujemy zapytania parametryczne (ang. *parameterized queries*; w języku SQL są one nazywane procedurami składowanymi, ang. *stored procedures*), wysyłamy do bazy danych parametry i nazwę kwerendy, jaką chcemy uruchomić, a nie tworzymy wiersza kodu przez włączenie danych wprowadzonych przez użytkownika w ciąg, który jest potem wysyłany jako polecenie do bazy danych. W przypadku zapytań parametrycznych różnica polega na tym, że jeśli atakujący spróbuje dodać własny kod poprzez dane wejściowe, aplikacja wyśle go w jednym z parametrów i próba się nie powiedzie. Jest tak dlatego, że parametry nie są nigdy interpretowane przez bazę danych jako kod, ale jako dane, co w zasadzie uniemożliwia atak polegający na wstrzyknięciu kodu.

Jeśli aplikacja łączy ciągi danych wprowadzonych przez użytkownika, a potem wysyła je do bazy danych bezpośrednio jako polecenie SQL-a, mówimy, że kod SQL-a jest „wkomponowany" (ang. *inline SQL*). Takie postępowanie prowadzi do podatności na wstrzyknięcie kodu SQL-a. Niezawodną ochronę daje tylko stosowanie w zamian zapytań parametrycznych. Zawsze ich używaj.

Parametry URL

Użytkownicy mają dostęp do paska adresu w przeglądarce. Tak samo atakujący. Nie umieszczaj w parametrach URL żadnych zmiennych istotnych dla aplikacji. Numer identyfikatora można zwiększyć, a wtedy, jeśli aplikacja nie była zabezpieczona przed taką sytuacją, użytkownik może zobaczyć konto innej osoby. Jeśli poufna wartość znajdzie się w parametrze URL, zostanie to zarejestrowane, co również będzie problemem. Manipulowanie wartościami parametrów URL jest bardzo proste, a wymówka, że „użytkownik nie powinien tego robić", nie przekona zespołu reagującego na incydenty. W parametrach URL przekazuj tylko nieistotne wartości, np. to, w jakim języku użytkownik chce przeglądać stronę. Nigdy nie przekazuj w ten sposób czegoś ważnego czy poufnego.

Najmniejsze uprzywilejowanie

Koniecznie stosuj w swojej aplikacji zasadę najmniejszego uprzywilejowania, szczególnie w zakresie dostępu do bazy danych lub interfejsów API. Poza tym aplikacja przy wywoływaniu interfejsów API, zapytań parametrycznych i wszystkich innych wywołań wymagających konta powinna korzystać tylko z konta usługi.

Oto kilka przykładów stosowania zasady najmniejszego uprzywilejowania:

- Konto usługi, z którego aplikacja odwołuje się do bazy danych, powinno mieć uprawnienia tylko do operacji CRUD, a nie prawa właściciela bazy danych (ang. *database owner* — DBO).
- Najlepiej, gdyby były dwa konta usługi: jedno z dostępem tylko do odczytu, dla zapytań typu „select" (wybieranie), a drugie z dostępem do funkcji CRUD, stosowane w razie konieczności wstawiania, aktualizowania lub usuwania danych. Jeśli to możliwe, używaj konta z dostępem tylko do odczytu.
- Utwórz oddzielne konta usługi dla każdego interfejsu API, z którego korzysta aplikacja. Każde z kont powinno mieć jedynie najbardziej niezbędne uprawnienia (np. tylko do wybierania, ang. *select*, lub odczytu, ang. *read*, jeśli dane mają być tylko zwracane, a nie modyfikowane).

Kup książkę Poleć książkę

- Uprawnień do odczytu i do zapisu projektów w bibliotece kodu udziel wyłącznie członkom zespołu. Wszyscy pozostali powinni mieć tylko dostęp do odczytu albo w ogóle nie powinni mieć dostępu, jeśli kod jest potencjalnie dość poufny lub wartościowy.

Wskazówka **Korzystanie tam, gdzie to tylko możliwe, z konta usługi z prawem tylko do odczytu zamiast ze wszystkimi uprawnieniami do operacji CRUD jest implementacją zasady „najmniejszego uprzywilejowania”.**

Lista kontrolna wymagań

Poniżej znajduje się lista kontrolna, której możesz użyć we wszystkich projektach aplikacji sieciowych. Każde z tych wymagań można zastosować w każdej takiej aplikacji. Proponuję uwzględnić je wszystkie jako wariant minimum i dodać własne, odpowiadające swoim unikatowym potrzebom biznesowym.

- Szyfruj wszystkie dane magazynowane (w bazie danych).
- Szyfruj wszystkie dane w trakcie przesyłania (po drodze od użytkownika, bazy danych, interfejsu API itp. i z powrotem).
- Niczemu nie ufaj: sprawdzaj (i oczyszczaj, jeśli zachodzą ku temu specjalne powody) wszystkie dane, nawet z własnej bazy danych.
- Koduj wszystkie dane wyjściowe (a w razie potrzeby dodawaj do nich znaki ucieczki).
- Skanuj wszystkie biblioteki i komponenty innych podmiotów pod kątem znanych luk w zabezpieczeniach, zanim ich użyjesz, a po wykorzystaniu rób to regularnie (cały czas pojawiają się nowe podatności i nowe wersje oprogramowania).
- Używaj wszystkich możliwych do zastosowania nagłówków bezpieczeństwa.
- Stosuj odpowiednie ustawienia bezpiecznych ciasteczek.
- Sklasyfikuj i oznacz wszystkie dane przechowywane, zbierane lub tworzone w aplikacji.
- Skracaj i sól wszystkie hasła użytkowników. Niech sól ma co najmniej 28 znaków.
- Wszystkie wpisy tajne aplikacji przechowuj w magazynie wpisów tajnych.
- Dopilnuj, by wszystkie używane w aplikacji konta były kontami usług (a nie osób).
- Niech wszyscy członkowie Twojego zespołu korzystają z menedżerów haseł i nigdy nie używają wielokrotnie tych samych haseł.
- Włącz we wszystkich ważnych kontach uwierzytelnianie MFA.
- Nie wymuszaj zmiany haseł według harmonogramu, ale wyłącznie po naruszeniu zabezpieczeń albo stwierdzeniu podejrzanej aktywności.
- Pozwalaj na dostęp do witryn publicznych (internetowych) tylko za pośrednictwem protokołu HTTPS. Przekierowuj ruch HTTP na HTTPS. Najlepiej stosować to zalecenie zarówno do aplikacji wewnętrznych, jak i zewnętrznych.

Kup książkę Poleć książkę

- Do szyfrowania koniecznie stosuj najnowszą wersję protokołu TLS (obecnie 1.3).
- Nigdy nie wpisuj wartości bezpośrednio w kodzie. *W żadnym razie.*
- Nigdy nie umieszczaj poufnych informacji w komentarzach. Dotyczy to ciągów połączeń i haseł; ich miejsce jest w magazynie wpisów tajnych.
- Korzystaj z zawartych na platformie programistycznej funkcji bezpieczeństwa, np. dotyczących kryptografii i szyfrowania, funkcji zarządzania sesją czy oczyszczania danych wejściowych. Nigdy nie pisz własnych, jeśli są dostępne na platformie.
- Używaj tylko ostatniej (albo przedostatniej) wersji platformy programistycznej i pilnuj jej aktualizacji. Dług techniczny = dług bezpieczeństwa.
- Jeśli pozwalasz na przekazywanie plików, koniecznie przestrzegaj wskazówek organizacji OWASP dla tej bardzo ryzykownej operacji. Obejmuje to również skanowanie wszystkich przekazanych plików za pomocą skanera, np. narzędzia AssemblyLine dostępnego za darmo na stronie Communications Security Establishment of Canada (CSE).
- Koniecznie rejestruj wszystkie błędy (ale bez poufnych informacji), a jeśli pojawi się błąd w zabezpieczeniach, wywołaj alert.
- Dopilnuj, by całą weryfikację (i oczyszczanie) danych wejściowych przeprowadzać po stronie serwera przy użyciu listy wyrażeń przyjmowanych czy aprobowanych (a nie blokowanych).
- Koniecznie przeprowadzaj testy bezpieczeństwa aplikacji przed jej wydaniem (więcej na ten temat w następnych rozdziałach).
- Przeprowadzaj modelowanie zagrożeń aplikacji przed jej wydaniem. Więcej na ten temat dowiesz się z rozdziału 3., z podrozdziału „Modelowanie zagrożeń”.
- Dokonuj inspekcji kodu (szczególnie funkcji bezpieczeństwa) aplikacji przed jej wydaniem.
- Dopilnuj, by aplikacja wychwytywała wszystkie błędy i w razie błędu powracała do bezpiecznego stanu albo była zamykana (by nigdy nie znajdowała się w nieprzewidzianym stanie).
- Dopilnuj, by o wszystkich błędach powiadamiać użytkowników w sposób ogólny, nie ujawniając stanu stosu, błędu zapytania czy innych szczegółów technicznych.
- W wymaganiach projektu zdefiniuj szczegóły dostępu opartego na rolach.
- Koniecznie zdefiniuj w wymaganiach projektu szczegółowe informacje o metodach uwierzytelniania i systemach tożsamości.
- Używaj tylko zapytań parametrycznych, nigdy nie stosuj wkomponowywanych poleceń SQL/NoSQL.
- Nie przekazuj w parametrach URL zmiennych, które miałyby jakieś znaczenie.
- Zgodnie z zasadami bezpieczeństwa koniecznie stosuj w aplikacji regułę najmniejszego uprzywilejowania, zwłaszcza pod względem dostępu do bazy danych lub interfejsów API.
- Jeśli to możliwe, zawsze minimalizuj powierzchnię ataku.
- Pozwalaj użytkownikom na wklejanie ciągu ze schowka do pola wprowadzania hasła, co pozwoli im korzystać z menedżerów haseł. Nie zezwalaj na stosowanie w przeglądarkach funkcji autouzupełniania, by nie dopuścić do zapisywania w nich haseł.

Kup książkę Poleć książkę

- Nie pozwalaj na buforowanie stron zawierających poufne informacje. Można to wymusić za pomocą nagłówka HTTP `Cache`, mimo że z technicznego punktu widzenia nie jest to nagłówek bezpieczeństwa.
- Dopilnuj, by hasła użytkowników aplikacji były długie, ale niekoniecznie skomplikowane. Im dłuższe, tym lepsze; zachęcaj do stosowania fraz hasłowych.
- Nie zmuszaj użytkowników do zmiany haseł po upływie określonego czasu, chyba że podejrzewasz, że doszło do naruszenia zabezpieczeń.
- Weryfikuj przy użyciu przeznaczonej do tego usługi, czy hasła nowych użytkowników nie zostały wcześniej złamane.

W zależności od tego, co robi aplikacja, być może trzeba będzie dodać kolejne wymagania, a niektóre istniejące usunąć. Celem tego rozdziału jest nakłonienie czytelników do pomyślenia o bezpieczeństwie przy zbieraniu wymagań dla projektu. Jeśli programiści będą od początku wiedzieli, że muszą się stosować do tych wymagań, będziecie na najlepszej drodze do tworzenia bardziej bezpiecznego oprogramowania.

Ćwiczenia

1. Podaj kolejne dwa (jeszcze nie wymienione) potencjalne wymagania dotyczące bezpieczeństwa aplikacji sieciowej.
2. Podaj kolejne dwa potencjalne wymagania dotyczące bezpieczeństwa dla systemu operacyjnego w samochodzie.
3. Podaj kolejne dwa potencjalne wymagania związane z bezpieczeństwem „inteligentnego” tostera.
4. Podaj kolejne dwa potencjalne wymagania dotyczące bezpieczeństwa aplikacji obsługującej karty kredytowe.
5. Które wymaganie dotyczące bezpieczeństwa jest najwartościowsze? Dlaczego przedstawia największą wartość dla Ciebie i (lub) Twojej organizacji?
6. W razie konieczności usunięcia jednego z podanych w tym rozdziale wymagań z założeń projektu aplikacji sieciowej z którego można by zrezygnować? Dlaczego?

Uwaga **Przypomnienie: „klucz odpowiedzi” na końcu książki jest zwięzły. Jeśli masz możliwość omówienia ich ze współpracownikiem albo znajomym, odpowiedzi będą bardziej szczegółowe. Dołącz do prowadzonej przeze mnie na żywo dyskusji nad pytaniami na stronie *youtube.com/shehackspurple* albo obejrzyj później jej zapis wideo.**

Kup książkę Poleć książkę

Kup książkę Poleć książkę


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

Media bezustannie donoszą o spektakularnych wpadkach w zakresie bezpieczeństwa aplikacji. Konsekwencją udanego ataku bywają straty finansowe, kompromitacja, niekiedy zagrożenie bezpieczeństwa narodowego. Aby tego uniknąć, wszyscy zainteresowani — od architekta po użytkownika — powinni stale uaktualniać i stosować w praktyce zasady bezpieczeństwa systemów informatycznych. Jednak szczególna odpowiedzialność spoczywa na projektantach i programistach aplikacji, gdyż podejmowanie działań zabezpieczających na wczesnych etapach opracowywania oprogramowania daje o wiele lepsze rezultaty niż rozwiązywanie problemów w fazie testowania.

To książka przeznaczona dla programistów, projektantów aplikacji i osób odpowiedzialnych za bezpieczeństwo informacji. Jest napisana w sposób bezpośredni, przystępny, bez fachowego żargonu i zawiłości. Zawarte w niej koncepcje bezpiecznego projektowania i programowania wzbogacono o praktyczne kody, ćwiczenia i przykłady. Aby ułatwić zrozumienie przedstawionych treści, posłużono się przykładem Alicji i Boba, których życie zawodowe, a także podejmowane przez nich przedsięwzięcia i realizowane zadania wpływają na decyzje dotyczące bezpieczeństwa aplikacji. Znajdziemy tu również odpowiedzi na wiele pytań nurtujących osoby zaczynające pracę w tej dziedzinie, a liczne wskazówki, wytyczne i opisy dobrych praktyk z pewnością ułatwią poprawne stosowanie zasad bezpieczeństwa w tworzonym oprogramowaniu.

W KSIĄŻCE MIĘDZY INNYMI:

— najważniejsze zasady bezpieczeństwa w koncepcjach projektowych
— wytyczne bezpiecznego programowania
— modelowanie zagrożeń i testowanie
— współczesne zagrożenia dla bezpieczeństwa aplikacji i metody obrony przed nimi
— protokoły bezpieczeństwa dla programistów i personelu informatycznego

Alicja i Bob JUŻ TO WIEDZĄ. BEZPIECZEŃSTWO JEST BEZCENNE!

Tanya Janca, znana jako **SheHacksPurple**, jest doświadczoną programistką specjalizującą się w zagadnieniach bezpieczeństwa aplikacji. Pracowała w startupach, administracji publicznej i ogromnych korporacjach technologicznych. Przeprowadziła setki wykładów i szkoleń na całym świecie. Jest laureatką licznych nagród. Ceni różnorodność, inkluzywność i życzliwość, co widać w jej niezliczonych inicjatywach.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

Sprawdź nasze szkolenia!
SZKOLENIA
AKADEMIA IT & BUSINESS
HELIONSZKOLENIA.PL

INFORMATYKA W NAJLEPSZYM WYDANIU

KOD KORZYŚCI
Sięgnij po więcej! ▶

ISBN 978-83-283-8283-1
9 788328 382831
Cena: 59,00 zł